Improvements to Syntax-based Machine Translation using Ensemble Dependency Parsers

Nathan David Green

Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Charles University in Prague

Supervisor: doc. Ing. Zdeněk Žabokrtský, Ph.D.

A thesis submitted for the degree of

Doctor of Philosophy

2013 September

Abstract

Dependency parsing is an integral part of Natural Language Processing (NLP) research for many languages. Research in dependency parsing has mainly dealt with improving accuracy for a limited number of languages. Current dependency parsing algorithms have developed mainly for languages with an ample amount of training data. Most of this data has been collected for shared tasks at conferences and are available mainly for European and resource-rich languages. New researchers into the area may not know which algorithm and techniques work best with a new, untested, language.

To address this issue, we will look at ensemble approaches to dependency parsing. More specifically, we look at three methods. First, stacking parsers' outputs into a weighted graph and extracting a tree structure using simple voting. Second, analyzing each parsers' errors distribution and using that as an input into the weighted graph through fuzzy clustering methods. Third, using a meta-classifier to choose the best parser for each and every word in our input. The parsers in each situation may come from a variety of techniques such as graph-based, transition-based, and constituent conversion. Using a variety of parsers allows us to study the errors associated with the parsers and choose the best combination or individual parser for each situation.

Even though many tools exist for these European and resource-rich languages, dependency parsing techniques are most commonly only tested using accuracy scores, both unlabeled and labeled. If a new technique is developed for a high accuracy such as English or Japanese, the results are often equivalent to existing techniques or sometimes

worse. Due to this, research is often only concerned with a very specific linguistic construction, domain, or localized feature. This often leads to a scenario, where one size does not fit all, particularly for under-resourced languages.

To make sure our techniques are useful for most languages, we analyzed them on large and small language data sets from a variety of language families. We want to give special attention to under-resourced languages, so we additionally show techniques on semi-supervised training via self-training. For under-resourced languages, self-training can be an important tool both for parser accuracy and for creating new annotated data. When using ensemble parsers, a fundamental self-training question arises on whether the individual parsers should be retrained on their own data or on ensemble data. Whether under-resourced or resource-rich, we feel that limiting the analysis to accuracy scores does not fully determine whether a technique is useful or not. To test our techniques down a typical NLP pipeline, we turn to machine translation.

Machine translation is often the first task people want solved for their language but often the last step in the process. Many components go into a successful system. These systems come in a variety of forms, whether rule-based or statistically based. One concern for machine translation is whether the early components of the pipeline are accurate. A 2% error in part-of-speech tagging may lead to a much higher percentage of parsing errors which in turn ends up in a double figure error rate in the final translation. Reducing the errors in early pipeline components is a prime concern so that researchers in machine translation can focus on the actual translation and not generalize earlier errors.

To examine the effects of dependency parsing down the NLP pipeline. Our dependency models will be evaluated using the Treex system and TectoMT translation system. This system, as opposed to other popular machine translation systems, makes direct use of the dependency

structure during the conversion from source to target languages via a tectogrammatical tree translation approach. We will compare UAS accuracy to corresponding NIST and BLEU scores from the start to finish of the machine translation pipeline.

Unfortunately any current approach to test dependency parsing's effect on machine translation is going to run into one major road block. There is no gold data for English dependency trees that has a corresponding gold standard translation. For the vast majority of English dependency parsers, the status quo is to train with data automatically converted from constituent trees. This leads to a final parse with at least an 8% error rate in UAS. This is too high of a rate to truly test the dependency's effect on the final output of the NLP pipeline. To address this issue we have hand annotated dependency trees for the WMT 2012 data set, commonly used to judge machine translation systems. Additionally, to improve future parser training and constituent conversions, we have hand corrected the dependency trees in one section of the Penn Treebank.

Within this dissertation, we aim to show both improvements to dependency parsing using ensemble methods for a variety of languages including under-resourced and resource-rich and show how these new dependency parsers effect the overall result in a machine translation pipeline. In addition to these results, we have developed new gold standard dependency trees for the purpose of machine translation. We have also determined an improved standard for constituent conversions through empirical means discovered from manual annotation of a part of the Penn Treebank.

Declaration

Acknowledgments

I would like to thank my supervisor and mentor during my time at Charles University, Zdeněk Žabokrtský, for keeping me on track and not letting any time go to waste during the dissertation. I would also like to thank the CLARA program for allowing me to come to Europe and study at Charles University and to my colleagues in the CLARA program for their support and advice. The number of experiments conducted in the time I was at Charles University would not of been possible with out the excellent work from the contributers to the Treex system, especially Zdeněk Žabokrtský, Martin Popel, and David Mareček. A special thanks to Loganathan Ramasamy and Septina Larasati for their help with the Tamil and Indonesian data sets.

I wouldn't be here without the support and love of my parents. Their support has always been needed and has always been given. I am especially grateful for their encouragement in academia throughout my life, starting early in life by giving me candy for good grades. Needless to say, I ate plenty of candy.

And last but not least, Tina, for making every day better than the last through love, friendship and unending support, even with increasing stress and pressure towards the end of writing this dissertation.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Improve Current Tools with Existing Data and Annotation

Problem: The focus of much of dependency parsing is on creating new modeling techniques and examining new feature sets for existing dependency models. Often these new models are lucky to achieve equivalent results with the current state-of-the-art results and often perform worse. These approaches are for languages that are often resource-rich and have ample training data available for dependency parsing. For this reason, the accuracy scores are often quite high. This, by its very nature, makes it quite difficult to create a significantly large increase in the current state-of-the-art. Research in this area is often concerned with small accuracy changes or very specific localized changes, such as increasing accuracy of a particular linguistic construction. With so many modeling techniques available to languages with large resources the problem exists on how to exploit the current techniques with the use of ensemble techniques along with this plethora of data.

With advancements in text and data mining, many languages in the world fall into a middle ground where they have some monolingual data resources but not a large amount of other types data. Often they will have little parallel data and often neither resource will have many linguistic annotations. For these languages, we can expect a small amount of annotation or a small treebank to be constructed by interested researchers. However, the techniques that are applied for resource-rich languages might not be applicable to these languages. New techniques need to be applied to get the maximal gain from the resources that do exist, while exploiting unannotated resources as well.

Research Questions: There are many questions to be examined when looking at state-of-the-art models and resource-rich languages. Models have similar accuracy but are they significantly different in their approach so that their construction of both correct and incorrect parse structures supply useful knowledge to an ensemble parse structure? Can the differences in parsers be exploited in both an ensemble system and in a discrete classification system? Not only can we determine the usefulness of a particular model, but can these differences be combined or selected in a fashion that still allows a system to construct a legitimate and logical dependency parse. Even with a logical parse, many systems falter when the domain of text is changed. Examination of whether ensemble systems can be weighted and constructed to aid in domain adaptation has yet to be fully researched.

With many languages in the world and few dependency treebanks, supervised techniques are not an optimal solution for the majority of languages. For many languages it is reasonable to assume enough labor or funds to create a very small treebank, but where do we go from there? Semi-supervised techniques have become increasingly popular due to the dichotomy of having very little information in a vast world of data, primarily due to the growth of the Internet in a variety of languages. Can lightly trained models use unannotated data in an ensemble way to help bootstrap parsers for new languages? Can using more than handful of parsers boost performance by giving a complementary view of the data? Given a seemingly unlimited amount of unannotated data that is growing daily, can self-training be used within an ensemble system?

Research Approach: To examine whether parser outputs can be combined in an effective manor, we look at creating one ensemble parse from any N amount of parsers, whether they are constituent-based, transition-based, or graph-based parsers. Looking at different combinