Does It Have To Be Trees? Data-Driven Dependency Parsing with Incomplete and Noisy Training Data

Dissertation von Kathrin Spreyer

Eingereicht bei der Humanwissenschaftlichen Fakultät
der Universität Potsdam
zur Erlangung des Grades des Doktors der Philosophie

Heidelberg, 11. Oktober 2011

Gutachter:
Prof. Dr. Jonas Kuhn
Prof. Dr. Manfred Stede

Datum der mündlichen Prüfung: 15. Dezember 2011

The research that led to this dissertation was funded by the DFG as part of the collaborative research center SFB 632.

Published online at the
Institutional Repository of the University of Potsdam:
URL http://opus.kobv.de/ubp/volltexte/2012/5749/
URN urn:nbn:de:kobv:517-opus-57498
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-57498

Erklärung

Ich erkläre, dass ich diese Dissertation selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Heidelberg, den 11. Oktober 2011.

Kathrin Spreyer

Abstract

We present a novel approach to training data-driven dependency parsers on incomplete annotations. Our parsers are simple modifications of two well-known dependency parsers, the transition-based Malt parser and the graph-based MST parser. While previous work on parsing with incomplete data has typically couched the task in frameworks of unsupervised or semi-supervised machine learning, we essentially treat it as a supervised problem. In particular, we propose what we call *agnostic* parsers which hide all fragmentation in the training data from their supervised components.

We present experimental results with training data that was obtained by means of *annotation projection*. Annotation projection is a resource-lean technique which allows us to transfer annotations from one language to another within a parallel corpus. However, the output tends to be noisy and incomplete due to cross-lingual non-parallelism and error-prone word alignments. This makes the projected annotations a suitable test bed for our fragment parsers. Our results show that (i) dependency parsers trained on large amounts of projected annotations achieve higher accuracy than the direct projections, and that (ii) our agnostic fragment parsers perform roughly on a par with the original parsers which are trained only on strictly filtered, complete trees. Finally, (iii) when our fragment parsers are trained on artificially fragmented but otherwise gold standard dependencies, the performance loss is moderate even with up to 50% of all edges removed.

Acknowledgments

First of all, I would like to thank my *Doktorvater*, Jonas Kuhn, for his support, encouragement and patience – and for putting up with my stubbornness. He provided the guidance I needed, but at the same time let me pursue things in my own manner, which I greatly appreciate. Needless to say, by sharing his ideas, he greatly contributed to this thesis.

Further, I would like to thank my former colleagues Gerlof Bouma, Lilja Øvrelid, Eleftherios Avramidis, Sina Zarrieß, Wolfgang Seeker for interesting and helpful discussions. Thanks also to Florian Marienfeld and Georg Jähnig for the effort they put into cleaning up the Europarl corpus.

I am also very grateful to Joakim Nivre for an interesting discussion back in 2008, which encouraged me to further pursue the idea of using fragmented parse trees; to Sebastian Padó for making his Europarl gold standard publicly available; and to Yi Zhang for sharing his dependency conversion software for the German treebank. Moreover, I am indebted to people at the Computational Linguistics department at Heidelberg for letting me use their computing resources: Anette Frank, Markus Kirschner and Patrick Simianer.

I would also like to thank the anonymous reviewers at CoNLL 2009, LREC 2010, and COLING 2010 for helpful comments.

Contents

Chapter 1

Introduction

In this dissertation we explore the induction of data-driven dependency parsers in the absence of manually annotated treebanks. Instead, our training data consists of parse tree fragments that are obtained by means of annotation projection within a large multilingual, parallel corpus.

The motivation for this resource-lean approach lies in the notorious shortage of manually annotated corpora, the so-called *resource bottleneck*. Resources such as treebanks are readily available for a handful of languages – foremost English – but for most of the world's languages no such data exists, or only in insufficient quantities. Moreover, considering the effort involved in creating a treebank, it seems unlikely that the situation will improve in the near future: the annotation process alone can take many years, and annotations need to be double-checked in order to ensure the desired quality and consistency; prior to annotation proper, annotation guidelines must be defined which describe linguistically plausible and computationally practical structures, and which are also easy to obey by the annotators. We will come back to the issue of annotation schemes and the role they play in a setup that uses annotation projection.

Thus, while state-of-the-art dependency parsers can achieve astonishing levels of accuracy, the applicability of the pivotal statistical and machine learning methods is dependent on large amounts of training data and therefore limited to the few resource-rich languages. This in turn affects a multitude of higher-level NLP tasks that build upon the dependency analysis produced by the parsers. It is therefore important to find ways of overcoming the resource bottleneck. The challenge can be approached from two perspectives. On the one hand, there are techniques for creating annotated resources (semi-)automatically; annotation projection (introduced in the next section) is one such technique. On the other hand, the parsing methods can be adapted to scenarios with less supervision. Although the abandonment of supervision is almost invariably accompanied by a loss in accuracy, it can be salvaged to some extent by benefiting from a nearly unlimited supply of unlabeled or partially labeled data. We propose a perspective on parsing with partial, projected training data, and we show how the two major paradigms in dependency parsing – graph-based and transition-based parsing – can be modified to the effect that they can handle the incomplete annotations.

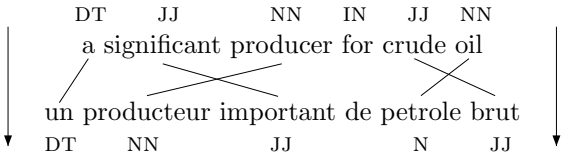


Figure 1.1: Annotation projection of part-of-speech tags from English (top) to French (bottom).

1.1 Annotation Projection

Annotation projection (Yarowsky et al., 2001) is a technique which exploits parallel corpora – text collections that contain translations in several languages – in order to transfer annotations from a (resource-rich) source language to a (resource-scarce) target language. It thereby enables us to bootstrap standalone tools for the target language without having to rely on manually annotated training data in that language. Besides the parallel corpus, the only resources required for annotation projection are the annotations in the source language portion of the corpus. These can be either manual annotations, or even automatic labelings produced by an *existing* source language analyzer of high quality. By choosing a source language for which such a tool is available, the resource bottleneck is sidestepped in that it is deflected from the resource-scarce target language to the resource-rich source language.

Annotation projection uses word-level correspondences (so-called *word alignments*) between the source and target language text as a bridge so that the target language words can "inherit" annotations from the aligned source words. This process induces an automatic labeling for the target language, as can be seen in the example shown in Figure 1.1. The example (from Yarowsky et al. (2001)) shows the projection of part-of-speech (POS) tags from English to French. In the upper half we see the English sentence with POS annotations above. The lower part shows the French translation, and the links between English and French words indicate the word alignments. The POS tag associated with a word in the English sentence is projected to the corresponding word in the French sentence as indicated by the word alignment. For example, consider the English adjective *significant* in the figure. It is aligned with the French *important*. The assumption underlying annotation projection is that the POS tag for *significant* (JJ) is also appropriate for its translation *important*. The French word is therefore tagged with the JJ tag. In the same manner, other words in the French sentence receive a POS tag from their English correspondents.

The word-level alignments that are required to perform projection can be induced from parallel text in an unsupervised manner, typically by a cascade of increasingly complex models (Brown et al., 1993). The success of these models, however, hinges on the availability of substantial amounts of parallel text. Despite being resource-lean with respect to *annotated* resources, annotation projection – including the technique proposed in this thesis – therefore needs to be backed by a large (unannotated) parallel corpus. This is because the accuracy of the word alignment is a crucial factor in the induction of high-quality pro-

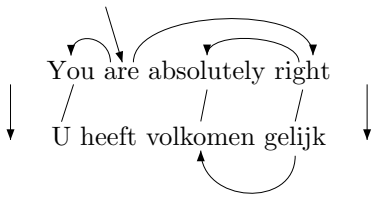


Figure 1.2: Dependency tree projection from English to Dutch.

jected annotations. Moreover, word alignments tend to be sparse in the sense that some words remain unaligned. The words *for* and *de* in Figure 1.1, for instance, do not have alignments in the other language. As a consequence, the English tag for the preposition (IN) cannot be projected, leaving the tag sequence for the French sentence incomplete. This is a minor problem when the projected annotations are *word-based,* as is the case for POS tags: the resulting annotations are interpretable irrespective of the context and do not depend on other words in the sentence.[1] The situation is different when we are dealing with *structured annotations.* Dependency trees, with which we are concerned in this dissertation, fall into this category. Consider the example in Figure 1.2, which shows the projection of a dependency tree from English to Dutch. The projection algorithm projects a dependency edge between two English words h (the *head* of the dependency) and d (the *dependent*) by finding corresponding (aligned) Dutch words h' and d', respectively, and establishing an edge between h' to d' in the Dutch tree under construction. For instance, the edge between *right* and *absolutely* in Figure 1.2 is projected by finding the Dutch correspondent of *right* (*gelijk*) and of *absolutely* (*volkomen*), and introducing an edge from *gelijk* to *volkomen*. In this setting, a missing word alignment (like the one between *are* and *heeft* in Figure 1.2) leads to a target language structure that is not a tree, but rather a set of tree fragments. This is because none of the source language edges involving the unaligned word can be projected. In the example, this affects the edge between *are* and *you*, as well as the one between *are* and *right*. The projected graph consists of three fragments: the isolated nodes *u* and *heeft*, and the subtree rooted in *gelijk*.

The basic assumption underlying the projection of dependency trees is the *Direct Correspondence Assumption* (DCA; Hwa et al., 2005). In simple terms, it states that the syntactic structures of two sentences that are literal translations of each other are isomorphic. We will discuss the DCA in detail in Chapter 3, where we also address the limitations of the assumption, especially its restricted empirical applicability: real-world translations are rarely one hundred percent literal. Thus even if the DCA holds in cases where the translation is in fact

[1] This is true for the annotations as such – the *prediction* of these annotations, as performed by automatic POS taggers, typically *does* require context information in order to resolve ambiguities. Without further pursuing the projection of POS tags in this thesis, we conjecture that our approach to the treatment of incomplete annotations can in principle be applied to that task as well. Moon and Baldridge (2007) present an alternative bootstrapping approach. We discuss their work in Chapter 2.3.

literal, its coverage is severely limited when we take the non-literal translations into account. Given these limitations, one may ask: Why bother at all? Can we gain anything from it? Hwa et al. (2002) provide a compelling argument against dismissing the approach all too early:

> For years, stochastic modeling of language has depended on the linguistically implausible assumptions underlying n-gram models, hidden Markov models, context-free grammars, and the like, with remarkable success. Having made the [underlying assumption] explicit, we would suggest that the right questions are: *to what extent* is it true, and *how useful* is it when it holds? (Hwa et al., 2002, p. 394)

We add to these questions the following: Is annotation projection still useful when the correspondence between source and target language is partial? We will explore this question for syntactic dependencies. Can we train parsers (we call them *fragment parsers*) on projected dependencies even if they do not constitute complete trees? Or are we better off restricting the training data to those sentences that received a *complete* analysis? How do fragment parsers compare to the latter parsers, which are trained only on the limited amount of complete projections? We show that the effective amount of training data is reduced to a minuscule fraction (less than 3%) of the sentences in the parallel corpus when we restrict ourselves to complete target language trees. Moreover, while recall can be improved by including additional, less reliable alignment links, we will see that the quantitative gain is almost completely canceled out by the simultaneous qualitative degradation. We therefore propose a precision–recall tradeoff of a different kind, which we call *partial correspondence projection*: instead of improving recall by admitting weaker alignments, we maintain a precision-oriented alignment filter but do not enforce completeness on the target side. The resulting high-precision tree fragments account for a substantially larger training set for the target language parser, and we present simple modifications of existing data-driven dependency parsers which can handle fragmented training data.

In order to avoid confusion later on, we would like to clarify right from the start that our goal is not so much the development of high-performance projection algorithms, but rather the investigation of the *usefulness* of – potentially imperfect – projected annotations for parser induction. We consider three projection settings that mainly differ in terms of the tradeoff between precision and recall. A number of additional improvements are conceivable beyond these purely structural variations. We ignore these options for the most part in favor of a methodologically more straightforward exploration of the impact of structural incompleteness on parsing performance. We discuss some extensions of the projection algorithms towards the end of Chapter 3.

1.2 Parsing with Tree Fragments

The use of incomplete training data for machine learning and stochastic modeling is not a novel idea, and particularly in the parsing community this issue has received considerable attention, as we will see in Chapter 2. In conjunction with annotation projection, two strategies have been proposed to deal with fragmentation in projected syntax trees. The first solution is to apply *correction rules* to the projected annotations. Such rules infuse target language-specific

knowledge about the appropriate attachment of unaligned words; they can even be designed to amend systematic precision errors in the projection output. The rule-based treatment of partial projections has proven very effective, and parsers trained on the transformed trees exhibit large gains in accuracy. However, the rules need to be hand-crafted anew for every target language and hence turn the conceptually language-independent projection method into a language-specific processing pipeline.

The second popular strategy completes the partial annotations in an unsupervised manner by *marginalizing over all possible completions*. In this scenario, model parameters are typically estimated via Expectation-Maximization (EM). Unfortunately, the EM algorithm is highly sensitive to the initialization of the parameters and its performance is hard to predict.

We present parsers which are also trained on the fragmented trees directly, but rather than explicitly summing over possible completions of the input structures like EM does, we *mask* the fragmentation in such a way that the parser essentially ignores missing edges. In graph-based dependency parsing, where parsing amounts to finding the highest-scoring set of edges that span the entire sentence, this "agnostic" training procedure will operate only on (the scores of) those edges that are present, and it will ensure that the training example receives a higher score than other possible trees that are inconsistent with those edges. The other parser we consider is a transition-based parser. It performs parsing actions like a shift-reduce parser and uses a data-driven component to predict the sequence of locally optimal actions (or *transitions*) at test time. Our fragment-compatible variant of the transition-based parser eliminates from the training data of the internal learner those transitions that concern the attachment of fragment roots. This leads to parsers which are trained only on those transitions that are in fact supported by the projected dependencies, without necessitating explicit provisions for incomplete input structures.

We will show empirically that our projected fragment parsers perform on a par with the corresponding parsers trained only on (the limited amount of) complete trees. Moreover, our analysis reveals that the fragment parsers are in fact superior on longer sentences and long-distance attachments. Both the fragment-trained as well as the tree-trained projected parsers outperform the direct projections that are output by the projection algorithm.

1.3 Evaluation of Projection-based Systems

This thesis touches upon a third topic, besides projection and fragment parsing: the evaluation of projection-based systems. Although annotation projection produces stand-alone target language annotations which can in turn be used to induce stand-alone target language tools, the target annotations are still loosely coupled with the source annotations through the *annotation scheme.* Consider again the example in Figure 1.2. The parse tree for the English sentence adheres to a certain schema that determines how syntactic relations are to be marked up. This includes decisions about the orientation of dependency edges (head-to-dependent versus dependent-to-head), the head word of nominal phrases (noun versus determiner), the depth of the tree (flat versus highly branching structures).

When the English dependency representation is projected to Dutch, the re-

sulting target language annotation – be it complete or partial – conforms to the same annotation scheme. This is desirable in a scenario where no language-specific annotation scheme exists for the target language. It is also unproblematic in an application-oriented setting when the concrete annotation style is irrelevant as long as the actual annotation scheme is known and can thus be referred to.

Complications only arise when we want to *compare* the projected annotations (or the output of parsers which are trained on projected annotations) with a *target language gold standard.* Gold standard annotations for the target language typically conform to an annotation scheme that is likely to differ from the source language scheme, since annotation schemes tend to be language-specific. A direct comparison of the projected annotations with the gold standard would be largely meaningless. The issue is usually addressed by a conversion step which consolidates the annotation schemes during projection (Spreyer and Kuhn, 2009), or it is sidestepped by choosing source and target language data that are annotated according to identical annotation schemes. For instance, Hwa et al. (2005) use the Penn Treebank for English and Chinese. By contrast, we propose an evaluation methodology in which the conversion between annotation schemes is performed *in the gold standard.* In particular, for each source language we create variants of our target language test sets which conform to the source language annotation scheme. For example, a parser for Italian that was trained on annotations projected from English is evaluated against a variant of the Italian gold standard converted to the English (PTB) scheme.

As we will see in Chapter 5, performing the conversion on the test data is preferable over various alternative scenarios because it can be done without introducing any additional noise. By performing the conversion on the test data, we further isolate the problem of diverging annotation schemes in the evaluation step and thus make the rest of the system independent of the presence or absence of an annotation scheme for the target language.

1.4 Overview of the Thesis

In this section we equip the reader with a more detailed picture of what to expect. Recall that our primary goal here is to assess the usefulness of our technique of fragment parsing as opposed to training on (i) a very small amount of conservatively filtered complete trees, and (ii) considerably larger amounts of complete trees that suffer from more noise. Of course there are many conceivable alternative approaches and heuristic improvements, but we will only pursue a few of them in order not to blur the focus on the main research question.

Chapter 2 presents **related work** in annotation projection, weakly-supervised (dependency) parsing, and finally training with incomplete annotations. In Section 2.1 we distinguish word-based from structural annotation projection and focus on the latter, where annotations for individual tokens cannot be (fully) interpreted in isolation, but only in relation to other tokens in the same sentence.

Section 2.2 begins with a brief overview of supervised data-driven dependency parsing, and then turns to weakly supervised and unsupervised approaches, including a thorough discussion of synchronous and multilingual parsing.

Finally, we review related approaches to learning from fragmented annotations in Section 2.3.

Chapter 3 introduces **parallel corpora** and word alignment (Section 3.1) and discusses limitations of the **Direct Correspondence Assumption** (DCA) which forms the basis of annotation projection (Section 3.2). We then present our basic algorithm for **dependency tree projection** in Section 3.3. The algorithm comes in three flavors of varying conservatism: we start out from a very **strict** projection setting which uses only *highly reliable alignment links* and discards dependency graphs unless they form *complete trees.* As we will see, the effective amount of data that passes this filter constitutes only a fraction of the initial data set, and is heavily biased towards simplistic structures. In order to retain a more varied data set of projected structures, we then relax the filter by using not only the highly reliable bidirectional word alignments as anchors for projection, but allowing a **fallback** onto *additional, weaker alignments* that are supported only in one direction (source-to-target or target-to-source). At the same time we still enforce the completeness constraint, that is, we still discard incomplete projections. Like fallback projection, the third variant of our projection algorithm strives to improve the recall of the projected structures. Unlike fallback projection, however, it does not incorporate weaker alignments in order to boost recall. Instead, it simply *lifts the completeness constraint,* thereby admitting fragmented projections in the output. That is, we acknowledge **partial correspondence** between a source sentence and a target sentence, by admitting projected structures even if they do not form trees because some edges are missing.

We first conduct a quantitative assessment of the output of the projection algorithms. It reveals that both fallback projection and partial correspondence projection amend the extreme data loss incurred by strict projection. We also provide a **qualitative evaluation** of our word alignments, the source language parse trees which form the base for projection, and the dependencies induced via projection. When compared to gold standard dependency trees for German, for example, the output of the strict projection algorithm exhibits an f-score of merely 1% due to extremely low recall. With an f-score of 10% the fallback approach shows some improvement, but is by far outperformed by partial correspondence projection, which achieves 46% f-score.

In Chapter 4 we turn to **dependency parsing**. We begin by providing formal definitions and notation for the basic concepts of dependency parsing such as head, dependent and dependency edge (Section 4.1). In Sections 4.2 and 4.4, respectively, we give a detailed description of two concrete (data-driven) dependency parsers: the Malt parser (Nivre et al., 2006), which implements *transition-based dependency parsing*, and the MST parser (McDonald et al., 2005), a representative of the *graph-based dependency parsing* paradigm. Both systems assume that the training data consists of complete trees. If we want to leverage fragmented dependency graphs – as obtained through partial correspondence projection – as training data, we therefore need to make adjustments to the original parsers, and more specifically to the training phase. Since the textual representation format for parse trees expected by the parsers requires that every word be attached to exactly one head (possibly the artificial root

token w_0), we are forced to introduce *spurious attachments* for fragment roots, and our modifications to the original parsers implement an awareness of the fact that these attachments are spurious. This **fragmentation awareness** is then exploited to simply disregard the attachments in question, resulting in parsers that are *agnostic towards missing edges,* as opposed to trying to estimate the plausibility of various possible attachments. We call our fragment-aware variant of the Malt parser **fMalt**, described in Section 4.3. The graph-based counterpart, **fMST**, is described in Section 4.5. We point out that both fragment parsers, despite being trained on partial annotations, produce complete trees as their output. This is because the fragmentation never percolates into the training data of the data-driven components.

Chapter 5 is devoted to the discussion of our **evaluation methodology**. We set out with a summary of the standard evaluation procedure for treebank parsers (Section 5.1) and continue with a general overview of the treebanks we use as **test data** (Section 5.2). The remainder of the chapter addresses two major issues in the evaluation of projection-based systems. First, the **annotation scheme** employed for the source language text is unlikely to coincide with the annotation scheme in the test data for the target language. Section 5.3 explores this topic in great detail, including a **comparison** of the annotation styles employed in our test sets, a proposal for the **consolidation** of diverging annotations, and an empirical assessment of the **learnability** of various annotation schemes.

In Section 5.4 we shift our focus to **variance assessment** and significance testing. In the parsing community, this is commonly tackled using cross-validation over the labeled training set, often in combination with randomized comparisons of the system outputs. In a projection-based setting, however, we are faced with the problem that the training set is typically disjoint from the test set. The training set is derived from a parallel corpus (otherwise no projection could be performed) and is usually not associated with any gold standard annotations. This rules out cross-validation over the training set. On the other hand, the gold standard test data that is used to evaluate projection-based systems is rarely part of a parallel corpus, so cross-validation over the test set is not an option, either. Even if a parallel dataset with manual annotations is available, it would have to be of considerable size in order to yield meaningful results, especially seeing as the training data tend to be noisy. We therefore propose an alternative validation scheme for our projected parsers which (i) does not reduce the amount of test data by partitioning, (ii) does not require parallel test data and is independent of the projection step, and (iii) takes advantage of the fact that training data is cheap and therefore abundant in projection-based settings.

In Chapter 6 we present experimental results. We first describe our **experimental setup** and procedures for **parameter tuning,** as well as simple heuristic **baselines** and **upper bounds** (treebank parsers).

Our experimental results with fMalt and fMST lead us to conclude that tree-based parsers trained on small amounts of data created using a precision-oriented projection algorithm perform roughly on a par with our fragment parsers trained on larger amounts of annotation obtained using partial cor-

respondence projection. For Dutch, for example, the tree-based Malt parser projected from English achieves an unlabeled attachment score of 73.1%, while the corresponding fMalt parser achieves 74.3%. The tree-based MST parser for the same language pair reaches 74.0%, outperforming fMST (73.3%) by a small margin.

Chapter 7 provides a detailed **error analysis** for the parsers evaluated in Chapter 6. We find that our fragment parsers largely outperform the tree-based parsers on *longer sentences* as well as *longer dependencies*. We further analyze the results relative to (gold standard) dependency type and word class, and discuss the dependency analyses predicted by our parsers for a concrete example.

Chapter 8 concludes with a summary and a discussion of future directions.

Chapter 2

Related Work

In this chapter we give an overview of related work. In doing so, we start out with an overview of work in annotation projection. We then turn to data-driven dependency parsing with a special focus on multilingual approaches, which are often closely related to or based in the annotation projection framework. Roughly speaking, the difference between multilingual learning on the one hand and annotation projection on the other hand is that the latter treats the word-aligned source language analysis as a hard constraint on the target language annotation that is being induced, whereas multilingual learning paradigms regard the source annotations as soft constraints which merely help steer model parameters in the right direction.

2.1 Annotation Projection

Annotation projection as a means to address the resource bottleneck for less researched languages was first introduced in the seminal works of Yarowsky and colleagues (Yarowsky et al., 2001; Yarowsky and Ngai, 2001). They employ annotation projection from English to induce stand-alone part-of-speech taggers, base NP bracketers, named-entity taggers and morphological analyzers for French, Chinese, Czech and Spanish. Furthermore, they present robust training methods that are capable of overcoming the noise in the automatic, direct projections. For POS tagging, this kind of robustness is achieved by reinforcing the bias towards the majority tag in the lexical prior model $\mathrm{P}(t|w)$, and subsequently weighting the contribution of each training sentence to the tag sequence model $\mathrm{P}(t_i|t_{i-1}, \ldots)$ proportional to its alignment score on the one hand, and its agreement with the biased lexical priors on the other hand. These re-estimation techniques are thoroughly discussed in Yarowsky and Ngai (2001). For improved NP bracketing accuracy, Yarowsky et al. (2001) exclude sentences with low alignment scores from the training data altogether. The performance of the morphological analyzer is boosted by the use of multiple translations on the source language side: redundant repeated, identical translations increase the confidence in those inflection–root pairs, whereas differences among the translations serve to improve coverage.

In the remainder of this section, we provide an overview of annotation projection approaches that followed the initial proposal in Yarowsky et al. (2001).

We distinguish between the projection of *word-based* and *structured* annotations. While word-based annotations allow for a straightforward isolation of high-precision projected data points (Section 2.1.1), structured annotations are typically subject to wellformedness constraints which on the one hand may facilitate the assessment of the plausibility of the projected annotations; on the other hand, such constraints also tend to complicate the filtering of unreliable data points (Section 2.1.2).

Since the basic principle of annotation projection is so generally applicable, differences between the works discussed below are often confined to task-specific idiosyncrasies, especially in the word-based projection paradigm. We will therefore have little to say about many articles mentioned in the next section, but list the prominent research for the sake of completeness and refer the interested reader to those articles for more details.

2.1.1 Word-based annotation projection

The projection of part-of-speech tags, pioneered by Yarowsky et al. (2001), has since been performed – at varying levels of tagset granularity, and often in tandem with other lexical and morphological information, such as case or (grammatical) gender – for Swedish (Borin, 2002), Czech and French (Drábek and Yarowsky, 2005), Polish (Ozdowska, 2006), Romanian, Kurdish, and Spanish (Cucerzan and Yarowsky, 2002, 2003), and French (Probst, 2003). The method has even been applied in the monolingual setting. For instance, Moon and Baldridge (2007) induce a POS tagger for Middle English via projection from modern English in a parallel diachronic corpus.

Other word-based annotation tasks for which annotation projection has been explored include the acquisition of word senses and semantic lexicons (Diab and Resnik, 2002; Bentivogli and Pianta, 2005; Padó and Lapata, 2005a), verb classification (Merlo et al., 2002), mention detection (Zitouni and Florian, 2008), temporal analysis (Saquete et al., 2006; Spreyer and Frank, 2008), information extraction (Riloff et al., 2002), identification of verb arguments (Bouma et al., 2008), or relation extraction (Kim et al., 2010).

The projection of annotations across parallel corpora is inherently noisy due to cross-language divergences (even in relatively literal translations) and errors in the automatic word alignment (cf. Chapter 3). This problem is commonly addressed by aggressive noise filters that identify and discard unreliable data points. Various filtering criteria have been used, for example Giza alignment scores (Yarowsky et al., 2001; Yarowsky and Ngai, 2001; Spreyer and Frank, 2008), consensus of multiple source languages (Bouma et al., 2008), alignment topology (Hwa et al., 2005; Spreyer and Frank, 2008), or the confidence of the source language tools (Kim et al., 2010).

2.1.2 Projection of structured annotations

In contrast to word-based annotations, structured (e.g., hierarchical) annotations often cannot be interpreted on a word-by-word basis, but rather connect words to form larger, more complex units. Examples include NP bracketing (chunking), semantic role labeling, and of course syntactic parsing. In these tasks, word labels often cannot be introduced in isolation without affecting the

wellformedness of the annotation as a whole. This means that noise filters typically have to discard entire sentences if some part of the annotation is deemed unreliable – even if the rest of the sentence could be labeled with high confidence.

A notable exception to this all-or-nothing approach has been presented in Padó and Lapata (2005b) and Padó and Lapata (2006), who automatically construct *constituent alignments* from sparse, high-precision word-level alignments in order to project semantic roles. Semantic roles are frequently assigned to contiguous spans of multiple words rather than a single word, thus constituent alignment captures the target unit of projection more adequately than mere word alignment. Constituent alignments can be computed locally (Padó and Lapata, 2005b) or globally (Padó and Lapata, 2006).

Of course, since dependency relations (introduced below in Section 2.2 and more thoroughly in Chapter 4) hold between individual words, word alignments are indeed the suitable link between source and target language units for the purpose of *dependency tree projection.* Hwa et al. (2005) (also Hwa et al., 2002) were the first to project dependency trees from English to Spanish and Chinese. They identify unreliable target parses (as a whole) on the basis of the number of unaligned or over-aligned words. In addition, they manipulate the tree structures to accommodate non-isomorphic sentences. Systematic non-parallelism between source and target language is subsequently addressed by hand-crafted correction rules in a post-projection step. These rules account for an enormous increase in the unlabeled f-score of the direct projections from 33.9 to 65.7 for Spanish and from 26.3 to 52.4 for Chinese. But they need to be designed anew for every target language, which is time-consuming[1] and, more importantly, requires knowledge of that language.

Ozdowska (2006) projects dependencies from English and French to Polish, and finds that the choice of source language makes no distinct difference. Unfortunately, she restricts the evaluation to precision and does not report recall figures.

Wróblewska and Frank (2009) present a framework for the projection of LFG f-structures (Bresnan, 2001) from English to Polish. F-structures basically encode syntactic dependencies, but do not necessarily encode dependency *trees* because of the possibility of structure sharing. Like Hwa et al. (2005), Wróblewska and Frank (2009) define post-projection transformations that implement linguistic knowledge specific to the target language. A notable difference with respect to the projection of plain dependencies is that an algorithm for f-structure projection (or, alternatively, the post-projection rules) needs to account for pro-drop phenomena, since unrealized pronouns are made explicit in the f-structure analysis. In the standard data sets for dependency parsing, dropped pronouns are not represented at all and hence do not pose a problem for projection into a pro-drop language.[2] In contrast to many other projection approaches, Wróblewska and Frank (2009) argue for the use of unidirectional alignments (cf. Section 3.1.2). They base this decision on the nature of the language pair: The source language, English, is an analytic language with impoverished morphology and the heavy use of function words. Polish, on the

[1] Hwa et al. (2005) mention an upper bound of one month for the design of the set of correction rules for one language. However, this assumes that a (linguistically trained) native speaker of the target language is available.

[2] Note, however, that projection in the *opposite* direction would systematically leave certain NPs unattached.

other hand, is a highly inflecting language and uses case marking in many cases that would be realized in analytic expressions in English. A unidirectional word alignment, permitting one-to-many links, can be expected to capture such correspondences, whereas they are inevitably lost when only bidirectional alignments are considered. The projected f-structures achieve an unlabeled f-score of 51% (labeled: 50%), and 63.5% when the correction rules are applied.

Ganchev et al. (2009) project English dependencies to Spanish and Bulgarian. However, rather than using the projected annotations directly to train a supervised parser, they accommodate uncertainty and partial correspondence by interpreting the English source analysis as constraints on the posterior expectations derived by the EM algorithm during the E-step (Graça et al., 2008). We will come back to this work in Section 2.3, where we discuss research that deals with fragmented training data. Similar in spirit to Hwa et al. (2005), Ganchev et al. (2009) introduce rules to deal with differences in annotation conventions between the treebanks. The (technically unsupervised) parsers that are obtained in this way generally outperform the corresponding supervised baselines trained on the hard (direct or transformed) projections.

Jiang and Liu (2009) also move away from hard projection and instead project trees from English to Chinese by searching for the Chinese tree that is *most consistent* with the English source tree, where consistency is defined as an aggregated score over the edges in the Chinese candidate tree, anchored in lexical translation probablities. They subsequently filter the projected trees by means of a threshold for the normalized consistency score, and then train the MST parser (cf. Section 4.4) on the remaining sentences.

The latter two approaches are both closely related to multilingual learning methods for dependency parsing, which we will discuss below.

We conclude the discussion of related work in annotation projection by briefly mentioning a proposal for treebank transfer between comparable corpora (Jansche, 2005), which dispenses with the need for strictly parallel data by treating target language trees as latent variables that are conditional on a monolingual n-gram model and a syntactic mapping from source language trees to target language trees. However, Jansche (2005) assumes that the syntactic mapping is given. Designing such a mapping is a non-trivial task in its own right.

2.2 Dependency Parsing

The notion of syntactic dependencies as asymmetrical binary relations between words dates back many centuries (cf. Kruijff (2002) for a historical overview of dependency grammar). The formal framework of dependency grammar is attributed to Tesnière (1959) and Mel'čuk (1988), and has become popular for languages with a high degree of non-configurationality, especially the Slavic languages. This is because, in contrast to constituent-based representations of syntactic structure, dependency grammar does not assume phrasal nodes and thereby facilitates a straightforward representation of so-called non-projective structures, which contain crossing dependencies.

Dependency grammar has since spawned many grammar formalisms and syntactic theories. Their discussion is beyond the scope of this dissertation. The interested reader is referred to Nivre (2006) and Kruijff (2002) for a list of

references.

2.2.1 Data-driven dependency parsing

In computational linguistics and the parsing community in particular, dependency representations have only recently received attention after efficient parsing algorithms have been proven to exist and implemented (Eisner, 1996; Yamada and Matsumoto, 2003; Nivre, 2008). In the wake of the CoNLL Shared Task on multilingual data-driven dependency parsing (Buchholz and Marsi, 2006) and subsequent shared tasks that focused on domain adaptation for dependency parsers (Nivre et al., 2007) and joint dependency parsing and semantic role labeling (Surdeanu et al., 2008; Hajič et al., 2009), a wealth of systems have been presented which achieve state-of-the-art performance on the emerging benchmark data sets.

Approaches to data-driven dependency parsing can be divided into two broad paradigms: *transition-based* and *graph-based* (McDonald and Nivre, 2007). Transition-based parsers construct a dependency tree in a step-wise fashion that is reminiscent of standard shift-reduce parsing algorithms for context-free parsing. They traverse the input sentence, typically from left to right, and perform parsing actions (or *transitions*) which either add a dependency arc, or push the current word onto a stack of partially processed words, to be attached at a later point in time. The data-driven component of such parsers is the decision model, which determines the locally optimal parser action on the basis of words in the input, on the stack, and previously constructed edges. Yamada and Matsumoto (2003) first proposed a transition-based parser which produces an unlabeled dependency tree in multiple passes over the input sentence, using SVMs to predict transitions. The Malt parser (Nivre et al., 2006), by contrast, is fully incremental in that it only makes a single pass over the input. Moreover, it produces labeled dependencies.

Graph-based methods, on the other hand, assume a global perspective on the parsing problem. They consider all possible dependency relations that could make up a well-formed dependency tree for a given sentence, and choose the tree that receives the highest aggregated score, which is obtained as a function of its component scores (e.g., individual arc scores in an arc-factored model). The models of Eisner (1996) fall into this category of graph-based models. More recently, the MST parser (McDonald et al., 2005, 2006; McDonald and Pereira, 2006) has been widely used. Given a sentence, the MST parser finds the maximum spanning tree for this sentence. The training data is used to adjust the scores assigned to the candidate trees, such that the margin between the correct tree and the other candidates is maximized. Computational speed-ups for MST parsing have been proposed in terms of hash kernels and parallelization across several CPUs (Bohnet, 2010), and in terms of arc filters that prune the search space by excluding unlikely edges (Bergsma and Cherry, 2010).

Other graph-based dependency parsers include the system of Carreras et al. (2006), which extends the feature set of the MST parser, or parsing in the belief propagation framework (Smith and Eisner, 2008).

Graph-based and transition-based parsers are known to have complementary strengths (McDonald and Nivre, 2007). Consequently, there have been proposals to integrate the two paradigms. Nivre and McDonald (2008) introduced a technique now known as *parser stacking,* in which the predictions of one parser

are included as features in the training phase of the other parser. An alternative strategy for the combination of graph-based and transition-based parsers has been proposed by Zhang and Clark (2008). They employ a beam search decoder to find the tree that achieves the highest *joint* score, expressed as the sum of the independent scores from both parsers.

2.2.2 Weakly supervised approaches

Research in the field of unsupervised and weakly supervised parsing ranges from various forms of EM training (Pereira and Schabes, 1992; Klein and Manning, 2004; Smith and Eisner, 2004, 2005) and Generalized Expectation constraints[3] for discriminative models (Druck et al., 2009) over bootstrapping approaches like self-training (McClosky et al., 2006; Smith and Eisner, 2007) or co-training (Søgaard and Rishøj, 2010) to feature-based enhancements of discriminative reranking models (Koo et al., 2008) and the application of semi-supervised SVMs (Wang et al., 2008).

The partial correspondence method we present in this thesis is compatible with such approaches and can be combined with other weakly supervised machine learning schemes.

2.2.3 Synchronous and multilingual parsing

Synchronous parsing is an area of NLP that deals with parsing systems which infer the syntactic structure of parallel texts in lock step and simultaneously infer the (hierarchical) alignment between these structures (Melamed, 2003), typically in an unsupervised fashion.

Melamed (2003) proposes a very general framework for synchronous parsing. His *Multitext Grammars* (MTGs) are expressive enough to account for various types of translation mismatches, while remaining computationally tractable. MTG subsumes many previous synchronous parsing approaches, including Inversion Transduction Grammars (Wu, 1997) and Alshawi's finite-state head transducers for synchronous dependency parsing (Alshawi et al., 2000). The left-hand side of MTG productions is a vector of non-terminals, with one component per language. The right-hand side of such a rule is either a vector of terminal symbols (including the empty word ϵ), or it consist of a vector of monolingual right-hand sides and a vector of permutations which specify the surface order of symbols in the respective right-hand sides. Shieber and Schabes (1990) present a similar formalism for Tree-Adjoining Grammar (Joshi, 1985).

Using the MTG framework, Smith and Smith (2004) improve monolingual Korean parsing by combining a state-of-the-art statistical parser for English with an unlexicalized PCFG for Korean. They train the latter on only a small amount of annotated data and benefit from the performance of the English parser by jointly maximizing the scores of the English tree, the Korean tree and the word alignment. They report small, but significant improvements over the monolingual Korean model. Taking this line of thought one step further, Burkett and Klein (2008) improve monolingual parsing performance for *both* languages by simultaneously reranking the n-best candidates of the source and target language parsers using bilingual features that infer the most likely tree

[3] See Mann and McCallum (2008).

alignment, given the sentence pair and word alignment. They recently extended this method for a semi-supervised scenario which uses automatically annotated parallel parse trees rather than relying on hand-annotated bitext (Burkett et al., 2010).

The transition between synchronous parsing and syntax-based machine translation (Galley et al., 2004; Zollmann et al., 2006) is seamless. For more details in this field see, for instance, Yamada and Knight (2001); Gildea (2003); Eisner (2003); Blunsom et al. (2009).

Multilingual learning is a technique that is closely related to annotation projection, but like synchronous parsing it is most commonly formulated in the context of syntax-based MT. In contrast to synchronous parsing, it makes weaker isomorphy assumptions about the source and target annotations, and pursues a feature-driven rather than a structural incorporation of parallel information to jointly model the syntactic structure in two or more languages. While the models described in this section rely on information from both languages at training time, they are typically only provided with target language text at test time. This means that the role of the aligned source language data amounts to guiding parameter search to good local optima.

One of the first to apply the multilingual learning paradigm to parsing was Kuhn (2004). He proposes an unsupervised method for PCFG induction which uses word alignments with a source language to derive a prior distribution over the PCFG parameters for the target language. In particular, the prior is designed to discourage certain string spans as constituent candidates. A similar, but more complex approach is pursued by Snyder et al. (2009), who jointly model the source and target language parse trees and tree alignments between them. The model extends the Constituent-Context Model (CCM) of Klein and Manning (2002) by duplicating the original constituent and context distributions of the CCM for each language so as to model monolingual preferences, and introducing a so-called coupling parameter which captures the compatibility of aligned constituent yield pairs.

For dependency parsing, Smith and Eisner (2006) introduce *quasi-synchronous grammars* (QGs), which are based on the unsupervised Dependency Model with Valence (DMV) of Klein and Manning (2004). In addition to the monolingual parameters of the DMV, QGs are equipped with bilingual parameters that model lexical translation probabilities, and a probability distribution over pairs of subtrees (so-called configurations). The latter explicitly allows structural divergences between source and target trees.

Smith and Eisner (2009) demonstrate how QGs can be used for parser adaptation and parser projection. In the projection setting, they train a QG which is conditioned on the source sentence and source tree; at test time, they back off from conditioning in order to obtain a truly monolingual parser. Conditioning the model on the source language information stirs the EM training procedure towards linguistically adequate target language structures without enforcing the DCA. In experiments with English–German and English–Spanish they show that the "projected" QG parser substantially outperforms a graph-based dependency parser trained on high-precision hard projections, with unlabeled attachment scores of 68.5 versus 66.2 (German) and 64.8 versus 59.1 (Spanish), respectively. They also compare the hard projection approach – which discards sentences with incomplete projected dependencies – to an EM-based variant of hard projection which uses the available high-precision dependencies to compute

expected counts for the remaining dependencies. This method is closely related to the research presented in this thesis; the difference between our approach and that of Smith and Eisner (2009) is that they explicitly model the uncertainty introduced by missing edges, while we deliberately ignore it. With attachment scores of 58.6 (German) and 53.0 (Spanish), their EM-based projected parser turns out to perform poorly in comparison to both the QG model and the supervised parser trained on aggressively filtered projected dependencies. As we will see, our technique for exploiting incomplete projection leads to parsers that achieve scores within 0.7 percentage points of the tree-oriented (aggressively filtered) counterparts, and in fact offers significant improvements for some of the language pairs we consider (including English–German).

2.3 Learning From Fragmented Annotations

This section outlines approaches that are more directly related to our fragment parsing proposal in that they involve training on incomplete – more specifically, fragmented – annotations, while still aiming at producing complete output at test time.

Eisner and Smith (2005, 2009) and Dreyer et al. (2006) suggest *vine parsing*. Vine parsing incorporates hard constraints on dependency length, which increases parsing speed, but may result in fragmented parses. There are two major differences between vine parsing and our approach of training on fragmented trees. First, vine parsers incorporate a probabilistic finite-state automaton which explicitly models sequences of fragment roots. Secondly, vine parsers produce fragmented output, whereas our parsers – despite being trained on fragments – still strive to build complete trees for new sentences.

Moon and Baldridge (2007) project POS tags conservatively and then train a simple bigram tagger on the sequences of target language words that received a tag this way. Using this tagger, they relabel the target text so as to obtain complete training data for a more sophisticated tagger. In an out-of-domain evaluation, the accuracy of their bootstrapped tagger (62%) surpasses that of a state-of-the-art supervised tagger for modern English (the "source" language) when applied to Middle English ("target" language) text (56%). The bootstrapped tagger also outperforms the initial bigram tagger (58%) that was trained on the partial, projected tag sequences. In contrast to our work, the training of the bigram tagger from the tagged (sub-)sequences does not use the unlabeled word sequences *at all*, while our fragment parsers do make use of unattached (i.e., unlabeled) words as lexical context.

Hwa (1999) exploits partially labeled data for domain adaptation. In particular, she compares two ways of training a parser on sparsely bracketed sentences from the new domain. The first strategy employs the Inside-Outside algorithm (Pereira and Schabes, 1992) to induce a grammar from the partial parse trees from scratch. The alternative and more effective strategy first trains a supervised parser on fully labeled, but out-of-domain data, and then uses that grammar to initialize the Inside-Outside algorithm.

Ganchev et al. (2009) (discussed in Section 2.1) require expectations to agree (approximately) with those parts of the structure that are present. Specifically, the learner is instructed to favor analyses that agree with *the majority* of the projected edges. This allows the learner to deviate from projected structures

if evidence from other training examples or previous iterations indicates that some of the projected information might be incorrect. At the same time, the expectation constraints say nothing about *missing* edges; missing structure is induced via the iterative EM procedure. Similarly, Smith and Eisner (2009) (discussed in the previous section) pursue the EM-alternative to our "agnostic" fragment parsing method.

Our proposal is closest to that of Tsuboi et al. (2008) and Clark and Curran (2006). They present similar methods for training with partial annotations, the former for sequential data, the latter for dependency trees. What is common to these two approaches – and what sets them apart from ours – is that they explicitly marginalize out the unknown parts of the training annotations. While Tsuboi et al. (2008) pursue this strategy in the framework of Conditional Random Fields (Lafferty et al., 2001), Clark and Curran (2006) use partial training data extracted from CCG lexical categories for the adaptation of a log-linear dependency model (Clark and Curran, 2004) to new domains. The lexical categories contain a considerable amount of information about *possible* dependencies, and as such can be considered an underspecified representation of the dependency tree. The rationale in training with the underspecified set of dependencies is then to restrict the training data to those dependencies that occur in k% of the derivations licensed by the lexical items. During training, missing attachments are addressed by summing over all trees *consistent* with the partial training example. As we will see in Chapter 4, this is very similar to our fragment-aware variant of the graph-based MST parser. The crucial difference between our projected, fragmented training data and Clark and Curran's lexical categories lies in the nature of the uncertainty they convey: dependencies that cannot be inferred from the lexical categories are indicative of true attachment ambiguities. By contrast, the fragmentation introduced during projection is due to missing word alignments and therefore not syntactically motivated.

Chapter 3

Cross-lingual Projection of Syntactic Dependencies

In the introduction, we have already discussed the dependence of supervised statistical NLP paradigms on labeled gold standards, and the bottleneck that arises from the limited availability of suitable resources. While it is true that a range of corpora, treebanks and tools for automatic annotation exist for English and a handful of other languages (e.g., German, Spanish, Japanese, Czech, Swedish, Chinese), these resources cover merely a small percentage of "the world's 200+ major languages" (Yarowsky and Ngai, 2001).

Annotation projection attempts to bridge the gap between these resource-rich languages and the remaining resource-scarce languages by transferring annotations from labeled sentences to their unlabeled translations. The technique depends on links that indicate translational equivalence between words, so-called *word alignments*. We discuss parallel data and word alignment in Section 3.1.

Before we describe our projection algorithms for dependency trees (Section 3.3), we address the limitations of the approach. First, natural languages are of course not one hundred percent isomorphic, but exhibit inter-language non-parallelism. These discrepancies are inherent in parallel data, and occur to a smaller or larger extent depending on factors such as language family, date of text creation, text genre, and even author. Second, professional translators regularly deviate from the strict word-to-word translation paradigm that would benefit the annotation projection approach, but tends to result in unnatural target language texts. Thus, non-literalness of translation further feeds the list of cross-language discrepancies. Third, automatic word alignment links are notoriously sparse and not always accurate. Section 3.2 is dedicated to these limiting factors.

Previous work on annotation projection (cf. Chapter 2) and similar weakly supervised annotation induction has chiefly relied on heuristics or other filtering techniques to deal with noise in unlabeled or automatically labeled training data. The objective is to select, from the pool of potential training examples, high-precision training data in sufficient quantities. While heuristic approaches can infuse linguistic knowledge that helps to detect and potentially adjust noisy data, they are usually specific to the language. This means they introduce a component of indirect supervision. Non-heuristic filtering techniques, on the

other hand, frequently employ reliability measures (often unrelated to the task) to predict high-precision data points. To reach a sufficient level of precision, filtering typically has to be aggressive, especially for highly structured tasks like parsing, where errors would propagate from one subpart of the annotation to many others. As we show in Section 3.3, such aggressive filters incur massive data loss and enforce harsh trade-offs between the quality and the amount of usable data. Moreover, they are particularly prone to distort the frequency spectrum of the data by systematically excluding certain types of examples.

Ideally, a general filtering strategy for weakly supervised training of structured analysis tools should eliminate noisy *sub*parts in the automatic annotation without discarding other, high-precision parts of that same structure. Thereby data loss and distortion of the data distribution would be kept to a minimum. This chapter presents a very simple method to reduce noise in projected annotations: *partial correspondence projection* eliminates unreliable components of an example, while greedily retaining a partial structure. This approach thus exploits correspondences that potentially cover only substructures of translated sentences. Section 3.3 describes how we approximate the reliability of projected annotations, and contrasts partial correspondence projection (Section 3.3.3) with a stricter as well as a laxer filtering alternative (Sections 3.3.1 respectively 3.3.2).

3.1 Parallel Data

3.1.1 Parallel corpora

Parallel corpora are multilingual corpora consisting of so-called *bitexts*, that is, documents in a *source language* L_s along with their translation in one or more *target languages* L_t. The translations are typically produced by professional translators or interpreters who are proficient in both L_s and L_t. In computational linguistics, parallel corpora have originally been collected for the purpose of training and testing (statistical) machine translation systems, which use the translations included in the corpus as reference translations.

Freely available parallel corpora often consist of proceedings and other documents of multinational organizations or governments of countries with multiple official languages. The most widely used of these bitexts is the Europarl corpus (Koehn, 2005) of proceedings of the European Parliament in 11 languages, with approximately 45 million words per language. The JRC-Acquis corpus (Steinberger et al., 2006) of legislative texts of EU member states comprises documents in 20 languages, but with only 8.8 million words per language, on average. The Hansards corpus[1] of proceedings of the Canadian Parliament consists of approximately 20.5 million words in English and French. Finally, bible translations are also frequently used as parallel data. They have the advantage of a standardized verse alignment, and electronic copies are available for nearly all languages. On the downside, the language is hardly representative of modern everyday usage, and the size does not exceed 1 million words per language.

The Europarl corpus. Throughout the dissertation, we use data from the Europarl corpus (Koehn, 2005). Europarl consists of the Proceedings of the Eu-

[1] `http://www.isi.edu/natural-language/download/hansard/`

lang. pair	# bisents	# words (L_s)	# words (L_t)
en–nl	1,317,788	36,734,246 (en)	36,604,096 (nl)
de–nl	1,312,187	34,300,673 (de)	36,071,014 (nl)
en–it	1,103,443	32,200,832 (en)	31,738,972 (it)
de–it	1,075,985	29,738,817 (de)	30,872,232 (it)
en–de	1,314,944	36,822,040 (en)	34,895,635 (de)

Table 3.1: Europarl parallel corpus data for selected language pairs.

ropean Parliament, which are transcribed and edited.[2] In release 3, the corpus encompasses the proceedings of the years 1996 through 2006, with translations in eleven languages, namely Danish, German, Greek, English, Spanish, Finnish, French, Italian, Dutch, Portuguese, and Swedish. The exact amount of data available for the language pairs considered in this thesis is summarized in Table 3.1.[3] With more than 1 million sentences per language, the Europarl corpus constitutes the largest parallel corpus of controlled origin.

3.1.2 Bilingual alignment

Preprocessing. The raw Europarl corpus does not indicate translational equivalence of text units at any level. The texts are, however, grouped into speaker turns which can be matched across translations. In order to establish sentence- and word-level alignments, the paragraphs first need to be broken down into smaller units. The necessary preprocessing steps for our data were performed using the Procep toolchain developed at the University of Potsdam.[4] The pipeline includes word tokenization and sentence splitting functionality that was carefully designed to avoid confusion between punctuation pertaining to abbreviations and truly sentence-final punctuation. It uses unsupervised machine learning techniques to identify sentence boundaries as well as handcrafted, language-specific rules for word tokenization.[5]

On the basis of the Procep tokenization, the bitexts were sentence-aligned using the implementation of the Church and Gale algorithm (Gale and Church, 1993) which is provided alongside the raw corpus on `statmt.org`. The preprocessed, sentence-aligned corpus we are using is the Golden Delicious release described in Jähnig and Marienfeld (2010). Figure 3.1 shows an excerpt from the sentence-aligned English–Italian bitext. The example in the upper part of the figure illustrates a straightforward one-to-one correspondence between an

[2] Obvious characteristics of spoken language, such as hesitations and corrections, are not featured in the transcriptions.

[3] The raw corpus as available from `http://www.statmt.org/europarl/` contains a superset of the data considered here. Some sessions were discarded in order to obtain cleaner alignments. Cf. Section 3.1.2.

[4] Procep is freely available at `http://sourceforge.net/projects/procep/`.

[5] For Dutch, Procep delegates preprocessing to Alpino (van Noord, 2006).

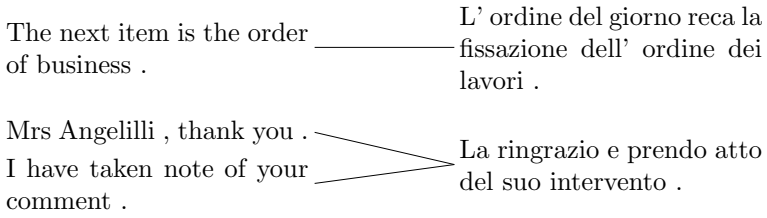


Figure 3.1: Excerpt from the tokenized, sentence-aligned English–Italian bitext, illustrating one-to-one (top) and many-to-one (bottom) sentence alignments.

English sentence and its Italian translation. As we can see in the second example, translators sometimes choose to split or merge sentences, resulting in many-to-one sentence alignments.

Word alignment. The sentence pairs (or *bisentences*) thus identified are subsequently aligned at the word level. Word alignments arise as a by-product in the estimation of translation probabilities. The latter are estimated in an iterative process which initially hypothesizes that any word in the sentence could be aligned to any word in the translation, but then successively takes cross-lingual patterns into account (e.g., when *heute* occurs in German, there tends to be an occurrence of *today* in English). As a side effect, particular word alignments (among the many possible options) gain the status of the best possible "explanation" of the co-occurrence of a source and target word string.

We use the Giza++ tool (Och and Ney, 2003) for the estimation of the word alignments. Giza++ implements a cascade of statistical word alignment models which estimate translation probabilities from large parallel corpora. In particular, Giza++ trains a sequence of successively more sophisticated models and uses the parameters of the simpler models to initialize the more complex ones in order to find good local optima.

More formally, statistical alignment models explain the relationship between a source language string f and a target language string e in reference to a hidden variable a, the word alignment, as follows:

$$\Pr(f|e) = \sum_{a} \Pr(f, a|e) \tag{3.1}$$

The models differ with respect to the decomposition of $\Pr(f, a|e)$ and hence the alignments they can model.[6] The Hidden Markov Model (HMM) approach to statistical word alignment (Vogel et al., 1996) breaks the translation probability down into an *alignment probablility* $\Pr(a_j|f_1^{j-1}, a_1^{j-1}, e)$ with a first-order dependence on the alignment of the preceding source word, and a *lexicon probability* $\Pr(f_j|f_1^{j-1}, a, e)$ which depends only on the alignment of the current word. In

[6] As the models become more complex, the summation over *all* alignments a in (3.1) is actually approximated by considering only the best predicted alignment so far, the so-called *Viterbi* alignment. It is precisely this Viterbi alignment that we are interested in – we do not make use of the translation probability $\Pr(f|e)$.

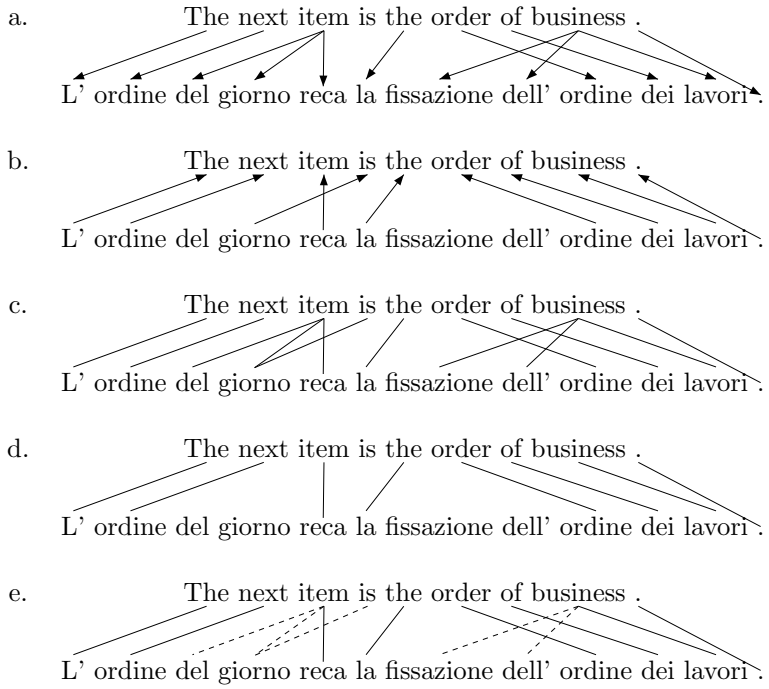


Figure 3.2: Viterbi word alignments for an English–Italian bisentence, showing both alignment directions separately (a and b, respectively) as well as the cumulative (c) and the intersective (d) alignment. A compact representation of the two combined alignments from c and d is shown in e.

addition, the HMM includes a term relating the length of the source and target language. The IBM Models 1 and 2 basically use the same decomposition as the HMM, but differ from the latter in that they assume zero-order dependence for the alignment probability and thereby eliminate any notion of word order.

The HMM as well as the IBM Models 1 and 2 allow source words to align to at most one target word. The IBM Models 3–5 (Brown et al., 1993) are more expressive (and more complex) because they also account for many-to-one alignments. This is achieved by modeling *fertility*: the fertility ϕ_i of a target language word e_i is defined as the number of source language words f_j that are aligned with e_i under a given alignment. Fertility affects the lexicon probability, which is now composed of the probability of all one-to-one alignments that a word is involved in. The fertility-based models 3 and 4[7] differ with respect to the alignment probabilities: like Models 1 and 2, Model 3 makes a zero-order Markov assumption. In Model 4, by contrast, the alignment of a target word e_i depends on the alignment(s) of the preceding word e_{i-1}. Och and Ney (2003) present a sixth model with combines the source language Markov dependence of the HMM with the target language dependence of Model 4.

[7]Model 5 is merely a non-deficient reformulation of Model 4, but considerably more complex. See Brown et al. (1993) for its formulation.

	en–nl		de–nl		en–it		de–it		en–de	
ϕ	en	nl	de	nl	en	it	de	it	en	de
0	21.0	22.1	20.0	24.6	21.1	19.1	28.3	27.4	22.6	19.7
1	69.3	67.8	69.0	67.2	68.5	69.9	57.6	61.0	68.5	67.8
2	6.7	6.6	7.2	5.3	8.0	8.1	8.7	8.1	6.6	8.5
3	1.7	1.9	1.8	1.2	1.4	1.7	2.7	1.8	1.3	2.2
4	0.7	0.7	0.8	0.6	0.5	0.6	1.0	0.8	0.5	0.8
5	0.3	0.4	0.4	0.3	0.2	0.2	0.5	0.4	0.2	0.4
6	0.2	0.2	0.3	0.2	0.1	0.1	0.3	0.2	0.1	0.2
7	0.1	0.1	0.2	0.1	0.1	0.1	0.3	0.1	0.1	0.2
8	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1
9	0.1	0.2	0.3	0.3	0.1	0.1	0.5	0.2	0.1	0.2

Table 3.2: Distribution of fertilities ϕ in unidirectional (non-cumulative) Viterbi alignments.

The parameters of the models are estimated iteratively by means of the EM algorithm. The most likely alignments (Viterbi alignments) for the parallel corpus can then be reconstructed based on the final set of parameters. Due to the asymmetry of the alignment models, it is common practice to compute the alignments in both directions (source–target and target–source) and then use a combination of the two, such as their intersection or their union. This is illustrated in Figure 3.2. The gloss for the Italian sentence is shown in (3.2).

(3.2) *L'* *ordine* *del* *giorno* *reca* *la* *fissazione* *dell'* *ordine*
The agenda of the day brings the fixing of the order
dei *lavori.*
of activities
'The next item on the agenda is the order of business.'

Figure 3.2a shows the Viterbi alignment by the alignment model which treats Italian as the source and English as the target; the Viterbi alignment for the opposite direction is given in Figure 3.2b. Figure 3.2c depicts the union of the two unidirectional alignments; we call this the *cumulative* alignment. The *intersection* of the unidirectional alignments (Figure 3.2d) is commonly used in scenarios that prioritize precision over recall (annotation projection typically falls into this category): the resulting *bidirectional* alignment links are sparse, but generally very reliable. Finally, Figure 3.2e illustrates how we represent the cumulative and intersective alignments compactly in one graph: the solid lines correspond to bidirectional links (which are a subset of the cumulative alignment) and are complemented by dashed lines which indicate links that are present in one of the two (but not both) unidirectional alignments.

For the alignment of our data, Giza++ was set to run five iterations of the IBM Model 1, five iterations of the HMM, and three iterations each of Models 3 and 4. Table 3.2 gives an impression of the distribution of fertilities in the aligned corpus. For each alignment direction it shows the percentage of target language

a.

lang. pair	% unaligned (L_s)	% unaligned (L_t)
en–nl	3.34 (en)	3.99 (nl)
de–nl	3.31 (de)	3.40 (nl)
en–it	3.47 (en)	3.73 (it)
de–it	3.41 (de)	3.14 (it)
en–de	3.07 (en)	3.44 (de)

b.

lang. pair	% unaligned (L_s)	% unaligned (L_t)
en–nl	28.25 (en)	27.48 (nl)
de–nl	27.61 (de)	30.96 (nl)
en–it	25.46 (en)	24.10 (it)
de–it	36.00 (de)	38.23 (it)
en–de	29.64 (en)	25.67 (de)

Table 3.3: Percentages of unaligned words in Europarl under a. the cumulative alignment (union) and b. the intersective alignment.

words that have a given fertility according to the Viterbi alignment. The second column from the left, for instance, tells us that 21.0% of all English words in the English–Dutch bitext remain unaligned in the alignment from Dutch to English, and that 69.3% of all English words are aligned to exactly one Dutch word, etc. The fertility distribution in the opposite alignment direction (English-to-Dutch) is shown in the next column, labeled 'nl.' It is immediately evident from the table that the majority (57–69%) of words, across all languages, are aligned to exactly one word in the other language, that is, have fertility $\phi = 1$. However, substantial portions (20–28%) also remain unaligned ($\phi = 0$). Fertilities of 4 or greater each account for 1% or less. The distribution of fertilities generally appears surprisingly stable across language pairs, with the exception of the alignment between German and Italian. Not only is the German–Italian word alignment particularly sparse at the expense of one-to-one alignments – the proportion of unaligned Italian words, for instance, exceeds that in the English–Italian alignment by more than 8 percentage points, while the number of words with fertility 1 drops by almost 9 percentage points – but at the same time there are conspicuously greater portions of words with higher fertility. This latter trend tends to be indicative of erroneous alignments for fertilities of 5 and beyond.

Table 3.3 summarizes the coverage of the combined Viterbi alignments. It shows the percentage of unaligned words for each of our language pairs. The most striking observation here is that the number of unaligned words is smaller by almost an order of magnitude under the cumulative alignment (Figure 3.3a) in comparison to the intersective alignment (Figure 3.3b). Moreover, the amount

	precision	recall	f-score
cumulative	72.78	79.22	75.86
intersection	94.88	62.04	75.02
Padó (2007) intersection	98.60	52.90	68.86

Table 3.4: Evaluation of English–German word alignment against the Padó gold standard.

of cumulatively unaligned words exhibits very little variation across languages, ranging from 3.07% to 3.99%. Differences are more pronounced when we look at the intersective setting. In particular, our suspicion is confirmed that the German–Italian word alignment must be considered an outlier. However, it is unclear if the unusually high proportion of unaligned words is merely due to an especially harsh precision–recall tradeoff, or if both measures are negatively affected, in which case we have to assume that this specific bitext – for one reason or another – is generally hard to align.

Evaluation of word alignment quality. Manually annotated word alignments are hard to come by, and unfortunately, for most of our language pairs we do not have gold standard alignments at our disposal. For the language pair English–German, however, there is a data set of 1,000 bisentences which are aligned manually, in accordance with the Blinker guidelines (Melamed, 1998a,b). The data is made available by Sebastian Padó,[8] and we refer to the data set as the *Padó gold standard*, or PGS. In addition to the manual word alignments, the annotations include syntactic as well as role-semantic analyses. In Section 3.3 we will use the syntax annotations to assess the quality of our direct projections from English to German. Table 3.4 shows the result of evaluating our automatic word alignments against the manually annotated alignments.[9] Precision and recall were computed in terms of individual links (i.e., many-to-one alignments are counted as multiple independent links); the f-score is the harmonic mean of precision and recall. It comes as no surprise that the cumulative and intersective alignments have complementary strengths: The union of the unidirectional alignments (cumulative) exhibits high recall (79.22%) but at a level of precision that is less than satisfactory (72.78%). Although the discrepancy between the two measures is even greater for the intersective alignment, the precision of this alignment (94.88%) exceeds its recall (62.04%). For comparison, we also list the figures reported in Padó (2007) for the evaluation of his automatic alignment against the same data set. We observe that Padó's automatic alignment exhibits an even harsher precision–recall tradeoff. As we shall see in Chapter 6, this is a favorable constellation for a noise-prone technique such as annotation projection if the projected annotations are used as training data for machine learning algorithms.

[8] `http://www.nlpado.de/~sebastian/srl_data.html`

[9] Due to differences in tokenization and sentence splitting, we had to exclude 346 of the 1,000 sentences. The figures in the table are based on the remaining 654 sentences.

3.2 Violations of Direct Correspondence

The basic assumption underlying annotation projection is the *Direct Correspondence Assumption* (DCA; Hwa et al., 2005).

> **DCA:** Given a pair of sentences E and F that are (literal) translations of each other with syntactic structures Tree_E and Tree_F, if nodes x_E and y_E of Tree_E are aligned with nodes x_F and y_F of Tree_F, respectively, and if syntactic relationship $R(x_E, y_E)$ holds in Tree_E, then $R(x_F, y_F)$ holds in Tree_F. (Hwa et al., 2005, p. 314)

For the DCA to be met, the source tree effectively has to be isomorphic to the intended target language tree. The violation of this assumption causes gaps or errors in the projected target annotations. Although the DCA is in fact valid in many cases – consider for instance the internal structure of nominal phrases, which witnesses little variation even across language families (Yarowsky and Ngai, 2001; Fox, 2002) – it clearly does not hold in general. Firstly, professional translators and interpreters trade off literalness against stylistic considerations in order to produce natural translations, leading to translation mismatches (Kameyama et al., 1991) as in (3.3) and, in the extreme, very loose translations (3.4), which are both examples from the Europarl corpus.

(3.3) *Bei einem der größten Unfälle* ***in jüngster Zeit*** *war*
in one of the greatest accidents in youngest time was
nicht die ***Ladung*** *an sich gefährlich.*
not the load in itself dangerous
'In one of the worst accidents **to have occurred recently**, the **goods being transported** were not dangerous in themselves.'

(3.4) *Wir sind der Auffassung, daß das richtig ist.*
we are of the opinion that this right is
'We think they should.'

Secondly, there are of course inherent limits to the extent of cross-language parallelism: so-called *translation shifts.* The term was introduced by Catford (1965), who defines translation shifts as "departures from formal correspondence in the process of going from the source language to the target language" (Catford, 1965, page 73). Translation shifts have been discussed, especially in translation studies, for several decades (Vinay and Darbelnet, 1958; Leuven-Zwart, 1989; Kameyama et al., 1991). Naturally, they are also an issue of great concern in machine translation, particularly the transfer-based flavors. Dorr (1994) formalizes systematic lexical-semantic differences between languages in terms of a set of divergence categories and subsequently defines translation mappings for each category, thus resolving the divergence by means of transfer via an interlingua. Examples for her divergence categories are shown in Figure 3.3 (Figure 1, page 598 in Dorr, 1994).

There have been attempts to create annotated resources that facilitate the empirical study of translation shifts (Cyrus, 2006; Čulo et al., 2008; Padó and Erk, 2010). However, since translation shifts are notoriously hard to annotate because they tend to interact, researchers have aimed at capturing such

a. **Thematic divergence:**
E: I like Mary ⇔ S: María me gusta a mí
'Mary pleases me'

b. **Promotional divergence:**
E: John usually goes home ⇔ S: Juan suele ir a casa
'John tends to go home'

c. **Demotional divergence:**
E: I like eating ⇔ G: Ich esse gerne
'I eat likingly'

d. **Structural divergence:**
E: John entered the house ⇔ S: Juan entró en la casa
'John entered in the house'

e. **Conflational divergence:**
E: I stabbed John ⇔ S: Yo le di puñaladas a Juan
'I gave knife-wounds to John'

f. **Categorial divergence:**
E: I am hungry ⇔ G: Ich habe Hunger
'I have hunger'

g. **Lexical divergence:**
E: John broke into the room ⇔ S: Juan forzó la entrada al cuarto
'John forced (the) entry to the room'

Figure 3.3: Divergence categories according to Dorr (1994)

shifts by investigating their correlation with more easily observable properties that are likely indicators of shifts. For example, Čulo et al. (2008) consider discrepancies between aligned words and chunks on the level of part-of-speech, grammatical function and position (sentence initial theme position versus later in the sentence). Cyrus (2006), on the other hand, describes a framework for annotating translation shifts directly, but restricts the enterprise to shifts that manifest themselves in the predicate-argument structure. She distinguishes grammatical shifts – category change, (de-)passivization, (de-)pronominalization, number change – and semantic shifts – semantic modification, explicitation/generalization, addition/deletion, mutation.

Fox (2002) measures phrasal cohesion between English and French, and finds that it is best preserved in a dependency representation (as opposed to a phrase-structure representation) of the syntactic structure. According to this study, which uses excerpts from the Canadian Hansards parallel corpus with manual word alignments (Och and Ney, 2000), 12% of all modifiers are incoherent with respect to their head, and 9% of all pairwise modifier comparisons reveal incoherence of modifiers. Closer inspection of the non-cohesive cases suggests that a considerable amount of divergence is due to non-literal translation.

Padó and Erk (2010) investigate – in an idealized setting with manual word alignments and manual annotations – to what extent the failure of annotation projection can predict the presence of semantic translation shift. In contrast to grammatical shifts, semantic shifts are typically characterized by much more subtle differences such as changes in perspective or reconceptualisations.

In the context of syntactic parsing, semantic translation shifts often do not harm the "projectability" of syntactic dependencies. Examples are generalization and explicitation (Cyrus, 2006), where a source language lexeme is translated by target language lexeme with a more general or a more specific meaning, respectively. Shifts of this kind do not influence the syntactic analysis. Moreover, mere frequency makes grammatical shifts the more severe problem. In their empirical evaluation of the DCA for dependency trees in English and Chinese, Hwa et al. (2002) find that the direct projection approach, which assumes full validity of the DCA, achieves precision and recall values in the range of 30–40%. Our partial correspondence approach (described in Section 3.3.3) achieves 66.69% precision and 35.74% recall when evaluated against gold standard German parse trees.

For the purpose of annotation projection, translation shifts can in principle be addressed with transformations that either manipulate the source language dependencies prior to projection, or amend the projected annotations. The latter strategy is pursued by Hwa et al. (2002, 2005). However, such a rule-based approach conflicts with the otherwise resource-lean trademarks of annotation projection in that it requires extensive linguistic knowledge of the target language if it is to attain reasonable coverage. A suitable alternative could be realized with machine learning methods. But again, the obvious supervised scenarios would rely on a systematic and consistent markup of the shifts, which is a challenge in its own right (Cyrus, 2006) and to the best of our knowledge is not available to date.

We pursue a radically different scheme in the present proposal in that we consider only those (partial) annotations that are supported by reliable alignment links. The resulting dependency structures may thus be *fragmented.* Since our goal is to bootstrap dependency parsers for the target language rather than

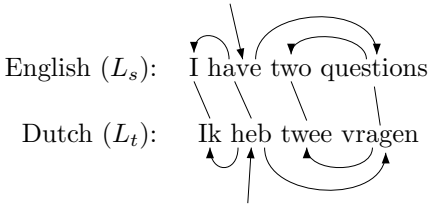


Figure 3.4: Dependency tree projection from English to Dutch.

producing gold standard treebanks, we accept those partial annotations *without further modification* and adapt the parsers so that they can handle incomplete training data, namely, the projected dependencies. Our take on parsing with fragmented training data is described in the next chapter. But first, the remainder of this chapter presents our algorithms for the cross-lingual projection of dependency trees.

3.3 Projection of Dependency Trees

The basic concept of annotation projection is simple: given a suitable resource in the source language L_s, and a word-aligned parallel corpus with languages L_s and L_t, label the L_s-portion of the parallel text and copy (or *project*) the resulting annotations to the corresponding (i.e., aligned) elements in language L_t.

In our case, the resource exploited in the source language L_s is a dependency parser, and the parallel corpus is Europarl. Figure 3.4 illustrates the projection of dependency trees with an example where L_s is English and L_t is Dutch. The links between English and Dutch words indicate the word alignment. Assuming that the source language parser produces the dependency tree shown for the English sentence (above), we build the projected tree for the Dutch sentence (below) by postulating dependency arcs between words w_d (e.g., *Ik*) and w_h (*heb*) if there are aligned pairs (w_d, w'_d) (*Ik* and *I*) and (w_h, w'_h) (*heb* and *have*) such that w'_h is the head of w'_d in the English tree.

We formally introduce the notation for dependency trees in Chapter 4; for now, suffice it to say that a dependency graph is a pair (S, A) consisting of the sequence of words $S = w_1 w_2 \dots w_n$ which make up the sentence, and a set A of dependency edges (or *arcs*). An arc (w_h, r, w_d) represents a dependency relation of type r (e.g., subject) between the head w_h and the dependent w_d.[10] Figure 3.5 shows pseudo-code for the basic projection algorithm. After initializing the arc set A for the target language tree in line 1, we iterate over the target language words w_d from left to right (line 2) and retrieve the aligned source language word w'_d (line 3). If there is no such word, we move on to the next word, otherwise we find the head w'_h of w'_d in the source tree and identify its correspondent w_h

[10] We assume that all arcs are labeled, but we ignore the labels for the most part. Cf. Section 5.5.1.

project$_{\text{strict}}(S, G', a)$
$S = w_0 w_1 \dots w_n$: target language sentence
$G' = (S', A')$: dependency tree for source language sentence
$a : S \rightarrow S'$: intersective word alignment

1 Initialization: $A = \emptyset$

2 **for** w_d $(1 \leq d \leq n)$
3 $w'_d = a(w_d)$
4 **if** $w'_d \neq$ **null**
then
5 $w_h = a^{-1}(w'_h)$ s.t. $(w'_h, r, w'_d) \in A'$
for some relation r
6 **if** $w_h \neq$ **null**
then
7 $A = A \cup \{(w_h, r, w_d)\}$

8 **return** A

Figure 3.5: The basic projection algorithm based on intersective word alignments.

in the target sentence (line 5). Provided that such an alignment exists (line 6), an arc from w_h to w_d is added to the target language tree (line 7).

In other words, for each source language dependency arc, if both the head and the dependent are aligned to a target language word, then the projected arc is added to the target graph under construction. Our algorithm is driven by the target language side, that is, the iteration in line 2 is over target language words. Alternative formulations driven by the source language sentence are of course possible.

It is easy to see that the target graph constructed by the algorithm in Figure 3.5 is not necessarily going to be a tree, even when the source graph G' is. The first two projection variants proposed in the following sections will simply discard analyses which do not form complete trees at the end of the projection phase (strict projection and constrained fallback projection), whereas the third (partial correspondence projection) does admit fragmented output. We discuss the exact treatment of fragmented analyses in the respective sections.

Preprocessing. Before we delve into the details of the projection algorithms, a few words are in order concerning the preprocessing steps that need to be performed in the L_s portion of the bitexts.

We use English as well as German as alternative instantiations of L_s. For both languages we POS-tag the respective Europarl portions with the TreeTagger (Schmid, 1994)[11] and subsequently parse the texts with the *source*

[11] We use the pretrained models available at `http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/`.

a.

	UAS	LAS
en	–	–
de	83.80	77.77

b.

	UAS	LAS
en	91.67	88.56
de	87.13	84.14

Table 3.5: Evaluation of German and English source parsers against a. the out-of-domain Padó gold standard (German only), b. the in-domain WSJ/Tiger test sets.

parsers. Our source parsers are Malt parser models (cf. Section 4.2) trained on a dependency-converted version of the Wall Street Journal (WSJ) part from the Penn Treebank and the German TIGER Treebank, respectively. The parsers are described in Øvrelid et al. (2009), where they are used as baseline parsers.[12]

We also POS-tagged the target language texts (Dutch, Italian, German). The resulting POS annotations are *not* used during projection, but they are exploited as features in the monolingual parsers that are derived from the projected dependency trees (Sections 4.3, 4.5 and Chapter 6). Although we strive for a resource-lean approach, we argue that the availability of POS taggers is less of an issue than the availability of parsers. Variants of our setup are conceivable in which POS tags are projected from a source language, just like we project the dependencies. This approach could then be improved by bootstrapping techniques like those proposed in Moon and Baldridge (2007). We leave this extension for future work.

Source parser quality. The performance of the German source parser on the Padó gold standard (PGS) parse trees is given in Table 3.5a.[13] Note that the test sentences in the PGS are from a different domain than the data used to train the parsers: neither the German nor the English parser were trained on Europarl, but on the WSJ respectively Tiger corpus instead. This evaluation – like the evaluation of our projected parsers in Chapter 6 – is therefore *out of domain*. For comparison, Table 3.5b shows the results of the in-domain evaluation reported in Øvrelid et al. (2009). The degradation in the out-of-domain setting (for German) is substantial, with a difference of 3.33 percentage points UAS (LAS: Δ-6.37).

[12] More specifically, we are using the baseline parsers in the setting with automatically assigned POS tags. That is, both training and testing is carried out using POS information that is derived by a tagger (as opposed to gold standard POS sequences).

[13] The Tiger-style annotations were converted to dependencies using software that was kindly provided by Yi Zhang. (It is the same conversion that was also used to prepare the data for the 2009 CoNLL Shared Task.) The conversion is specific to German, however, and to our knowledge there is no tool to convert Padó's TigerXML-encoded Penn Treebank annotations to dependencies. This is why we omit the out-of-domain evaluation of the English source parser. Results from the literature suggest that a substantial drop in performance must be expected. For example, the system of Sagae and Tsujii (2007), which performed best in the domain adaptation track of the CoNLL 2007 Shared Task (Nivre et al., 2007), sacrifices 6.45 percentage points UAS (LAS: Δ-7.95) when parsing out-of-domain chemical research abstracts.

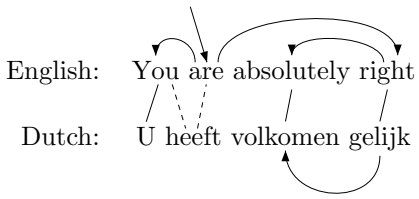


Figure 3.6: Dependency tree projection from English to Dutch: projection fails due to weak alignments.

3.3.1 Strict projection

In the general description of the projection algorithm above we have neglected the directionality of the word alignments. Recall from Section 3.1.2 that we compute both the $L_s \rightarrow L_t$ and the $L_t \rightarrow L_s$ alignments. It is common practice to intersect the two alignments and consider only the intersection, which contains those links that are supported *bidirectionally*. Bidirectional alignments have proven to be highly reliable, albeit very sparse. The high-precision aspect of the intersective alignment seemingly makes bidirectionality a promising candidate for a reliability measure on which we can base a noise filter. The first noise filter we examine is therefore the one that considers bidirectional alignments exclusively. We call this filter the *strict* or *bidirectional filter* and sometimes refer to bidirectional links as *strong* alignments, as opposed to *weak* unidirectional links which are only supported in one direction. The bidirectional filter admits dependency arcs to be projected only if the alignment between the heads w'_h and w_h as well as the alignment between the dependents w'_d and w_d are supported under the intersection of the two unidirectional alignments. In practice this means that if one of the target words or any source word with an outgoing dependency arc does not have such a strongly corresponding word in the other language, then the projected structure is not a tree. Strict projection therefore rejects the entire sentence.

Let us return briefly to Figure 3.4. We observe that all alignment links are bidirectional (indicated by solid rather than dashed links). All dependency edges can thus be projected and the resulting target language structure forms a tree; that is to say, the target sentence receives an analysis under strict projection. Figure 3.6 shows an example where this is not the case: the Dutch verb *heeft* is only weakly aligned with the English translation *are*, while a second weak alignment links it to the pronoun *you*. This means that under the strict filter, none of the dependencies involving *are* are projected, and the projected structure is not connected.

Although bidirectional alignments may be a reliable indicator of alignment quality, we now proceed to show that their modest recall hardly allows any trees to be projected completely. Subsequent sections then discuss less restricted projection methods which can incorporate larger portions of the data.

Table 3.6 quantifies the data loss in terms of the number of sentences that receive a parse tree under projection (a), the vocabulary size observed in those

		unfiltered	bidirectional	fallback	bi+frags$_{\leq 3}$
a.	en–nl	1,317,788	32,066 (2.4%)	134,375 (10.2%)	114,351 (8.7%)
	de–nl	1,312,187	54,953 (4.2%)	188,191 (14.3%)	163,191 (12.4%)
	en–it	1,103,443	25,131 (2.3%)	128,804 (11.7%)	100,204 (9.1%)
	de–it	1,075,985	20,003 (1.9%)	94,626 (8.8%)	58,423 (5.4%)
	en–de	1,314,944	35,367 (2.7%)	142,094 (10.8%)	125,992 (9.6%)
	avg.	1,224,869	33,504 (2.7%)	137,618 (11.2%)	112,432 (9.2%)
b.	en–nl	203,026	11,565 (5.7%)	34,493 (17.0%)	27,509 (13.5%)
	de–nl	201,135	17,583 (8.7%)	46,980 (23.4%)	35,945 (17.9%)
	en–it	75,897	8,287 (10.9%)	21,366 (28.2%)	16,479 (21.7%)
	de–it	74,771	6,075 (8.1%)	18,527 (24.8%)	11,723 (15.7%)
	en–de	250,468	12,975 (5.2%)	40,067 (16.0%)	30,058 (12.0%)
	avg.	161,059	11,297 (7.0%)	32,287 (20.0%)	24,343 (15.1%)
c.	en–nl	27.78	7.33 (26.4%)	11.85 (42.7%)	10.30 (37.1%)
	de–nl	27.49	8.33 (30.3%)	12.96 (47.1%)	11.31 (41.1%)
	en–it	28.76	7.09 (24.7%)	13.06 (45.4%)	10.90 (37.9%)
	de–it	28.69	5.78 (20.1%)	11.29 (39.4%)	8.61 (30.0%)
	en–de	26.54	7.17 (27.0%)	11.43 (43.1%)	10.06 (37.9%)
	avg.	27.85	7.14 (25.6%)	12.12 (43.5%)	10.24 (36.8%)

Table 3.6: Data reduction effect of noise filters. a. Number of target language sentences with a projected parse, b. vocabulary size (number of distinct lemmas), c. mean sentence length. The percentages in parentheses are relative to the full data set ('unfiltered').

	unfiltered	bidirectional	fallback	bi+frags$_{\leq 3}$
en–nl	1.89	28.54	11.00	10.59
de–nl	1.92	22.96	9.13	9.11
en–it	1.83	32.50	9.45	9.94
de–it	1.86	42.94	12.75	16.08
en–de	2.02	26.99	10.25	10.43
avg.	1.90	30.78	10.51	11.23

Table 3.7: Percentage of duplicate sentences in the original and filtered data sets.

remaining sentences (b), and their mean sentence length (c). The first column ('unfiltered') states these statistics for the unfiltered Europarl data for reference. The two rightmost columns ('fallback' and 'bi+frags$_{\leq 3}$') refer to projection algorithms described in later sections. We ignore them for now and focus on the column labeled 'bidirectional.'

We see that out of all sentences in the Europarl corpus (on average 1.2 million per language pair), only between 1.9% (German–Italian) and 4.2% (German–Dutch) of the sentences pass the bidirectional filter. This corresponds to a data reduction of up to 98.1%. Consequently, the vocabulary size diminishes, leaving data sets that contain merely 5% (English–German and English–Dutch) to 11% (English–Italian) of the lemmas that occur in the original, unfiltered Europarl (Table 3.6b). More crucially, the mean sentence length drops considerably from an average of almost 28 words to 7 words per sentence Table 3.6c). This is noteworthy because it suggests that most non-trivial examples are lost. Inspection of the filtered data confirms that not only is the range of sentences reduced to rather simplistic phrases, but it is also highly repetitive. In Table 3.7 we report the percentages of duplicate sentences in the data. Again, we ignore the columns 'fallback' and 'bi+frags$_{\leq 3}$' for the moment. We first note that even in the unfiltered data sets the portion of duplicates is not negligible, ranging from 1.83% in the English–Italian bitext to 2.02% in the English–German text. Sentences that occur multiple times are typically part of the parliamentary protocol, such as declarations of resumption and adjournment of sessions or indications of applause; we will see some examples shortly. Under strict projection ('bidirectional'), repetition is amplified to the extent that roughly one third (30.78% on average) of all projected parses are duplicates.

Figure 3.7 lists the most prominent target language sentences among the strictly filtered projections, along with their frequency and projected dependency tree.[14] A pleasant fact to notice is that the projected trees (shown here

[14]Here and in the rest of the dissertation we omit the attachment of punctuation in our examples. The implementation of the projection algorithm consistently attaches punctuation to the root node. In the case of sentence-internal punctuation, this procedure is likely to intro-

en–nl **32,066 sentences, 22,915 distinct**

2,844 Het debat is gesloten .
844 (Applaus)
409 Applaus
264 Waarom ?
223 Stemming

de–nl **54,953 sentences, 42,338 distinct**

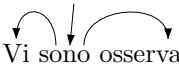
2,842 Het debat is gesloten .
842 (Applaus)

660 (Het Parlement neemt de resolutie aan)

656 De stemming vindt morgen om 12.00 uur plaats .
393 Applaus

en–it **25,131 sentences, 16,965 distinct**

2,349 La discussione è chiusa .
643 (Applausi)
436 Applausi
273 Perché ?



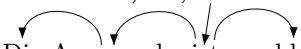
257 Vi sono osservazioni ?

de–it **20,003 sentences, 11,414 distinct**

2,329 La discussione è chiusa .
645 (Applausi)

478 (Il Parlamento approva la risoluzione)
421 Applausi
283 Perché ?

en–de **35,367 sentences, 25,824 distinct**



2,941 Die Aussprache ist geschlossen .
902 (Beifall)
436 Beifall
288 Warum ?
240 Herr Präsident !

Figure 3.7: Most frequent sentences with projected dependency tree. The exact frequency is shown to the left of each sentence. For each language pair, we further state the number of non-unique sentence tokens and non-unique sentence types ("distinct").

project$_{\text{fb}}(S, G', a_1, a_2)$
$S = w_0 w_1 \ldots w_n$: target language sentence
$G' = (S', A')$: dependency tree for source language sentence
$a_1, a_2 : S \times S'$: unidirectional word alignments

1 $A = \text{project}_{\text{strict}}(S, G', a_1 \cap a_2)$

2 $a = a_1 \cup a_2$
3 **for** w_d $(1 \leq d \leq n)$
4 **if** w_d has no incoming edge in A
then
5 $w'_d = \text{select}(a(w_d))$
6 **if** $w'_d \neq$ **null**
then
7 $w_h = \text{select}(a^{-1}(w'_h))$
8 s.t. $(w'_h, r, w'_d) \in A'$ and $\neg\text{connected}(w_d, w_h, A)$
9 **if** $w_h \neq$ **null**
then
10 $A = A \cup \{(w_h, r, w_d)\}$

11 **return** A

Figure 3.8: Pseudo-code for constrained fallback projection. We use the functional notation $a(w)$ as a shorthand for the set $\{w' : (w, w') \in a\}$.

only for non-trivial utterances with two or more words) are in fact correct analyses of the target language sentences. However, it is also obvious that a large portion of the sentences are one-word utterances which convey no parsing-relevant information. Moreover, the most frequent sentence in all five language pairs is the equivalent of *The session is adjourned* with 2,000–3,000 occurrences. While the sentence constitutes a valid and informative training example, seeing over 2,000 copies of it during training does not help a dependency parser attain broader coverage or higher accuracy.

In summary, the data loss incurred by the bidirectional filter goes beyond a mere reduction of the sample size. It also leads to highly skewed data samples that can no longer be considered representative of the underlying corpus: the filter limits the projected annotations to extremely short, simple sentences that have little lexical coverage.

3.3.2 Constrained fallback projection

The immense data loss and frequency distortion of the strict projection approach are a result of the sparseness of bidirectional alignments. A denser mapping between source and target language words can be obtained if *all* alignment links are taken into consideration, including the unidirectional ones. But we have to keep in mind that the unidirectional alignments are weaker than the

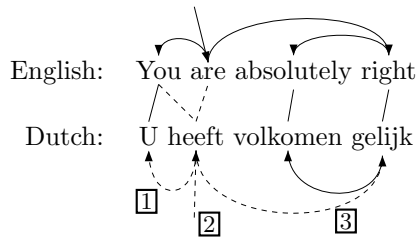


Figure 3.9: Constrained fallback projection from English to Dutch.

alignments that are also supported in the opposite direction.

In this section we lay out a more relaxed projection method, which we call *constrained fallback projection.* Constrained fallback projection considers all alignments, but takes reliability into account by prioritizing bidirectional alignments. The pseudo-code in Figure 3.8 outlines the algorithm. The approach essentially comes down to a fallback mechanism which projects further dependencies only *after* a partial structure has been built based on the more reliable bidirectional links (line 1). Moreover, the dependencies established via unidirectional alignments in lines 3–10 are constrained to be compatible with the existing subtrees, and are subject to the wellformedness conditions for dependency trees: single-headedness (line 4) and acyclicity (line 8).

In contrast to the intersective alignment, the cumulative alignments are not necessarily one-to-one. As a consequence, if a word is multiply aligned (like *heeft* in Figure 3.6, for example) a choice has to be made as to which alignment should be used for projection. This choice is ideally led by the existing structure established via strict projection. If these constraints still permit multiple projection avenues, we have to take recourse to a secondary selection strategy, denoted select in Figure 3.8. Our implementation simply picks the leftmost alignment link; more sophisticated strategies are of course conceivable.

Figure 3.9 demonstrates how constrained fallback projection recovers a complete parse tree for the weakly aligned sentence pair in Figure 3.6. The graph is initialized with the unconnected structure built with the bidirectional filter. Starting with the leftmost word in the Dutch sentence and its English translation (*U* and *You*), there is a unidirectional alignment for the head of *You*: *are* is aligned to *heeft*, so *U* is established as a dependent of *heeft* via fallback. Likewise, *heeft* can now be identified as the root node. The (incorrect) alignment between *heeft* and *You* is not pursued because it would lead to *heeft* being a dependent of itself, thus violating the wellformedness conditions. Finally, the subtree rooted in *gelijk* is incorporated as the second dependent of *heeft*.

The data obtained with constrained fallback projection is summarized in Table 3.6 (p. 36) as 'fallback.' As expected, the proportion of examples that pass this filter increases – to 11.2% on average – and 20.0% of the Europarl vocabulary are preserved with this filter. Moreover, with a mean length of

duce many crossing edges; we come back to this issue when we discuss *pseudo-projectivization* (Nivre and Nilsson, 2005) in Chapter 4.

approximately 12 words, the sentences are not quite as short as those that pass the strict filter. Roughly 10% of the data are duplicate sentences, so the repetition rate is also considerably lower than in the data obtained under strict projection, where 30% of all sentences are repetitions (Table 3.7).

To summarize, we have thus far seen two rather extreme noise filters. Situated on one end of the spectrum is the strict projection approach, which admits projected dependency trees only if they are projected via bidirectional alignment links. As we will see later in this chapter, the bidirectional filter produces comparatively high-precision annotations. but on the downside, it induces immense data loss and frequency distortion (Tables 3.6 and 3.7). In the other extreme, we have explored the less conservative fallback projection, which looks at all alignments, whether bidirectional or unidirectional. Quantitatively, data loss and distortion can be somewhat dampened in this way. However, we will show shortly that the quality of the annotations projected via weak alignments is rather modest.

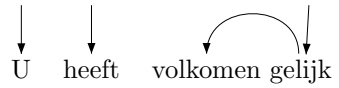
A questionable aspect of both strict projection and constrained fallback projection as they are formulated here is their completeness assumption: target language annotations are rejected unless the projected dependency edges form a connected tree structure spanning the entire sentence. By demanding complete trees we practically exclude all instances of L_t insertion from the projected data, that is, bisentences where the target language part consists of more words than the source language part (cf. Section 3.2). Since insertion is a frequent phenomenon in translation, the completeness requirement for the projected annotations contributes to the distortion of the data sample. We therefore propose a perspective on projection which combines the advantages of strict and fallback projection by allowing partial annotations in the data.

3.3.3 Partial correspondence projection

So far, we have only considered complete trees, that is, projected dependency graphs with exactly one root node. This is a rather strict requirement, given that even state-of-the-art parsers sometimes fail to produce plausible complete analyses for long sentences, and that non-sentential phrases such as complex noun phrases still contain valuable, non-trivial information. This section discusses *partial correspondence projection.* In addition to the complete annotations produced by tree-oriented projection, partial correspondence projection yields partial structures: it admits fragmented analyses in cases where neither strict nor fallback projection can construct a spanning tree.

As an example, consider again Figure 3.6. This example is discarded in a tree-oriented, bidirectional scenario. Under partial correspondence, it is included as a partial analysis consisting of three fragments:

(3.5)



Although the amount of information provided in this analysis is limited, the arc between *gelijk* and *volkomen*, which is strongly supported by the alignment, can be established without including potentially noisy data points that are only weakly aligned. Partial correspondence projection thus combines the

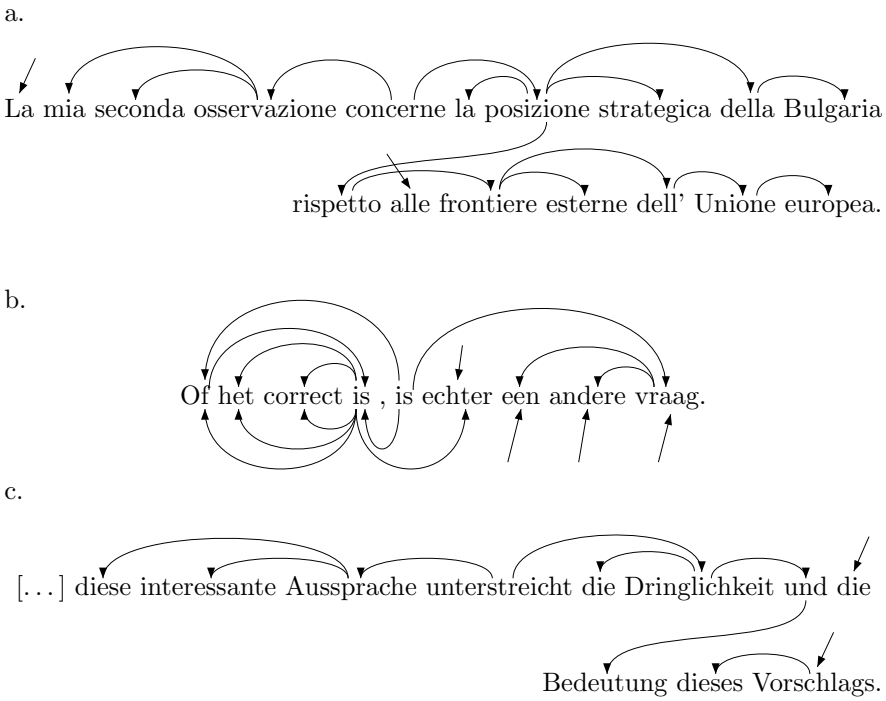


Figure 3.10: Examples projected under partial correspondence projection. The dependency graph in (a) shows an analysis projected from English to Italian, (b) is projected from English (top) respectively German (bottom) to Dutch, and (c) is projected from English to German.

high-precision aspect of strict projection with the greedy, recall-oriented perspective of a fallback approach. The result is a less dismissive rendition of the bidirectional filter. We show a few more elaborate examples in Figure 3.10. In Figure 3.10a we see the dependency structure that was projected onto the Italian sentence from English. (3.6) provides a gloss for the sentence, along with the English translation from the corpus.

(3.6) *La mia seconda osservazione concerne la posizione strategica*
the my second remark concerns the position strategic
della Bulgaria rispetto alle frontiere esterne dell'
of the Bulgaria with respect to the borders external of the
Unione europea.
Union European
'My second comment concerns the strategic position of Bulgaria on the external borders of the European Union.'

In this example, only two edges are missing, namely the dependency between *La* and *osservazione*, and the one between *rispetto* and *alle*. The first dependency, which should attach the determiner to the noun, cannot be recovered because *La* has no corresponding word in the English translation. The second fragment root, *alle*, is aligned with the English *the* (cf. (3.6)) via a weak, unidirectional link, which is not considered by partial correspondence projection. Note, however, that even if that alignment were strongly supported, the syntactic structure annotated by the English (Penn Treebank) source parser would in fact lead to *alle* being attached to *frontiere*, instead of the desired attachment to *rispetto*, with *frontiere* a dependent of *alle*. Apart from the two missing edges (and the questionable attachment of *frontiere* to *rispetto*), the projected dependencies constitute a fully legitimate syntactic analysis for the sentence.

For the Dutch example in Figure 3.10b we show the dependency structure projected from English above and the structure projected from German below the sentence.[15] The sentence is glossed in (3.7).

(3.7) *Of het correct is, is echter een andere vraag.*
whether it right is is however a different issue
'Whether it is right is a different issue.'

One arc is missing in the English-based projection: *echter* remains unattached because it has no corresponding word in the English translation. The rest of the dependency structure is a correct parse for the sentence. The German-based projected dependencies (depicted below the Dutch sentence) are slightly more sparse since the entire predicative NP *'een andere vraag'* remains unanalyzed due to the presence of the idiomatic expression *'steht auf einem anderen Blatt'* in the German translation. Furthermore, *echter* is attached to the wrong occurrence of *is*, which can be traced back to an error in the automatic word alignment. The rest of the analysis is fully acceptable; it differs from the English-based dependencies simply because the source language annotation schemes prescribe diverging treatments for subordinate clauses.

[15] The German translation reads as follows:

Ob das aber richtig ist, steht auf einem anderen Blatt.
whether that however right is stands on a different leaf/sheet

#frags	1	2	3	4–15	>15
sent. length (in words)					
<4	4,740	1,297	180	–	–
4–9	**19,185**	**20,326**	**20,551**	46,007	–
10–19	**7,621**	**15,817**	**24,547**	298,050	3,148
20–30	**486**	**1,646**	**3,688**	285,384	92,198
>30	**34**	**126**	**237**	96,999	375,393

Table 3.8: Fragmentation in the Dutch annotations produced with partial correspondence projection from English. Number of corpus sentences according to sentence length and number of fragments obtained. The sentences included in the data set 'bi+frags$_{\leq 3}$' are in boldface.

	unfiltered			bi+frags$_{\leq 3}$		
	words/sent	words/frag	frags/sent	words/sent	words/frag	frags/sent
en–nl	27.78	1.95	14.25	10.30	4.71	2.19
de–nl	27.49	1.98	13.92	11.31	5.50	2.06
en–it	28.76	2.26	12.79	10.90	4.85	2.25
de–it	28.69	1.66	17.33	8.61	3.99	2.16
en–de	26.54	2.05	12.98	10.06	4.57	2.20
avg.	27.85	1.98	14.25	10.24	4.72	2.17

Table 3.9: Average fragmentation in the partial correspondence projections.

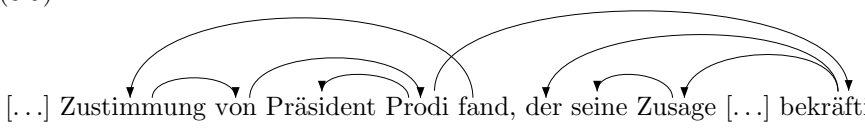
Finally, the German example in Figure 3.10c illustrates the almost complete projection of a coordinate structure. There are again two edges that could not be projected: the determiner in the second NP-conjunct is not realized in the English sentence, and the genitive phrase *'dieses Vorschlags'* is not attached to the rest of the sentence because the discrepancy between the analytical *'of the'*-construction in English, where the intervening preposition has not direct correspondence in the German NP where genitive case is marked morphologically:

(3.8) [...] *diese interessante Aussprache unterstreicht die Dringlichkeit*
this interesting debate highlights the urgency
und die Bedeutung dieses Vorschlags.
and the importance of this proposal
'[...] this interesting debate highlights the urgency and importance of this proposal.'

To give an impression of the composition of the data sets obtained via partial correspondence projection, Table 3.8 shows how fragment size varies with

sentence length in the Dutch annotations projected from English (en–nl), while aggregated statistics for all language pairs are given in Table 3.9. It is evident from Table 3.8 that the projected dependency graphs are indeed highly fragmented, with most parses being split into four or more fragments (columns '4–15' and '>15'). Table 3.9 confirms that this is the case across all languages: taking into consideration all sentences irrespective of the degree of fragmentation ('unfiltered'), we find that the projected analyses are split into 14.25 fragments on average (column 'frags/sent'). At an average sentence length of 27.85 words, this amounts to approximately 2 words per fragment. The degree of fragmentation thus needs to be restricted so as to exclude data with no (interesting) dependencies. For instance, a sentence of five words with a projected parse consisting of five fragments provides virtually no information about dependency structure because no words are interconnected. We therefore impose a limit (fixed at 3 for the remainder of the chapter) on the number of fragments that can make up an analysis. When we discard the projections that exceed this threshold, the remaining data (shown in boldface in Table 3.8, and in the right-hand part of Table 3.9) contains larger fragments (4.72 words per fragment on average). In other words, the filtered annotations are more densely populated with proper dependency edges. Alternative filters could require a minimum fragment size, or even set a threshold dynamically, depending on sentence length. We will explore the suitibility of different fragmentation constraints in Chapter 6. Note that we do not require the fragments to be contiguous; this allows long distance and more complex dependencies to pass the filter, for instance the relative clause in (3.9).

(3.9)



(3.10) *Frau* ***Präsidentin!*** *Die Vorstellung des politischen*
madam president! the presentation of the political

Programms ***der*** *Kommission Prodi für die gesamte*
programme of the commission Prodi for the whole

Wahlperiode ***ging auf*** *einen* ***Vorschlag*** *der* ***Fraktion***
legislative period went to a proposal of the faction

der Sozialdemokratischen Partei Europas zurück, der
of the social democratic party of Europe back, which

die *einhellige* ***Billigung*** *der* ***Konferenz*** *der Präsidenten*
the unanimous approval of the conference of the presidents

im September ***und*** *auch die ausdrückliche Zustimmung von*
in the September and also the explicit acceptance of

Präsident Prodi fand, der seine Zusage in ***seiner***
president Prodi found, who his commitment in his

Antrittsrede *bekräftigte.*
inaugural speech affirmed.

'Madam President, the presentation of the Prodi Commission's political programme for the whole legislature was initially a proposal by the Group of the Party of European Socialists which was unanimously approved by the Conference of Presidents in September and which was also explicitly accepted by President Prodi, who reiterated his commitment in his inaugural speech.'

(3.11)

In this example, the dependencies between the modified proper name (*Prodi*), the head of the relative clause (*bekräftigte*) and the relative pronoun (*der*) are correctly projected. However, the sentence as a whole is highly fragmented, as shown in (3.10), where all *unattached* words are highlighted in boldface. A total of 18 words remain unattached, that is, 18 edges are missing for the projected dependency structure to form a complete tree. Clearly, this analysis would be discarded under strict projection, but it can in principle be admitted under partial correspondence projection, depending on the fragmentation constraint used for filtering. Besides the relative clause attachments, the sentence contains an almost complete analysis of a complex noun phrase, shown in (3.11). The only disruption in this fragment is the lack of an attachment for the determiner *der*, which was not projected from the English source because of the discrepancy between the analytic *'of the'*-construction in English and the morphological case marking on the determiner in German.

In Table 3.6 (column 'bi+frags$_{\leq 3}$'), we see that partial correspondence projection boosts the amount of usable data to a range similar to that of the fallback technique for trees: 9.2% of all target language sentences are deemed valuable, retaining 15.1% of the vocabulary at an average sentence length of 10.24 words. Duplicate sentences account for 11.23% of the data, rather than 30.78% under strict projection.

Of course, in the partial correspondence setting, not all words in the target language corpus are actually attached. That is, some words do not encode information about the dependency structure directly. However, they provide *contextual information* for those attachments that were projected successfully. Table 3.10 complements Table 3.6 by stating the number of *attached* word tokens ('words attached') and the number of attached word types ('vocabulary attached') in the partial correspondence data sets. We see that, on average, 88.83% of all words in the fragmented data are attached by means of a projected dependency edge. In other words, 11.17% of the edges are missing. The number of word *types* that are thus seen as proper dependents in the data is also reduced to 89.15% on average. Although there are words in the data projected under partial correspondence that do not directly carry information about the

	words	words attached	vocabulary	vocabulary attached
en–nl	1,178,332	**1,036,332 (89.0%)**	27,509	**22,917 (83.3%)**
de–nl	1,845,121	**1,671,529 (90.6%)**	35,945	**32,869 (91.4%)**
en–it	1,092,387	**955,090 (87.4%)**	16,479	**15,737 (95.5%)**
de–it	502,993	**431,231 (85.7%)**	11,723	**11,075 (94.5%)**
en–de	1,268,084	**1,108,885 (87.5%)**	30,058	**25,906 (86.2%)**
avg.	1,177,383	**1,040,613 (88.8%)**	24,343	**21,701 (89.2%)**

Table 3.10: Summary of the partial correspondence data sets (bi+frags$_{\leq 3}$), ignoring unattached words.

dependency structure, the fragmented data sets still leave us with considerably more information than can be obtained through strict, tree-oriented projection, and the information is considerably more varied, as we have seen in Table 3.7.

3.4 Quality of Direct Projections

This section is concerned with the qualitative assessment of the direct projections. We have already seen the quantitative effect of the noise filters on the data in previous sections, but in order to evaluate their potential when it comes to producing training examples for data-driven dependency parsers, we need to compare the projected annotations to a gold standard.

Before we delve into the details of this comparison, we need to point out that there is one crucial fact which complicates the evaluation of projected annotations: the annotation scheme that is projected from the source language typically does not coincide with the annotation scheme employed in the target language gold standard. Chapter 5 is devoted to this issue and describes our methodology to circumvent the problem. For now, suffice it to say that we are not actually comparing the projected annotations against the original gold standard, but against a version of the gold standard which is converted so as to reflect the *source language annotation scheme*.

Recall that we are considering two source languages (German and English) and three target languages (Dutch, Italian and German), giving rise to the five language pairs German–Dutch, English–Dutch, German–Italian, English–Italian and English–German. For German, we can once more use the Padó gold standard (PGS) with its dependency-converted syntax trees. We parse the English portion of the test set, project the annotations to German, and evaluate the projected dependencies against the manual annotation for the German sentences. For the other two target languages – Dutch and Italian – the situation is more complicated, for we are not aware of any dependency-annotated resources

that are based on a parallel corpus. To work around this bottleneck, we use treebank parsers as a proxy for manual annotations and thus compare the projected graphs against the predictions of those parsers. Doing this not only for Dutch and Italian, but also for German will allow us to estimate (if only very roughly) to what extent potential errors of the treebank parsers distort the results of the proxy evaluation. But let us begin with the conventional evaluation procedure with proper gold standard annotations for German.

3.4.1 Gold standard evaluation (German)

We evaluate our projection methods in two different settings. In the first setting (PGS_t), the direct projections are obtained using automatically annotated source trees and the automatic Giza word alignment, but they are evaluated against the gold standard *target* trees. The second setting ($\text{PGS}_{t,a}$) uses automatic source trees, too, but projects them along the gold standard *alignment* instead of the Giza alignment.[16] Note that the fallback projections elude evaluation in this latter setting because the manual alignments annotated in the PGS are undirected. Hence, they do not embody a notion of strong versus weak alignment links.

The primary metric employed to evaluate syntactic dependencies is the (labeled or unlabeled)[17] *attachment score* (3.12), which measures the percentage of words that are attached to the correct head:

$$\text{AS} = P = R = \frac{|G \cap S|}{|S|} \tag{3.12}$$

where G and S denote the set of gold and predicted dependencies, respectively.[18] Since all *complete* dependency trees for a given sentence consist of the same number of dependency relations (namely, one per word), $|S|$ equals $|G|$, so that the attachment score coincides with both precision and recall. However, if we exclude missing edges from the evaluation instead of taking their default attachment to ROOT at face value, then we can define more meaningful precision and recall measures:

$$P_f = \frac{|G \cap S_f|}{|S_f|} \qquad R_f = \frac{|G \cap S_f|}{|G|} \tag{3.13}$$

where S_f denotes the *filtered* set of predicted dependencies that are *not* fragment roots. Thus, $|S_f| \leq |G|$, and missing edges affect recall, but not precision. Under tree-oriented strict and fallback projection, S_f contains dependency edges that form complete trees – but only for those sentences that do indeed receive a complete projected analysis. By contrast, the predictions S_f under partial correspondence projection contain all projected edges, irrespective of the completeness of the analysis, but there may be words within a sentence that are not mentioned in S_f at all due to missing attachments.

[16] We omit the third setting, $\text{PGS}_{t,s,a}$, which would make use of gold standard trees for both the source and the target language, as well as the manual alignment. As mentioned in footnote 13, the reason for this omission is that the annotations in the PGS are encoded in TigerXML emulating the Penn Treebank annotation style, and there is no straightforward conversion from TigerXML-encoded Penn Treebank annotations to pure dependencies.

[17] We only consider the *unlabeled* attachment score throughout the thesis.

[18] Following standard practice, we exclude punctuation from the evaluation.

			filtered		
		UAS	P_f	R_f	F_f
PGS_t	strict	5.54	78.90	0.66	1.30
	fallback	10.02	67.20	5.78	10.64
	frags	37.66	66.69	35.74	46.54
$PGS_{t,a}$	strict	9.64	60.36	5.30	9.75
	fallback	n.a.	n.a.	n.a.	n.a.
	frags	48.37	56.33	47.50	51.54
parser (Tiger)		83.80	83.80	83.80	83.80
baseline (next)		29.53	29.53	29.53	29.53

Table 3.11: Evaluation of direct projections from English to German against the Padó gold standard (PGS), using automatic source parses and automatic alignments (PGS_t) or manual alignments ($PGS_{t,a}$). The baseline attaches every word to the word on its right. The evaluation metrics are stated as percentages.

The results of the evaluation are summarized in Table 3.11. In the first column, we report the unlabeled attachment score according to the definition in (3.12). The remaining columns record precision and recall as defined in (3.13) as well as their harmonic mean, the F_1 score.

Let us first compare the two tree-oriented projection methods 'strict' and 'fallback' in the PGS_t setting, where we project automatic source trees along automatic word alignments and then evaluate the projections against gold standard target language trees. Both strict and fallback projection result in a very low attachment score of 5.54% and 10.02%, respectively. This can be explained by the fact that both methods discard any target language analysis that does not form a complete tree. We have already quantified the resulting data loss across the Europarl corpus in the previous chapter. In the PGS data set, the filters reduce the effective amount of data from 654 to 13 (strict) and 97 (fallback) sentences. Given the definition of the attachment score in (3.12), the sentences that do not receive an analysis not only incur recall errors, but also precision errors due to the implicit attachment of unanalyzed words to the root node, which for the majority of words is not supported by the reference annotation in the gold standard. The filtered variants of precision and recall (3.13) eliminate the impact of those spurious attachments. According to P_f, the quality of the strict, tree-oriented projection exceeds the fallback projections as well as the fragments by more than 10 percentage points. Moreover, it is noteworthy that the trees projected under fallback projection – that is, using weak word alignments – achieve a precision that is higher than that of the fragments, which are projected using only bidirectional alignment links. This means that the completeness constraint that is enforced by tree-oriented projection methods is indeed a valid noise filter. When we turn to the R_f measurements, however, it is evident that this noise filter comes at the price of greatly reduced recall: at 0.66% filtered recall, the strict projection method successfully projects less

		1	2	3–6	≥7
PGS_t	strict	1.57	1.23	1.29	0.12
	fallback	12.96	9.48	11.64	5.25
	frags	60.62	48.25	42.55	33.34
$PGS_{t,a}$	strict	12.69	8.35	11.16	6.77
	fallback	n.a.	n.a.	n.a.	n.a.
	frags	69.73	51.98	55.30	50.95
parser (Tiger)		93.29	87.22	84.60	83.88
baseline (next)		58.87	0.00	0.00	0.00

Table 3.12: F-score of the direct projections relative to dependency length.

than one edge in one hundred. The 5.78% R_f of the fallback method constitute an improvement but are still far from satisfactory. When fragmented analyses are taken into consideration ('frags') the precision–recall tradeoff is less harsh. In fact, we sacrifice only very little precision in comparison to fallback projection (Δ0.51 P_f), while at the same time the recall is increased by a factor of six (35.74% R_f). This gives rise to an overall F-score of 46.54% in the PGS_t evaluation setting.

In the $PGS_{t,a}$ setting, which is also shown in Table 3.11, the word alignment is not the automatically induced Giza++ alignment that we use in PGS_t, but rather manual alignment link annotations. As mentioned above, this scenario is not applicable to the fallback projection method because the manual alignment in the PGS is undirected. It comes as no surprise that we witness an increase in UAS (+4.10% and +10.71%, respectively) as well as recall (+4.64% and +11.76%) for both strict projection and partial correspondence projection: the manual alignments are more dense than the recall-impaired intersective alignments produced by Giza++. While this has a positive effect on recall, it also reduces precision by 18.54% ('strict') respectively 10.36% ('frags').

For comparison, we also report the performance of a state-of-the-art treebank parser which was trained on the Tiger treebank ('parser (Tiger)', the source parser from Table 3.5), and the performance of a very simple attach-right baseline ('baseline (next)'; see Section 6.3, page 105 for details).

In order to arrive at a more detailed picture of the strengths and shortcomings of our projection algorithms, we further analyze the direct projections with respect to dependency length (Table 3.12) and dependency type (Table 3.13). We discuss the results in turn.

Dependency length. Table 3.12 shows the results of evaluating the projected annotations relative to dependency length, that is, the distance (in words) between a dependent and its head. It should come as no surprise that the f-scores aggregate a precision-recall tradeoff (not shown) which is highly biased towards precision, at the cost of recall. We further notice a general drop in f-score as de-

pendency length increases, with the exception that the tree-oriented strict and fallback projections achieve slightly higher f-scores on dependencies of length 3–6 than on dependencies of length 2.

The corresponding results for the German treebank parser ('parser (Tiger)') show that the same degradation occurs with a state-of-the-art parser. However, the magnitude of the effect is much smaller: the difference in UAS between dependencies of length one and dependencies longer than six amounts to approximately 10%. By contrast, the partial correspondence projections ('frags') sacrifice Δ27.28 (PGS_t) and Δ18.78 ($\text{PGS}_{t,a}$).

An explanation is in order concerning the baseline performance. It is situtated at 58.87% UAS on dependencies between adjacent words (length one), but at 0.00% UAS on all edges longer than one. This is not surprising since our baseline, by design, only introduces dependencies of length one by attaching every word to the word to its right.

Dependency type. Table 3.13 shows the results of factoring the performance by gold standard dependency labels.[19] As a point of reference, let us first look at the scores achieved by the supervised source parser for German ('parser (Tiger)') and the baseline. Note that the baseline actually outperforms the treebank parser on proper names (pnc). This is due to annotation differences between the CoNLL X dependency encoding which was used to train the source parser on the one hand, and the dependency encoding of the CoNLL 2009 shared task for which the TigerXML-to-dependencies conversion tool was designed (cf. footnote 13). The performance of the treebank parser gives an indication of which dependency types are especially hard or easy to recover. According to this reasoning, appositions (app), comparative complements (cc), placeholder phrases (ph), relative clauses (rc), repeated elements (re) and vocatives (vo) are difficult to identify: the treebank parser does so with less than 60% accuracy. On the other hand, expletives (ep) and noun kernel material (nk) are easily identified (>90% UAS). In contrast to the treebank parser, the results of the baseline system do not reflect the *difficulty* of recovering dependencies of a certain type, but instead reflect how often a particular dependency type holds *between adjacent words*. The results show that this is frequently (>80%) the case for components of numeral expressions (nmc) and proper nouns (pnc), as well as for morphological particles (pm). The majority of dependency types, however, can never be identified by our simple baseline (typically by design). This holds true for appositions, comparative complements, conjuncts of a coordinate structure (cj), collocational verb constructions (cvc), phrasal genitives (pg), relative clauses, repeated elements, passivized subjects (sbp), and vocatives.

Turning to the results obtained with the projected annotations, we find that the attachment scores of both tree-oriented projection algorithms ('strict' and 'fallback') are devastating: they rarely exceed 5% UAS, and in fact the large amount of zero-valued cells in the table tell us that many dependency types are *never* projected correctly under these projection methods. Taking fragmented analyses into consideration ('frags') results in improved coverage across most

[19]The list of edge labels shown in Table 3.13 does not exhaustively cover all labels of the Tiger treebank: we ignore some highly infrequent labels. Furthermore, the label CP for complementizers is not included because it does not occur in the converted test set where complementizers govern the subordinate verb rather than being analyzed as a CP-dependent of the latter.

		ag	app	cc	cd	cj	cm	cvc
PGS_t	strict	0.00	0.00	0.00	0.25	0.18	0.00	0.00
	fallback	1.46	5.00	0.00	3.96	5.11	0.00	0.00
	frags	**18.25**	**12.50**	0.00	**43.07**	**35.21**	**8.33**	0.00
$PGS_{t,a}$	strict	1.09	2.50	0.00	4.46	4.23	5.56	0.00
	frags	**18.25**	**17.50**	**5.71**	**52.48**	**42.25**	16.67	**11.76**
baseline (next)		0.73	0.00	0.00	4.21	0.00	**33.33**	0.00
parser (Tiger)		89.05	57.50	31.43	78.71	68.31	83.33	88.24

		da	ep	mo	ng	nk	nmc	oa
PGS_t	strict	0.00	0.00	0.45	0.00	0.53	0.00	1.12
	fallback	6.20	0.00	3.67	5.97	5.81	0.00	6.59
	frags	**22.48**	**25.00**	**22.26**	**14.93**	**48.59**	**62.50**	**32.54**
$PGS_{t,a}$	strict	4.65	4.55	3.85	0.75	5.37	0.00	5.19
	frags	**31.78**	**45.45**	**33.99**	20.90	**57.24**	62.50	**44.04**
baseline (next)		21.71	27.27	22.79	**38.06**	51.87	**87.50**	27.63
parser (Tiger)		68.22	97.73	71.26	62.69	93.96	75.00	77.70

		oc	op	pd	pg	ph	pm	pnc
PGS_t	strict	0.88	0.00	1.15	0.00	0.00	0.00	0.00
	fallback	5.74	7.75	6.32	5.26	0.00	0.00	23.53
	frags	**29.73**	**31.01**	**40.80**	**34.21**	**21.05**	**1.40**	**88.24**
$PGS_{t,a}$	strict	5.68	3.10	9.77	7.89	0.00	0.00	5.88
	frags	**48.72**	**47.29**	**53.45**	**55.26**	**26.32**	**3.50**	**88.24**
baseline (next)		21.76	9.30	31.03	0.00	31.58	85.31	**88.24**
parser (Tiger)		88.14	81.40	83.33	86.84	68.42	100.00	23.53

		rc	re	sb	sbp	svp	vo
PGS_t	strict	0.00	0.00	1.20	0.00	0.00	0.00
	fallback	1.85	0.00	8.49	0.00	8.33	9.30
	frags	**18.52**	**5.97**	**39.84**	**28.57**	**4.17**	**20.93**
$PGS_{t,a}$	strict	2.47	2.99	7.86	4.76	0.00	2.33
	frags	**30.25**	**14.93**	**60.23**	**38.10**	**10.42**	**41.86**
baseline (next)		0.00	0.00	32.27	0.00	**10.42**	0.00
parser (Tiger)		48.15	34.33	91.52	76.19	87.50	44.19

Table 3.13: Evaluation of direct projections from English to German across gold standard dependency types (UAS).

	source lang.		UAS	filtered P_f	filtered R_f	filtered F_f
EP_{de}	en	strict	6.22	85.26	0.69	1.36
		fallback	8.39	62.59	3.28	6.23
		frags	33.14	63.03	30.29	40.92

Table 3.14: Pseudo-evaluation of direct projections for German against the output of the treebank parser.

labels, but cannot remedy the failure on comparative complements and collocational verb constructions. We note that even the partial correspondence projections do not always outperform the baseline (cm, nmc, ng, pm) and attribute this finding to the fact that the baseline is more "consistent" than projection in the sense that it always assigns the same analysis irrespective extraneous factors such as the word alignment.

Another interesting observation that emerges from Table 3.13 is that for some dependency types, namely attributive genitives (ag) and components of numerals and proper nouns, the use of gold standard word alignments for projection ($PGS_{t,a}$) does not improve the projection quality in comparison to the PGS_t setting, which uses automatic alignments. We draw two conclusions: firstly, the automatic word alignment already establishes the correct links for the words involved in those dependencies; secondly, the remaining attachment errors on the labels in question originate from true cross-language divergences.

3.4.2 Pseudo-evaluation against treebank parsers

As explained above, the PGS data covers English–German bitexts. In order to assess the quality of the projections to the remaining target languages considered here (Dutch and Italian), we therefore conduct a *pseudo-evaluation* in which we compare the projected annotations to *treebank parsers*, that is, parsers that were trained on manually annotated sentences, albeit from a different domain. In particular, we use parsers which we will discuss in greater detail in Chapter 5. The parsers are trained on treebank data of the target languages, but they employ the annotation scheme of the respective source language (Section 5.3.3).

We know from Table 3.11 that a parser trained on the German treebank fares reasonably well when evaluated against the manual annotations in the PGS (83.80%). Table 3.14 confirms that the results obtained using the German pseudo-gold standard are to some extent comparable to the results we observe when using the manual PGS annotations: The ranking among the systems (strict < fallback < frags) is the same, by approximately the same magnitudes. We therefore conclude that the output of treebank parsers can be used as a proxy for gold standard data, but more so in relative than in absolute terms.

With this in mind, we now proceed to the pseudo-evaluation results for Dutch (Table 3.15) and Italian (Table 3.16). Both paint a picture similar to the one we observe for German: strict projection produces annotations that exhibit high precision but unacceptably low recall. Relaxation to fallback projections

	source lang.		UAS	filtered P_f	filtered R_f	filtered F_f
EP_{nl}	en	strict	4.90	76.54	0.55	1.10
		fallback	6.77	61.60	2.79	5.34
		frags	33.27	65.16	31.25	42.24
	de	strict	5.51	79.34	1.26	2.49
		fallback	9.01	59.14	5.35	9.81
		frags	27.98	54.09	26.46	35.54

Table 3.15: Pseudo-evaluation of direct projections for Dutch against the output of the treebank parser.

	source lang.		UAS	filtered P_f	filtered R_f	filtered F_f
EP_{it}	en	strict	4.19	58.09	0.31	0.62
		fallback	6.30	47.31	2.81	5.30
		frags	29.60	50.90	27.91	36.05
	de	strict	4.02	66.67	0.11	0.22
		fallback	5.36	41.35	1.79	3.43
		frags	16.77	40.60	14.77	21.66

Table 3.16: Pseudo-evaluation of direct projections for Italian against the output of the treebank parser.

improves recall, but not by much (roughly 1.5%). Finally, partial correspondence projection boosts the f-scores through considerable recall gains at only moderate loss of precision.

Recall from previous sections that the German–Italian alignment and the projections based on this alignment were outliers from a quantitative perspective. The results in Table 3.16 strongly suggest that the language pair is an outlier in qualitative terms, too. Due to the sparse nature of the tree-oriented projection approaches, this manifests itself more clearly in the partial correspondence setting, where we observe a difference of 12.83 percentage points UAS and 14.39 percentage points F_f.

3.5 Summary and Discussion

In this chapter, we have introduced the techniques and ideas underlying annotation projection, and we proposed three algorithms for the projection of syntactic dependencies. The first algorithm (tree-oriented bidirectional projection) is very conservative in that a dependency edge is projected only if the alignment of both the head and the dependent is supported by the word alignment in both directions (source-to-target and target-to-source). Moreover, projected target language analyses are discarded unless they form complete trees. The second algorithm (tree-oriented constrained fallback projection) builds on top of the structures created by bidirectional projection: it attempts to complete the dependency graph by taking weaker (unidirectional) alignment links into consideration as long as they are *compatible* with the bidirectionally projected partial structure. This algorithm is also tree-oriented: incomplete target language structures are rejected. Finally, we proposed partial correspondence projection, which differs from the strict bidirectional projection algorithm in that incomplete target analyses are *not* discarded.

Various heuristic methods are conceivable that could populate incomplete projections of dependency graphs in manners more informed than our constrained fallback or partial correspondence approaches. For example, the fallback to weaker alignments could be confined to configurations (e.g., POS-based patterns) previously observed with strong alignment support. Similarly, the fragmented output of partial correspondence projection could be supplemented by patterns over POS tags, or correction rules as proposed by Hwa et al. (2005). It would also seem promising to employ such heuristic approaches in conjunction with a bootstrapping cycle. However, we focus on techniques that are entirely language independent and will therefore not pursue any heuristic improvements in this thesis. We concentrate instead on a comparison of our three basic, non-enhanced projection settings, neither of which uses additional knowledge, heuristic or otherwise.

Our projection algorithms target different precision-recall tradeoffs. Bidirectional projection is clearly geared towards precision, ignoring all but the most confident alignment links and admitting only complete trees in the output. Constrained fallback projection, on the other hand, is more focused on recall in that it tries to produce a larger number of fully connected dependency graphs even when this requires drawing on alignment links which are poorly supported. In terms of the precision-recall tradeoff, partial correspondence projection opts for a compromise between the two. Like the strict bidirectional projection algo-

rithm, it considers only strongly supported alignment links. But unlike either tree-oriented projection algorithm, partial correspondence projection does not enforce the tree constraint on the output structure. It thereby admits even fragmented dependency graphs in the output.

We quantitatively assessed the output of our three projection algorithms and found that the precision-oriented bidirectional projection induces immense data loss. Since incomplete target language analyses are discarded, the remaining sentences are overly short (7 words or 25% of the sentence length in the original data set), impoverished in terms of vocabulary (15% of the vocabulary in the original data), and highly redundant in that more than 30% of all parse trees projected under bidirectional projection are duplicates (non-unique in the projected data).

Both constrained fallback projection and partial correspondence projection succeed at enhancing the projected dependencies from the purely quantitative perspective. The former boosts the mean sentence length to 12 words and the vocabulary size to 20% of the original (unfiltered) data sets. Partial correspondence projection achieves similar quantitative results with a mean sentence length of 10 words on average and an effective (attached) vocabulary that covers 13.5% of the original vocabulary. Throughout this quantitative assessment, the alignment – and consequentially the projections – from German to Italian stood out as an outlier.

We further conducted a qualitative evaluation of the projection algorithms, using manual gold standard annotations for German and the output of state-of-the-art treebank parsers for Dutch and Italian. The results show that both tree-oriented projection approaches have severe recall issues which lead to f-scores well below 10%. Constrained fallback projection outperforms bidirectional projection in terms of recall, but at the same time sacrifices precision. Partial correspondence projection, which is not geared towards complete tree structures, improves over the tree-oriented algorithms by an order of magnitude, in terms of both recall and f-score. In summary, we can say that partial correspondence imposes a high-precision filter (bidirectionality) while improving recall through relaxed structural requirements (partial annotations).

We conjecture that the idea and usefulness of partial correspondence is not limited to projection frameworks. Given a suitable reliability metric/measure, partial correspondence filters can be devised for any structural or sequential annotations that potentially contain noisy subparts. However, if the annotations are intended to be used as training data for a machine learning algorithm, the algorithm is likely to require adjustment to the presence of partial, incomplete labels.

In the remainder of this thesis, we show how existing dependency parsers can be modified to process training data of this kind. The resulting parsers predict dependency structures with an accuracy which by far exceeds the quality of the direct projections.

Chapter 4

Training Parsers on Fragmented Trees

Models for data-driven dependency parsing can be roughly divided into two paradigms: *graph-based* and *transition-based* models (McDonald and Nivre, 2007; Kübler et al., 2009). In this chapter, we review representative implementations of both paradigms, and present alternative formulations to handle fragmented training data as produced by partial correspondence projection. In particular, we deal with the Malt parser (Nivre et al., 2006) as a representative of the transition-based parsing paradigm, and the MST parser (McDonald et al., 2005) as an instance of graph-based parsing.

Section 4.1 covers the basic notions underlying data-driven dependency parsing and describes the data format used to represent dependency graphs. In Section 4.2, we discuss transition-based parsing, exemplified by the Malt parser, and introduce fMalt as a fragment-aware variant. Section 4.4 describes the graph-based MST parser and proposes fMST to handle fragments in the training data

4.1 Background: Data-driven Dependency Parsing

The term *data-driven dependency parsing* is used to describe parsing frameworks that use machine learning techniques to assign dependency trees to sentences. The systems we consider here employ supervised learning methods in the sense that they rely on annotated training data in order to learn a mapping from input sentences to dependency graphs. Moreover, since neither one of the parsers is grammar-based, any input string is deemed "grammatical" and therefore receives an analysis, even if it would be rejected by a formal grammar of the language at hand. This results in robust parsing models that can be readily applied to a wide range of natural language texts.

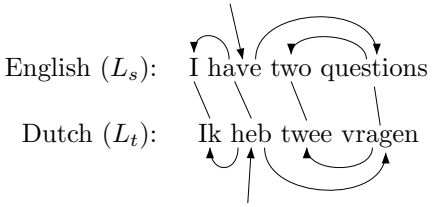


Figure 4.1: Dependency tree projection from English to Dutch.

4.1.1 Basic notions of dependency parsing

In the framework of dependency grammar, the syntactic structure of a sentence is represented as a binary, assymmetric relation over the words in the sentence. The elements of this relation are called *dependencies*, and are represented as *dependency edges* (or *arcs*) between the *head* of the dependency and its *dependent*. To indicate the type of a dependency, the edges are typically labeled. The concrete set of dependency types varies with the annotation scheme, but most distinguish at least subjects, objects, and modifiers (adjuncts). A dependency edge with label r between a head w_i and a dependent w_j is thus represented as a triple (w_i, r, w_j), or graphically as $w_i \rightarrow^r w_j$. Since dependency grammar postulates edges between words and does not assume intermediate constituents, the nodes in a dependency graph correspond directly to the words in the sentence. In addition, most formulations include an artificial *root token* $w_0 =$ ROOT, which is prepended to the sentence as the root of the dependency graph. The role of ROOT will become clear when we discuss the properties of admissible dependency structures below.

Adopting the definition of Kübler et al. (2009), a *dependency graph* for a sentence $S = w_0 w_1 \dots w_n$ is a labeled directed graph $G = (V, A)$ consisting of nodes V and arcs A such that

1. $V \subseteq \{w_0, w_1, \dots, w_n\}$
2. $A \subseteq V \times R \times V$, where R is the set of possible edge labels.
3. if $(w_i, r, w_j) \in A$ then $(w_i, r', w_j) \notin A$ for all $r' \neq r$

Given this definition, the dependency structure for the English sentence in Figure 3.4 – repeated here as Figure 4.1 for convenience – would be represented as follows (labels are not shown in the figure):

$V =$ { ROOT, I, have, two, questions }

$A =$ { (ROOT, ROOT, have), (have, SBJ, I), (have, OBJ, questions), (questions, NMOD, two) }.

The set of *well-formed dependency graphs* is usually restricted to directed spanning trees originating out of node w_0. This definition implies various fundamental properties of dependency trees.

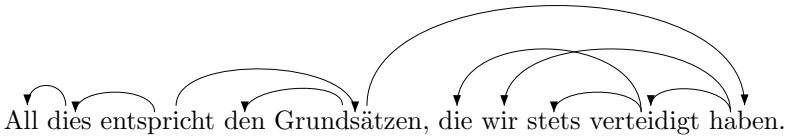
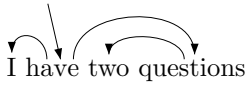


Figure 4.2: Non-projective dependency tree for a German sentence.

First, the *root property* demands that no node dominate w_0. Second, the node set V is required to contain every word in the sentence, that is, $V = V_S = \{w_0, w_1, \ldots, w_n\}$ (*spanning property*), and the graph is *connected*. The spanning property and connectedness obviously cannot be guaranteed when we allow fragmented analyses in the training data. However, they can be reinstated – from a technical point of view – with the help of the artificial ROOT node by creating a dependency edge from ROOT to every word that would otherwise remain unattached. Finally, dependency trees are *acyclic* and they satisfy the *single-head property*, which states that for all $w_i, w_j \in V_S$, if $w_i \rightarrow w_j$ then there does not exist $w_{i'} \in V_S$ such that $i' \neq i$ and $w_{i'} \rightarrow w_j$ (Kübler et al., 2009).

Many parsing algorithms further require dependency trees to be *projective*, that is, without crossing edges. Non-projective dependency edges regularly occur in languages with relatively free word-order, such as Dutch or German, but they can also be found in English, for example in cases of extraposition. Figure 4.2 shows an example from the German Europarl. Although there are parsing algorithms that allow non-projective trees, their flexibility comes at the cost of complexity, and hence reduced efficiency. An attractive alternative to more expressive parsing algorithms is a technique called *pseudo-projectivization* (Nivre and Nilsson, 2005). Pseudo-projectivization methods are applied to the training data prior to training. They reattach non-projective edges at points higher up in the tree where they no longer cross other edges. The information about the original attachment is encoded by means of an augmented label set, so that pseudo-projective edges in the output of the parser can be "deprojectivized" in order to create truly non-projective dependencies where appropriate. The complexity introduced by non-projective arcs is thus shifted from the level of graph structure (A) to the level of dependency labels (R), thus avoiding the need for more complex parsing algorithms. Depending on the machine learning strategy, the augmented label set might itself add complexity to the parsing process, but at most by a constant factor.

The pseudo-projectivization framework of Nivre and Nilsson (2005) is implemented as part of the Malt parser and in fact makes provisions for the treatment of isolated tree fragments, which are called *covered roots* in that context. The Malt parser can attach covered roots to the left, to the right, or to the head of the shortest covering arc. This strategy is intended for cases like dangling mid-sentence punctuation that would otherwise be attached to ROOT and cause spurious non-projective edges. Re-attaching fragments of this kind in a consistent manner streamlines the projectivized label set without sacrificing faithfulness to the original annotation scheme. And in fact, we make use of this option;



ID	FORM	LEMMA	CPOSTAG	POSTAG	HEAD	DEPREL
1	I	I	PP	PP	2	SBJ
2	have	have	VHP	VHP	0	ROOT
3	two	two	CD	CD	4	NMOD
4	questions	question	NNS	NNS	2	OBJ

Figure 4.3: Textual representation of a dependency tree in the CoNLL-X format.

however, we do not incorporate the resulting attachments into the training data of the machine learning component of the parsers (as discussed in subsequent sections). The covered root treatment in our case therefore merely serves to reduce the number of pseudo-projectivized dependency edges, but does not affect attachment proper.

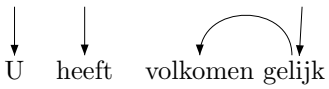
4.1.2 Textual representation of dependency graphs

We use the CoNLL-X data format for dependency trees (Buchholz and Marsi, 2006) to represent dependency graphs, both complete and fragmented.

The CoNLL-X data format encodes one word per line. The columns in each line contain information that describes the word (ID, FORM, LEMMA, (C)POSTAG) and identifies the word's head (HEAD) as well as the dependency type (DEPREL).[1] Figure 4.3 exemplifies the data format for the dependency tree in Figure 4.1. We see that every dependent in the tree lists the index (ID) of its head in the HEAD field, with the corresponding edge label in the DEPREL column. As a special case, the head of the word *have* is the artifical (implicit) root token w_0 referred to by the index 0. This dependency is labeled ROOT.

Now, when we want to represent a fragmented analysis, we will also use the ROOT token to represent the unique root node of the dependency graph, thus encoding a wellformed tree in the technical sense. That is, every word that is the root of a fragment specifies w_0 as its head. However, the edge is labeled by a special relation "FRAG" in order to distinguish it from a true root dependency. Thus, sentences with a fragmented parse are represented as a single sentence with a single dependency graph, just like sentences with complete analyses; the only difference from a fully parsed sentence is that unconnected substructures

[1]The original format further defines the columns PHEAD and PDEPREL which can be used to specify alternative projective edges, and a column FEATS to store additional properties (e.g., morphological). These fields are always empty in our data and hence omitted here. We also note that none of our parsing models actually employs the LEMMA field, and the CPOSTAG field, which stores a coarse-grained part-of-speech tag, is only used to split the training data for the SVMs (cf. Section 4.2).



ID	FORM	LEMMA	CPOSTAG	POSTAG	HEAD	DEPREL
1	U	u	pron	pronpers	0	FRAG
2	heeft	hebben	verb	verbpressg	0	FRAG
3	volkomen	volkomen	adj	adj	4	AMOD
4	gelijk	gelijk	adj	adj	0	FRAG

Figure 4.4: Textual representation of a fragmented dependency graph.

are attached directly under w_0. For instance, the partial projected parse in Figure 3.6 is represented as depicted in Figure 4.4.

4.2 Background: Transition-Based Parsing with Malt

Transition-based models for dependency parsing construct dependency structure in a stepwise fashion by applying parser actions according to a transition system. In this and the following sections, we use the terminology laid out in Kübler et al. (2009). A detailed discussion of data-driven transition-based dependency parsing can also be found in Nivre (2006).

The specific parser we describe in the following sections is the Malt parser (version 1.4) of Nivre et al. (2006).

4.2.1 Transition system

The default transition system implemented in the Malt parser amounts to basic shift-reduce parsing, with two internal data structures: an *input buffer* containing the input tokens that remain to be processed, and a *stack* of partially processed words. The current state of these two data structures together with A, the set of dependency edges created so far form a *parser configuration* $c = (\sigma, \beta, A) \in \mathcal{C}$, where

1. σ is a stack of words $w_i \in V$
2. β is a buffer of words $w_i \in V$
3. A is a set of dependency arcs $(w_i, r, w_j) \in V \times R \times V$.

Furthermore, for any sentence $S = w_0 w_1 \dots w_n$, there is exactly one *initial* configuration $c_0(S) = ([w_0], [w_1, \dots, w_n], \emptyset)$. Any configuration of the form $(\sigma, [], A)$, for any σ and A, is called a *terminal* configuration. In other words,

Shift

$$(\sigma, w_i|\beta, A) \quad \Rightarrow \quad (\sigma|w_i, \beta, A)$$

Left-Arc$_r$ if $i \neq 0$

$$(\sigma|w_i, w_j|\beta, A) \quad \Rightarrow \quad (\sigma, w_j|\beta, A \cup \{(w_j, r, w_i)\})$$

Right-Arc$_r$

$$(\sigma|w_i, w_j|\beta, A) \quad \Rightarrow \quad (\sigma, w_i|\beta, A \cup \{(w_i, r, w_j)\})$$

Figure 4.5: Arc-standard transition system.

the parser is initialized with an empty arc set and with $w_0 = \text{ROOT}$ on the stack and all other words in the input buffer. The parser is done when all words in the input buffer have been processed.

Given a configuration, the transition system defines the set $\mathcal{T}$ of permissible *transitions* that can be applied to derive the next configuration. A transition is a parser action, and the Malt parser defines three types of actions: SHIFT, LEFT-ARC, and RIGHT-ARC. The two latter actions are further parametrized over edge labels. Figure 4.5 shows the definition of the transition system for shift-reduce dependency parsing.[2] The SHIFT action simply removes the next input token from the buffer and pushes it on top of the stack. It does not introduce any dependency arcs and is applicable whenever the input buffer is non-empty. The transition LEFT-ARC$_r$ introduces a dependency of type r from the word on top of the buffer to the word on top of the stack, and the dependent is removed from the stack. The condition that $i \neq 0$ ensures that the resulting dependency graph satisfies the root property. Conversely, RIGHT-ARC$_r$ adds an arc with label r from the word on top of the stack to the next input token. The input token is then removed from the buffer and replaced by the word on top of the stack.

A *transition sequence* for a sentence S in this transition system is a sequence of configurations $(c_0, c_1, \ldots, c_m)$ such that c_0 is the initial configuration for S, c_m is terminal configuration, and each c_i $(1 \leq i \leq m)$ can be derived from the preceding configuration by means of a permissible transition $t \in \mathcal{T}$: $c_i = t(c_{i-1})$. The dependency tree derived for S is then $G_{c_m} = (V_S, A_{c_m})$ where A_{c_m} is the arc set in the terminal configuration c_m. Figure 4.6 shows the transition sequence for the tree in Figure 3.4. Nivre (2008) formally proves that the transition

[2]The transition system discussed above is of the "arc-standard" flavor. This means that a word is only attached to its head when all its dependents have already been identified. An arc-eager variant which introduces edges as soon as possible is also available in the Malt parser system. The arc-eager transition system uses an additional action REDUCE and requires slight modifications of LEFT-ARC and RIGHT-ARC:

Reduce if $(w_k, r', w_i) \in A$

$$(\sigma|w_i, \beta, A) \quad \Rightarrow \quad (\sigma, \beta, A)$$

Left-Arc$_r$ if $(w_k, r', w_i) \notin A$ and $i \neq 0$

$$(\sigma|w_i, w_j|\beta, A) \quad \Rightarrow \quad (\sigma, w_j|\beta, A \cup \{(w_j, r, w_i)\})$$

Right-Arc$_r$

$$(\sigma|w_i, w_j|\beta, A) \quad \Rightarrow \quad (\sigma|w_i|w_j, \beta, A \cup \{(w_i, r, w_j)\})$$

$$
\begin{array}{rlll}
 & ([w_0], & [\text{Ik}, \text{heb}, \text{twee}, \text{vragen}], & \emptyset) \\
\text{SH} \Rightarrow & ([w_0, \text{Ik}], & [\text{heb}, \ldots], & \emptyset) \\
\text{LA}_{\text{SBJ}} \Rightarrow & ([w_0], & [\text{heb}, \ldots], & A_1 = \{(\text{heb}, \text{SBJ}, \text{Ik})\}) \\
\text{SH} \Rightarrow & ([w_0, \text{heb}], & [\text{twee}, \ldots], & A_1) \\
\text{SH} \Rightarrow & ([w_0, \text{heb}, \text{twee}], & [\text{vragen}], & A_1) \\
\text{LA}_{\text{NMOD}} \Rightarrow & ([w_0, \text{heb}], & [\text{vragen}], & A_2 = A_1 \cup \{(\text{vragen}, \text{NMOD}, \text{twee})\}) \\
\text{RA}_{\text{OBJ}} \Rightarrow & ([w_0], & [\text{heb}], & A_3 = A_2 \cup \{(\text{heb}, \text{OBJ}, \text{vragen})\}) \\
\text{RA}_{\text{ROOT}} \Rightarrow & ([], & [w_0], & A_4 = A_3 \cup \{(w_0, \text{ROOT}, \text{heb})\}) \\
\text{SH} \Rightarrow & ([w_0], & [], & A_4)
\end{array}
$$

Figure 4.6: Transition sequence for the dependency tree in Figure 3.4, annotated with transitions.

a. **parse**(S, o)
$c \leftarrow c_0(S)$
while c is not terminal
$\boxed{t^* \leftarrow o(c)}$
$c \leftarrow t^*(c)$
return G_c

b. **parse**(S, g)
$c \leftarrow c_0(S)$
while c is not terminal
$\boxed{\hat{t} \leftarrow g(\mathbf{f}(c))}$
$c \leftarrow \hat{t}(c)$
return G_c

Figure 4.7: Deterministic transition-based algorithm with a. an oracle and b. a classifier.

system in Figure 4.5 derives exactly the set of projective dependency trees.

4.2.2 Parsing algorithm

What remains to be addressed is the parsing algorithm: How does the parser derive a concrete transition sequence given an input sentence? Note that the transition system as such is non-deterministic, that is, there may be more than one transition permissible for a given configuration. A greedy, deterministic parsing algorithm is given in Figure 4.7a. The algorithm starts with the initial configuration and deterministically derives the correct transition sequence – and thus the correct parse tree – by applying the transition returned by the *oracle* o. The oracle is a function which returns the optimal transition t^* given the current configuration c.

Unfortunately, oracle functions are somewhat tricky to procure in real life. But recall that the Malt parser is a supervised parser which assumes the availability of a training set $\mathcal{D}$ annotated with complete trees of gold standard quality:

$$\mathcal{D} = \{(S_d, G_d)\}_{d=1}^{|\mathcal{D}|} \tag{4.1}$$

Then for each training example $(S_d, G_d = (V_d, A_d)) \in \mathcal{D}$, the oracle $o_{\mathcal{D}}$ can be constructed by "reverse-engineering" the transition system using the knowledge from A_d as follows:

	Attribute	**Address**
f_1	POSTAG	$\sigma[0]$
f_2	POSTAG	$\beta[0]$
f_3	POSTAG	$\beta[1]$
f_4	POSTAG	$\beta[2]$
f_5	POSTAG	$\beta[3]$
f_6	POSTAG	$\sigma[1]$
f_7	DEPREL	$\mathsf{ldep}(\sigma[0])$
f_8	DEPREL	$\mathsf{rdep}(\sigma[0])$
f_9	DEPREL	$\mathsf{ldep}(\beta[0])$
f_{10}	DEPREL	$\mathsf{rdep}(\beta[0])$
f_{11}	FORM	$\sigma[0]$
f_{12}	FORM	$\beta[0]$
f_{13}	FORM	$\beta[1]$
f_{14}	FORM	$\mathsf{head}(\sigma[0])$

Table 4.1: Feature model for arc-standard parsing.

$$o(c = (\sigma, \beta, A)) = \begin{cases} \text{LEFT-ARC}_r & \text{if } (\beta[0], r, \sigma[0]) \in A_d \\ \text{RIGHT-ARC}_r & \text{if } (\sigma[0], r, \beta[0]) \in A_d \text{ and, for all } w, r', \\ & \text{if } (\beta[0], r', w) \in A_d, \text{then } (\beta[0], r', w) \in A \\ \text{SHIFT} & \text{otherwise} \end{cases}$$

At first glance, $o_{\mathcal{D}}$ may seem to be of little use, since it is only defined for sentences that are already annotated with the correct parse. But we will see shortly how it allows the Malt parser to train a classifier $g_{\mathcal{D}}$ which can replace the oracle to parse free text. This data-driven version of the parsing algorithm is outlined in Figure 4.7b. It differs from the oracle-driven algorithm in that it chooses the transition $\hat{t}$ that is predicted by the classifier g (specifically, $g_{\mathcal{D}}$) given a feature representation $\mathbf{f}(c)$ of the current configuration c.

4.2.3 Feature model

The Malt parser approximates the oracle with a classifier g that is trained on the training set $\mathcal{D}$. More specifically, the function $o_{\mathcal{D}}$ is applied successively to each training example $(S_d, G_d) \in \mathcal{D}$ to derive the transition sequence $C_d = (c_0, c_1, \ldots, c_m)$ corresponding to G_d. Each non-terminal configuration $c_i (0 \leq i < m)$ then gives rise to a training instance $(\mathbf{f}(c_i), t_i)$ for the classifier g. t_i is the transition that is applied to c_i in C_d, that is, $t_i(c_i) = c_{i+1}$, and $\mathbf{f}(c_i)$ is a feature representation of c_i.

The exact make-up of the feature representation is defined in terms of *feature functions*. In the Malt parser, a feature function $(v \circ a)(c) : \mathcal{C} \rightarrow Y$ is composed of

1. an *address function* $a(c) : \mathcal{C} \rightarrow V$, and
2. an *attribute function* $v(w) : V \rightarrow Y$.

The Malt parser defines a wide range of address functions; we restrict our discussion to those used in the default feature model for the arc-standard parser, which is given in Table 4.1.[3] The most basic address functions extract the node at a given position in the stack or the input buffer. For instance, $\sigma[0]$ extracts the word on top of the stack, $\sigma[1]$ the second word from the top of the stack, $\beta[3]$ the fourth word from the front of the buffer, and so forth. The functions head, ldep and rdep extract the head, the leftmost dependent and the rightmost dependent of a given node, respectively. The attribute functions used in Table 4.1 extract the value in the corresponding input column, as described in Section 4.1.2.

To illustrate the mapping from configurations to feature representations, Figure 4.8 shows the training instances $(\mathbf{f}(c_i), t_i)$ constructed from the transition sequence in Figure 4.6. Note that the final SHIFT transition generates no training instance. This is because SHIFT trivially is the only permissible action when $\beta[0] = w_0$.

Finally, the training instances $(\mathbf{f}(c_i), t_i)$ are passed on to the machine learning mechanism of choice to train the oracle approximation $g_{\mathcal{D}}$. In our case, this is the libsvm package of Chang and Lin (2001), which implements support vector machines (SVMs; Vapnik, 1995).

4.3 fMalt

The system we have described so far (i.e., the original Malt parser system) expects that the training data consists of complete trees. This section proposes a variant of Malt which relaxes this assumption. We call the system fMalt, for *filtering Malt*, or *fragment-aware Malt*.

The ultimate challenge which we need to address in defining fMalt lies in the discrepancy between the training data and the expected output structures: our training data consists of projected dependency graphs, which may or may not be complete. To reiterate the point made in Chapter 3, we have observed that aggressive filtering of the projected annotations leads to a sharply pronounced, undesirable bias towards simplistic sentences. A data set that is restricted to examples with a complete projected tree can therefore hardly be considered a representative sample. We have shown that the situation can be improved if we take into account partial correspondences and project tree fragments. However, we do not merely want to train a partial parser, but expect the output of our parser to be complete trees, just like those produced by a regular treebank parser. This means we have to find a way to "hide" the fragmentation from the machine learning component of the parser in order to overcome the gap between incomplete training annotations and complete output annotations.

At the same time, we need to consider that although the oracle approximation $g_{\mathcal{D}}$ performs local optimization, it does take into account context information about neighboring words. In the feature model at hand (Table 4.1), the context takes the form of the next three words in the input buffer β as well as the preceding word on the stack σ. The context contributes with the attributes POSTAG and FORM and hence provides valuable lexical information. However, some of the features in the model refer to dependency structure. Taking

[3] See `http://maltparser.org/userguide.html#featurespec` for a full list of available feature functions.

$\mathbf{f_1}$	$\mathbf{f_2}$	$\mathbf{f_3}$	$\mathbf{f_4}$	$\mathbf{f_5}$	$\mathbf{f_6}$	$\mathbf{f_7}$	$\mathbf{f_8}$	$\mathbf{f_9}$	$\mathbf{f_{10}}$	$\mathbf{f_{11}}$	$\mathbf{f_{12}}$	$\mathbf{f_{13}}$	$\mathbf{f_{14}}$	$\mathbf{t}$
ROOT	pron	verb	num	noun	NULL	NULL	NULL	NULL	NULL	ROOT	Ik	heb	NULL	SH
pron	verb	num	noun	NULL	ROOT	NULL	NULL	NULL	NULL	Ik	heb	twee	NULL	LA_{SBJ}
ROOT	verb	num	noun	NULL	NULL	NULL	NULL	SBJ	NULL	ROOT	heb	twee	NULL	SH
verb	num	noun	NULL	NULL	root	SBJ	NULL	NULL	NULL	heb	twee	vragen	NULL	SH
num	noun	NULL	NULL	NULL	verb	NULL	NULL	NULL	NULL	twee	vragen	NULL	NULL	LA_{NMOD}
verb	noun	NULL	NULL	NULL	ROOT	SBJ	NULL	NMOD	NULL	heb	vragen	NULL	NULL	RA_{OBJ}
ROOT	verb	NULL	NULL	NULL	NULL	NULL	NULL	SBJ	OBJ	ROOT	heb	NULL	NULL	RA_{ROOT}

Figure 4.8: Training instances $(\mathbf{f}(c), t)$ for classifier g.

$\mathbf{f_1}$	$\mathbf{f_2}$	$\mathbf{f_3}$	$\mathbf{f_4}$	$\mathbf{f_5}$	$\mathbf{f_6}$	$\mathbf{f_7}$	$\mathbf{f_8}$	$\mathbf{f_9}$	$\mathbf{f_{10}}$	$\mathbf{f_{11}}$	$\mathbf{f_{12}}$	$\mathbf{f_{13}}$	$\mathbf{f_{14}}$	$\mathbf{t}$
ROOT	adj	adj	NULL	NULL	NULL	NULL	NULL	NULL	NULL	ROOT	volkomen	gelijk	NULL	SH
adj	adj	NULL	NULL	NULL	ROOT	NULL	NULL	NULL	NULL	volkomen	gelijk	NULL	NULL	LA_{AMOD}

Figure 4.9: Training instances $(\mathbf{f}(c), t)$ for fMalt classifier g_f.

$$
\begin{array}{rlll}
 & ([w_0], & [\text{U}, \text{heeft}, \text{volkomen}, \text{gelijk}], & \emptyset) \\
\triangleright \text{RA}_{\text{FRAG}} \Rightarrow & ([], & [w_0, \text{heeft}, \ldots], & A_1 = \{(w_0, \text{FRAG}, \text{U})\}) \\
\text{SH} \Rightarrow & ([w_0], & [\text{heeft}, \ldots], & A_1) \\
\triangleright \text{RA}_{\text{FRAG}} \Rightarrow & ([], & [w_0, \text{volkomen}, \ldots], & A_2 = A_1 \cup \{(w_0, \text{FRAG}, \text{heeft})\}) \\
\text{SH} \Rightarrow & ([w_0], & [\text{volkomen}, \ldots], & A_2) \\
\text{SH} \Rightarrow & ([w_0, \text{volkomen}], & [\text{gelijk}], & A_2) \\
\text{LA}_{\text{AMOD}} \Rightarrow & ([w_0], & [\text{gelijk}], & A_3 = A_2 \cup \{(\text{gelijk}, \text{AMOD}, \text{volkomen})\}) \\
\triangleright \text{RA}_{\text{FRAG}} \Rightarrow & ([], & [w_0], & A_4 = A_3 \cup \{(w_0, \text{FRAG}, \text{gelijk})\})
\end{array}
$$

Figure 4.10: Transition sequence for the incomplete dependency graph in Figure 4.4. Filtered transitions are marked with arrowheads.

the dependencies in the training data at face value, the feature model as defined above is prone to incorporate aspects of the parser configuration that are residual of the provisional attachment of fragments to ROOT, which is misleading at the least, or even plain wrong. The remainder of this section describes the differences between Malt and fMalt in detail.

When we described the textual representation of parse tree fragments as dependents of the artificial root node in a strictly technical sense (Section 4.1.2), we said that these arcs are distinguished from proper root dependencies by means of a special label FRAG. Now, to the Malt parser, FRAG is of course a label like any other and consequently is not given any special treatment. In the training phase, FRAG edges are accordingly assumed to indicate the correct attachment of the fragment root. By contrast, their intended meaning is that the correct attachment is unknown. Basically, what we aim for is a parser that learns from whatever dependencies are provided in the training data but acknowledges the fact that the input is incomplete by ignoring FRAG edges.

We achieve this behavior by modifying the parser in two respects. First, when constructing training instances $(\mathbf{f}(c_i), t_i)$ for a training example (S_d, G_d), fMalt simply omits those instances where c_i is of the form $(\sigma, w|\beta, A)$ and there exists an arc (ROOT, FRAG, w) $\in A_d$ in the "gold standard" tree. This change prevents the provisional attachment decisions concerning fragment roots from entering into the training data for the classifier $g_{\mathcal{D}}$ directly. Figure 4.10 shows the transition sequence for a sentence with a fragmented analysis. The transitions for which fMalt does not generate a training instance are marked with arrowheads.

The second modification ensures that information about fragment attachment does not leak into the feature representations $\mathbf{f}(c_i)$. Recall that the feature model in Table 4.1 refers to the dependency structure built by $o_{\mathcal{D}}$ in the transitions that have derived c_i. This means that the address functions head, ldep and rdep as well as the attribute function deprel potentially introduce faulty information into the feature vector representing an otherwise perfectly legitimate transition. Again, fMalt solves this problem by omission: while letting the oracle $o_{\mathcal{D}}$ access the complete training example (in particular, A_d including FRAG edges) so as not to disrupt the transition sequence, we do prevent it from actually adding FRAG edges to the arc set A. It is this set A that the feature functions in question draw from when constructing the feature vectors.

Conceptually, fMalt accommodates the distinguished treatment of FRAG edges as a special case in the transition system:[4]

$$\text{(4.2)} \quad \textbf{Right-Arc}_{\text{FRAG}} \quad (\sigma|w_i, w_j|\beta, A) \quad \Rightarrow \quad (\sigma, w_i|\beta, A)$$

Sticking with the example in Figure 4.10, this means that none of the FRAG arcs are actually added and A_4 contains only the AMOD dependency. The resulting feature representations are shown in Figure 4.9.

Given that the affected RIGHT-ARC transitions are also excluded from the training data for the classifier, their purpose is thereby reduced to advancing the state of the data structures σ and β. In summary, the two modifications allow fMalt to emulate complete transition sequences for fragmented training examples without letting the data-driven component g be affected by the spurious attachments necessitated by wellformedness constraints.

4.4 Background: Graph-Based Parsing with MST

The graph-based approach to dependency parsing differs from transition-based parsing in two important respects. Firstly, the parsing algorithm operates directly on trees; that is, there is no equivalent of transition sequences or similar intermediate representations. Secondly, the objective in graph-based parsing is global optimization (or at least some approximation thereof); by contrast, the transition-based approach implemented in Malt and fMalt proceeds in a greedy, locally optimal fashion.

Graph-based parsing, as the name suggests, employs techniques and algorithms from graph theory to solve the parsing problem. The motivation for such an approach is obvious: dependency structures as defined in Section 4.1 lend themselves quite naturally to an interpretation as directed acyclic graphs, for which computer science provides many well-understood algorithms (see, e.g., Cormen et al., 2001).

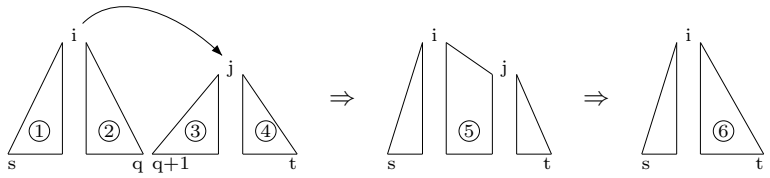
The graph-based parser we consider in this dissertation is the MST parser (McDonald et al., 2005). It solves the parsing task by finding maximum spanning trees (hence the name MST): given a sentence S, it finds the graph $\hat{G}$ such that

$$\hat{G} = \underset{G \in \mathcal{G}_S}{\operatorname{argmax}}\, s(G) \tag{4.3}$$

where $\mathcal{G}_S$ is the set of all well-formed spanning trees for S rooted in w_0, and s is a *scoring function*.

In Section 4.4.1, we briefly describe an efficient parsing algorithm that solves the maximization for projective dependency trees. We then describe in Section 4.4.2 how the MST parser uses the training data to learn the scoring function s, and finally propose the fragment-aware variant fMST in Section 4.5.

[4] In the implementation of fMalt, the transition system is not actually altered. Instead, the feature functions are redefined to the effect that address functions return the value NULL if the path to the requested address involves a FRAG edge. Likewise, the attribute function deprel returns NULL instead of FRAG.



① $E[s][i][left][true]$ ③ $E[q+1][j][left][true]$ ⑤ $E[i][j][right][false]$
② $E[i][q][right][true]$ ④ $E[j][t][right][true]$ ⑥ $E[i][t][right][true]$

Figure 4.11: Addition of an arc from w_i to w_j with Eisner's algorithm.

4.4.1 Parsing algorithm

In the MST parser, parsing is couched in an arc-factored parametrization. This means that the score $s(G)$ for a graph $G = (V, A)$ is computed as a combination of the scores of its arcs:

$$s(G) = \sum_{(w_i,r,w_j)\in A} s(w_i, r, w_j), \tag{4.4}$$

and the parsing problem can be reformulated as finding

$$\hat{G} = \operatorname*{argmax}_{G=(V,A)\in\mathcal{G}_S} \sum_{(w_i,r,w_j)\in A} s(w_i, r, w_j) \tag{4.5}$$

The MST parser implements two alternative parsing algorithms. One is the Chu-Liu-Edmonds spanning tree algorithm (Chu and Liu, 1965; Edmonds, 1967), which is used for non-projective dependency parsing. Since we are assuming (pseudo-)projective dependency graphs throughout the thesis, we do not discuss this algorithm here and instead refer the interested reader to Kübler et al. (2009).

The other spanning tree algorithm used in the MST parser is the projective algorithm of Eisner (1996, 2000). While a naïve CKY-style chart-parsing algorithm would require an $O(n^5)$ runtime to populate the chart (Kübler et al., 2009), Eisner's algorithm runs in cubic time. This is achieved by processing and storing left and right dependents separately, and introducing new edges (w_i, r, w_j) by combining w_i together with its left/right children with w_j and its right/left children in an intermediate subgraph that has w_i at one periphery and w_j at the other. This is illustrated in Figure 4.11. The left and right dependents of w_i are represented by the triangles labeled ① and②, respectively, and likewise for the dependents of w_j (③ and ④). When adding w_j as a new right dependent of w_i, the algorithm first creates a chart entry for the subgraph⑤, which spans the right dependents of w_i (②) and the left dependents of w_j (③). In a second step, the intermediate subgraph is merged with the right dependents of w_j (④) to form the new subtree ⑥ which covers the complete extent of w_i's right dependents.

Subgraphs are stored in the dynamic programming table E. Each entry $E[s][t][d][c]$ contains the score of the highest weighted subgraph spanning w_s to w_t, and since left and right dependents are stored separately, the arc direction

parse(S, score)
sentence $S = w_0 w_1 \dots w_n$
scoring function score (cf. Section 4.4.2)

1 Initialization: $E[s][s][d][c] = 0$
for all $0 \leq s \leq n, d \in \{\mathit{left}, \mathit{right}\}, c \in \{\mathit{true}, \mathit{false}\}$
2 **for** $m : 1..n$
3 **for** $s : 0..n$
4 $t = s + m$
5 **if** $t > n$ **then break**

6 $E[s][t][\mathit{left}][\mathit{false}] = \max_{\substack{s \leq q < t \\ r \in R}} \text{score}(w_t, r, w_s) + E[s][q][\mathit{right}][\mathit{true}] + E[q+1][t][\mathit{left}][\mathit{true}]$
7 $E[s][t][\mathit{right}][\mathit{false}] = \max_{\substack{s \leq q < t \\ r \in R}} \text{score}(w_s, r, w_t) + E[s][q][\mathit{right}][\mathit{true}] + E[q+1][t][\mathit{left}][\mathit{true}]$

8 $E[s][t][\mathit{left}][\mathit{true}] = \max_{\substack{s \leq q < t \\ r \in R}} E[s][q][\mathit{left}][\mathit{true}] + E[q][t][\mathit{left}][\mathit{false}]$
9 $E[s][t][\mathit{right}][\mathit{true}] = \max_{\substack{s \leq q < t \\ r \in R}} E[s][q][\mathit{right}][\mathit{false}] + E[q][t][\mathit{right}][\mathit{true}]$

10 **return** backpointers$[0][n][\mathit{right}][\mathit{true}]$

Figure 4.12: Arc-factored projective parsing with Eisner's algorithm.

$d \in \{\mathit{left}, \mathit{right}\}$ implies that the head is either the rightmost word w_t (when $d = \mathit{left}$) or the leftmost word w_s ($d = \mathit{right}$). The flag c indicates whether the subgraph is complete ($c = \mathit{true}$) or intermediate ($c = \mathit{false}$).

The algorithm is outlined in Figure 4.12. It starts by initializing all subgraphs of length 1 to a weight of 0 in line 1. It then builds larger subgraphs by iterating over spans of increasing lengths (line 2), from left to right (lines 3–5). In lines 6 and 7, the chart is filled with entries for intermediate subgraphs like ⑤ in Figure 4.11. The algorithm finds the optimal such subgraph by maximizing over possible split points q and arc labels r. Similarly, lines 8 and 9 define how the intermediate subgraphs are combined with the remaining, adjacent dependents to form complete subgraphs.

Finally, the score of the maximum spanning tree for the entire sentence can be found in the entry $E[0][n][\mathit{right}][\mathit{true}]$, which spans words w_0 through w_n and is rooted in the leftmost word w_0. Since the chart E merely contains the *score* of the optimal tree, retrieval of the actual graph structure requires some additional book-keeping. We leave the details of the reconstruction unspecified here and assume that an appropriate auxiliary table *backpointers* contains the relevant information about split points and arc labels (line 10).

	w_{i-1}	w_i	w_{i+1}	w_k $i<k<j$	w_{j-1}	w_j	w_{j+1}
Unigram							
		form, pos					
		form&pos					
						form, pos	
						form&pos	
Bigram							
		form&pos				form&pos	
		form&pos				form, pos	
		form, pos				form&pos	
		form				form	
		pos				form	
Intervening							
		pos		pos		pos	
Surrounding							
		pos	pos		pos	pos	
	pos	pos			pos	pos	
		pos	pos			pos	pos
	pos	pos				pos	pos

Table 4.2: Feature model for first-order arc-factored parsing. The ampersand ('&') denotes feature conjunction.

4.4.2 Scoring function

The scoring function $s : V \times R \times V \rightarrow \mathbb{R}$ is the crucial element of the parsing algorithm in finding the optimal spanning tree. It is defined as a function that assigns real-valued scores to dependency edges. More precisely, it arrives at the score for a given edge (w_i, r, w_j) by multiplying the feature representation $\mathbf{f}(w_i, r, w_j) \in \mathbb{R}^m$ with a *weight vector* $\mathbf{w} \in \mathbb{R}^m$:

$$s(w_i, r, w_j) = \mathbf{w} \cdot \mathbf{f}(w_i, r, w_j) = \sum_{1 \leq k \leq m} w_k f_k(w_i, r, w_j) \tag{4.6}$$

A schematic representation of the specific features that make up the feature vector $\mathbf{f}(w_i, r, w_j)$ is shown in Table 4.2. Unigram features refer to the surface form ('form') or the POS tag ('pos') of either w_i or w_j in isolation, whereas bigram features combine the unigram features of these two words. The features listed under the heading 'intervening' additionally include information about the words intervening between the head and the dependent (namely, their POS tags). Features over surrounding words combine the POS-based bigram features with the POS tag of directly adjacent words. In addition, all features in the table are also combined with the arc direction and the distance between w_i and w_j. In contrast with the Malt feature models, MST incorporates no parse history, that is, the feature representation of an edge does not encode information about other edges.

mira$(\mathcal{D}, N)$
$\mathcal{D}$: training data
N: number of epochs

1 $\mathbf{w}^{(0)} = \mathbf{0}; \mathbf{v} = \mathbf{0}; i = 0$
2 **for** $n : 1..N$
3 **for** $d : 1..|\mathcal{D}|$

4 $\min \|\mathbf{w}^{(i+1)} - \mathbf{w}^{(i)}\|$
5 s.t. $s_{i+1}(G_d) - s_{i+1}(G') \geq L(G', G_d)$
6 where $G' = \text{parse}(S_d, s_i)$

7 $\mathbf{v} = \mathbf{v} + \mathbf{w}^{(i+1)}$
$i = i + 1$
8 $\mathbf{w} = \mathbf{v}/(N \times |\mathcal{D}|)$
9 **return w**

Figure 4.13: MIRA algorithm for online learning.

The MST parser learns $\mathbf{w}$ from the training data $\mathcal{D}$ using an online learning algorithm called MIRA (Margin Infused Relaxed Algorithm; Crammer and Singer, 2003). Online algorithms are presented with one training example at a time. They immediately update the weights so as to correctly classify the current example, while keeping the change to the weight vector as small as possible. Upon seeing a new training example $(S_d, G_d) \in \mathcal{D}$, MIRA adjusts the weight vector $\mathbf{w}$ to establish a margin between the correct tree G_d and the tree that receives the highest score according to the old $\mathbf{w}$. Pseudo-code for the algorithm is given in Figure 4.13. First, the weights $\mathbf{w}$ and an auxiliary vector $\mathbf{v}$ are initialized to zero (line 1). The algorithm makes a pre-specified number N of passes over the training set (line 2). In each pass, each of the training examples $(S_d, G_d) \in \mathcal{D}$ is considered in turn (line 3), and MIRA attempts to keep the new weight vector $\mathbf{w}^{(i+1)}$ as close to the old one as possible (line 4), subject to scoring the correct tree G_d highest with a margin at least as large as the loss of the highest scored incorrect tree (lines 5 and 6). The loss of a tree is defined as the number of words with incorrect parents relative to the correct tree:

$$L(G', G_d) = |\{w_j : (w_i, r, w_j) \in A' \setminus A_d\}| \tag{4.7}$$

We write s_i to refer to the scoring function s using the weight vector $\mathbf{w}^{(i)}$. Thus, the margin in line 5 is computed using the new weights $\mathbf{w}^{(i+1)}$, whereas the highest scoring tree G' is of course determined using the weights $\mathbf{w}^{(i)}$ from the previous iteration. Note that when G' already coincides with G_d the weight vector will not be updated because the loss is zero and the constraint in line 5 is satisfied trivially.

In line 7, the weights are accumulated in the auxiliary vector $\mathbf{v}$, and the final weight vector $\mathbf{w}$ that is returned in line 9 is the average of all vectors $\mathbf{w}^{(i)}$.[5]

[5]This is very similar to the averaged perceptron of Collins (2002) who shows that averaging

4.5 fMST

In the training phase, the original MST parser tries to maximize the scoring margin between the correct training example and the candidate that currently scores best. However, if the training example is fragmented, it is not strictly speaking correct, in the sense that it does not coincide with the desired parse tree. In fact, this desired tree is among the other possible trees that MST assumes to be suboptimal. In order to relax this assumption, a fragment-aware version of MST has to ensure that the loss of the desired (complete) tree is zero. This section presents fMST as one such parser. Much like fMalt, fMST masks fragmentation in the training data by essentially ignoring FRAG edges.

The vast pool of literature on the topic of unsupervised and semi-supervised parsing makes it abundantly clear that we are not the first to suggest learning from incomplete annotations. But to our knowledge, the idea of dealing with missing annotations by *omission* (as opposed to, e.g., explicit marginalization) is novel. Chapter 2 provides an overview of some of the proposals most closely related to ours, and in Section 4.6 we briefly outline a concrete alternative to the "agnostic" way of handling incomplete training data which we advocate here.

As mentioned above, the key objective in deriving the fragment-aware fMST from the original MST parser is to ascertain zero loss for the intended complete tree in the face of an incomplete training example. While it is impossible to single out this one tree (since we do not know which one it is), we can steer the margin in the right direction with a loss function that assigns zero loss to all trees that are *consistent* with the training example, that is, trees that differ from the training example at most on those words that are fragment roots. To formalize the notion of consistency at the level of dependency edges, we define the *consistent attachments* C_G^j for a word w_j given a graph $G = (V, A)$ as an equivalence class over dependency edges as follows:

$$C_G^j = \{(w_i, r, w_j) : w_i \in V \wedge ((w_i, r, w_j) \in A \vee \exists w_k \in V : (w_k, \text{FRAG}, w_j) \in A)\} \tag{4.8}$$

If w_j is *not* a fragment root, then C_G^j will only contain one edge, namely the attachment of w_j in G. On the other hand, if w_j is a fragment root (that is, is attached to some w_k[6] by a dependency of type FRAG), then C_G^j contains all possible edges that attach w_j to any word in V. We can then define the set $\mathcal{C}_G$ of dependency graphs consistent with G as

$$\mathcal{C}_G = \{G' = (V, A') : (w_i, r, w_j) \in C_G^j \text{ for all } (w_i, r, w_j) \in A'\}, \tag{4.9}$$

and a loss function with the desired property of assigning zero loss to graphs consistent with the training example G_d:

$$L(G, G_d) = |\{w_j : (w_i, r, w_j) \in A \setminus C_{G_d}^j\}| \tag{4.10}$$

It follows from the definition of $\mathcal{C}_G$ (4.9) that $L(G, G_d) = 0$ for all $G \in \mathcal{C}_{G_d}$.

Finally, we adjust the score of all graphs $G = (V, A) \in \mathcal{G}_{G_d}$ for a given training example $(S_d, G_d = (V_S, A_d))$ in order to make the margin constraints

the model parameters helps prevent overfitting.

[6] In our setup, this w_k is always the root node w_0.

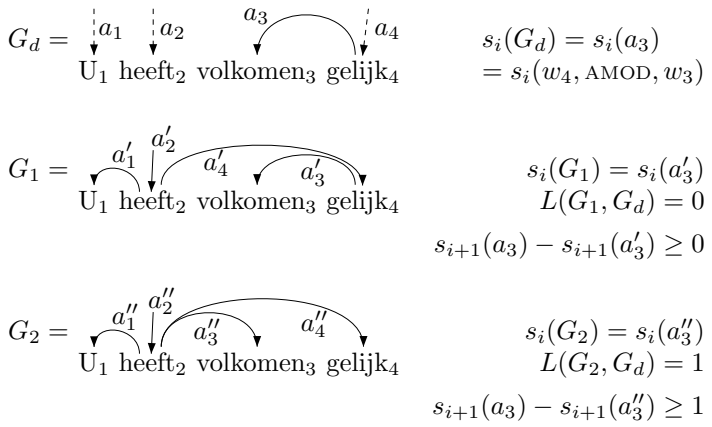


Figure 4.14: Scoring, loss, and margin constraints in fMST.

compatible with the fMST notion of loss:

$$s(G) = \sum_{\substack{(w_i,r,w_j)\in A \\ \text{s.t. } \forall w_k \in V_S : (w_k, \text{FRAG}, w_j) \notin A_d}} s(w_i, r, w_j) \tag{4.11}$$

This scoring function sums only over those edges in G for which G_d has a definitive attachment. This modification of the scoring function is important because the margin constraints would otherwise enforce unmotivated updates of the weight vector $\mathbf{w}$. To see this, consider the example in Figure 4.14. The "correct" graph G_d contains a single non-FRAG edge $a_3 =$ (gelijk, AMOD, volkomen); all other edges are FRAG edges that link fragment roots to the artificial root token. Let us assume that G_1, which is in fact the desired tree, is the highest weighted tree under the current weights $\mathbf{w}^{(i)}$, which implies that $s_i(G_1) \geq s_i(G_d)$. If s_i were to sum over *all* edges, this would further imply that $s_i(a'_1) + s_i(a'_2) + s_i(a'_4) \geq s_i(a_1) + s_i(a_2) + s_i(a_4)$, because $s_i(a'_3) = s_i(a_3)$. Now, according to the fMST definition of L, the loss of G_1 is zero, and the margin constraint with complete scoring would become

$$s_{i+1}(a_1) + s_{i+1}(a_2) + s_{i+1}(a_4) - (s_{i+1}(a'_1) + s_{i+1}(a'_2) + s_{i+1}(a'_4)) \geq 0, \tag{4.12}$$

thus forcing the algorithm to transfer weight from the correct G_1 edges to the spurious FRAG edges. However, if we score only the non-FRAG edges a_3 and a'_3, the spurious updates are avoided because the constraint is already satisfied.

Figure 4.14 shows a second possible scenario, where the highest weighted graph G_2 has non-zero loss because it contains an edge a''_3 which is not consistent with G_d. Under complete scoring, the corresponding constraint would require that

$$s_{i+1}(a_1) + s_{i+1}(a_2) + s_{i+1}(a_3) + s_{i+1}(a_4) - (s_{i+1}(a''_1) + s_{i+1}(a''_2) + s_{i+1}(a''_3) + s_{i+1}(a''_4)) \geq 1, \tag{4.13}$$

but the updates would by no means be localized to the relevant edge a_3''. In fact, the constraint resolution implemented in the MST parser distributes the updates uniformly over all features in the distance vector $\mathbf{f}(G_d) - \mathbf{f}(G_2)$. On the other hand, if we limit the score (and the updates) to the non-FRAG edges, the resulting margin constraint gives rise to much more succinct updates:

$$s_{i+1}(a_3) - s_{i+1}(a_3'') \geq 1 \tag{4.14}$$

4.6 Summary and Discussion

This chapter revolved around two dependency parsers, Malt and MST, and how we adapted them in order to accommodate fragmented training data but at the same time output complete trees.

The Malt parser is a transition-based parser, which means that it models a parse tree as a sequence of parser actions. The data-driven component of the parser learns the locally optimal choice for the next parser action, given the current state of the parser, which is in turn characterized in terms of three data structures: the list of remaining input tokens, the stack of partially processed words, and the set of dependency relations established so far. Our fragment-aware variant of the Malt parser (fMalt) differs from the original system in two respects. First, training instances (pairs of a parser state and the suitable transition action) are only passed on to the machine learning algorithm if they concern actual attachments supported in the training example; training instances that describe the spurious attachment of a fragment root to the artificial root node are discarded. This filter enables the parser to learn from whatever informative structure is encoded in the data, without ever knowing that the training example as a whole may not have been a connected tree. The second modification which distinguishes fMalt from Malt simply ensures that (misleading) information about fragment attachment does not leak into the feature representation of the parser states. At the same time, however, lexical and categorial information about unattached words *is* available as context information.

The second parser we discussed is the MST parser. It is a graph-based parser which formulates the parsing problem as a search for the maximum spanning tree, where the score of a tree is defined in terms of the scores of its edges, and each edge is represented by features over the head and the dependent of edge, as well as intervening and surrounding words. During training, the corresponding feature weights are updated so as to maximize the scoring margin between the correct parse tree on the one hand, and the candidate that currently scores best on the other hand. It is exactly this assumption which is relaxed in our modified version of the MST parser (fMST): When the training data are fragmented, the structures encountered during training do *not* necessarily coincide with the desired trees. In fact, the desired (complete) parse tree may be among the other possible trees – which the original MST parser would assume to be suboptimal. fMST solves this discrepancy by introducing a modified loss function which assigns zero loss to all trees *consistent* with the training example. In order to make the margin constraints compatible with this consistency-based notion of loss, we further adjusted the scoring function to disregard the edges which attach words that are fragment roots in the training example.

Both fMalt and fMST *hide* the partial status of the training data from their respective data-driven components. fMalt achieves this by ignoring certain tran-

sitions, whereas fMST ignores certain edges. These modifications allow us to essentially treat parsing with incomplete annotations as a supervised problem. This is in contrast to traditional, EM-based approaches, which infer an *explicit* model of the uncertainty introduced by partial annotations.

We now proceed to the empirical part of this thesis, beginning with a discussion of issues related to the evaluation methodology of projected parsers in Chapter 5. In Chapter 6, then, we will present empirical results that allow us to assess the quality of the parse trees predicted by our parsers. These results will further be subjected to a detailed error analysis in Chapter 7.

Chapter 5

Evaluation Methodology

Figure 5.1 shows an overview of the framework we are developing in this thesis. So far, we have covered the upper part of the figure: we have described the projection step in Chapter 3, and training and parsing with (f)Malt and (f)MST in Chapter 4. The intervening step labeled 'sample' indicates a simple sampling procedure that randomly selects sentences from the entire pool of projected trees. This will become relevant in Chapter 6, where we conduct an empirical evaluation of projected parsers: the sampling step ensures that the parsers are all trained on equal amounts of training data and thereby enables us to control the data loss factor incurred by the noise filters (Chapter 3). Before presenting any empirical results, however, a few words are in order about the evaluation methodology, depicted in the lower part of the figure. The evaluation methodology is the focus of this chapter, and we will occasionally return to Figure 5.1 when we explain the individual steps in the following sections.

Two problems arise in the evaluation of parsers trained on projected data (henceforth, *projected parsers*). First, the annotations projected from the source language usually differ stylistically from those found in the target language test data, rendering any immediate comparison between the predictions of the

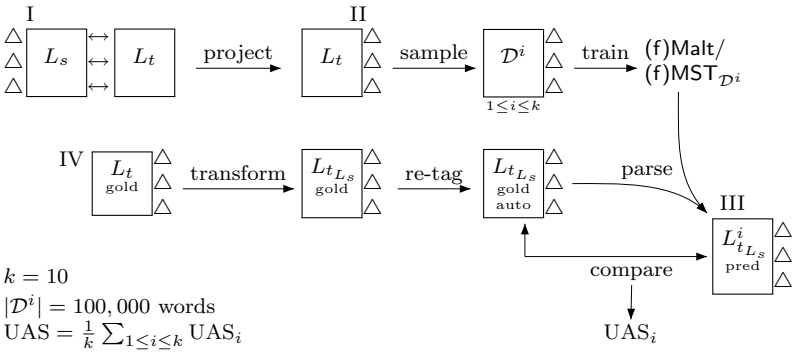


Figure 5.1: Evaluation of projected dependency parsers. The roman numbers refer to potential conversion sites and will be explained in Section 5.3.2.

projected parser and the gold standard meaningless. We discuss the use of tree transformations to consolidate discrepancies between the annotation schemes. We also present experiments (with treebank parsers, not projected parsers) that investigate the learnability of different annotation schemes, that is, how suitable they are from a machine learning perspective.

The second problem we address is the assessment of variance in the training data, and hence in parser quality. The standard procedure for this purpose would be *k-fold cross-validation*. However, the popular data sets used for benchmarking parsers are typically based on monolingual text. This means that cross-validation is unavailable for projection-based frameworks, because no projection can be performed for the training splits in the absence of a translation in the source language. We therefore propose an alternative validation scheme.

We begin by reviewing the evaluation of dependency parsers in the standard supervised setting, where the parser is trained and tested on manually annotated treebank data (Section 5.1). Of course, as far as testing is concerned, we also need treebank data in the projection setting; Section 5.2 describes the target language treebanks we use for this purpose, and also the source language treebanks that the source parsers are trained on. We then proceed to compare the annotation schemes employed in the various treebanks, and describe how we consolidate the differences by means of treebank conversions (Section 5.3). Finally, Section 5.4 addresses the issue of variance assessment and significance testing in projection-based frameworks.

5.1 Evaluation of Treebank Parsers

In order to evaluate dependency parsers, they are compared against a test set, usually an excerpt from a treebank. Since treebanks are manually annotated by trained linguists, they can be regarded as a *gold standard*. Applying the parser to the sentences in the test set yields the predictions which can subsequently be compared to the gold standard trees. The comparison allows us to estimate the correctness of the parser, relative to the gold standard. The *evaluation metric* defines how correctness is measured. The most common evaluation metric for dependency parsers is the *attachment score*: the *labeled attachment score* (LAS) is the percentage of words for which the parser predicts the correct head and dependency label, whereas the *unlabeled attachment score* (UAS) measures the percentage of words that have the correct head, irrespective of the label. These metrics have been implemented in the `eval.pl` script in the context of the CoNLL-X Shared Task.[1]

Most treebanks are annotated with constituent structure rather than dependency structure, and need to be converted to a dependency format if they are to be used for dependency parsing. To perform this conversion, it is necessary to recursively identify the head of each constituent and attach all siblings of the head as dependents. This procedure is rather straightforward if the original treebanks is annotated with grammatical functions which explicitly distinguish the head; if no grammatical functions are available, the conversion has to resort to pattern matching on the basis of constituent labels (Magerman, 1994; Johansson and Nugues, 2007).

[1] `http://nextens.uvt.nl/depparse-wiki/SoftwarePage#eval07.pl`

Once the treebank is thus prepared, it is traditionally split into a training set, a development set, and a test set. The parser is trained on the training set, and parameters (if any) are tuned based on preliminary results on the development set. The resulting parser is then applied to the held-out test set, and the predicted trees are compared to the reference trees in the treebank in terms of LAS and UAS. In practice, however, *cross-validation* over the training and test set is preferred over one designated test set because it allows a more reliable estimate of performance which takes variance into account. To perform k-fold cross-validation, the data is partitioned into k splits of equal size, and one of the splits is used as test data, while the remaining $k-1$ splits serve as training data. The train–test cycle is repeated until each of the k subsamples has been used as test data exactly once. The k pairs of gold standard and predicted annotations give rise to k LAS and UAS values, so that the average in conjunction with the standard deviation summarizes performance and variance.

Furthermore, performance differences across systems are usually subjected to statistical significance tests. In parsing, the standard software for this task is Dan Bikel's implementation of a "Randomized Parsing Evaluation Comparator,"[2] which repeatedly (e.g., 10,000 times) swaps the predictions of the two systems for randomly selected sentences, reassesses the difference, and finally determines a p-value based on the distribution of the 10,000 artificial difference values, as compared to the actual difference.

In a projection-based approach, evaluation basically follows the same principles, but it is complicated by two factors, as laid out above. First, the annotation scheme that is projected from the source language and learned by the projected parsers is likely to differ from the annotation scheme employed in the target language test set. Sections 5.3 deals with this issue. Second, cross-validation is not applicable unless a parallel gold standard is available. An alternative is presented in Section 5.4. For reasons that will become clear in Section 5.3, we ignore labels throughout our experiments, and therefore report only UAS. We also note that we treat punctuation marks as non-scoring tokens, following common practice.

5.2 Treebanks

In Chapter 6 we will present experiments with two different source languages (English and German) and three target languages (Dutch, German, and Italian). We are thus dealing with four languages, and four treebanks.

The treebanks are used for different purposes. On the one hand, the Penn Treebank (Marcus et al., 1993) for English and the Tiger Treebank (Brants et al., 2002) for German figure indirectly, as training data for the source language parsers. On the other hand, the Alpino Treebank (van der Beek et al., 2002) for Dutch, the Turin University Treebank[3] (TUT) for Italian, and again the German Tiger Treebank serve as (i) test data for the projected target language parsers, and (ii) training data for our upper bounds. The Dutch and German data sets are those provided for the CoNLL-X Shared Task (Buchholz and Marsi, 2006). Italian was not included in the 2006 Shared Task; the TUT data was released in the context of the EVALITA workshop (Magnini et al.,

[2] `http://www.cis.upenn.edu/~dbikel/software.html#comparator`
[3] `http://www.di.unito.it/~tutreeb`

	PTB (en)	Tiger (de)	Alpino (nl)	TUT (it)
train				
words	950,348	699,610	195,069	23,336
sents	39,832	39,216	13,349	794
words/sent	23.9	17.8	14.6	29.4
% non-scoring	11.5	11.5	11.3	11.0
test				
words	56,702	5,694	5,585	6,712
sents	2,416	357	386	306
words/sent	23.5	15.95	14.47	21.93
% non-scoring	12.1	12.0		11.0
% unseen	2.0	5.7	11.3	15.7
genre	news	news	mixed	news

Table 5.1: Characteristics of the treebank data sets.

2008). Table 5.1 summarizes the data sets. As we can see, all test sets that will be directly involved in the evaluation of the projected parsers are of roughly the same size (approximately 5,000 words), while the number of training examples varies greatly, from 23,000 words for Italian to 700,000 words for German. The WSJ portion of the Penn Treebank constitutes of course a much larger data set with 950,000 words of training data and almost 57,000 words of test data. We will describe the treebanks and in particular the (dependency-converted) annotations in greater detail in the next section.

Beside the syntactic annotations, all four treebanks also include manually annotated POS tags. Part-of-speech labels constitute a crucial source of information for data-driven parsers, and as we have seen in Chapter 4 both the (f)Malt and the (f)MST feature models employ this feature. However, Øvrelid et al. (2009) argue that any results based on such gold standard tags (as opposed to automatically assigned tags) overestimate the performance that one can realistically expect if the parser is applied to free text. This is because the free text will be tagged automatically rather than manually, and even the best state-of-the-art taggers are not one hundred percent accurate. Tagging errors in the input will in turn affect parsing performance. In our setup, where the training data is created in an entirely automatic fashion, there are in fact no gold standard tags available in the first place. We therefore replace the manual tags in the test sets with the tags assigned automatically by the TreeTagger (Schmid, 1994).[4] Assuming that the errors of the tagger occur systematically, this should ensure equal conditions in training and testing.

[4]Again, we use the pre-trained models available from `http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/`.

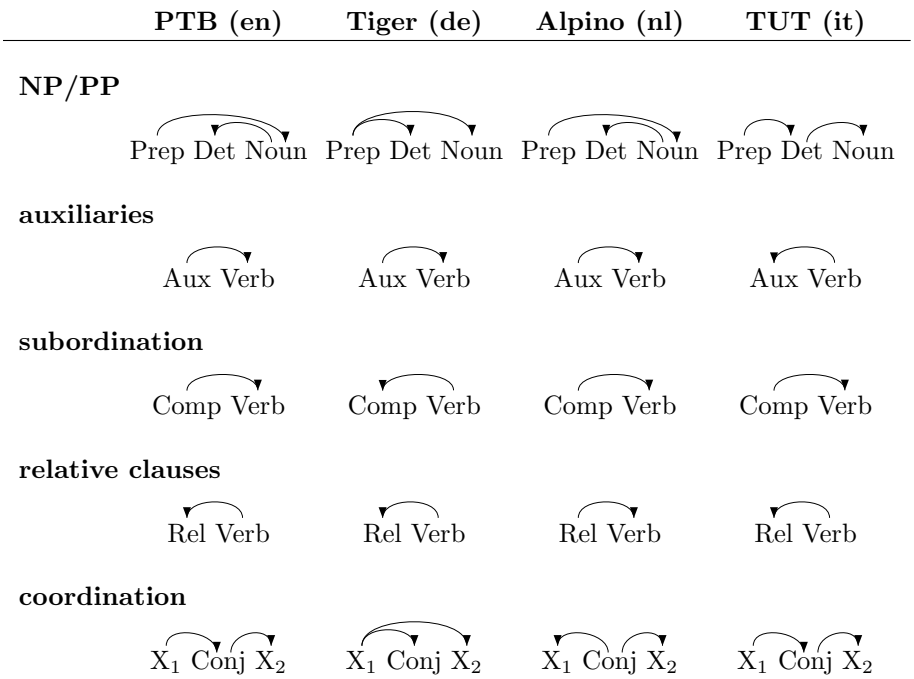


Figure 5.2: Different annotation schemes in the dependency-converted treebanks.

5.3 Annotation Schemes

Except for the Italian TUT annotation scheme, none of the treebanks is annotated with pure dependency structure. Specifically, the Penn Treebank contains only constituent structure, whereas the Dutch and the German treebank adopt hybrid strategies: in Alpino, dependency trees are embedded explicitly in Head-driven Phrase Structure Grammar (HPSG; Pollard and Sag, 1994) analyses, and the Tiger Treebank combines relatively flat phrase-structures with grammatical function labels. As mentioned earlier, these treebanks have been converted to the CoNLL dependency format in the context of the Shared Tasks. In the following discussion of the annotation schemes, we refer to these converted annotations, which are strictly word-based, without traces or other relics of a constituent-based analysis.

5.3.1 Comparison

The main phenomena in which the four annotation schemes differ are depicted in Table 5.2.

As shown in the table, both the English and the Dutch treebank annotate prepositional phrases hierarchically, with an embedded NP. The flat annotation scheme of the German treebank, on the other hand, makes every word in the PP a dependent of the preposition (with some exceptions). The Italian annotation

scheme assumes a hierarchical structure like English and Dutch, but declares the determiner rather than the noun as the head of nominal phrases. Another idiosyncrasy of the Italian annotation scheme is the treatment of fused prepositions such as *della*, which incorporate the determiner of the embedded NP. In the dependency-converted TUT, such fused prepositions are represented as two separate tokens, one tagged as a preposition, the other as a determiner.

Next, auxiliaries take the lexical verb as their dependent in all treebanks except the Italian TUT, which inverts the dependency, resulting in a flat structure where the lexical verb is the head of the auxiliary. The structure of subordinate clauses is hierarchical according to the English, Dutch and Italian annotation schemes, but flat in Tiger, with the complementizer as a dependent of the embedded verb. Relative clauses, on the other hand, are assigned a flat structure in all but the Dutch scheme, where the relativizer is the head of the embedded verb. Finally, coordination is annotated in three different ways: while the English and Italian treebanks implement a strictly right-branching strategy, the German annotation scheme attaches both the conjunction and the second conjunct to the first conjunct. The Dutch treebank annotates coordinations as flat symmetrical structures, with all conjuncts depending on the conjunction.

The annotation of PPs in TUT and (to some extent) in Tiger might conceivably be motivated on language-specific grounds, in that the flat structure adopted in the German treebank as well as the DP analysis of the Italian TUT facilitate a reasonably uniform treatment of fused prepositions, which occur in Italian and German, but not in English or Dutch. The precautionary approch is, however, not the only possibility given that prepositions and determiners form closed classes; that is to say, fused prepositions can in principle be treated like regular prepositions and their presence does not preclude the use of an annotation scheme like, say, the English one. The deviant treatment of auxiliaries as dependents in the Italian treebank may also give a more adequate account of the language. This is because Italian is a relatively free word order language where the primary role of the auxiliary is one of tense and aspect marking, rather than fulfilling a syntactic function. This is in contrast to a fully configurational language like English, where the auxiliary is an important anchor for word order constraints. But again, there is no technical reason to prevent an analysis that postulates the auxiliary as the head rather than the dependent of the lexical verb. Interestingly, the remaining discrepancies between the annotation schemes do not seem to be based on language-specific arguments, but rather reflect stylistic choices, subscribing to one flavor of dependency grammar or another. This observation leads us to hypothesize that the annotation schemes are in fact more or less interchangable between the languages considered here, and consequently, that we can apply straightforward treebank conversions techniques to consolidate the discrepancy between the annotations that we project from the source language and the target annotations dictated by the test data.

5.3.2 Conversions

In order to evaluate our projected parsers, we need to overcome the differences between the source and target annotations. A straightforward way of doing so is by means of treebank conversion, which matches the input structures expected under one annotation scheme and transforms them into another annotation scheme. Naturally, this begs the question of where such transfor-

	I source trees	**II** projected trees	**III** predicted trees	**IV** test set
direction	$L_s \rightarrow L_t$	$L_s \rightarrow L_t$	$L_s \rightarrow L_t$	$L_t \rightarrow L_s$
training	L_t	L_t	L_s	L_s
noise	POS, (L_s parser)	POS, projection	POS, L_t parser	none
reliability	$(-)$	$-$	$-$	$+$
applicability	full	limited	full	full

Table 5.2: Comparison of potential conversion sites.

mations should take place: one could transform the projected annotations to conform to the reference annotations encountered in the test set; alternatively, one could manipulate the test set to reflect the annotation decisions adopted in the source annotations. A variant of the former approach has been implemented by Hwa et al. (2005). They apply post-projection transformations to the Chinese training data projected from English in order to infuse L_t-specific information which has no counterpart in the source language. Ganchev et al. (2009) and Wróblewska and Frank (2009) propose similar correction rules, albeit more general in nature, and fewer in number.

By contrast, we argue that in a real-world scenario it is conceivable that the source language annotation scheme would be adopted unaltered for the target language parser; devising a tailored annotation scheme for the target language requires linguistically trained personnel with extensive knowledge of the language at hand.

Examining the project–train–evaluate pipeline laid out so far, we find that there are in fact four places where consolidation of the source and target annotation schemes could take place. Namely, on the source language annotations (labeled I in Figure 5.1); on the projected target language annotations (labeled II); on the predicted annotations (III); or on the test data (IV). Table 5.2 assesses the alternative sites with respect to various criteria. The first aspect we consider is the direction of conversion, which also bears upon the annotation scheme that the projected parsers are ultimately trained on. Conversion sites I and II involve conversion from the source language annotation scheme to the target language annotation scheme: If the source annotations are converted prior to projection, as in scenario I, then the projected training data will immediately conform to the target language annotation scheme; alternatively, the conversion from the source to the target language scheme can take place on the projected structures (II). In either case, the projected parsers would be trained on annotations that correspond to those in the target language test set. By contrast, if we consolidate annotation differences on the predicted trees (III), the training data is annotated according to the source language scheme. In the fourth scenario (IV), the gold standard trees in the test set are converted from the original target language annotation scheme to the annotation scheme

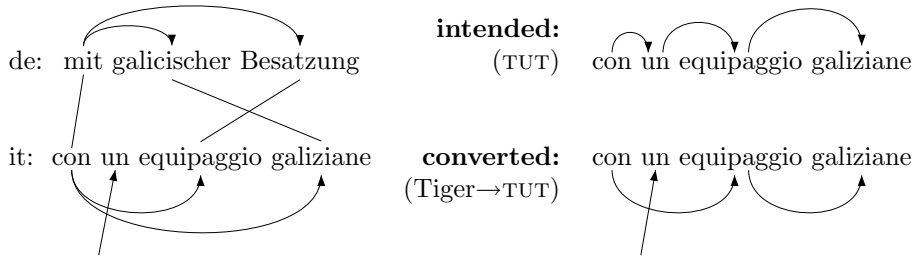


Figure 5.3: Conversion in scenario II fails to produce the desired DP structure of the TUT annotation scheme due to fragmentation in the projected L_t dependencies for the phrase *con un equipaggio galiziane* ('with a Galician crew').

employed by the source parser.

Next, we look at potential error sources that could perturb the conversion. We discussed above that the target language training data for projected parsers – and thus also their test data – is necessarily POS-tagged automatically. Any part of the conversion procedure that refers to part-of-speech at sites II or III is therefore likely to be affected by tagging errors. The same is true for the source language data that is the source for projection (I). However, recall that the treebanks do contain the manually assigned tags; in scenario IV it is therefore possible to base the conversion on the highly reliable manual POS tags, and only then insert the automatic tags. These two steps are labeled 'transform' and 'retag' in Figure 5.1. There are additional error sources: In scenario I, the source language annotations may be incorrect due to errors of the source parser.[5] The noise introduced by projection (relevant in scenario II) was discussed in Chapter 3, and clearly the projected target language parsers introduce some additional noise, too (scenario III). Each of these error sources must be expected to trigger inadequate conversion decisions and thus introduce additional noise. Solely the conversion of the test set (IV) – prior to automatic re-tagging – can be deemed reliable.

Finally, we note that the applicability as well as the quality of the individual transformations in conversion scenario II may be limited by fragmentation in the projected dependencies. The example in Figure 5.3 illustrates the problem for the NP/PP-to-DP transformation that is part of the conversion from Tiger to TUT annotations. On the left, we see the German source parse ('de') and the projected dependencies for the Italian translation ('it'). Since the Italian indefinite article *un* has no counterpart in the German translation, it cannot be attached during projection. The conversion, which in scenario II is performed on the basis of the projected dependencies, tries to match the Tiger-style annotations and transform them into the format of the Italian TUT test data. Unfortunately, given that the determiner is not part of the PP-structure in the

[5]Although we assume high-quality source parsers, they cannot be expected to be one hundred percent accurate. Especially in the out-of-domain setting in which they are applied here, they constitute a non-negligible error source (cf. Section 3.3, Table 3.5).

projected dependencies, it cannot be included in the converted PP, either,[6] so that the resulting PP-internal dependencies (shown at the bottom right) do not conform to the TUT annotation scheme, which calls for the structure depicted at the top right of Figure 5.3. At first glance, one might think that these kinds of errors are tied directly to the fragmentation and are unrelated to the conversion process. But upon closer inspection we find that missing edges in the projected dependencies are really just that: *missing* information. It is only when the conversion mechanism tries to interpret these structures as *complete* dependency graphs that the absent attachments trigger inappropriate transformations; in Figure 5.3, the conversion constructs the TUT analysis for PPs that embed bare NPs – despite the presence of a determiner.

In conclusion, conversion of the test data presents itself as the most promising option because it is based on gold standard annotations. By contrast, scenarios I–III are prone to introducing additional noise or amplifying the effect of other error sources. Moreover, the separation of evaluation-related processes from the training and parsing phase of the parsers is certainly appropriate in an application-oriented context, where gold standard annotations or even a dedicated annotation scheme may not be available at all. In this case, one would conceivably resort to a task-based evaluation, settling for whatever annotation scheme has been projected from the source language, rather than devising a new annotation scheme for the target language and annotating test data from scratch. In the light of these arguments, we couch the conversion in scenario IV as shown in Figure 5.1, and derive transformed versions of the Dutch, German and Italian test sets for each source language: one version according to the Penn Treebank annotation scheme to evaluate the parsers projected from English, and another version according to the Tiger-style annotations to evaluate parsers projected from German. The relevant transformations are summarized in Figure 5.4.[7] The conversions under the heading 'Alpino⇒Tiger' are applied to the Dutch test set when we evaluate the Dutch parsers projected from German, while for the evaluation of parsers projected from English, we apply the 'Alpino⇒PTB' transformations. Likewise, the Italian test set is converted with the 'TUT⇒Tiger' transformations in order to evaluate the Italian parsers projected from German, and with the 'TUT⇒PTB' transformations to evaluate those projected from English. The evaluation of the German projected parsers is performed on the German Tiger test set after it has been converted using the transformations labeled 'Tiger⇒PTB.'

5.3.3 Learnability experiments

If the annotation scheme is carried over from the source language as we suggest above, we may ask: Is one annotation scheme more appropriate than the other? When more than one source language (annotation scheme) is available, will one produce more "learnable" target language annotations than the other?

[6] This is unless we want the transformation to *build* structure, which is an arguable enterprise in real-world applications where a well-defined target language annotation scheme may not exist.

[7] The conflation of the two tokens representing fused prepositions under the TUT scheme is not strictly speaking required in order to consolidate the annotation schemes. It is, however, necessitated by our tokenization of the Italian Europarl data, which does not distinguish fused prepositions from base form prepositions.

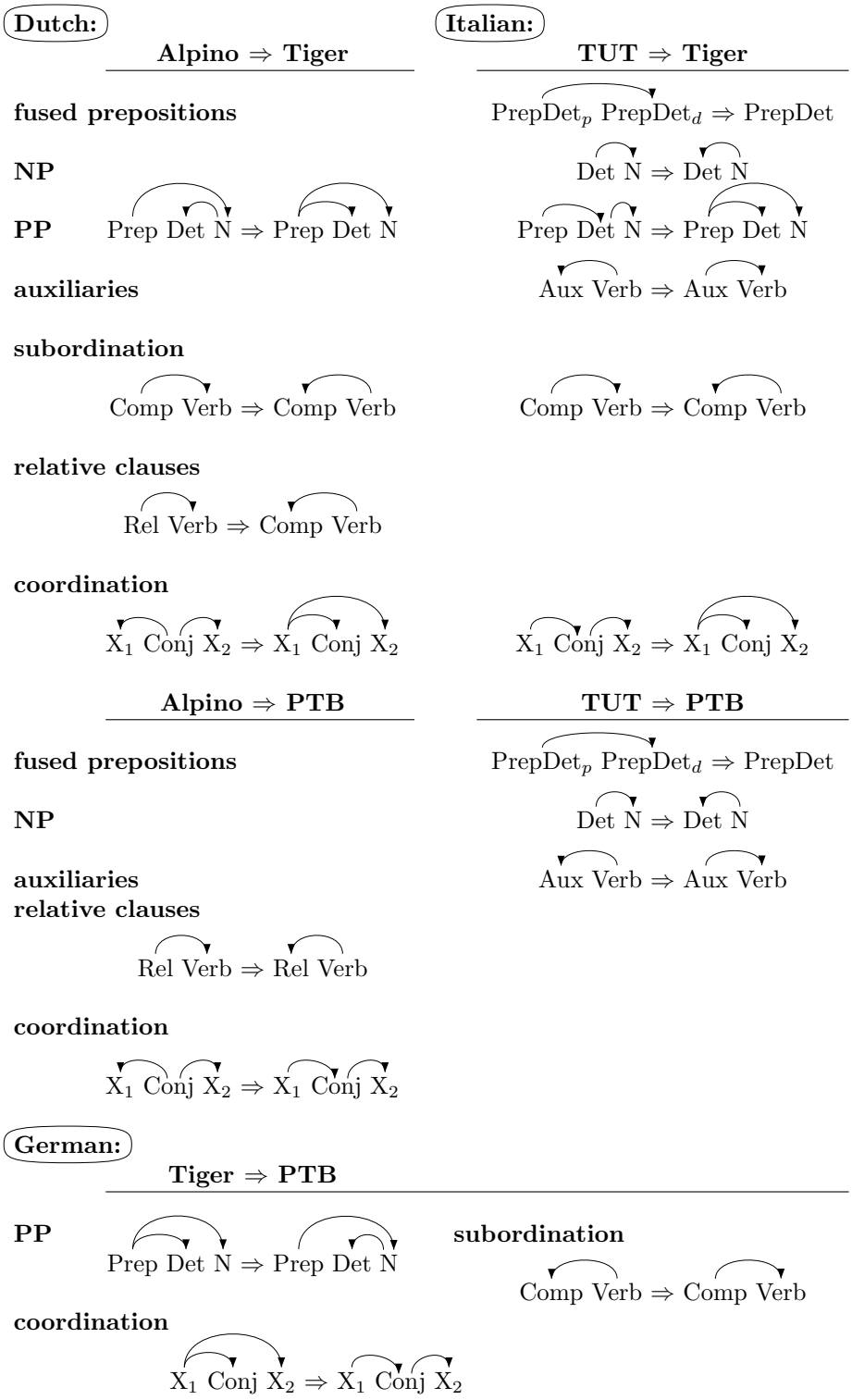


Figure 5.4: Test set conversions from target language annotation schemes to source language annotation schemes.

a.

lang	orig	PTB	Tiger
nl	79.23	**80.79**	79.19
		PTB > *orig* ≈ *Tiger*	
it	**88.52**	86.88	84.02
		orig > *PTB* > *Tiger*	
de	86.92	87.12	cf.'orig'
		PTB ≈ *orig*	

b.

lang	orig	PTB	Tiger
nl	81.41	**83.01**	**83.87**
		Tiger ≈ *PTB* > *orig*	
it	**90.23**	89.02	84.11
		orig > *PTB* > *Tiger*	
de	**89.86**	87.76	cf.'orig'
		orig > *PTB*	

Table 5.3: UAS of Malt parsers (a) and MST parsers (b) trained on gold standard dependencies in different annotation schemes.

This section concludes the discussion of annotation schemes by exploring these questions experimentally.

The training data for the parsers in this section is taken from treebanks; in particular, we are using the training sets described in Table 5.1. Although supervised (i.e., treebank) parsers are not at the center of discussion in this dissertation, they are more suitable for approaching the above questions than the projected parsers, which will be analyzed extensively in Chapter 6. First, the impact of the annotation scheme on parser performance can only be assessed in a meaningful way if the results are not distorted by noise stemming from the source parser, the word alignment, or projection issues. Second, the treebank parsers presented in the following experiments will give us an approximate idea of where the upper bound lies for dependency parsing with our target languages and the two parsers we use. We did not attempt parameter optimization, so the figures reported here represent *weak* upper bounds. For a survey of the state of the art in dependency parsing, we refer the interested reader to McDonald and Nivre (2011).

The results, in terms of unlabeled attachment scores (UAS), are shown in Table 5.3. The first column of each table ('orig') lists the results for training and testing on the original annotation scheme, that is, Alpino for Dutch, TUT for Italian, and Tiger for German. In the 'PTB' column, we see the results for parsers trained and tested on annotations converted to the Penn Treebank scheme. Similarly, the column labeled 'Tiger' contains the results with annotations converted to the Tiger annotation scheme.

Inspection of the results for Dutch reveals a rather surprising outcome: the Malt parser (Table 5.3a) trained on the Alpino-style annotations (79.23%) performs on a par with the parser based on Tiger-style annotations (79.19%), and is in fact significantly[8] ($p<0.01$) outperformed by the Malt parser trained with PTB-style annotations (80.79%). The ranking of annotation schemes is similar for the MST parser (Table 5.3b), where the parser based on PTB-style annotations again outperforms the Alpino-based parser. These results are surprising because the Alpino scheme was devised specifically for Dutch, whereas the PTB

[8]According to Dan Bikel's Randomized Parsing Evaluation Comparator, cf. Section 5.1.

annotation scheme was developed with English in mind.

Note also that the Dutch Malt parser responds better to the hierarchical PTB-based annotation scheme than to the flat Tiger scheme ($p<0.01$). We observe no significant difference between the corresponding MST parsers. We will have more to say on this topic below.

For Italian, training on the PTB-transformed treebank is again significantly ($p<0.01$) more effective than training on the Tiger-transformed treebank. The original TUT scheme is even more effective ($p<0.01$), which comes as no surprise given that the TUT guidelines were tailored to the traits of the Italian language.

Finally, the Malt parsers for German exhibit no significant difference with respect to the annotation scheme used in training, whereas the MST counterparts show that the Tiger-style annotations are preferable over the PTB-style annotations in a graph-based parsing scenario.

The results in Table 5.3 affirm that the performance of a parser hinges on the annotation scheme that it is trained on. However, the learnability of a given scheme depends not only on the annotation decisions, but also on the parsing algorithm implemented by the parser. McDonald and Nivre (2007) present a systematic comparison of the errors of the graph-based MST parser versus the transition-based Malt parsers. They conclude that the graph-based parser tends to be superior when it comes to long distance dependencies and dependencies that are close to the root of the tree in general, whereas the strength of the Malt parser lies in local attachments. For instance, it has been pointed out (Joakim Nivre, p.c. 2008) that flat coordination structures like those in the Alpino Treebank generally pose a challenge to incremental, deterministic parsers like the Malt parser. And in fact, when we compare the performance of the transition-based Malt parser on the one hand (Table 5.3a) with the results of the graph-based MST parser on the other hand (Table 5.3b), we see this trend confirmed for Dutch: the Alpino-based Malt parser (along with its German-based counterpart trained on the Tiger-style annotations, which are also comparatively flat) performs significantly worse than the PTB-based Malt parser, which employs far more hierarchical structures. Among the corresponding MST parsers, the ranking of English and German annotation schemes is reversed, indicating that the graph-based approach appears to be advantageous when dealing with relatively flat annotations like those in the Tiger corpus. The Alpino-annotated data, however, still appears to be less adequate for training dependency parsers, even in the graph-based paradigm.

We observe no such shift in annotation scheme adequacy for Italian, where the original TUT annotations consistently outperform the PTB-trained parsers, and the advantage over the Tiger annotations is even more pronounced (slightly more so among the MST results). Among the German parsers, we witness a shift similar to that in Dutch: the flat Tiger annotations are more effective in conjunction with the graph-based parser.

To shed some light on the unexpected ranking of the Alpino annotation scheme, we look at the impact of the individual transformations separately in Table 5.4. The upper part of the table shows how the transformations of the Alpino data towards PTB-style annotations affects learnability. We find that both the MaltParser and the MST parser benefit from the right-branching coordination markup of the PTB scheme. The attachment of relativizers in relative clauses seems to play only a minor role and makes no significant difference.

Turning to the Tiger-style transformations, first note that the semi-flat co-

trans	Malt	MST
none	79.23	81.41
$\text{coordination}_{\text{en}}$	**80.91**	**83.01**
$\text{relative}_{\text{en}}$	79.21	81.81
all_{en}	**80.79**	**83.01**
$\text{coordination}_{\text{de}}$	79.39	82.19
$\text{relative}_{\text{de}}$	79.21	81.81
$\text{subord}_{\text{de}}$	79.47	**82.67**
np/pp_{de}	**80.73**	**83.83**
all_{de}	79.19	**83.87**

Table 5.4: Impact of individual transformations on Dutch treebank parsers. Significant improvements ($p<0.01$) over original Alpino annotation ('none') are in bold face.

ordination adopted in the German treebank does not seem to be superior to the flat annotations in Alpino: no significant improvement is achieved for either parser by using the former ('$\text{coordination}_{\text{de}}$'). Surprisingly, both parsers benefit from the flat annotation of prepositional phrases ('np/pp_{de}'). The MST parser, but not the MaltParser, further takes advantage of the flat subordination structure annotated in Tiger. As mentioned earlier, this is in line with the fundamentally different parsing paradigms represented by Malt and MST.

We tentatively conclude that the MST parser is in fact better at exploiting the flat aspects of the Tiger annotations, while both parsers largely benefit from the highly hierarchical coordination structure of the PTB annotation scheme. A more detailed exploration of these issues is clearly in order, and subject to future research.

5.4 Variance Assessment

We now turn to the second factor that complicates the evaluation of projected parsers, namely the assessment of variance in the training data, and hence in parser quality. As mentioned earlier, the standard procedure for this purpose would be cross-validation, which partitions the data into k bins of equal size, and uses each of the bins once as test data while training on the remaining $k-1$ bins. The problem in a projection-based setting is the following: the training data has to be projected from a source language and must therefore be available as a multilingual parallel corpus. At the same time, cross-validation requires that all data be used as test data once. In combination, this means that only multilingual gold standards of appropriate size can be used for evaluation, and these are scarce. Moreover, the expected noise level in the projected dependencies requires that there be a considerable amount of training data for the results to be meaningful. So even if parallel test data is available, the data partitioning performed in cross-validation may compromise the results.

We therefore propose a validation scheme which (i) does not reduce the amount of test data by partitioning (this may be a problem when only a small

number of gold standard annotations is available), (ii) does not require parallel test data and is independent of the projection step, and (iii) takes advantage of the fact that projected training data is cheap and therefore abundant in projection-based settings.

In particular, given that we have plenty of training data, we can train a particular parser multiple (say, k) times, each time sampling a fixed number of training examples from the pool of training data. In Figure 5.1 on page 77 these samples are denoted $\mathcal{D}^i$, and each gives rise to a parsing model $\mathsf{(f)Malt}/\mathsf{(f)MST}_{\mathcal{D}^i}$. The k parsers can then each parse the unseen test set, and subsequent comparison against the gold standard annotations yields k values of the performance metric at hand (here, UAS). As in conventional cross-validation, these k values can then be averaged to provide an aggregated score, and they can be used to derive standard deviations etc. The arrays of measurements for different systems can further be subjected to significance tests such as the two-sample t-test to verify that observed performance differences are not merely random effects.

For the experiments in Chapter 6, we set $k = 10$, and we sample 100,000 words training data in each round (unless mentioned otherwise). We report the average of the 10 UAS values. Statistical significance of the difference between two systems A and B is determined by comparing the 10 results of A with the 10 results of B in a two-sample t-test. Unless stated explicitly, we imply a p-value smaller than 0.01 when we say that system A significantly outperforms system B.

5.5 Summary and Discussion

In this section, we have introduced the evaluation methodology that will be applied in the next chapter, where we present a systematic empirical evaluation of our projected parsers.

We have discussed two issues that arise in the evaluation of frameworks that involve cross-lingual projection of annotations, and in particular the projection of dependency trees. The first obstacle one needs to overcome in order to evaluate projected parse annotations and tools derived from them is the consolidation of different annotation schemes. More specifically, the parse trees projected onto the target language sentences conform to whichever annotation scheme was employed in the source language treebank. A given test set for the target language, on the other hand, is likely to be annotated according to different guidelines. In order to evaluate the projected annotations and parsers against the gold standard annotations, we therefore propose to convert the target language test sets to the annotation scheme employed in the respective source language.

Diverging annotation schemes are problematic not only for cross-lingual approaches, but they also blur the results of domain adaptation if the data sets for the source and target domains are annotated according to different guidelines (Dredze et al., 2007). In fact, the conversions we propose here could be replaced by parser adaptation techniques (e.g., Smith and Eisner, 2009; Jiang and Liu, 2009). However, this would require for the training data of the adaptation model to be annotated with both annotation schemes. The definition of transformations would therefore still be necessary.

Kübler et al. (2008) present an extensive comparison of two German treebanks: the Tiger treebank with its rather flat annotation scheme, and the

TüBa/DZ treebank with more hierarchical structures. They find that the flat Tiger annotation scheme is more easily learned by constituent-based (PCFG) parsers when evaluated at a dependency level. Our results suggest the opposite, but this may well be due to the differences in the experimental setup: our training data represent dependency trees directly, and we learn incremental, deterministic dependency parsers rather than PCFGs. In line with our results, Seeker et al. (2010) report results which clearly indicate that a restructuring of prepositional phrases can yield improvements in parsing accuracy: they add more structure to the entirely flat PP annotation of the Tiger corpus by explicitly representing the PP-internal nominal phrase.

We performed our evaluation of the learnability of various annotation schemes for Dutch, Italian and German. For English, we refer the reader to Buyko and Hahn (2010), who conduct an indirect (task-based) evaluation comparing the suitability of CoNLL dependencies versus Stanford dependencies (Marneffe et al., 2006) for event extraction. They employ six different dependency parsers and find that the CoNLL dependency representation turns out more helpful for the IE task with four of these parsers.

In this chapter, we have further proposed a validation scheme which unlike cross-validation does not require parallel test data. Instead, it exploits the fact that training data is usually available in abundance in projection scenarios, so parsers can be trained on multiple random samples and evaluated against a single, independent test set which need not be further partitioned.

Classical cross-validation and the validation method described here do measure slightly different things. First, in cross-validation it is not only the training data that is varied, but the test data as well. Second, when two systems are compared under the cross-validation regime, the k rounds can usually be considered *paired* samples because both systems are trained and evaluated on identical partitionings of the data. In contrast, projection-based settings typically involve some form of filtering on the basis of the projected annotations; in our case, the filter restricts the degree of fragmentation in the projected dependency tree. This filtering makes it all but impossible to pair the training samples without seriously diminishing the pool from which the samples are drawn. For instance, when comparing an Italian parser projected from English (it_{ptb}) and one projected from German (it_{tig}), then a training sentence may receive a complete analysis from the English translation, and hence be included in the training pool for it_{ptb}; but the same (Italian) sentence may receive a highly fragmented analysis under projection from German (e.g., due to missing alignment links) and be discarded from the training pool for it_{tig}.

With samples that cannot be paired, it is also not obvious how evaluation strategies like the randomized comparison mentioned above (fn. 2) could be employed in a sound way.

5.5.1 Labeling schemes

An issue we have not addressed thus far is the labeling of dependency edges. Just like the structural annotation schemes differ across treebanks, so do the labeling schemes. Hence, there is no one-to-one mapping between the labels employed in different treebanks. In fact, the labels are usually so intricately language specific and in complex interaction with attachment decisions that a sound mapping of labels is an enterprise by itself. Yarowsky and Ngai (2001);

Moon and Baldridge (2007) discuss tagset differences in POS tagging, and solve the issue by manually defined mappings to a (coarse-grained) consensus tagset.

We make no attempt at such a mapping for syntactic dependency types. We instead rely on the unlabeled attachment score (UAS) to reflect parser performance. The fact that the annotation scheme conversions may render the labeling inconsistent in the absence of an appropriate mapping is acceptable in our setup because the converted test sets are never used as training data.

Chapter 6

Experiments

This chapter provides empirical results showing that the dependency structures projected under partial correspondence projection are perferable – not only in terms of quantity, but also qualitatively – to the parse trees obtained with strict projection.

We describe a series of experiments performed in order to evaluate various projected parsers. In particular, we focus on the following questions: Can quantity really make up for compromised quality when it comes to training data? And if so, to what extent? How does the presence of fragmented analyses in the training data affect parsing performance? Can certain error types be traced back to this fragmentation? Do the transition-based and the graph-based parser react differently? Is parsing with fragments more appropriate for one language than for another?

The majority of these experiments has been described previously in Spreyer and Kuhn (2009), Øvrelid et al. (2009), and Spreyer et al. (2010). The concrete numbers given here, however, differ slightly from those reported in the publications due to parameter tuning, which has not been performed until recently. In addition, minor bug-fixes were introduced after the aforementioned articles were published, and the random sampling involved in the evaluation cycles witnesses some variation in the training data.

We describe our experimental setup in Section 6.1. Our core experiments and results are presented in Sections 6.4 and 6.5. A detailed error analysis will be presented in Chapter 7.

6.1 Experimental Setup

The principal aspects of our experimental methodology have already been addressed in Chapter 5 when we discussed the evaluation of projected parsers. We repeat Figure 5.1 here as Figure 6.1. To recapitulate, our test data consist of sentences from the Alpino Treebank for Dutch (5,600 words), the Tiger Treebank for German (5,700 words), and the TUT Treebank for Italian (6,700 words). In order to evaluate our projected parsers in a meaningful way, we convert each of the test sets to the annotation scheme employed by the respective source parsers (English or German). For instance, when we report results for a Dutch parser projected from English, these results are obtained by comparing

the predictions of the projected parser against the Alpino test set with gold standard annotations *converted to the Penn Treebank annotation style.*

We emphasize that the projected parsers are subjected to an *out-of-domain* evaluation: They are trained on the Europarl corpus, which consists of proceedings of parliamentary debates, whereas the test sets are from the newspaper (Dutch: mixed) domain. This means that the results presented in Sections 6.4 and 6.5 are likely to underestimate the true performance of our parsers.

Unless stated otherwise, all results refer to *unlabeled attachment scores* (UAS), that is, the percentage of words for which the parser predicts the correct head, but not necessarily the same dependency label. The reason why we disregard the labels is that the label set employed by the source parser – and hence the projected parsers – does not usually coincide with the dependency types assumed in the test data.

Recall from Chapter 5 that we train the parsers multiple times in order to account for variance in the training data. The concrete training sets ($\mathcal{D}^i$ in Figure 6.1) are sampled randomly from the pool of all projected parse trees. Naturally, when we want to train parsers on the entire corpus, the samples are all random permutations the same set of sentences. Our k-fold training (and evaluation) procedure may at first glance seem redundant in this case. However, both the Malt parser and the MST parser are sensitive to the order in which training examples are presented to the learner. The MIRA algorithm employed by the MST parser is order-dependent by virtue of being an online learning scheme. But even the libsvm package used for training the SVM classifiers in the Malt parser arrives at different feature weights when presented with different permutations of the training set. We therefore have two reasons to maintain k-fold training even if the training sample overlap is complete: (i) Doing so enables us to estimate the impact of order-sensitivity in terms of variation in accuracy across the permutations; and (ii) it allows us to maintain a consistent – and hence comparable – way of significance testing, namely by means of the t-test: While the nature of the variance captured by training on permutations certainly differs from that captured by training on different sentences, the k-arrays of UAS scores can be compared in a meaningful way *within* the respective sampling paradigms.

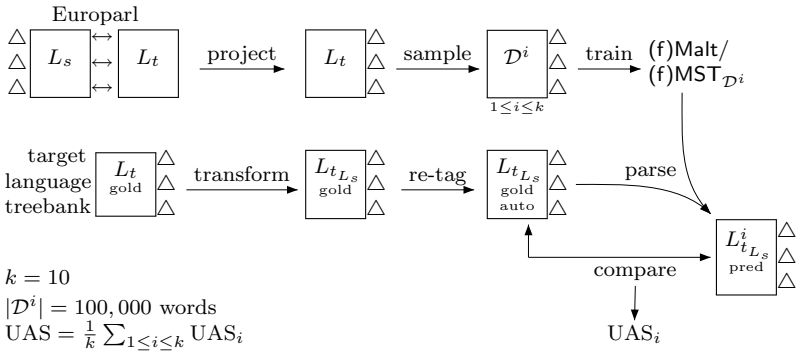


Figure 6.1: Evaluation of projected dependency parsers.

Parameter	Values considered
parsing algorithm	nivrestandard, nivreeager
root handling	normal, strict, relaxed
marking strategy	baseline, head, path, head+path
covered root	left, right
learner	libsvm
data split column	CPOSTAG of $\beta[0]$
data split threshold	5,000

Table 6.1: Malt parser parameters and their values.

What remains to be addressed is the set of training parameters, including both parser-specific parameters like the parsing algorithm, as well as the parameters restricting the degree of fragmentation in the training data.

6.2 Parameter Tuning

6.2.1 Parser-specific training parameters

Parser-specific parameters in our setup refer to parser options that modify the behavior of the parser. These are parameters defined for the original parsers Malt and MST.

Malt/fMalt. The relevant Malt parser parameters are summarized in Table 6.1. Parameters not mentioned in the table assume their default value.[1] Similarly, values that are not mentioned are not considered in parameter optimization. We consider two parsing algorithms, namely the *arc-standard* and *arc-eager* variants of the Nivre algorithm (Nivre, 2008). The arc-standard variant discussed in Chapter 4 does not attach right dependents until they have been assigned all their dependents. The arc-eager version of the transition system adds all arcs, including right arcs, as soon as possible, thus eliminating (spurious) uncertainty as to whether or not to attach right dependents (Kübler et al., 2009). The *root handling* parameter determines the treatment of ROOT dependents: normal root handling attaches dependents of ROOT with RIGHT-ARC transitions during parsing. Neither relaxed nor strict root handling allow these attachments during parsing, but instead attach ROOT dependents in a post-processing phase; strict root handling further prohibits the reduction of ROOT dependents from the stack.[2] Normal root handling turned out to outperform the two alternative strategies across the board. We therefore omit this parameter when we discuss the results of parameter optimization in Section 6.2.2.

[1] We are using version 1.4.1 of the Malt parser, and version 0.2 of the MST parser.

[2] It may at first seem that root handling counteracts fragment parsing, since it seemingly connects fragments to the rest of the graph. However, our implementation of fMalt identifies fragment roots solely on the basis of the designated FRAG label, and hence supersedes the re-attachment of fragments performed during projectivization.

Parameter	Values considered
decoder	projective (Eisner)
loss function	Hamming loss incl. punctuation
order	1, 2
training-k	1, 2, 5, 10
pre-/post-processing (Malt):	
marking strategy	baseline, head, path, head+path
covered root	left, right

Table 6.2: MST parser parameters and their values.

The parameters called *marking strategy* and *covered root* specify the details of pseudo-projectivization (cf. Section 4.1). The marking strategy determines to what extent the arc lifts are encoded in the labels; the covered root option prevents unnecessary lifts by re-attaching ROOT dependents to nearby nodes if they are covered by crossing edges.

Finally, we note that we use the libsvm package to train Malt's oracle approximation. The training instances are split into bins according to the value of the CPOSTAG of the word in $\beta[0]$ (*data split column* in Table 6.1), and separate SVMs are trained for bins that contain at least 5,000 instances (*data split threshold*).

MST/fMST. The training parameters specific to the MST parser are shown in Table 6.2. We consider only the projective parsing algorithm, that is, the Eisner algorithm described in Section 4.4.1. As with the Malt parser, we employ pseudo-projectivization to ensure that the training examples are in fact projective; the specifics of this procedure are again determined by the marking strategy and covered root options of the Malt parser, which is run in (de-)projectivization mode as pre- and post-processing steps.

The *loss function* implements the standard Hamming distance, which counts attachment discrepancies across all words of a sentence, including punctuation. The *order* parameter of MST determines the feature set used to encode dependency graphs. In the first-order feature model, edges are described without reference to adjacent edges. The second-order model also includes features over pairs of adjacent edges.

When we discussed the MST parser in Chapter 4, we presented a k-best formulation with $k = 1$. This means that the margin is enforced only between the correct example and the single highest scoring candidate parse. Thus, in the general case, k-best training imposes margin constraints between the correct graph and each of the k highest scoring graphs, respectively. This parameter k is set with MST's *training-k* option. We consider the values 1, 2, 5, and 10.

optimize-parameters$(\mathcal{D}, k, \mathcal{G})$
$\mathcal{D}$: training data
k: number of folds
$\mathcal{G}$: development data

1 split $\mathcal{D}$ into k folds $\mathcal{D}_i$ of equal size
2 **for** each fold $\mathcal{D}_i$
3 **for** each parameter combination p

4 train parser with options p on remaining folds $\bigcup_{j \neq i} \mathcal{D}_j$
5 $\mathcal{D}_{i,p}^{\text{pred}} \leftarrow \text{parse}(\mathcal{D}_i)$
6 $\text{UAS}_i^p \leftarrow \text{compare}(\mathcal{D}_{i,p}^{\text{pred}}, \mathcal{D}_i)$
7 $\mathcal{G}_{i,p}^{\text{pred}} \leftarrow \text{parse}(\mathcal{G})$
8 $\text{UAS}_{\mathcal{G},i}^p \leftarrow \text{compare}(\mathcal{G}_{i,p}^{\text{pred}}, \mathcal{G})$

9 **for** each parameter combination p
10 $\text{UAS}^p \leftarrow \frac{1}{k} \sum_i \text{UAS}_i^p$
11 $\text{UAS}_{\mathcal{G}}^p \leftarrow \frac{1}{k} \sum_i \text{UAS}_{\mathcal{G},i}^p$

12 **return** $(\text{argmax}_p \text{UAS}^p, \text{argmax}_p \text{UAS}_{\mathcal{G}}^p)$

Figure 6.2: Pseudo-code for parameter optimization.

6.2.2 Parameter optimization with manually annotated development data

In order to find the ideal combination of the parameters discussed above, we assess the performance of the resulting parsers on development data. For each target language, we randomly sampled a development set of 2,000 words from the treebanks that are also used for final testing, but distinct from the test sets. Like the test sets, the development sets are subsequently prepared to match the annotation scheme of the source language as described in Chapter 5.

In addition to the optimization on the basis of the manually annotated gold standard, we simultaneously investigate (in Section 6.2.3) the impact of the parameters when evaluated against projected data. We believe that a comparison of the parameter combinations selected by optimization with manually annotated data on the one hand and projected data on the other hand will be instructive in that it should shed some light on which factors are helpful in overcoming noise in the training data, as opposed to merely fitting the training data.

Parameter optimization with automatically labeled data is realized as 10-fold cross-validation on training samples of 100,000 words for the Malt parameters, and 50,000 words for MST parameters.[3] In this setup, parsers are trained

[3]We decided to reduce the amount of training data for the parameter tuning of MST due to time constraints. For the same reason, we adopted the marking strategy and covered root treatment of the corresponding (f)Malt systems, and we use the 'trees (bi.)' parameters for

	training samples	parameters: algorithm	marking	cov. root	UAS (dev)
en–nl	trees (bi.)	standard	head	left	68.97
	trees (fb.)	standard	head	left	68.65
	frags (fMalt)	standard	head	left	70.60
de–nl	trees (bi.)	standard	head	left	69.03
	trees (fb.)	standard	head	left	58.98
	frags (fMalt)	standard	head+path	left	67.25
en–it	trees (bi.)	standard	baseline	left	64.50
	trees (fb.)	standard	baseline	left	62.53
	frags (fMalt)	standard	baseline	left	64.60
de–it	trees (bi.)	standard	head+path	left	53.97
	trees (fb.)	standard	head+path	left	46.81
	frags (fMalt)	standard	head+path	left	53.75
en–de	trees (bi.)	eager	head	left	61.28
	trees (fb.)	standard	head	left	58.54
	frags (fMalt)	standard	head	left	59.89

Table 6.3: Optimal parameter settings for Malt and fMalt parsers, as determined on development sets of 2,000 words.

on 90,000 (45,000) words of projected data – trees or fragments – and their predictions for the remaining fold are compared against the dependency structures projected from the source language. Since the test folds for fMalt and fMST parsers contain fragmented analyses, we modify the comparison to consider only non-FRAG attachments, much like the loss function formulated for fMST. Pseudo-code for the parameter optimization procedure is given in Figure 6.2. Note that parameter optimization is run separately for each projection scenario: The parameters ideal for training with projected trees are determined independently of those for fragment training.

While we defer the comparison between the two optimization approaches to Section 6.2.3, we mention the cross-validation approach here because the resulting parsers (trained on 10 times 9 out of 10 folds of projected data) are the very same models we use to parse the development data to tackle the ideal parameter combinations.

Malt/fMalt. Table 6.3 lists the optimal parameter settings for Malt and fMalt parsers along with the corresponding attachment scores against the development sets. For German parsers trained on trees, the arc-eager version of Nivre's algorithm in combination with head marking for pseudo-projectivization and left-attachment of covered roots achieves the best data fit. Head marking and left-attachment also yield the best results for trees projected under fallback

the 'trees (fb)' parsers, too.

	training samples	parameters order	k	marking	cov. root	UAS (dev)
en–nl	trees (bi.)	2	10	head	left	70.45
	frags (fMST)	1	10	head	left	70.56
de–nl	trees (bi.)	2	10	head	left	70.58
	frags (fMST)	2	1	head+path	left	65.83
en–it	trees (bi.)	2	10	baseline	left	64.01
	frags (fMST)	2	10	baseline	left	65.09
de–it	trees (bi.)	2	10	head+path	left	52.92
	frags (fMST)	2	10	head+path	left	50.53
en–de	trees (bi.)	2	2	head	left	63.23
	frags (fMST)	2	10	head	left	62.82

Table 6.4: Optimal parameter settings for MST and fMST parsers, as determined on development sets of 2,000 words.

projection, and for fragmented training sets. However, in these settings the arc-standard formulation of the parsing algorithm outperforms the eager variant.

The arc-standard transition system also performs best with the Dutch projections, from either source language. Left-attachment is chosen for both language pairs to deal with covered roots, and arc-lifting is marked using the head strategy for all Dutch parsers except the fragment-trained parser projected from German, which takes advantage of the additional information encoded in head+path marking.

In line with the Dutch and the majority of the German results, the Italian parsers consistently prefer the arc-standard algorithms and left-attachment for covered roots. However, parameter optimization reveals that different marking strategies are in order for Italian: dependency structures projected from English do not require any encoding of arc-lifts performed during pseudo-projectivization (baseline strategy), while parsers trained on projections from German benefit from the elaborate head+path encoding.

MST/fMST. The parameter settings ideal for the graph-based parsers are summarized in Table 6.4. As mentioned above, the projectivization parameters have been adopted from the corresponding (f)Malt systems. The remaining parameters are *order* and *training-k*. For all language pairs and parsing systems except one (namely, en–nl fragment parsing), the second-order feature model is preferable. The preferred number k of suboptimal parse trees for which margin constraints are generated is 10 (the largest possible value in our search space) with two exceptions: de–nl fragment parsing and en–de tree parsing select $k = 1$ and $k = 2$, respectively.

	training samples	parameters			UAS (proj)
		algorithm	marking	cov. root	
en–nl	trees (bi.)	eager	path	right	80.52
	trees (fb.)	standard	head	left	67.65
	frags (fMalt)	eager	head	left	79.37
de–nl	trees (bi.)	eager	head	right	79.80
	trees (fb.)	standard	head	left	64.37
	frags (fMalt)	eager	head	left	74.63
en–it	trees (bi.)	eager	head	left	80.57
	trees (fb.)	standard	head	left	69.59
	frags (fMalt)	standard	baseline	left	78.83
de–it	trees (bi.)	standard	head+path	left	80.43
	trees (fb.)	standard	head / baseline	right / left	53.40
	frags (fMalt)	standard	head	left	73.77
en–de	trees (bi.)	standard	head	right	79.11
	trees (fb.)	standard	head	left	67.62
	frags (fMalt)	eager	head	left	79.78

Table 6.5: Optimal parameter settings for Malt and fMalt parsers, as determined through cross-validation against projected data.

6.2.3 Parameter optimization with projected development data

In this section we compare the ideal parameters determined in the previous section to the parameter combinations that a cross-validation on projected data arrives at. The comparison allows us to identify those parameters that enable the parsers to deal with noisy training data. This is because optimization on automatically labeled data singles out the parameters that result in the best data fit, while the optimization with gold standard dependencies reveals the parser settings that are best capable of learning the desired tree structures despite the erroneous or missing attachments in the training data.

Malt/fMalt. Table 6.5 shows the (f)Malt parameter combinations that lead to the best results when evaluated against automatically labeled data. Comparing these settings to the "proper" parameters in Table 6.3, we observe that the arc-standard transition system is chosen over the arc-eager variant in almost every instance when the optimization procedure is based on gold data. By contrast, the arc-eager algorithm performs better when evaluated against projected annotations, which leads us to conclude that the eager transition system is prone to fit the noise in the training data. However, we also see that even with automatically labeled test data, the optimization procedure consistently selects the arc-standard formulation for fallback trees (rows labeled 'trees (fb.)'). This is interesting because fallback projection constitutes the weakest noise fil-

	training samples	parameters order	k	marking	cov. root	UAS (proj)
en–nl	trees (bi.)	2	5	head	left	89.73
	frags (fMST)	1	5	head	left	81.58
de–nl	trees (bi.)	2	10	head	left	84.753
	frags (fMST)	1	5	head+path	left	76.60
en–it	trees (bi.)	2	10	baseline	left	88.73
	frags (fMST)	1	10	baseline	left	82.14
de–it	trees (bi.)	2	10	head+path	left	88.70
	frags (fMST)	1	10	head+path	left	78.18
en–de	trees (bi.)	2	10	head	left	89.88
	frags (fMST)	1	10	head	left	82.18

Table 6.6: Optimal parameter settings for MST and fMST parsers, as determined through cross-validation against projected data.

ter and hence produces the noisiest annotations. So it seems that optimization against projected data in a cross-validation scheme is capable of discovering favorable parameters (here, the arc-standard parsing algorithm), but it requires a prohibitive amount of noise (as in the fallback trees) to do so.

For German and Dutch, the cross-validated, unsupervised optimization procedure further selects right-attachment of covered roots for bidirectionally projected trees, across both source languages. Interestingly, parameter tuning on *fallback* projections from German to Italian does not exclude a covered root attachment to the right, either. Given the pronounced language-specific preference for left-attachment, as witnessed by the strong baseline performance for Italian, this emphasizes once more that these annotations are of questionable quality.

MST/fMST. The results of unsupervised parameter optimization for (f)MST in Table 6.6 reveal that a first-order model is sufficient to fit the fragmented data, across all language pairs. This is in contrast to the second-order parsers that were selected for almost all language pairs during optimization against gold standard annotations. This discrepancy can be explained by the fragmentary nature of the annotations that make up the training data for the fMST parsers: few edges actually have adjacent edges, thus second-order features are very sparse and have little discriminative power when compared against fragmented data. Evaluation against gold standard data, on the other hand, requires that the parsers learn that it is precisely the denser parts of the dependency structure which are highly informative for building complete trees.

constraint	description	values
$T \leq x$	less than x missing edges	$x \in \{2, 3, 4, 5\}$
$T \leq x \wedge W \geq y$	less than x missing edges, sentence length at least y words	$y \in \{4, 5, 6\}$
$W/T \geq x$	avg. fragment size at least x	$x \in \{2, 3, 4, 5\}$
$W/T \geq x \wedge W \geq y$	avg. fragment size at least x, sentence length at least y words	$y \in \{4, 5, 6\}$
MAX $\geq x$	largest fragment has at least x nodes	$x \in \{2, 3, 4, 5\}$
MAX $\geq x \wedge W \geq y$	largest fragment has at least x nodes, sentence length at least y words	$y \in \{4, 5, 6\}$

Table 6.7: Schemata for fragmentation constraints.

6.2.4 Fragment size

The training parameters for fMalt (Table 6.3) and fMST (Table 6.4) were determined on the basis of a fixed fragmentation constraint: the training data contained only sentences with at most two missing edges, and at least 4 words. We now explore various other constraints on fragmentation.

The constraint schemata we consider are shown in Table 6.7. We define three variables to describe fragmentation in the data: T denotes the number of fragments that make up the dependency graph for a sentence, W refers to the number of words in the sentence, and MAX provides the size of the largest fragment in the graph.[4]

The first constraint schema ($T \leq x$) allows only sentences with a dependency structure that consists of at most x fragments. In other words, the *number of missing edges* must be smaller than x. The second schema is a variant of the first which additionally requires a minimum sentence length of y words in order to exclude trivial, uninformative training examples. The second group of schemata ($W/T \geq x$ and $W/T \geq x \wedge W \geq y$) directly relates the number of fragments T to the sentence length W by imposing a *minimum average fragment size.* Finally, the last pair of constraints effectively ensures that each sentence contains at least one *coherent fragment of size* x or larger.

The impact of fragmentation constraints on the performance of the fMalt parsers on the development set (the one that was also used for tuning the other parameters, cf. Section 6.2.2) is plotted in Figure 6.3. The figure shows results for those constraints where y equals 6; the interested reader will find the complete results in Appendix A. A brief glance at the plots reveals that the variation among the fragmentation constraints is surprisingly small. In fact, there is no single data set that results in significantly better results than all other data sets (cf. appendix). While no individual constraint stands out as superior, we can discern more general tendencies from the results. First, we generally observe

[4]The size of a fragment is defined as the number of nodes it consists of, which is equivalent to the number of words in the fragment's yield.

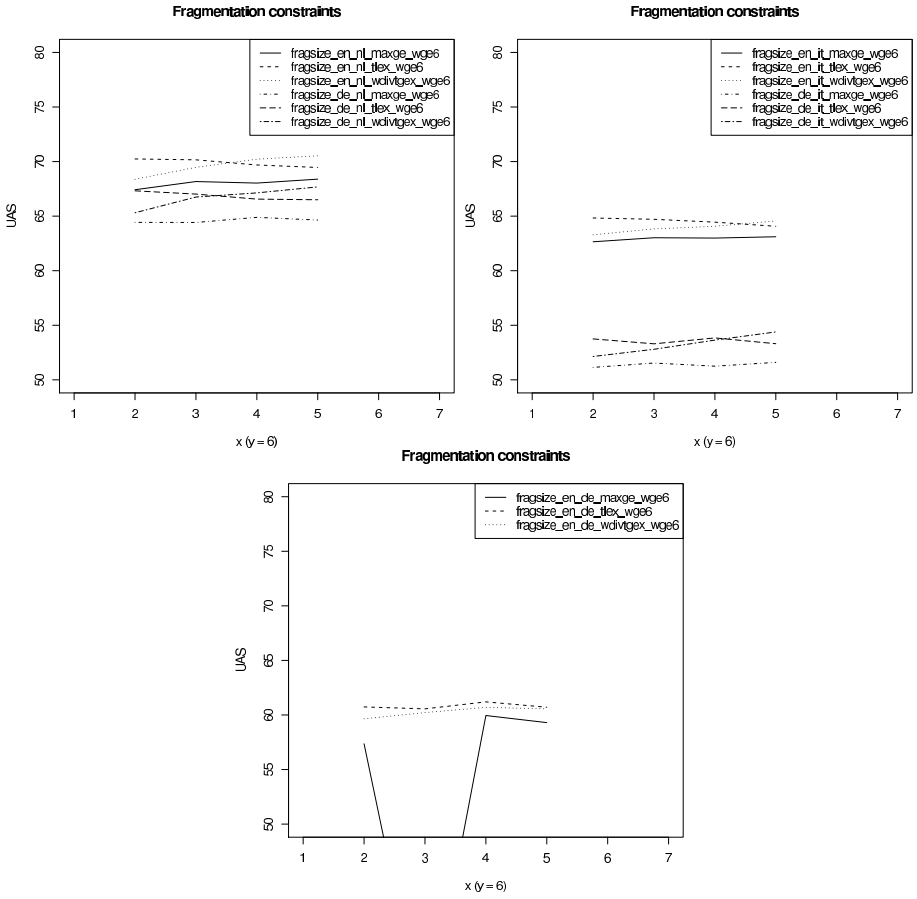


Figure 6.3: Effect of fragmentation constraints on the performance of fMalt parsers.

system		(f)Malt algorithm	(f)MST marking	order	k	fragmentation
en–nl	trees	standard	head	2	10	
	frags	standard	head	1	10	$W/T \geq 5 \wedge W \geq 6$
de–nl	trees	standard	head	2	10	
	frags	standard	head+path	2	1	$W/T \geq 5 \wedge W \geq 6$
en–it	trees	standard	baseline	2	10	
	frags	standard	baseline	2	10	$T \leq 3 \wedge W \geq 4$
de–it	trees	standard	head+path	2	10	
	frags	standard	head+path	2	10	$W/T \geq 5$
en–de	trees (bi.)	eager	head	2	2	
	trees (fb.)	standard				
	frags	standard	head	2	10	$T \leq 4 \wedge W \geq 6$

Table 6.8: Final parameter settings as determined on the development sets.

a.

lang	orig	PTB	Tiger
nl	79.23	**80.79**	79.19
it	**88.52**	86.88	84.02
de	86.92	87.12	cf.'orig'

b.

lang	orig	PTB	Tiger
nl	81.41	**83.01**	**83.87**
it	**90.23**	89.02	84.11
de	**89.86**	87.76	cf.'orig'

Table 6.9: UAS of Malt parsers (a) and MST parsers (b) trained on gold standard dependencies.

performance improvements with increasing restrictivity of the constraints, that is, fewer missing edges and greater average fragment size. In line with this observation, the results further suggest that constraints that merely require that there be at least one fragment of sufficient size (MAX $\geq x$) are not restrictive enough. None of the systems trained under these conditions can compete with the respective best constraints.

Table 6.8 summarizes the final parameter values used in the experiments in the remainder of this chapter. We omit the root handling parameter here since it always takes the value left.

6.3 Baselines and Upper Bounds

In order to situate our results with respect to supervised (treebank) parsing on the one hand, and simpler (heuristic and semi-heuristic) approaches on the other hand, we establish upper and lower bounds.

The treebank parsers presented in Section 5.3.3 – repeated here in Table 6.9 – can be considered upper bounds for the respective language pairs: they give

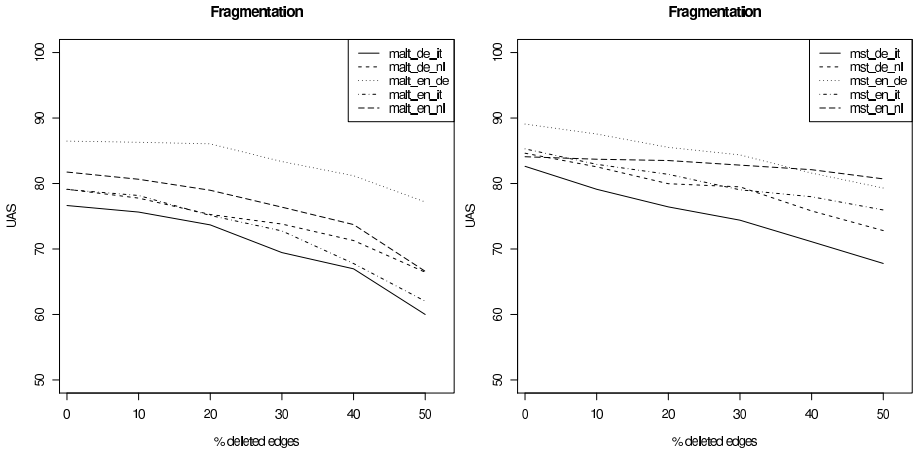


Figure 6.4: UAS of fMalt (left) and fMST parsers (right) trained on artificially fragmented gold standard dependencies (with predicted POS tags).

an indication of the level of accuracy that can be achieved when the training data is noise-free. Taking this line of thought further, we may also ask which evaluation figures can realistically be expected with noise-free, but fragmented training data. This amounts to an upper bound specifically for our approach to fragment parsing. We arrive at this upper bound by randomly removing a portion of the dependency edges from the training portions of the treebanks and then training fMalt and fMST on the resulting, artificially fragmented data sets. The results of this experiment are shown in Figure 6.4. Note that, in contrast to the results reported for supervised treebank parsers in chapter 5, the parsers evaluated in Figure 6.4 are trained using predicted rather than gold POS tags. As expected, the accuracy of the parsers drops as the degree of fragmentation (i.e., the proportion of missing edges) increases. Nonetheless, it is noteworthy that the slope of most of the curves is rather shallow. Especially the fMST parsers seem rather robust towards fragmentation: for instance, even with 50% missing edges, the Dutch fMST system projected from English sacrifices less than 3.5 percentage points UAS.

In addition to the upper bounds, we report the performance of simple, resource-lean baselines in Table 6.10. The first three columns of the table refer to purely heuristic baselines which attach each word to the preceding word ('prev'), the following word ('next'), or the ROOT node ('root'). These baselines reveal a clearly preferred attachment direction for German and Italian, across all annotation schemes. For Dutch, however, the right-attachment preference encoded in the original Alpino scheme diverges from the left-attachment favored under the PTB and Tiger schemes. We attribute this divergence to the flat, symmetrical treatment of coordinate structures in Alpino, which introduces right-attachments for all conjuncts to the left of the conjunction.

The rightmost column in Table 6.10 shows the results of a more sophisticated baseline ('pos'), which attaches words to the neighboring left *or* right word

		prev	next	root	pos
nl	PTB	**26.91**	24.77	10.28	**45.50**
	Tiger	**28.75**	16.71	10.28	**32.97**
	orig	23.65	**27.63**	10.28	**41.84**
it	PTB	**50.06**	14.41	5.12	**59.91**
	Tiger	**41.76**	13.00	5.12	**50.66**
	orig	**62.78**	8.95	4.71	**64.13**
de	PTB	18.67	**26.58**	7.13	**31.23**
	orig.	21.03	**22.38**	7.13	**27.16**

Table 6.10: Heuristic baselines (prev, next, root) and a weakly supervised baseline (pos).

on the basis of their POS tag. The attachment direction for a given tag is estimated from a small set of 10 annotated sentences (distinct from the test sets); alternatively, the direction could be provided by a native speaker. If a tag has not been encountered in the training sentences, the direction is assigned by the best performing heuristic baseline.

Clearly, the supervised POS-based baseline substantially outperforms the heuristic strategies across all languages and annotation schemes, with improvements ranging from Δ1.35 for Italian (under the TUT scheme) to Δ18.59 for Dutch (PTB scheme). As the strong performance of the 'prev' baseline for Italian already indicates, attachment to the left is predominant in this language under all three annotation schemes, albeit to varying degrees. It is therefore not surprising that a differentiation based on part-of-speech adds only little accuracy, especially if we take into account that neither baseline system generates arcs of length greater than one. This also explains the weak baseline performance on the Tiger annotation scheme, which calls for longer arcs due to its flat structures. Based on the advantage it gives for Dutch and Italian, one might expect that the German baseline should achieve considerably higher attachment scores when evaluated against the PTB scheme, but although we see some improvement ('next': Δ4.20, 'pos': Δ4.07), the performance is still mediocre in comparison to the other languages. We conjecture that may be due to the German STTS tagset on which the flat Tiger (Negra) annotation scheme is based.

Having established upper and lower bounds, the remainder of this chapter presents experimental results with transition-based (Section 6.4) and graph-based parsers (Section 6.5) that are trained on projected dependencies. A detailed error analysis will be provided in Chapter 7.

	trees (bi.)	trees (fb.)	baseline (pos)
en–nl	**72.38**∗∗	70.74	45.50
de–nl	**70.11**∗∗	59.40	32.97
en–it	**67.86**∗∗	64.90	59.91
de–it	**52.57**∗∗	47.28	50.66
en–de	**59.83**∗∗	57.79	31.23

Table 6.11: Unlabeled attachment scores of projected Malt parsers trained on 100,000 words.

6.4 Malt and fMalt

In this section, we consider transition-based Malt and fMalt parsers that are trained on projected data. We begin with the evaluation of conventional (original) Malt parsers, which assume that the training data consists of complete trees and are thus restricted to the fraction of training data consisting of complete trees.

6.4.1 Malt: parsers with completeness assumptions

When we use the original Malt parser for training, irrespective of the exact makeup of the training data, the implicit assumption underlying the training procedure is that the training examples are complete trees.

Table 6.11 shows the performance of two different Malt parsers that incorporate this "completeness assumption." The parsers are trained on samples of 100,000 words (after filtering). Note that a much larger pool of parallel data is required to obtain a data set of 100,000 words containing *only* complete projected analyses. This detail should not be underestimated: although there are relatively large amounts of unlabeled training data for many languages, it is still in limited supply for smaller languages, especially seeing as the projection method also requires a translation in the source language. Furthermore, as discussed in Chapter 3, the strictness of the completeness constraint tends to result in data sets made up of short, simple and repetitive sentences.

In the first column of Table 6.11 ('trees (bi.)') we see the results for models trained exclusively on trees as obtained under strict projection, that is, using only bidirectional word alignments. By contrast, the parsers in the 'trees (fb.)' column are derived from trees that were projected using the fallback mechanism, which imposes a more relaxed filtering criterion on projected dependency structures. The POS-based baselines are repeated in the rightmost column for comparison. Statistical significance of a parser A over the next best parser B is marked by asterisks on A; the number of asterisks indicates the level of confidence ($** : p < 0.01, * : p < 0.05$).[5] For readability, the highest score for each

[5]No significance tests were performed with respect to the baseline systems because the sampling approach pursued for the (f)Malt parsers (cf. Section 5.4) is not applicable to the

language pair is printed in bold face.

The most striking observation from Table 6.11 is that the bidirectionally projected trees give rise to the best parsers *across all language pairs,* and in each case the difference to the parser based on fallback projection is statistically significant ($p < 0.01$). The parsing accuracy in absolute terms varies, however. The highest attachment scores are obtained by the Dutch parsers, and in particular the Dutch parsers projected from English: both systems exhibit unlabeled attachment scores over 70%, the one trained on strict tree projections reaches a UAS of 72.38%. Both parsers outperform the baseline by 25 percentage points UAS or more.

The ranking among the Dutch parsers projected from German is analogous to that of the English–Dutch systems, but there is a noteworthy difference. The drop in UAS from the strict parser (trees (bi.): 70.11%) to the fallback parser (trees (fb): 59.40%) by more than 10 percentage points is considerably more pronounced than is the case with the parsers projected from English (Δ <2). As with the English–Dutch parsers, the baseline performance is exceeded by more than 25 percentage points UAS.

Turning to the Italian parsers we witness a substantial decrease in accuracy, for parsers projected from English, but even more so for those projected from German. While the test sets and hence the results across target languages are of course not immediately comparable, comparison across *source* languages is, to some extent, warranted. Recall from Chapter 5 that the test sets are converted to conform to the annotation scheme of the source language, and we have seen that the English (PTB) annotation scheme does indeed seem to be preferable over the German Tiger-style annotations when it comes to training transition-based parsers for Italian. However, the difference of more than 15 percentage points UAS which we observe here between the best English–Italian parser (67.86%) and the best German–Italian parser (52.57%) suggests that other factors are at play. In fact, we have seen in Chapter 3 that projection from German to Italian is more prone to data reduction – especially reduction of average sentence length – under all projection scenarios we investigated (Table 3.6 on page 36). Moreover, the percentage of duplicate sentences in the projected data is substantially higher for this language pair (Table 3.7, page 37), as is the degree of fragmentation under partial correspondence projection (Table 3.9, page 44). These observations lead us to conclude that the word alignment between the German and Italian Europarl sections are of poorer quality – or simply more sparse – than the word alignments between the other language pairs; this claim is substantiated by the percentage of unaligned tokens in Table 3.3. We have seen in the previous section that even the simple attach-left heuristics constitutes a strong baseline, and we observe now that only the 'trees (bi.)' parser achieves higher accuracy by a small margin (Δ1.91).

The results for German follow the same pattern: the parser projected under strict projection fares significantly better than the fallback parser. Both parsers clearly outperform the baseline (Δ >26).

The parsers in Table 6.11 are all trained on data sets of equal size, namely 100,000 words. However, we have seen in Chapter 3 that the effective amount of training data that can be obtained from a given corpus varies immensely depending on the projection technique and noise filter that is being used: precision-

baselines.

	trees (bi.)		trees (fb.)	
en–nl	(235,000)	**73.09**∗∗	(1,592,000)	72.79
de–nl	(458,000)	**72.28**∗∗	(2,438,000)	oom
en–it	(178,500)	**68.31**∗∗	(1,682,000)	65.92
de–it	(115,500)	**52.68**∗∗	(1,068,500)	48.38
en–de	(253,500)	**59.86**∗∗	(1,623,500)	58.10

Table 6.12: Unlabeled attachment scores of projected Malt parsers trained on the entire Europarl corpus, subject to filtering (effective training set size (words) given in parentheses).

oriented bidirectional projection in conjuction with a filter that discards all partial analyses yields very small data sets of presumably high-quality dependencies, while the amount of trees projected under fallback projection is comparatively large, closely followed by partial correspondence projection in terms of quantity. Given that the assumption underlying the relaxation of the noise filters is that quantity – and thereby variety – can make up for quality, a realistic picture of the merit of the various projection methods and filters can only be obtained by taking this quantitative aspect into account. We therefore repeat the experiments discussed above, but rather than controlling the effective training set size, we train the parsers on whatever data remains after filtering the projections of the entire Europarl corpus. The results are shown in Table 6.12, along with the number of words in the effective training sets. They indicate that the increase in sheer amount of data does not help the parser in overcoming the poor quality of the fallback projections. However, the *difference* between the scores of the respective systems is diminished. The results for the parser projected from German to Dutch under fallback projection is missing due to memory limitations.

6.4.2 fMalt: parsers with fragment awareness

So far we have only considered parsers that assume complete trees in the training data. In this section, we compare those parsers to fMalt, our fragment-aware variant of the Malt parser. As before, we first investigate parsers trained on equal (i.e., fixed) amounts of training data, namely 100,000 words. The results are summarized in Table 6.13. For comparison, we also repeat the results of the best tree-oriented parsers from Table 6.11.[6]

It is immediately evident that the results here are less clear-cut than those discussed above, since the fragment-aware parsers generally seem to achieve a level of accuracy very similary that of their tree-oriented counterparts (Malt, 'trees (bi.)'). No general trend is discernible when all models are trained on

[6]These results are at odds with the results reported in previous publications (Spreyer and Kuhn, 2009; Spreyer et al., 2010; Spreyer, 2010). This is due to an erroneous feature model used in training the tree-oriented parsers, which decreased their performance.

	Malt trees (bi.)	fMalt frags
en–nl	72.38	**72.88**∗
de–nl	**70.11**∗∗	68.75
en–it	**67.86**∗∗	67.29
de–it	52.57	52.80
en–de	59.83	59.19

Table 6.13: Unlabeled attachment scores of projected fMalt parsers trained on 100,000 words. For comparison, we also repeat the corresponding results of the original Malt parser trained only on bidirectionally projected complete trees, cf. Table 6.11.

	Malt trees (bi.)		fMalt frags	
en–nl	(235,000)	73.09	(1,138,500)	**74.26**∗∗
de–nl	(458,000)	**72.28**∗∗	(2,114,500)	71.62
en–it	(178,500)	68.31	(1,092,500)	68.08
de–it	(115,500)	52.68	(317,500)	**53.92**∗∗
en–de	(253,500)	59.86	(1,972,000)	60.21

Table 6.14: Unlabeled attachment scores of projected fMalt parsers trained on the entire Europarl corpus. For comparison, we repeat the corresponding results of the original Malt parser trained only on bidirectionally projected complete trees, cf. Table 6.12.

the same amount of data, neither across source languages nor across target languages: The tree-oriented (Malt) parsers significantly outperform the fragment parsers for the language pairs German–Dutch and English–Italian, while the opposite is true for English–Dutch, albeit at a weaker level of significance ($p < 0.05$). For the pairs German–Italian and English–German, the difference between the Malt and fMalt parsers is not significant.

But as before, we argue that we can only obtain realistic results when we take the quantitative aspect of projection and noise filtering into account by fixing the amount of training data *prior to filtering.* The results obtained in this setting are displayed in Table 6.14. And indeed, once we take the quantitative aspect into account, a clear picture emerges: except in one instance, none of the fMalt parsers fall short of their tree-trained Malt counterparts with statistical significance. On the contrary, the English-based Dutch fMalt and the German-based Italian fMalt both significantly outperform the corresponding Malt parsers.

	trees (bi.)	trees (fb.)	baseline (pos)
en–nl	**73.98**∗∗	68.27	45.50
de–nl	**71.02**∗∗	60.07	32.97
en–it	**66.82**∗∗	60.77	59.91
de–it	**51.02**∗∗	42.57	50.66
en–de	**63.21**∗∗	61.50	31.23

Table 6.15: Unlabeled attachment scores of projected MST parsers trained on 100,000 words.

6.5 MST and fMST

In this section we present results for the graph-based MST and fMST parsers. We proceed in essentially the same manner as we did for the Malt parsers in the previous section. However, due to time constraints and limited computing resources, we had to resort to smaller training sets and fewer sampling steps in some settings. Moreover, we are adopting the fragmentation constraints that were identified as ideal for the *fMalt* parsers, which presumably constitutes a suboptimal tuning for fMST. We explicitly identify diverging training circumstances when we discuss the affected parsers. The full battery of experimental results under comparable training conditions will be made public in the course of future work.

6.5.1 MST: parsers with completeness assumptions

Tables 6.15 and 6.16 show the results of MST parsers trained on 100,000 words and the first 500,000 words of Europarl, respectively. As for the corresponding Malt parsers in Section 6.4.1, we consider complete trees projected under strict projection ('trees (bi.)') and complete trees projected under constrained fallback projection ('trees (fb.)'). The parsers discussed in this section are instantiations of the original MST parser, which expects the training sentences to be annotated with complete trees. The results obtained with fragmented training data and fragment-aware fMST parsers will be discussed in Section 6.5.2.

The results for tree-oriented parsing with MST are largely analogous to the corresponding Malt results in that the parsers trained on bidirectional projections exhibit higher accuracy throughout. No clear trend is discernible that indicates that one of the parsers (Malt or MST) fares clearly better than the other. MST achieves higher absolute figures for Dutch and German, whereas Malt is superior on the Italian data. Seeing as both German and Dutch allow relatively free word order, whereas Italian does not, this is in line with the observations of McDonald and Nivre (2007). They show that the graph-based MST parser is more successful at learning about long distance dependencies, whereas the transition-based Malt parser models local dependencies very well.

	trees (bi.)		trees (fb.)	
en–nl	(90,000)	**73.72**∗∗	(607,000)	68.22
de–nl	(174,000)	**71.89**∗∗	(928,500)	59.83
en–it	(80,500)	**66.47**∗∗	(758,500)	60.51
de–it	(53,500)	**51.26**∗∗	(495,500)	42.53
en–de	(96,800)	**62.63**∗∗	(614,500)	61.90

Table 6.16: Unlabeled attachment scores of projected MST parsers trained on the first 500,000 sentences of the Europarl corpus, subject to filtering (effective training set size (words) given in parentheses).

	MST trees (bi.)	fMST frags
en–nl	73.98	73.90
de–nl	**71.02**∗∗	66.52
en–it	66.82	**68.84**∗∗
de–it	51.02	**53.04**∗∗
en–de	**63.21**∗∗	62.28

Table 6.17: Unlabeled attachment scores of projected fMST parsers trained on 100,000 words. For comparison, we repeat the corresponding results of the original MST parser trained only on bidirectionally projected complete trees, cf. Table 6.15.

6.5.2 fMST: parsers with fragment awareness

The result of training fragment-aware fMST parsers on 100,000 words of training data are summarized in Table 6.17. As before with fMalt, the results are not clear-cut. The fragment parsers significantly outperform the tree-oriented parsers for Italian, but the opposite is the case for German and for the Dutch parsers projected from German. The difference between the Dutch parsers projected from English is not significant.

Even when we train the parsers on data samples that reflect the aggressiveness of the underlying noise filters, no clear picture emerges (Table 6.18). With only one fifth of the amount of training data, the Dutch tree-oriented parsers still outperform their fragment-aware counterparts (73.72 versus 73.26 from English, 71.89 versus 65.27 from German). The reverse is true for German (62.63 versus 63.23) and for the Italian parser projected from German (51.26 versus 52.10). The difference between MST and fMST is not significant for the language pair English–Italian.

	MST trees (bi.)		fMST frags	
en–nl	(90,000)	**73.72**∗	(449,000)	73.26
de–nl	(174,000)	**71.89**∗∗	(797,500)	65.27
en–it	(80,500)	66.47	(492,500)	66.60
de–it	(53,500)	51.26	(145,000)	**52.10**∗∗
en–de	(96,800)	62.63	(484,000)	**63.23**∗

Table 6.18: Unlabeled attachment scores of projected fMST parsers trained on the first 500,000 sentences of the Europarl corpus.

6.6 Summary and Discussion

The empirical results presented in this chapter show that fragment parsers (fMalt and fMST) perform roughly on a par with their tree-based counterparts (Malt and MST). In fact, the fMalt parsers achieve unlabeled attachment scores as high as or higher than the Malt parsers, with the exception of one language pair (German–Dutch). The results with MST and fMST are not as clear-cut: The original MST parser significantly outperforms fMST on the language pairs English–Dutch and German–Dutch, while the opposite is true for the Italian parser projected from German and the German parser projected from English. There is no significant difference between MST and fMST for the language pair English–Italian.

We further find that the parsers projected under constrained fallback projection consistently fail to outperform the parsers based on strict (bidirectional) projections, despite the fact that the former are trained on considerably more training data. This finding is not surprising given the qualitative evaluation of the direct projections in Chapter 3: While the fallback projections improve recall to some extent, this cannot make up for the loss in precision and thus leads to substantially more noise in the training data of the fallback parsers. The results obtained with the fragment parsers, on the other hand, suggest that the even more pronounced gain in recall and thus greater quantity of training annotations obtained under partial correspondence projection does help overcome the noise we introduce when we allow fragmented projections.

In the next chapter we will explore the strengths and weaknesses of fragment and (strict) tree-based parsers in more detail. In particular, we will conduct evaluations by sentence length, dependency length, dependency type, and word class.

Chapter 7

Error Analysis

The error analysis presented in this chapter focuses on the differences between *bidirectional* (strict) tree-oriented parsers and the *fragment* parsers. It is based on the output of the parsers trained on the complete (MST parsers: half) Europarl corpus (Tables 6.14 and 6.18). We investigate the performance of our parsers relative to sentence length (Section 7.1), dependency length (Section 7.2), and dependency type (Section 7.3). We conclude the chapter with two concrete examples in Section 7.4.

7.1 Sentence Length

In order to observe the performance gradient with respect to sentence length, we divided the test sentences into bins of six different length ranges, namely sentences with one to four words, five to nine words, 10 to 19, 20-29, 30-39 and finally 40 words or more. Each of these bins was then evaluated separately. The results are given in Table 7.1 for (f)Malt, in Table 7.2 for (f)MST.

Comparing tree with fragment parsers across all language pairs and bins, we find that fragment parsers generally tend to outperform their tree-oriented counterparts on long sentences, and vice versa for the shorter sentences. This is especially obvious for the fMalt parsers (Table 7.1). With the exception of the English–Dutch system, fMalt outperforms the tree-oriented Malt parser on sentences longer than 40 words across all languages. The Italian and German fMalt systems further perform better on sentences with 30 to 39 words. The German fMalt parser also outperforms the tree-oriented counterpart on sentences of length 10 or longer.

For the fMST parsers the trend is not as pronounced. But even so, we find that most fragment parsers outperform the tree-oriented parser on the set of extremely long sentences ($\geq$40 words), and the English–Dutch, English–Italian and English–German fMST systems extend this tendency to sentences of 30 to 39 words.

The observation that the fragment parsers tend to fare better on longer sentences can of course be explained by the nature of the training sets that the parsers learned from: as we saw in Chapter 3, the training examples projected under bidirectional trees-only projection comprise only six to eight words per sentence on average. The fragment parsers, on the other hand, are trained on

		1–4	5–9	10–19	20–29	30–39	≥40
en-nl	trees	**66.42**	**72.33**	74.60	73.68	73.47	**55.33**
	frags	66.10	71.46	**76.02**	**75.55**	**74.15**	53.80
de-nl	trees	**69.68**	70.75	**74.20**	**74.40**	**68.75**	49.53
	frags	68.05	**71.37**	74.14	73.08	67.74	**50.47**
en-it	trees	**65.42**	**68.71**	**69.21**	**68.71**	64.79	70.09
	frags	47.08	66.60	68.01	68.22	**65.72**	**70.55**
de-it	trees	73.33	**70.61**	**58.61**	52.59	47.58	48.86
	frags	**86.67**	67.77	57.49	**53.30**	**50.47**	**51.70**
en-de	trees	**69.57**	**71.35**	63.06	58.91	55.02	47.35
	frags	65.92	69.14	**63.33**	**59.47**	**56.22**	**48.94**

Table 7.1: UAS relative to sentence length (Malt). The higher of the two scores (trees versus frags) is in bold face.

		1–4	5–9	10–19	20–29	30–39	≥40
en-nl	trees	**66.04**	72.70	**76.74**	**74.57**	71.25	52.67
	frags	65.66	**73.68**	75.68	73.86	**71.92**	52.67
de-nl	trees	68.93	**69.35**	**75.23**	**73.36**	**67.79**	46.66
	frags	**69.31**	64.83	68.74	64.91	61.33	**48.40**
en-it	trees	66.67	**69.48**	**68.97**	**66.40**	62.93	66.95
	frags	**74.17**	68.26	68.68	65.44	**64.28**	**67.56**
de-it	trees	68.34	**72.39**	55.45	50.44	48.77	**46.99**
	frags	**75.00**	71.82	**56.78**	**52.21**	**49.39**	46.81
en-de	trees	**76.56**	**76.29**	**67.41**	60.88	55.65	47.91
	frags	75.91	74.20	67.09	**61.16**	**57.80**	**52.94**

Table 7.2: UAS relative to sentence length (MST). The higher of the two scores (trees versus frags) is in bold face.

sentences with an average length of 11 to 18 words. Thus, while the superior performance of the fragment parsers on longer sentences is a straightforward consequence of the makeup of the underlying training sets, it also shows that the strengths of the fragment parsers are complementary to those of the tree-oriented parsers.

7.2 Dependency Length

A factor related to sentence length is dependency length, that is, the distance (in words) between a dependent and its head. Long distance dependencies are typically harder for most parsers to model and discover than short dependencies between adjacent or close words. This is because longer dependencies cannot be modeled using only local context. In fact, dozens of words can in principle intervene between a head and its dependent in a long distance dependency. In this section, we evaluate our parsers with respect to their performance on dependencies of increasing length. This is again accomplished by aggregating dependencies of similar length into bins. Specifically, the first bin contains dependencies of length 1, the second bin those of length 2, the third combines dependencies of length 3 to 6, and the last bin contains all remaining dependencies (length $\geq$7). The results of this evaluation are presented in Figures 7.1 and 7.2 for (f)Malt and (f)MST, respectively. In the left column of each figure, we show the f-score for each bin. The right column plots the corresponding precision–recall tradeoff. The tables underlying these plots are provided as Appendix B.

Malt/fMalt. Let us first consider the (f)Malt results in Table 7.1. The obvious observation is that performance degrades with increasing dependency length, which can be seen in the downward slope of the f-score curves (left column). While this slope is rather "constant" for the Dutch parsers, it levels off for (most of) the Italian parsers, and even more so for the German systems. Comparing the Dutch parsers projected from German, we find that Malt (de) and fMalt (de) perform almost identical on dependencies up to and including length 6. Dependencies in the fourth bin are captured more accurately by the fragment parser. The plot on the right-hand side further shows that both parsers find a relative balance between recall and precision, with a slight orientation towards recall, which is more pronounced for very long dependencies (+).

A similar picture emerges for the parsers projected from English: the respective f-score curves are close. In addition, fMalt outperforms Malt on dependencies of length 2. The precision-recall plot reveals an interesting detail which holds for both parsers: they are rather recall-oriented on short dependencies (length 1 and 2), but geared towards precision when it comes to longer dependency edges. The fMalt system exhibits higher precision and higher recall across all bins except bin 3, where it lags behind on recall by less than 0.5 points.

We now turn to the Italian (f)Malt parsers in the second row of Table 7.1. The f-score curves have a similar shape as those for Dutch in that the fragment parsers maintain a considerably higher level of quality on dependencies of length 3 and longer. Especially the tree-oriented parser from German is seriously limited on very long edges, which is due to drops in both recall and precision. The

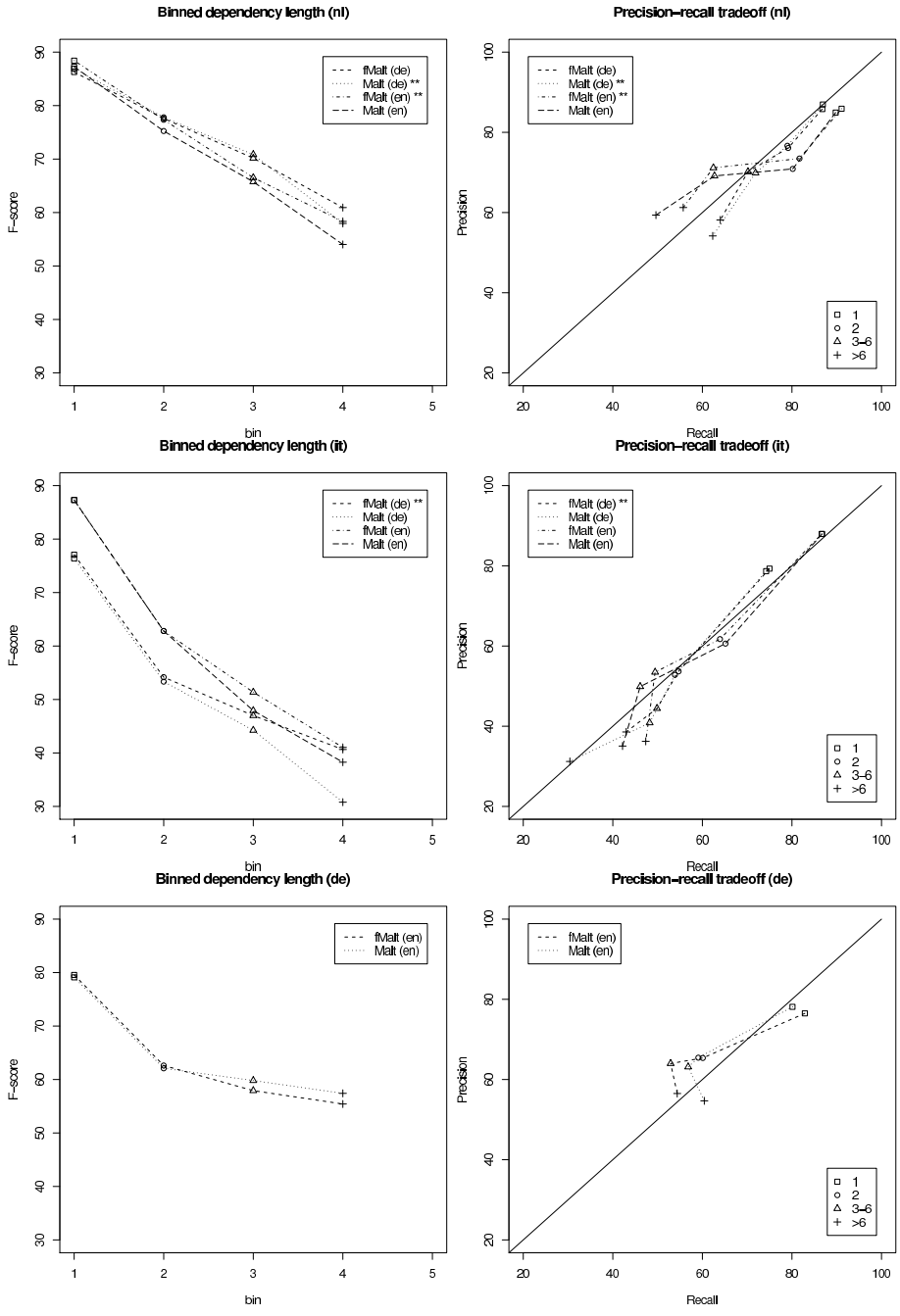


Figure 7.1: F-score and precision-recall tradeoff for dependencies of length 1 (bin 1, □), 2 (bin 2, ○), 3–6 (bin 3, △) and longer than 6 (bin 4, +) for Malt and fMalt parsers.

English-based parsers both have very low precision on dependencies in bin 4, but exhibit a fairly balanced tradeoff otherwise.

Finally, the German parsers perform almost alike on short dependencies (bins 1 and 2). In contrast to the other languages, however, it is the *tree-oriented* Malt parser that is clearly superior on longer dependencies. If we break down the f-score into precision and recall, we see that the performance drop is really a recall issue: the fragment parser lags behind by more than 6 percentage points on dependencies longer than 6 words (+), and by almost 4 percentage points on dependencies of length 3 to 6 (△), while the precision remains above that of the tree-oriented Malt parser in both cases.

MST/fMST. The plots for the (f)MST parsers are shown in Figure 7.2. The general trends conveyed by these plots are not entirely unlike the corresponding (f)Malt behavior. However, there are noteworthy differences.

For Dutch, the fMST parser projected from German underperforms consistently, and especially on dependencies in bin 3. The corresponding precision-recall curve reveals that this parser fails to strike a balance between precision and recall. While it suffers from relatively low precision on dependencies of length 1 (□), it lags behind in terms of recall on dependencies longer than 1.

The Italian parsers suffer immense drops in f-score on longer dependencies, with the English-based fMST parser being affected most. For this parser, the degradation is due to recall limitations, whereas the other (f)MST parsers for Italian sacrifice precision on longer dependencies.

Finally, the German parsers ('fMST (en)' and 'MST (en)') perform roughly alike, and even exhibit nearly the same precision-recall tradeoffs, except on very long dependencies, where fMST maintains a relatively high precision at the cost of recall, and MST does the inverse.

7.3 Dependency Type

In this section we analyze the performance of our parsers by dependency type. In particular, we partition the dependency edges in the test sets according to their *gold standard label*, and subsequently obtain *unlabeled* attachment scores for each of these bins. Tables 7.3 through 7.8 contain results for most dependency types found in the target language treebanks.[1]. We focus our discussion on three major dependency types – subjects, objects, and modifiers – which manifest themselves in one or more edge labels, depending on the annotation scheme.

7.3.1 Subjects

All treebanks considered here have a single label for subjects proper: su in the Dutch Alpino Treebank, subj in the Italian TUT, and sb in the German Tiger treebank. In addition, the label sbp is employed in the German treebank to mark subjects in passive constructions. The Dutch treebank uses the label sup to distinguish provisional subjects, that is, expletives in subject position.

The results for subject dependencies are mixed. While the Dutch fMalt parsers projected from English outperform the tree-based Malt system on both

[1] We omit some highly infrequent labels.

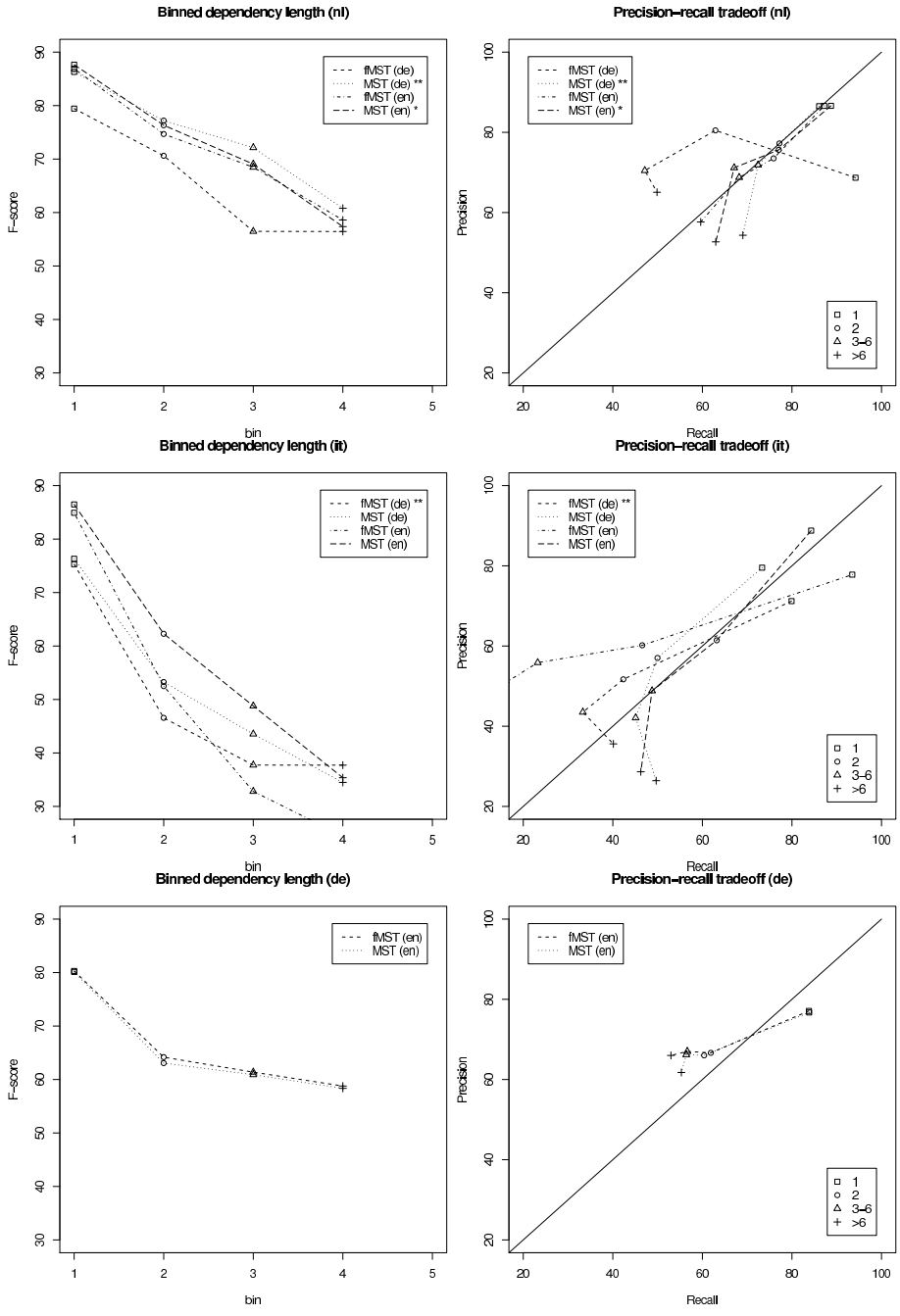


Figure 7.2: F-score and precision-recall tradeoff for dependencies of length 1 (bin 1, □), 2 (bin 2, ○), 3–6 (bin 3, △) and longer than 6 (bin 4, +) for MST and fMST parsers.

		app	rel	body	cnj	crd	det	hdf
en-nl	Malt	**43.33**	61.20	60.70	**65.58**	57.30	93.27	**30.00**
	fMalt	41.73	**80.80**	**61.95**	63.81	**59.19**	**93.93**	26.67
de-nl	Malt	52.80	**78.33**	–	55.15	64.42	91.30	6.67
	fMalt	**54.13**	76.67	–	**55.22**	**64.52**	**91.39**	**33.33**

		ld	me	mod	obcomp	obj1	obj2	pc
en-nl	Malt	**67.39**	20.00	67.96	35.56	82.10	**72.50**	**56.83**
	fMalt	65.22	**66.67**	**69.70**	**44.44**	**83.04**	67.50	56.63
de-nl	Malt	59.13	23.33	**65.49**	**21.11**	**81.90**	43.75	64.36
	fMalt	**60.87**	**40.00**	64.03	12.22	79.78	**56.25**	**69.50**

		pobj1	pred	se	su	sup	svp	vc
en-nl	Malt	**50.00**	69.70	33.33	74.80	97.50	**53.57**	**79.96**
	fMalt	45.00	**72.53**	**86.67**	**76.80**	**100.00**	50.00	79.65
de-nl	Malt	50.00	**75.15**	66.67	**79.25**	**100.00**	59.52	**82.44**
	fMalt	50.00	68.18	**96.67**	77.88	95.00	**63.33**	82.29

		cmp
en-nl	Malt	–
	fMalt	–
de-nl	Malt	85.25
	fMalt	**86.75**

Table 7.3: Dutch (f)Malt parsing accuracy (unlabeled) across gold standard dependency types. rel is the label we introduced in the conversion step to label the dependency from the subordinated verb in a relative clause to the relative pronoun.

		apposition	arg	coord	empty-compl	extra-obj	ind-compl	ind-obj
en-it	Malt	**37.76**	**81.43**	**43.98**	55.56	100.00	**74.16**	61.29
	fMalt	36.63	81.01	41.43	**59.44**	100.00	70.07	**64.84**
de-it	Malt	**29.49**	64.26	24.59	54.44	100.00	**70.42**	37.42
	fMalt	29.29	**64.81**	**28.28**	**60.56**	100.00	69.58	**42.58**

		obj	pred	rmod	rmod+relcl	subj	visitor
en-it	Malt	66.67	**71.50**	68.44	53.79	**56.39**	**48.33**
	fMalt	**67.91**	66.54	**69.50**	**57.06**	55.45	40.00
de-it	Malt	48.31	67.38	47.45	36.86	42.49	29.17
	fMalt	**49.34**	**71.78**	**48.49**	**36.93**	**43.76**	**30.83**

Table 7.4: Italian (f)Malt parsing accuracy (unlabeled) across gold standard dependency types.

		ag	app	cc	cd	cj	cm	cp
en-de	Malt	9.33	**16.67**	0.00	64.61	64.07	**8.89**	–
	fMalt	**13.87**	10.95	0.00	**66.48**	**66.63**	7.78	–

		cvc	da	ep	mo	ng	nk	nmc
en-de	Malt	**100.00**	46.82	**92.73**	43.92	**37.73**	**76.80**	**82.08**
	fMalt	95.00	**58.18**	91.82	**48.30**	35.45	76.74	80.83

		oa	oc	op	pd	pg	ph	pm
en-de	Malt	59.56	**64.89**	49.57	**78.71**	90.91	100.00	1.11
	fMalt	**64.37**	59.25	**62.61**	73.23	90.91	100.00	**2.22**

		pnc	rc	re	sb	sbp	svp	vo
en-de	Malt	1.04	29.72	0.77	**66.96**	29.00	**49.68**	100.00
	fMalt	**1.49**	**32.22**	**7.69**	65.92	**30.00**	33.23	100.00

Table 7.5: German (f)Malt parsing accuracy (unlabeled) across gold standard dependency types.

		app	rel	body	cnj	crd	det	hdf
en-nl	MST	39.62	**74.86**	**52.90**	**62.89**	57.41	93.65	0.00
	fMST	**40.53**	72.80	47.97	58.35	**59.91**	**93.82**	0.00
de-nl	MST	**47.47**	**76.67**	–	**57.22**	**65.71**	**89.99**	**26.67**
	fMST	43.47	54.44	–	48.21	57.14	87.46	0.00

		ld	me	mod	obcomp	obj1	obj2	pc
en-nl	MST	72.67	33.33	67.68	**36.51**	**80.19**	62.50	64.78
	fMST	**80.00**	**66.67**	**68.76**	22.22	76.84	**67.50**	**67.13**
de-nl	MST	**66.96**	20.00	**62.65**	0.00	**81.82**	**75.00**	**68.32**
	fMST	60.87	**66.67**	50.17	0.00	76.89	47.50	65.74

		pobj1	pred	se	su	sup	svp	vc
en-nl	MST	**50.00**	**75.47**	28.57	**81.42**	**89.29**	60.54	**82.78**
	fMST	20.00	72.73	**33.33**	80.78	80.00	**64.29**	80.23
de-nl	MST	**50.00**	**82.42**	33.33	**80.07**	**100.00**	**75.24**	**82.48**
	fMST	0.00	70.91	**100.00**	74.25	95.00	56.67	81.24

		cmp
en-nl	MST	–
	fMST	–
de-nl	MST	**80.50**
	fMST	74.00

Table 7.6: Dutch (f)MST parsing accuracy (unlabeled) across gold standard dependency types.

		appo-sition	arg	coord	empty-compl	extra-obj	ind-compl	ind-obj
en-it	MST	28.98	79.74	**41.43**	**47.78**	0.00	69.11	**61.94**
	fMST	**37.96**	**82.98**	39.90	36.67	0.00	**81.84**	58.06
de-it	MST	**18.98**	62.57	20.59	**54.44**	**100.00**	70.52	41.94
	fMST	15.10	**62.71**	**24.13**	44.44	60.00	**74.12**	41.94

		obj	pred	rmod	rmod+ relcl	subj	visi-tor
en-it	MST	**67.35**	58.50	**66.76**	**47.45**	**59.36**	31.67
	fMST	66.46	**67.85**	62.28	40.00	46.48	**38.33**
de-it	MST	50.21	62.24	**46.20**	**24.18**	45.49	31.67
	fMST	**53.97**	**72.90**	45.58	24.05	**46.48**	**40.00**

Table 7.7: Italian (f)MST parsing accuracy (unlabeled) across gold standard dependency types.

		ag	app	cc	cd	cj	cm	cp
en-de	MST	12.27	**20.95**	0.00	64.06	64.42	0.00	–
	fMST	**21.07**	7.62	0.00	**66.56**	**68.37**	0.00	–

		cvc	da	ep	mo	ng	nk	nmc
en-de	MST	90.00	**48.18**	**100.00**	**49.33**	33.18	77.88	64.17
	fMST	**100.00**	35.45	92.73	49.11	**36.36**	**78.99**	**90.83**

		oa	oc	op	pd	pg	ph	pm
en-de	MST	**59.61**	**69.57**	60.43	79.35	**84.55**	100.00	**5.93**
	fMST	56.50	67.21	**63.91**	**81.29**	77.27	100.00	4.44

		pnc	rc	re	sb	sbp	svp	vo
en-de	MST	**1.19**	**29.44**	0.00	**71.56**	34.00	49.68	100.00
	fMST	0.00	26.11	**3.08**	70.99	**46.00**	**50.32**	100.00

Table 7.8: German (f)MST parsing accuracy (unlabeled) across gold standard dependency types.

regular and provisional subjects, the opposite is true for parsers projected from German (Table 7.3). If we look at the corresponding (f)MST parsers in Table 7.6, we find that the original system outperforms fMST across both source languages, for regular as well as provisional subjects.

The results for Italian in Tables 7.4 and 7.7 are more clear-cut: the original Malt and MST parsers prevail when they are trained on dependencies projected from English, while the fragment parsers are superior in retrieving subject dependencies when the training examples are projected from German.

Finally, Tables 7.5 and 7.8 show that both fragment parsers outperform their tree-based counterparts on sbp dependencies, but not on regular subjects (sb).

7.3.2 Objects

Object dependencies constitute a much more diverse dependency type than subjects. We consider direct and indirect (nominal) objects, verbal and clausal object complements, and prepositional objects. In the treebanks, the corresponding labels are the following. The Dutch treebank uses obj1 and obj2 for direct and indirect objects, respectively. Verbal complements are denoted as vc, and prepositional objects bear the label pc. We will further consider the labels pobj1 (provisional direct objects) and ld (locative and directional complements). The Italian treebank provides the labels obj, indobj and indcompl for direct and indirect objects and verbal complements, respectively. Moreover, there is a distinguished label extraobj for duplicated objects. In the German annotations, we encounter oa (direct objects), da (indirect objects), oc (clausal objects), and op (prepositional objects).

7.3.3 Modifiers

In order to assess the accuracy of modifier attachments, we consider the labels mod (Alpino), mo[2] (Tiger), and rmod (TUT). We further consider relative clauses, which have separate labels in the Italian (rmod+relcl) and German (rc) treebanks.

The Dutch fMalt and fMST parsers projected from English recover modifier dependencies with almost 70% UAS. In the German–Dutch setting, however, the tree-based parsers outperform the fragment parsers, and the attachment score is generally lower (Malt: 65.49%, MST: 62.65%).

Turning to the Italian parsers (Tables 7.4 and 7.7) we find that the fMalt systems outperform the tree-oriented Malt parsers across the board on modifier dependencies, including relative clauses. In the graph-based paradigm, however, the opposite is the case: the original MST parser consistently outperforms fMST on those dependencies (again including relative clause attachments). We find the same pattern among the German systems: fMalt outperforms Malt, while fMST lags behind MST (although only by $\Delta 0.22$ UAS on mo-dependencies).

[2]The figures reported for mo in Tables 7.5 and 7.8 summarize the performance on the original labels mo and mnr.

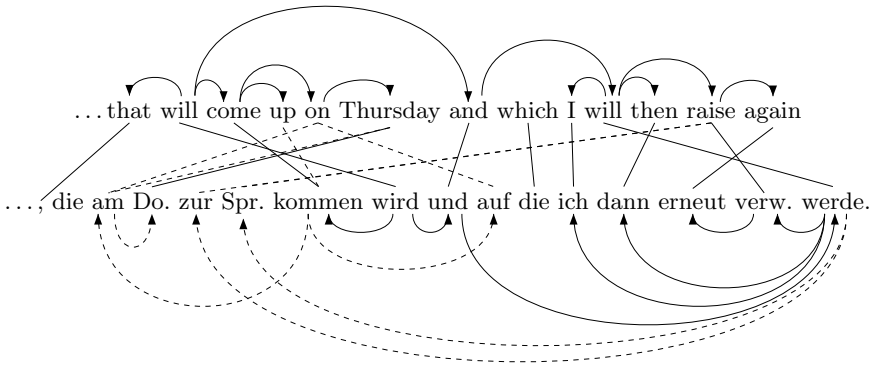


Figure 7.3: English–German sentence pair with automatic word alignment and projected dependency structure. Solid arcs indicate edges supported bidirectionally, dotted arcs are licensed under fallback projection

7.4 Concrete Examples

In this section we look at two example sentences in detail. Our first example is a sentence from the German section of the Europarl corpus, while the second example is from the Italian test set, that is, from the TUT treebank. For each of these sentences, we will compare the gold standard parse tree to the dependency structures predicted by our parsers.

Let us begin with the German example. A gloss for the sentence is given in (7.1); for reasons of space and ease of readability, we will omit parts of the sentence in the discussion that follows.

(7.1) *Meine Frage betrifft eine Angelegenheit, die am*
my question concerns a matter which on
Donnerstag zur Sprache kommen wird und auf die ich
Thursday to the speech come will and to which I
dann erneut verweisen werde.
then again refer will
'My question relates to something that will come up on Thursday and which I will then raise again.'

Since the example is taken from Europarl, we also have access to the English translation and can thus examine the dependency structure projected onto the German sentence on the basis of the automatic Giza++ word alignment. The aligned sentence pair is depicted in Figure 7.3. We show the English sentence above, with the dependency graph as predicted by the English source parser (cf. Section 3.3). The dependency edges shown below the German sentence in the lower half of Figure 7.3 are those projected from English. Solid arcs denote edges supported under the intersection of the two alignment directions – that is, strict as well as partial correspondence projection – whereas dotted edges are licensed only under constrained fallback projection.

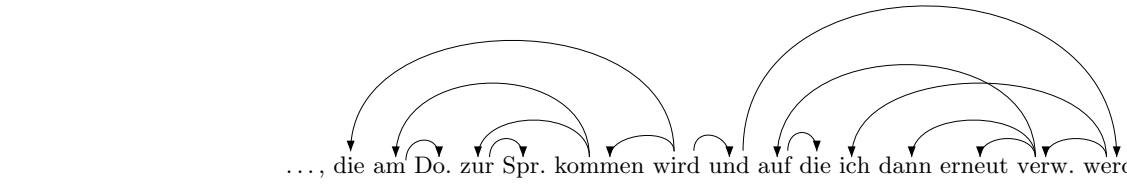


Figure 7.4: Gold standard parse for the German example (transformed to PTB conventions).

We first note that the analysis proposed by the English source parser is not complete: the relative pronoun *which* is left unattached. This directly leads to a missing attachment of the corresponding German pronoun *die*, despite the fact that Giza actually recovered the correct alignment between the two words. The word alignment identifies many of the correspondences, but not all of them. The bidirectional (strong) links are naturally sparse, but those that are posited are also correct, except the alignment of *that* with the comma. The unidirectional (weak) alignment links boost the coverage to some extent, but also introduce considerable noise. For instance, while the particle *up* is correctly aligned to the German verb *kommen*[3] and the correspondence between the prepositions *on* and *am* is identified, the unidirectional alignment postulates a second, erroneous alignment of *on* with the preposition *auf*, and further suggests that *raise* in the English sentence is equivalent to the German prepositional phrase *zur Sprache*. The latter might be the case in a different context, but is not correct in the present sentence.

We can now compare the analysis projected to the German sentence via this word alignment to the desired dependency structure for the sentence, shown in Figure 7.4. Some arcs are missing in the projected dependencies due to missing alignments. This concerns the attachment of both occurrences of the relative pronoun *die*. Second, we find that five out of the six edges established under strict projection (solid arcs in Figure 7.3) are in accordance with the desired tree. Only the attachment of *dann* to the auxiliary (*werde*) deviates from the "gold" analysis, which stipulates an attachment to the main verb *verweisen*. Turning to the fallback projections, we are faced with quite a few incorrect edges, namely three out of five. All of these errors are due to the misalignments discussed above: the word *auf* is mistakenly attached to *kommen* because of the uncertain alignments involving the English *on*. Both *zur* and *Sprache* are attached to *werde* because they are aligned with *raise*, the projected head of which is *werde*.

When we evaluate the projected trees – one obtained under strict, the other under fallback projection – according to the evaluation metrics P_f and R_f introduced in Chapter 3 (page 48), they both achieve scores of zero in the tree-oriented paradigm, since they are discarded due to incompleteness. Un-

[3]It is of course arguable whether such heterogenous alignment links – between a particle/preposition and a verb in this case – are indeed useful for a task like dependency parsing when the syntactic properties of the aligned categories differ so obviously.

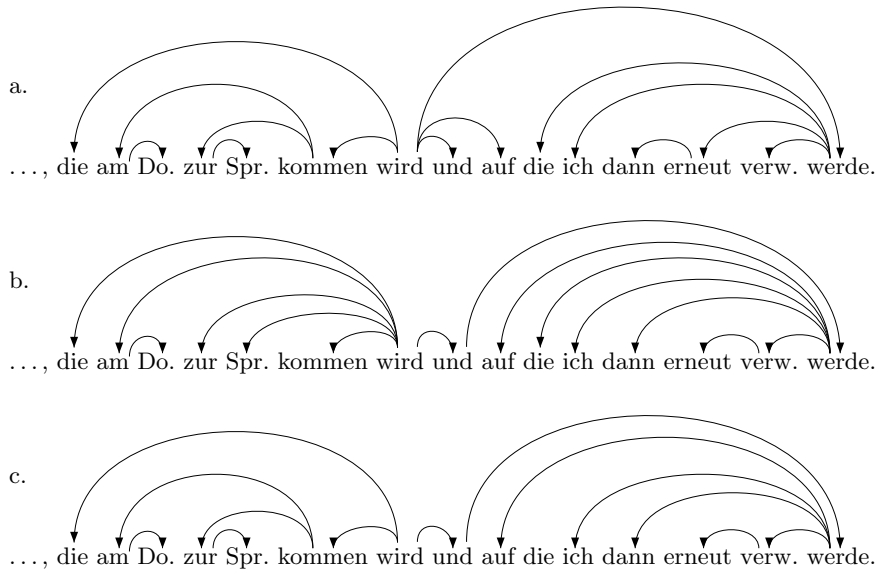


Figure 7.5: Predicted parse trees for the German example sentence: a. Malt (strict trees), b. Malt (fallback trees), c. fMalt.

der partial correspondence projection (considering only bidirectionally projected edges), we have $P_f = \frac{6}{7} = 85.7\%, R_f = \frac{6}{15} = 40.0\%$ and thus $F_f = 54.6\%$.[4]

Now that we have discussed the gold and projected dependencies for the sentence, we contrast these with the dependency structures predicted by our projected parsers. The predictions from the Malt and fMalt parsers are shown in Figure 7.5. We first note that every single one of these trees looks by far more reasonable than the projected structure from Figure 7.3, which confirms that, at least to some degree, the multitude of (potentially fragmented, potentially noisy) projected annotations enables the parsers to construct a model of the target language which assigns mostly complete dependency trees to new sentences.[5]

Figure 7.5a shows the prediction of the original Malt parser trained on strictly projected trees. Comparison with the "gold" parse in Figure 7.4 re-

[4] If we were to also consider the weakly projected edges in partial correspondence projection – thus abandoning the high-precision filter imposed by bidirectionality – we could achieve an F_f score of 64.3% brought about by an increased recall of $R_f = \frac{9}{15} = 60.0\%$, but at the cost of severely impaired precision $P_f = \frac{9}{13} = 69.2$.

[5] At this point we should point out that the projected annotations depicted in Figure 7.3 are not actually part of the training data of any of the parsers. Since the projections do not form a complete tree, they are discarded by both of the tree-oriented projection algorithms. Partial correspondence projection, when restricted to bidirectional alignments as we do here, also dismisses the analysis due to the high degree of fragmentation: eight fragments, with 1.9 words on average, are too much to pass the filter we determined as appropriate during parameter tuning in Chapter 6.

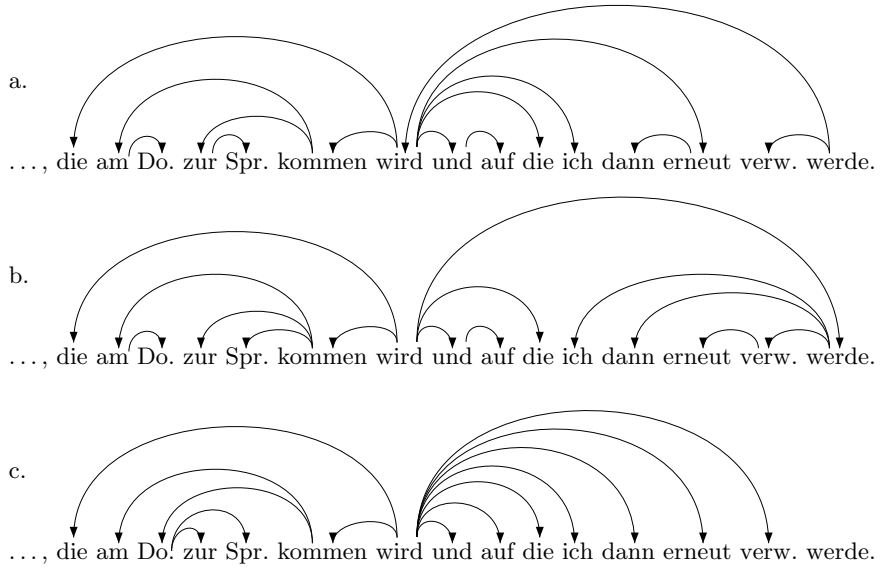


Figure 7.6: Predicted parse trees for German example sentence: a. MST (strict trees), b. MST (fallback trees), c. fMST.

veals that five out of the 15 proposed attachments are incorrect. Two of these five involve the attachment of modifiers (*dann* and *erneut*), which should both modify the main verb *verweisen*. Furthermore, the second occurrence of the relative pronoun *die* should be a dependent of *auf*, *auf* in turn the prepositional object of *verweisen*. Finally, the parser did not correctly identify the coordinate structure (*. . . wird und . . . werde*), but rather attached the second conjunct directly to the first. The analysis proposed for the first conjunct is completely correct.

The Malt parser trained on fallback projections (Figure 7.5b) produces six erroneous attachments, three in each conjunct. The prepositions *am* and *zur* are incorrectly attached to the auxiliary, and so is the noun *Sprache*, which should be a dependent of *zur*. This parser also fails when it comes to attaching *auf*, *die* and *dann*. In contrast to the bidirectionally projected parser, however, the fallback-based parser successfully identifies (i) the coordination of the auxiliaries and (ii) *verweisen* as the head of *erneut*.

In Figure 7.5c we finally see the analysis proposed by the fMalt fragment parser. It leaves one word unattached (the relative pronoun *die*, which is attached wrongly by the two other parsers) and produces two errors in identifying the head of *auf* and *dann*, respectively.

We conclude the discussion of the German example by comparing the numerical scores achieved by the respective analyses. Malt trained on strict trees (Figure 7.5a) correctly predicts the heads of ten out of 15 words, thus

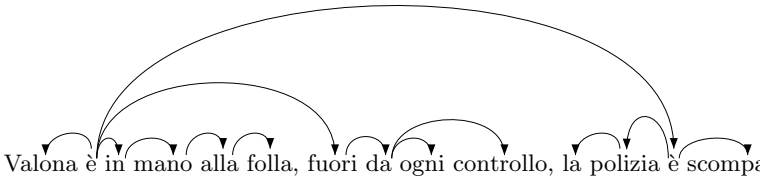


Figure 7.7: Gold standard parse for Italian example (transformed to Tiger conventions).

$P_f = R_f = \frac{10}{15} = 66.7\%$. The Malt parser based on fallback trees (Figure 7.5b) makes one more error, hence $P_f = R_f = \frac{9}{15} = 60.0\%$. Finally, the analysis proposed by fMalt for this sentence leaves one word unattached and makes two errors: $P_f = \frac{12}{14}, R_f = \frac{12}{15}$ and $F_f = 82.8\%$.

For the sake of completeness we also show the corresponding analyses predicted by the (f)MST parsers in Figure 7.6. These analyses are, however, somewhat disappointing, and neither approaches the quality of the Malt-predicted dependencies. The Italian example we will discuss now illustrates more reasonable predictions by the MST parsers.

Figure 7.7 shows the gold standard parse tree for our second example, which is part of the test set for our Italian parsers, taken from the TUT treebank. The sentence is glossed[6] in (7.2).

(7.2) *Valona è in mano alla folla, fuori da ogni controllo,*
Valona is in hand of the crowd, outside of every control,
la polizia è scomparsa, [...].
the police is passing
'Valona is in the hands of the crowds, out of control, the police are vanquished.'

We show the predicted trees of the MST and fMST parsers in Figure 7.8. None of the parsers correctly identifies the coordination of the first and second occurrence of *è*. The MST parser trained on strict trees (Figure 7.8a) makes six mistakes altogether. In particular, it attaches *in* to *alla* rather than to *è*, and the prepositions *alla* and *da* to a verb in a later part of the sentence not shown here.[7] Furthermore, *fuori* is analyzed as a dependent rather than the head of *da*, which is also postulated to be the head of *polizia*. The latter is in turn mistaken as the head of the auxiliary *è*. These six errors lead to $P_f = R_f = F_f = \frac{14-6}{14} = 57.1\%$.

Turning to the analysis of the fallback parser in Figure 7.8b we observe nine errors overall. Six of these errors involve the participle *scomparsa*, which is

[6]The gloss and translation are constructed on the basis of the automatic translation obtained using `babelfish.yahoo.com` since the author of this thesis does not speak Italian.

[7]The entire sentence reads *Valona è in mano alla folla, fuori da ogni controllo, la polizia è scomparsa, rintanata nelle caserme.* The attachment of *alla* and *da* in Figure 7.8a points to *rintanata*.

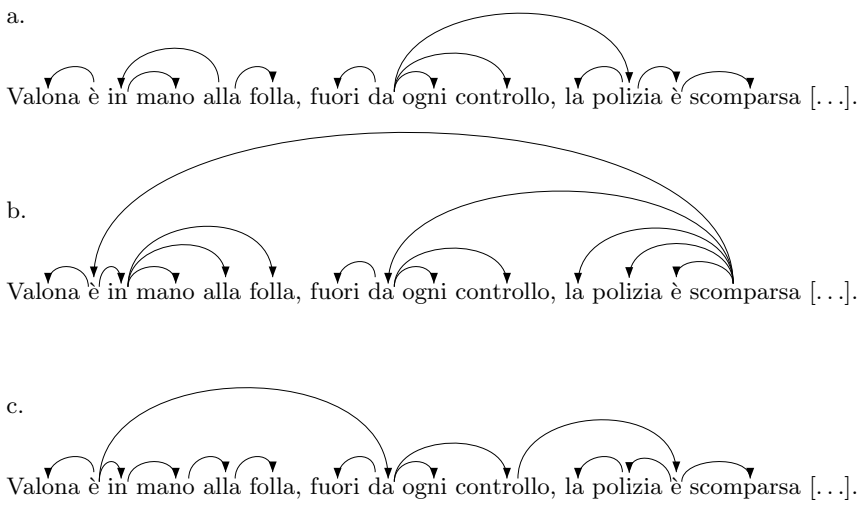


Figure 7.8: Predicted parse trees for Italian example sentence: a. MST (strict trees), b. MST (fallback trees), c. fMST.

the dependent of *è* according to the gold standard annotations (Figure 7.7), but plays the role of the matrix predicate (root node) in the dependency tree suggested by the fallback parser. All of the proposed dependents of *scomparsa* – *la, polizia, da,* as well as both occurrences of *è* – are wrongly identified. The erroneous attachments of *alla, folla* and *fuori* account for the remaining three errors, thus $P_f = R_f = F_f = \frac{14-9}{14} = 35.7\%$.

The output of fMST on this example is shown in Figure 7.8c. It attaches 11 out of the 14 words correctly, thus achieving $P_f = R_f = F_f = \frac{11}{14} = 78.6\%$. Like the two MST parsers, fMST fails at identifying *fuori* as the head of the adverbial modifier phrase *fuori da ogni controllo*, thereby introducing wrong attachments for *fuori* and *da*. The third error can be found in the attachment of *è* in the second conjunct, which should be attached to the first occurrence of *è*.

The outputs of the Malt and fMalt parsers for the Italian example are given in Figure 7.9. There is considerable overlap between the errors made by the (f)MST parsers and their (f)Malt counterparts, and they achieve rather similar scores. The scores for both examples and all parsers are summarized in Table 7.9. For the German example (Table 7.9a.) we also repeat the scores obtained with the direct projections (cf. Figure 7.3).

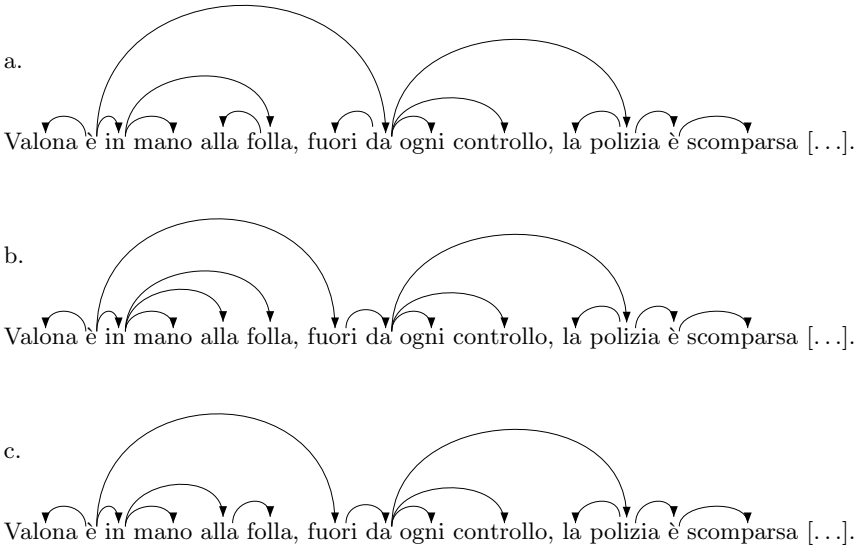


Figure 7.9: Predicted parse trees for Italian example sentence: a. Malt (strict trees), b. Malt (fallback trees), c. fMalt.

a. **German example**

	P_f	R_f	F_f
projected			
trees (strict)	0.0	0.0	0.0
trees (fb.)	0.0	0.0	0.0
frags	85.7	40.0	54.6
predicted			
Malt (strict)	66.7	66.7	66.7
Malt (fb.)	60.0	60.0	60.0
fMalt	85.7	80.0	82.8
MST (strict)	53.3	53.3	53.3
MST (fb.)	66.7	66.7	66.7
fMST	42.9	40.0	41.4

b. **Italian example**

	P_f	R_f	F_f
projected	–	–	–
trees (strict)	–	–	–
trees (fb.)	–	–	–
frags	–	–	–
predicted			
Malt (strict)	57.1	57.1	57.1
Malt (fb.)	71.4	71.4	71.4
fMalt	78.6	78.6	78.6
MST (strict)	57.1	57.1	57.1
MST (fb.)	35.7	35.7	35.7
fMST	78.6	78.6	78.6

Table 7.9: Precision and recall scores achieved by the parsers on the German example (a) and the Italian example (b).

7.5 Summary and Discussion

In this chapter we have provided a detailed error analysis for our parsers, namely by taking into account sentence length, dependency length, and gold standard dependency type.

The analysis by sentence length revealed that the tree-oriented parsers (Malt and MST) tend to perform better than the fragment parsers on short sentences, while the fragment parsers outperform the tree-oriented parsers on longer sentences. We observed a similar pattern for the impact of dependency length: fMalt and to some degree fMST are clearly superior on longer dependencies. Moreover, the fragment parsers often strike a better balance between precision and recall.

Our evaluation relative to gold standard dependency labels did not yield conclusive results. We suspect that this is due to the absence of a label mapping. Since we do not attempt to reconcile the projected labels – that is, the labels posited by the source parser, for the source language dependencies – with the labels that occur in the target language test data, the labels used to group the dependencies for the evaluation by dependency type do not necessarily coincide with a well-defined label (set) in the inventory of the projected parser. Furthermore, cross-lingual non-parallelism can lead to situations during projection where the projected attachment may be correct, but the dependency type from the source language parse is no longer appropriate in the target language. This affects the semantics of the dependency types in the projected annotations and makes a mapping between label inventories even harder. Nevertheless, or for exactly this reason, the induction of a consistent labeling scheme for the target language – be it (weakly) supervised, unsupervised or even devised manually – should bear potential for improvement of the parsers.

Chapter 8

Conclusions

This thesis deals with three broad topics. The first is cross-lingual projection of annotations. Annotation projection is a resource-lean way of attaining automatic annotations for languages with limited resource coverage. The only resources required are a parallel corpus of the source language and the target language, and a tool which can reliably produce the desired annotations for the source language. The annotations in the source language portion of the parallel corpus are then transferred (projected) onto the target language using the word alignment between corresponding words in the bitexts. This word alignment can be obtained automatically with tools such as Giza++ (Och and Ney, 2003).

In this thesis we were concerned with the projection of syntactic dependency relations as produced by off-the-shelf data-driven dependency parsers (Nivre et al., 2006). We used the Europarl parallel corpus of European Parliament proceedings (Koehn, 2005), and in particular considered English and German as alternative source languages, and Dutch, Italian and German as target languages, thus dealing with the language pairs English–Dutch, German–Dutch, English–Italian, German–Italian and English–German.

Three error sources can compromise the quality of the projected target language annotations. First, the source language labeler whose annotations are the basis for projection may introduce annotation mistakes. The source labelers are parsers in our case, and although both source parsers achieve unlabeled attachment scores of 92% (English) and 87% (German) on in-domain test sets, we have also seen that their performance drops somewhat on out-of-domain sentences such as those in the Europarl corpus: the German source parser experiences a loss of 3.5 percentage points (unlabeled). A second source of error is the non-literalness of the translations, brought about by stylistic considerations of the interpreter on the one hand, and true cross-lingual divergencies on the other, making a fully literal word-by-word translation undesirable or even impossible. These divergences from parallelism in turn contribute to the third error source, the automatic word alignment. Especially the intersected variant which is typically used for annotation projection tasks is well known to exhibit high precision at the expense of recall, and while content words are aligned with relative accuracy, the alignment of function words is often erroneous or missing.

All three of these error sources can cause noise in the projected annotations. Moreover, they can lead to the projected annotations being incomplete. In the case of parsing, where the annotations are structured and built of successively

larger subtrees, this problem needs to be addressed if the target language annotations are to constitute the training data for a stand-alone labeler for the target language. A common solution to the issue of noise and gaps in the projected annotations is the definition of more or less aggressive filters (Yarowsky and Ngai, 2001; Hwa et al., 2005, among others). In this thesis we have investigated two such filters, both of which discard incomplete trees. The first, which we call strict projection, considers only those edges for which both the head and the dependent have a strongly (i.e., bidirectionally) aligned equivalent in the target language. The second, constrained fallback projection, further considers weaker (unidirectional) alignment links, but also discards the resulting structure unless it forms a tree. We have shown that the former projection algorithm leads to prohibitive data loss, outputting only very short, simplistic, repetitive sentences. The latter, on the other hand, introduces lots of noise in the output because the unidirectional alignment links are notoriously unreliable. Our preferred solution is therefore a compromise between strict and fallback projection. In what we call partial correspondence projection, we consider only bidirectionally supported alignment links, but do not enforce the completeness constraint. That is, the output of partial correspondence projection is potentially fragmented.

This leads us to the second topic addressed in this dissertation: the training of data-driven dependency parsers on fragmented training data. This problem has been addressed in the past by means of techniques from unsupervised and semi-supervised machine learning (cf. Chapter 2). By contrast, our approach essentially turns the task into an instance of supervised parsing by simply masking the fragmentation from the learning component of the parser. We have shown how two specific dependency parsers can be modified to achieve this: the transition-based Malt parser (Nivre et al., 2006) and the graph-based MST parser (McDonald et al., 2005).

The transition-based parser consists of a transition system which defines the set of legitimate parser actions given the current state of the parser, much like a shift-reduce parser. During training, an oracle reconstructs the correct sequence of transitions for each training example, and each pair of intermediate parser state and subsequent parser action then constitutes a training instance for the data-driven component of the parser: a classifier which determines locally optimal parser actions. Our fragment-enhanced variant of the Malt parser, which we call fMalt, handles fragments as follows. The oracle performs a modified RIGHT-ARC action when it encounters a fragment root. This modified action does not actually add a dependency edge which attaches the fragment, but it is crucial in order to advance the state of the parser and allow it to process the rest of the input. Moreover, the fact that no edge is added for the fragment root also prevents misleading information to leak into the feature representation of the parser state, which may reference the dependency structure built so far. Finally, the modified RIGHT-ARC actions are of course withheld from the machine learning component. The latter is therefore never aware of the fact that the training tree may have been fragmented, but instead learns from whatever informative structure is present.

Unlike the Malt parser, the graph-based MST parser determines the most likely dependency tree for a given sentence by summing over the scores of component edges. These edge scores are adjusted during training by solving margin constraints which separate the gold standard training tree from (a subset of) the suboptimal trees for the sentence which are currently ranked highest. In

order to train such a parser with fragmented training data, our modified variant fMST scores only those edges that are not fragment roots in the training example, and simultaneously relaxes the loss function which triggers the relevant margin constraints so that the deviation of a proposed tree from the training tree does not incur any loss as long as the differences pertain to the attachment of a fragment root.

In our experiments we trained these fragment parsers (fMalt and fMST) on the annotations previously obtained by means of annotation projection. That is, the parsers were faced with highly fragmented data (although constrained in most cases to contain at least 5 words per fragment). Our experimental results show that the fragment parsers perform roughly on a par with the original parsers trained on only conservatively projected, complete trees. A more detailed error analysis further revealed that the fragment parsers are superior on longer sentences as well as on longer dependencies. This is a natural consequence of the more diverse makeup of their training data, but it also confirms that the parsers are indeed capable of exploiting the additional information contained in the fragments.

Finally, the third topic we discussed in this thesis is the evaluation of projection-based systems. This includes the evaluation of the projected annotations directly, as well as the evaluation of the parsers trained on projected annotations. Evaluation against existing gold standards is problematic in two respects. First, the annotations which are projected onto the target sentences from the source language typically do not follow the same guidelines that were used to construct the target language gold standard. This requires a conversion between the two annotation schemes, which is error prone. We argue in this thesis that the most convenient venue for this conversion is the gold standard test set. That is, the gold annotations, which follow the target language annotation scheme, are converted to the source language annotation scheme. The advantage of this conversion direction is that the training and application of the parser (or more generally, labeling tool) are ultimately independent of the existence of designated annotation guidelines for the target language, and the conversion need only be performed when the system is to be evaluated. Furthermore, the venue as such (a gold standard test set) ideally prevents the introduction of annotation errors due to misguided conversion steps.

The second aspect which complicates the evaluation of projection-based systems is the fact that the training data need to be paired with a translation so that projection can be performed in the first place. The standard treebanks from which test sets are taken are usually monolingual. This shortage of parallel test data usually precludes the application of cross-validation. We therefore did not employ cross-validation in this dissertation, but instead trained each of our parser ten (five) times, each time drawing a random sample from the pool of all projected annotations. The arrays of ten results were then used to test for significant differences between the systems, by means of the t-test.

8.1 So, *does* it have to be trees?

Based on the empirical evidence presented in this dissertation, the answer would be: It doesn't hurt. But we can add, based on the same evidence, that parsers trained on tree fragments can achieve the same and sometimes even a higher level

of accuracy than their tree-based counterparts trained on aggressively filtered examples. However, the success of this method hinges on the choice of the source language, the quality of the word alignment, and, as we have shown in Chapter 5, on the (source language) annotation scheme. Note, however, that the same is true for any projected parsers and does not apply to fragment parsers specifically.

Our experiment with artificially fragmented but otherwise gold standard training data (cf. Figure 6.4 on page 105) indicates that fragmentation as such, in an environment with little to no noise, is handled well by our fragment parsers. For instance, even with 50% of all edges removed from the gold standard trees, the fMST parser projected from English to Dutch suffers a decrease of less than 3.5 percentage points in unlabeled attachment score, from 84.1% to 81.7% UAS. These results suggest that fMalt and maybe even more so fMST could be highly useful to process genuinely partial annotations, which brings us to future directions.

8.2 Future Directions

As mentioned above, our fragment parsers could turn out to be a valuable tool for genuinely partial annotations, that is, annotations which are largely noise-free, but where some parts of the dependency structure are left unspecified. This could be useful for problematic or underspecified dependencies, or even for dependencies which are already well represented in the training set, for instance the annotation of nominal and prepositional phrases: given a treebank which contains ample examples of this kind, the annotation of more NPs or PPs will most likely not improve a parser by much; instead, annotators could focus their attention on more fruitful phenomena such as the attachment of the PP, the core structure of coordinations, or the markup of long distance dependencies, etc. Using a parser like fMalt or fMST, the remainder of the sentence could simply remain unanalyzed, and the parser could still benefit from the partial annotations provided. In a similar vein, the parsers could be used for targeted domain adaptation.

Some questions remain of course with respect to the fragment parsers themselves. In this dissertation we considered various fragmentation constraints, that is, constraints on the degree of fragmentation which is acceptable for a dependency graph to be included in the training set for the parser. However, we have certainly not explored all possible kinds of constraints. For a more elaborate tweaking of the fragment size, for instance, one could devise POS-based heuristics which allow missing edges in constellations which are typical for word alignment sparsity (conjunctions, pronouns) or cross-language non-parallelism (prepositions).

Finally, it would also be interesting to conduct a more thorough comparison – both theoretical and empirical – of our fragment parsers with related approaches such as the EM-based fragment treatment of Smith and Eisner (2009) and also with more recent work such as the parsers of Naseem et al. (2010), Søgaard (2011), or McDonald et al. (2011).

Appendix A

Evaluation of Fragmentation Constraints

L_s		$W \geq 1$	$W \geq 4$	$W \geq 5$	$W \geq 6$
en:	$T \leq 2$	–	69.75	**70.04**	**70.24**
	$T \leq 3$	–	**70.18**	**70.23**	**70.16**
	$T \leq 4$	–	–	69.24	69.69
	$T \leq 5$	–	–	–	69.46
	$W/T \geq 2$	68.24	68.43	68.28	68.38
	$W/T \geq 3$	**69.81**	**69.76**	**70.14**	69.47
	$W/T \geq 4$	**70.32**	–	**70.16**	**70.22**
	$W/T \geq 5$	**70.46**	–	–	**70.53**
	MAX ≥ 2	67.42	67.15	67.20	67.42
	MAX ≥ 3	68.13	67.83	67.08	68.17
	MAX ≥ 4	68.66	–	68.11	68.03
	MAX ≥ 5	68.60	–	–	68.39
de:	$T \leq 2$	–	**67.31**	**67.05**	**67.32**
	$T \leq 3$	–	66.94	66.80	67.02
	$T \leq 4$	–	–	66.65	66.56
	$T \leq 5$	–	–	–	66.50
	$W/T \geq 2$	65.64	65.05	65.24	65.31
	$W/T \geq 3$	66.82	66.66	66.44	66.75
	$W/T \geq 4$	67.04	–	66.82	**67.13**
	$W/T \geq 5$	66.69	–	–	**67.68**
	MAX ≥ 2	64.02	64.47	64.07	64.43
	MAX ≥ 3	65.02	64.60	64.53	64.42
	MAX ≥ 4	64.72	–	64.64	64.89
	MAX ≥ 5	65.45	–	–	64.64

Table A.1: Performance of different fragmentation constraints for Dutch fMalt parsers. Boldface indicates equivalence class of significantly best systems ($p < 0.05$).

L_s		$W \geq 1$	$W \geq 4$	$W \geq 5$	$W \geq 6$
en:	$T \leq 2$	–	**64.75**	**64.92**	**64.83**
	$T \leq 3$	–	**64.94**	**64.49**	**64.71**
	$T \leq 4$	–	–	64.20	**64.45**
	$T \leq 5$	–	–	–	64.07
	$W/T \geq 2$	62.47	62.63	62.77	63.29
	$W/T \geq 3$	63.58	63.85	63.96	63.83
	$W/T \geq 4$	63.87	–	64.03	64.07
	$W/T \geq 5$	**64.81**	–	–	**64.56**
	MAX ≥ 2	62.94	62.87	62.70	62.65
	MAX ≥ 3	63.26	62.80	62.42	63.02
	MAX ≥ 4	62.86	–	62.99	62.99
	MAX ≥ 5	63.16	–	–	63.11
de:	$T \leq 2$	–	**53.65**	53.38	**53.75**
	$T \leq 3$	–	53.53	53.46	53.30
	$T \leq 4$	–	–	53.22	**53.84**
	$T \leq 5$	–	–	–	53.31
	$W/T \geq 2$	52.14	52.00	52.15	52.14
	$W/T \geq 3$	53.47	53.28	52.88	52.80
	$W/T \geq 4$	**54.18**	–	**53.78**	53.64
	$W/T \geq 5$	**54.47**	–	–	**54.39**
	MAX ≥ 2	51.00	50.84	50.90	51.14
	MAX ≥ 3	50.76	50.70	50.75	51.54
	MAX ≥ 4	51.20	–	51.69	51.25
	MAX ≥ 5	51.35	–	–	51.60

Table A.2: Performance of different fragmentation constraints for Italian fMalt parsers.

L_s		$W \geq 1$	$W \geq 4$	$W \geq 5$	$W \geq 6$
en:	$T \leq 2$	–	**60.56**	**60.81**	**60.74**
	$T \leq 3$	–	**60.92**	**60.68**	60.56
	$T \leq 4$	–	–	**60.61**	**61.20**
	$T \leq 5$	–	–	–	**60.71**
	$W/T \geq 2$	60.35	59.47	59.89	59.65
	$W/T \geq 3$	60.23	60.13	60.51	60.22
	$W/T \geq 4$	**60.79**	–	**60.57**	**60.70**
	$W/T \geq 5$	**60.68**	–	–	60.56
	MAX ≥ 2	58.63	29.61	55.20	57.36
	MAX ≥ 3	59.68	30.75	30.89	30.70
	MAX ≥ 4	59.50	–	60.02	59.95
	MAX ≥ 5	59.66	–	–	59.30

Table A.3: Performance of different fragmentation constraints for German fMalt parsers.

Appendix B

Analysis by Dependency Length

		F-score				
		root	1	2	3–6	≥7
en-nl	trees	73.95	87.28	75.24	65.75	54.03
	frags	**75.54**	**88.41**	**77.35**	**66.52**	**58.34**
de-nl	trees	**80.14**	**86.92**	**77.81**	**70.85**	57.96
	frags	79.06	86.28	77.60	70.17	**60.93**
en-it	trees	**58.12**	**87.33**	62.80	47.92	38.27
	frags	55.17	87.29	62.80	**51.37**	**41.04**
de-it	trees	59.99	76.39	53.34	44.24	30.79
	frags	**62.14**	**77.09**	**54.17**	**46.99**	**40.65**
en-de	trees	**65.68**	79.11	62.10	**59.81**	**57.41**
	frags	59.92	**79.56**	**62.62**	57.92	55.42

Table B.1: F-score relative to dependency length (Malt).

		Recall					Precision				
		root	1	2	3–6	≥7	root	1	2	3–6	≥7
en-nl	trees	65.53	89.77	80.18	**62.70**	49.63	84.86	84.93	70.88	69.13	59.36
	frags	**67.69**	**91.09**	**81.67**	62.46	**55.72**	**85.46**	**85.89**	**73.46**	**71.15**	**61.25**
de-nl	trees	71.27	**86.90**	79.00	**71.87**	62.31	**91.55**	**86.94**	**76.65**	69.86	54.17
	frags	**71.44**	86.78	**79.16**	70.18	**64.00**	88.50	85.80	76.10	**70.16**	**58.15**
en-it	trees	**56.70**	**86.74**	**65.15**	46.11	42.17	**59.61**	87.94	60.61	49.88	35.06
	frags	55.30	86.64	63.92	**49.44**	**47.35**	55.05	**87.95**	**61.71**	**53.47**	**36.23**
de-it	trees	60.10	74.24	53.87	48.23	30.41	59.88	78.67	52.82	40.85	31.21
	frags	**62.52**	**74.99**	**54.68**	**49.86**	**42.98**	**61.77**	**79.30**	**53.67**	**44.43**	**38.57**
en-de	trees	**72.55**	80.12	59.10	**56.83**	**60.43**	**60.00**	**78.12**	**65.44**	63.14	54.68
	frags	66.98	**82.88**	**60.10**	52.91	54.38	54.21	76.49	65.37	**63.99**	**56.51**

Table B.2: Recall and precision relative to dependency length (Malt)

		F-score				
		root	1	2	3–6	≥7
en-nl	trees	**77.25**	**87.66**	**76.33**	**69.06**	57.37
	frags	75.66	86.89	74.68	68.47	**58.60**
de-nl	trees	**79.15**	**86.30**	**77.20**	**72.14**	**60.80**
	frags	77.64	79.44	70.62	56.48	56.45
en-it	trees	**57.31**	**86.48**	**62.29**	**48.79**	**35.37**
	frags	54.62	84.92	52.50	32.81	24.48
de-it	trees	56.72	**76.32**	**53.26**	**43.54**	34.49
	frags	**62.53**	75.30	46.56	37.73	**37.72**
en-de	trees	**71.44**	80.12	63.07	60.90	58.35
	frags	68.65	**80.30**	**64.18**	**61.38**	**58.78**

Table B.3: F-score relative to dependency length (MST).

		Recall					Precision				
		root	1	2	3–6	≥7	root	1	2	3–6	≥7
en-nl	trees	70.82	**88.72**	**77.05**	67.10	**62.99**	**84.99**	**86.61**	**75.63**	**71.16**	52.69
	frags	**72.14**	87.19	75.90	**68.22**	59.63	79.54	86.60	73.50	68.74	**57.64**
de-nl	trees	**71.52**	86.07	**77.19**	**72.43**	**69.04**	88.62	**86.52**	77.22	**71.86**	54.34
	frags	68.91	**94.20**	62.91	47.14	49.87	**88.90**	68.68	**80.48**	70.46	**65.06**
en-it	trees	**58.63**	84.29	**63.20**	**48.76**	**46.25**	**56.06**	**88.78**	**61.42**	48.82	28.64
	frags	57.32	**93.47**	46.57	23.23	16.13	52.16	77.80	60.18	**55.86**	**50.92**
de-it	trees	60.06	73.34	**49.98**	**45.10**	**49.68**	53.74	**79.54**	**57.02**	42.08	26.42
	frags	**62.91**	**79.91**	42.35	33.31	40.12	**62.15**	71.20	51.70	**43.50**	**35.60**
en-de	trees	79.33	**83.84**	60.38	56.40	**55.31**	**65.00**	76.72	66.02	66.18	61.76
	frags	**82.07**	83.78	**61.88**	**56.63**	52.98	59.00	**77.10**	**66.65**	**67.00**	**66.00**

Table B.4: Recall and precision relative to dependency length (MST).

Bibliography

Alshawi, H., Douglas, S., and Bangalore, S. (2000). Learning dependency translation models as collections of finite-state head transducers. *Computational Linguistics*, 26(1):45–60.

van der Beek, L., Bouma, G., Malouf, R., and van Noord, G. (2002). The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*.

Bentivogli, L. and Pianta, E. (2005). Exploiting parallel texts in the creation of multilingual semantically annotated resources: the MultiSemCor Corpus. *Natural Language Engineering*, 11(3):247–261.

Bergsma, S. and Cherry, C. (2010). Fast and accurate arc filtering for dependency parsing. In *Proceedings of Coling*, pages 53–61, Beijing, China.

Blunsom, P., Cohn, T., and Osborne, M. (2009). Bayesian synchronous grammar induction. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 161–168. MIT Press.

Bohnet, B. (2010). Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of Coling*, pages 89–97, Beijing, China. Coling 2010 Organizing Committee.

Borin, L. (2002). Alignment and Tagging. In Borin, L., editor, *Parallel corpora, parallel worlds. Selected papers from a symposium on parallel and comparable corpora 1999*, pages 207–218, Uppsala University, Sweden. Rodopi, Amsterdam.

Bouma, G., Kuhn, J., Schrader, B., and Spreyer, K. (2008). Parallel LFG Grammars on Parallel Corpora: a Base for Practical Triangulation. In *Proceedings of the LFG Conference*, Sydney, Australia.

Brants, S., Dipper, S., Hansen, S., Lezius, W., and Smith, G. (2002). The TIGER treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, pages 24–41.

Bresnan, J. (2001). *Lexical-Functional Syntax*. Blackwell.

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

Buchholz, S. and Marsi, E. (2006). CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of CoNLL-X*, pages 149–164, New York City.

Burkett, D. and Klein, D. (2008). Two languages are better than one (for syntactic parsing). In *Proceedings of EMNLP*, Honolulu, Hawaii.

Burkett, D., Petrov, S., Blitzer, J., and Klein, D. (2010). Learning better monolingual models with unannotated bilingual text. In *Proceedings of CoNLL*, pages 46–54, Uppsala, Sweden. Association for Computational Linguistics.

Buyko, E. and Hahn, U. (2010). Evaluating the impact of alternative dependency graph encodings on solving event extraction tasks. In *Proceedings of EMNLP*, pages 982–992, MIT, MA.

Carreras, X., Surdeanu, M., and Marquez, L. (2006). Projective dependency parsing with perceptron. In *Proceedings of CoNLL-X*, pages 181–185, New York City, NY.

Catford, J. C. (1965). *A Linguistic Theory of Translation: An Essay in Applied Linguistics*. Oxford University Press, Oxford.

Chang, C.-C. and Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

Clark, S. and Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *Proceedings of ACL*, pages 104–111, Barcelona, Spain.

Clark, S. and Curran, J. R. (2006). Partial training for a lexicalized-grammar parser. In *Proceedings of HLT*, pages 144–151, New York.

Collins, M. (2002). Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, Philadelphia, PA.

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

Crammer, K. and Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Reseach*, 3:951–991.

Cucerzan, S. and Yarowsky, D. (2002). Bootstrapping a multilingual part-of-speech tagger in one person-day. In Roth, D. and van den Bosch, A., editors, *Proceedings of CoNLL*, pages 132–138, Taipei, Taiwan.

Cucerzan, S. and Yarowsky, D. (2003). Minimally supervised induction of grammatical gender. In *Proceedings of HLT-NAACL*, pages 40–47.

Čulo, O., Hansen-Schirra, S., Neumann, S., and Vela, M. (2008). Empirical studies on language contrast using the English-German comparable and parallel CroCo corpus. In *Proceedings of the LREC Workshop on Building and Using Comparable Corpora*, pages 47–51, Marrakech, Morocco.

Cyrus, L. (2006). Building a resource for studying translation shifts. In *Proceedings of LREC*, pages 1240–1245, Genoa, Italy.

Diab, M. and Resnik, P. (2002). An unsupervised method for word sense tagging using parallel corpora. In *Proceedings of ACL*, pages 255–262, Philadelphia, PA.

Dorr, B. J. (1994). Machine Translation Divergences: A Formal Description and Proposed Solution. *Computational Linguistics*, 20(4):597–635.

Drábek, E. F. and Yarowsky, D. (2005). Induction of fine-grained part-of-speech taggers via classifier combination and crosslingual projection. In *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, pages 49–56, Ann Arbor, MI.

Dredze, M., Blitzer, J., Pratim Talukdar, P., Ganchev, K., Graça, J. a., and Pereira, F. (2007). Frustratingly hard domain adaptation for dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 1051–1055.

Dreyer, M., Smith, D. A., and Smith, N. A. (2006). Vine parsing and minimum risk reranking for speed and precision. In *Proceedings of CoNLL-X*, pages 201–205, New York City.

Druck, G., Mann, G., and McCallum, A. (2009). Semi-supervised learning of dependency parsers using generalized expectation criteria. In *Proceedings of ACL-IJCNLP*, pages 360–368, Suntec, Singapore. Association for Computational Linguistics.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71(B):233–240.

Eisner, J. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of Coling*, pages 340–345, Copenhagen, Denmark.

Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In Bunt, H. and Nijholt, A., editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer Academic Publishers.

Eisner, J. (2003). Learning non-isomorphic tree mappings for machine translation. In *Proceedings of ACL, Companion Volume*, pages 205–208, Sapporo, Japan.

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

Eisner, J. and Smith, N. A. (2009). Favor short dependencies: Parsing with soft and hard constraints on dependency length. In Bunt, H., Merlo, P., and Nivre, J., editors, *Trends in Parsing Technology*, chapter 9. Springer.

Fox, H. J. (2002). Phrasal cohesion and statistical machine translation. In *Proceedings of EMNLP*, pages 304–311.

Gale, W. A. and Church, K. W. (1993). A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(1):75–102.

Galley, M., Hopkins, M., Knight, K., and Marcu, D. (2004). What's in a translation rule? In *Proceedings of HLT-NAACL*, pages 273–280, Boston, MA.

Ganchev, K., Gillenwater, J., and Taskar, B. (2009). Dependency grammar induction via bitext projection constraints. In *Proceedings of ACL-IJCNLP*, pages 369–377, Suntec, Singapore. Association for Computational Linguistics.

Gildea, D. (2003). Loosely tree-based alignment for machine translation. In *Proceedings of ACL*, pages 80–87, Sapporo, Japan.

Graça, J. a., Ganchev, K., and Taskar, B. (2008). Expectation maximization and posterior constraints. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA.

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., Straňák, P., Surdeanu, M., Xue, N., and Zhang, Y. (2009). The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of CoNLL (Shared Task)*, pages 1–18, Boulder, CO. Association for Computational Linguistics.

Hwa, R. (1999). Supervised grammar induction using training data with limited constituent information. In *Proceedings of ACL*, pages 73–79, College Park, MD.

Hwa, R., Resnik, P., Weinberg, A., Cabezas, C., and Kolak, O. (2005). Bootstrapping parsers via syntactic projection across parallel texts. *Natural Language Engineering*, 11(3):311–325.

Hwa, R., Resnik, P., Weinberg, A., and Kolak, O. (2002). Evaluating Translational Correspondence using Annotation Projection. In *Proceedings of ACL*, Philadelphia, PA.

Jähnig, G. and Marienfeld, F. (2010). Word tokenization, sentence splitting, sentence alignment and word alignment of europarl. Technical report, University of Potsdam.

Jansche, M. (2005). Treebank transfer. In *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT)*, pages 74–82, Vancouver, BC. Association for Computational Linguistics.

Jiang, W. and Liu, Q. (2009). Automatic adaptation of annotation standards for dependency parsing – using projected treebank as source corpus. In *Proceedings of IWPT*, pages 25–28, Paris, France. Association for Computational Linguistics.

Johansson, R. and Nugues, P. (2007). Extended constituent-to-dependency conversion for English. In Nivre, J., Kaalep, H.-J., and Koit, M., editors, *Proceedings of NODALIDA*, pages 105–112.

Joshi, A. K. (1985). *Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?*, chapter 6, pages 206–250. Natural Language Parsing. Cambridge University Press, Cambridge.

Kameyama, M., Ochitani, R., and Peters, S. (1991). Resolving translation mismatches with information flow. In *Proceedings of ACL*, pages 193–200, Berkeley, CA. Association for Computational Linguistics.

Kim, S., Jeong, M., Lee, J., and Lee, G. G. (2010). A cross-lingual annotation projection approach for relation detection. In *Proceedings of Coling*, pages 564–571, Beijing, China.

Klein, D. and Manning, C. D. (2002). A generative constituent-context model for improved grammar induction. In *Proceedings of ACL*, pages 128–135.

Klein, D. and Manning, C. D. (2004). Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of ACL*, pages 478–485, Barcelona, Spain.

Koehn, P. (2005). Europarl: A Parallel Corpus for Statistical Machine Translation. In *Proceedings of the MT Summit*.

Koo, T., Carreras, X., and Collins, M. (2008). Simple semi-supervised dependency parsing. In *Proceedings of ACL-HLT)*, pages 595–603, Columbus, Ohio.

Kruijff, G.-J. M. (2002). Formal and computational aspects of dependency grammar: History and development of dg. Technical report, ESSLLI-2002.

Kübler, S., Maier, W., Rehbein, I., and Versley, Y. (2008). How to Compare Treebanks. In *Proceedings of LREC*, pages 2322–2329.

Kübler, S., McDonald, R., and Nivre, J. (2009). *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool.

Kuhn, J. (2004). Experiments in parallel-text based grammar induction. In *Proceedings of ACL*, pages 470–477.

Lafferty, J., McCullum, A., and Pereira, F. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, San Francisco, CA. Morgan Kaufmann.

Leuven-Zwart, K. M. (1989). Translation and original: Similarities and dissimilarities, i. *Target*, 1(2):151–181.

Magerman, D. M. (1994). *Natural language parsing as statistical pattern recognition*. PhD thesis, University of Pennsylvania, Philadelphia.

Magnini, B., Cappelli, A., Tamburini, F., Bosco, C., Mazzei, A., Lombardo, V., Bertagna, F., Calzolari, N., Toral, A., Lenzi, V. B., Sprugnoli, R., and Speranza, M. (2008). Evaluation of natural language tools for Italian: EVALITA 2007. In *Proceedings of LREC*, Marrakesh, Morocco.

Mann, G. S. and McCallum, A. (2008). Generalized expectation criteria for semi-supervised learning of conditional random fields. In *Proceedings of ACL-HLT*, pages 870–878, Columbus, OH.

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

Marneffe, M.-C. d., MacCartney, B., and Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, pages 105–112, Genoa, Italy.

McClosky, D., Charniak, E., and Johnson, M. (2006). Effective self-training for parsing. In *Proceedings of HLT-NAACL*, pages 152–159, New York.

McDonald, R., Lerman, K., and Pereira, F. (2006). Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of CoNLL-X*.

McDonald, R. and Nivre, J. (2007). Characterizing the errors of data-driven dependency parsing models. In *Proceedings of EMNLP-CoNLL*, pages 122–131.

McDonald, R. and Nivre, J. (2011). Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230.

McDonald, R. and Pereira, F. (2006). Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, pages 81–88, Trento, Italy.

McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT-EMNLP*, pages 523–530, Vancouver, BC.

McDonald, R., Petrov, S., and Hall, K. (2011). Multi-source transfer of delexicalized dependency parsers. In *Proceedings of EMNLP*, pages 62–72, Edinburgh, Scotland.

Melamed, D. I. (1998a). Annotation style guide for the blinker project. Technical Report 98-06, IRCS.

Melamed, D. I. (1998b). Manual annotation of translational equivalence: The blinker project. Technical Report 98-07, IRCS.

Melamed, D. I. (2003). Multitext grammars and synchronous parsers. In *Proceedings of HTL-NAACL*, pages 79–86, Edmonton.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press.

Merlo, P., Stevenson, S., Tsang, V., and Allaria, G. (2002). A Multilingual Paradigm for Automatic Verb Classification. In *Proceedings of ACL*, pages 207–214, Philadelphia, PA.

Moon, T. and Baldridge, J. (2007). Part-of-speech tagging for middle English through alignment and projection of parallel diachronic texts. In *Proceedings of EMNLP-CoNLL*, pages 390–399, Prague, Czech Republic. Association for Computational Linguistics.

Naseem, T., Chen, H., Barzilay, R., and Johnson, M. (2010). Using universal linguistic knowledge to guide grammar induction. In *Proceedings of EMNLP*, pages 1234–1244, MIT, MA.

Nivre, J. (2006). *Inductive Dependency Parsing*, volume 34 of *Text, Speech and Language Technology*. Springer.

Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., and Yuret, D. (2007). The CoNLL 2007 shared task on dependency parsing. In *Proceedings of EMNLP-CoNLL*, pages 915–932, Prague, Czech Republic.

Nivre, J., Hall, J., Nilsson, J., Eryiğit, G., and Marinov, S. (2006). Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of CoNLL-X*, pages 221–225.

Nivre, J. and McDonald, R. (2008). Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-HLT*, pages 950–958, Columbus, Ohio.

Nivre, J. and Nilsson, J. (2005). Pseudo-projective dependency parsing. In *Proceedings of ACL*, pages 99–106.

van Noord, G. (2006). At last parsing is now operational. In Mertens, P., Fairon, C., Dister, A., and Watrin, P., editors, *TALN06. Verbum Ex Machina. Actes de la 13e conference sur le traitement automatique des langues naturelles*, pages 20–42.

Och, F. J. and Ney, H. (2000). Improved statistical alignment models. In *Proceedings of ACL*, pages 440–447.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Øvrelid, L., Kuhn, J., and Spreyer, K. (2009). Cross-framework parser stacking for data-driven dependency parsing. *Traitement Automatique des Langues (TAL) Special Issue on Machine Learning for NLP*, 50(3):109–138.

Ozdowska, S. (2006). Projecting POS tags and syntactic dependencies from English and French to Polish in aligned corpora. In *Proceedings of the EACL Workshop on Cross-Language Knowledge Induction*, pages 53–60, Trento, Italy.

Padó, S. (2007). *Cross-Lingual Annotation Projection Models for Role-Semantic Information*. PhD thesis, Saarland University.

Padó, S. and Erk, K. (2010). Translation shifts and frame-semantic mismatches: A corpus analysis. *International Journal of Corpus Linguistics*. To appear.

Padó, S. and Lapata, M. (2005a). Cross-lingual Bootstrapping for Semantic Lexicons: The Case of FrameNet. In *Proceedings of AAAI*, pages 1087–1092, Pittsburgh, PA.

Padó, S. and Lapata, M. (2005b). Cross-lingual projection of role-semantic information. In *Proceedings of HLT-EMNLP*, Vancouver, BC.

Padó, S. and Lapata, M. (2006). Optimal constituent alignment with edge covers for semantic projection. In *Proceedings of Coling-ACL*, Sydney, Australia.

Pereira, F. and Schabes, Y. (1992). Inside-outside reestimation from partially bracketed corpora. In *Proceedings of ACL*, pages 128–135.

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase-Structure Grammar*. University of Chicago Press, Chicago.

Probst, K. (2003). Using 'smart' bilingual projection to feature-tag a monolingual dictionary. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL*, pages 103–110, Edmonton, Canada.

Riloff, E., Schafer, C., and Yarowsky, D. (2002). Inducing Information Extraction Systems for New Languages via Cross-language Projection. In *Proceedings of Coling*.

Sagae, K. and Tsujii, J. (2007). Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 1044–1050, Prague, Czech Republic.

Saquete, E., Martínez-Barco, P., Muñoz, R., Negri, M., Speranza, M., and Sprugnoli, R. (2006). Multilingual extension of a temporal expression normalizer using annotated corpora. In *Proceedings of the EACL Workshop on Cross-Language Knowledge Induction*, pages 1–8, Trento, Italy.

Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *International Conference on New Methods in Language Processing*, pages 44–49, Manchester, England.

Seeker, W., Bohnet, B., Øvrelid, L., and Kuhn, J. (2010). Informed ways of improving data-driven dependency parsing for German. In *Proceedings of Coling (Poster volume)*, pages 1122–1130, Beijing, China.

Shieber, S. M. and Schabes, Y. (1990). Synchronous tree-adjoining grammars. In *Proceedings of ACL*, pages 253–258.

Smith, D. A. and Eisner, J. (2006). Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 23–30, New York City. Association for Computational Linguistics.

Smith, D. A. and Eisner, J. (2007). Bootstrapping feature-rich dependency parsers with entropic priors. In *Proceedings of EMNLP-CoNLL*, pages 667–677, Prague.

Smith, D. A. and Eisner, J. (2008). Dependency parsing by belief propagation. In *Proceedings of EMNLP*, pages 145–156, Honolulu, Hawaii.

Smith, D. A. and Eisner, J. (2009). Parser adaptation and projection with quasi-synchronous grammar features. In *Proceedings of EMNLP*, pages 822–831.

Smith, D. A. and Smith, N. A. (2004). Bilingual parsing with factored estimation: Using english to parse korean. In *Proceedings of EMNLP*, pages 49–56.

Smith, N. A. and Eisner, J. (2004). Annealing techniques for unsupervised statistical language learning. In *Proceedings of ACL*, pages 487–494, Barcelona.

Smith, N. A. and Eisner, J. (2005). Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of ACL*, pages 354–362, Ann Arbor, MI.

Snyder, B., Naseem, T., and Barzilay, R. (2009). Unsupervised multilingual grammar induction. In *Proceedings of ACL-IJCNLP*, pages 73–81, Suntec, Singapore. Association for Computational Linguistics.

Søgaard, A. (2011). Data point selection for cross-language adaptation of dependency parsers. In *Proceedings of ACL-HLT*, pages 682–686, Portland, OR.

Søgaard, A. and Rishøj, C. (2010). Semi-supervised dependency parsing using generalized tri-training. In *Proceedings of Coling*, pages 1065–1073, Beijing, China.

Spreyer, K. (2010). Notes on the evaluation of dependency parsers obtained through cross-lingual projection. In *Proceedings of Coling (Poster volume)*, pages 1176–1184, Beijing, China. Coling 2010 Organizing Committee.

Spreyer, K. and Frank, A. (2008). Projection-based acquisition of a temporal labeller. In *Proceedings of IJCNLP*, Hyderabad, India.

Spreyer, K. and Kuhn, J. (2009). Data-driven dependency parsing of new languages using incomplete and noisy training data. In *Proceedings of CoNLL*, pages 12–20, Boulder, CO.

Spreyer, K., Øvrelid, L., and Kuhn, J. (2010). Training parsers on partial trees: A cross-language comparison. In ELRA, editor, *Proceedings of LREC*.

Steinberger, R., Pouliquen, B., Widiger, A., Ignat, C., Erjavec, T., Tufiş, D., and Varga, D. (2006). The JRC-acquis: A multilingual aligned parallel corpus with 20+ languages. In *Proceedings of LREC*, Genoa, Italy.

Surdeanu, M., Johansson, R., Meyers, A., Màrquez, L., and Nivre, J. (2008). The CoNLL 2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of CoNLL*, pages 159–177, Manchester, UK.

Tesnière, L. (1959). *Eléments de syntaxe structurale*. Editions Klincksieck.

Tsuboi, Y., Kashima, H., Mori, S., Oda, H., and Matsumoto, Y. (2008). Training conditional random fields using incomplete annotations. In *Proceedings of Coling*, pages 897–904, Manchester, UK. Coling 2008 Organizing Committee.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer.

Vinay, J.-P. and Darbelnet, J. (1958). *Stylistique Comparée du Français et de l'Anglais: Méthode de Traduction*. Didier, Paris.

Vogel, S., Ney, H., and Tillmann, C. (1996). HMM-based word alignment in statistical translation. In *Proceedings of Coling*, pages 836–841, Copenhagen, Denmark.

Wang, Q. I., Schuurmans, D., and Lin, D. (2008). Semi-supervised convex training for dependency parsing. In *Proceedings of ACL-HLT*, pages 532–540, Columbus, Ohio.

Wróblewska, A. and Frank, A. (2009). Cross-lingual projection of LFG f-structures: Building an f-structure bank for polish. In Passarotti, M., Przepiórkowski, A., Raynaud, S., and Eynde, F., editors, *Proceedings of the Eighth International Workshop on Treebanks and Linguistic Theories (TLT)*, pages 209–220, Milan, Italy.

Wu, D. (1997). Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora. *Computational Linguistics*, 23(3):377–404.

Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In van Noord, G., editor, *Proceedings of IWPT*, pages 195–206.

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In *Proceedings of ACL*, pages 523–530, Toulouse, France. Association for Computational Linguistics.

Yarowsky, D. and Ngai, G. (2001). Inducing Multilingual POS Taggers and NP Bracketers via Robust Projection across Aligned Corpora. In *Proceedings of NAACL*, pages 200–207.

Yarowsky, D., Ngai, G., and Wicentowski, R. (2001). Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of HLT*.

Zhang, Y. and Clark, S. (2008). A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of EMNLP*, pages 562–571, Honolulu, Hawaii.

Zitouni, I. and Florian, R. (2008). Mention detection crossing the language barrier. In *Proceedings of EMNLP*, pages 600–609, Honolulu, Hawaii.

Zollmann, A., Venugopal, A., Vogel, S., and Waibel, A. (2006). The CMU-UKA Syntax Augmented Machine Translation System for IWSLT-06. In *Proceedings of the International Workshop on Spoken Language Translation*, Kyoto, Japan.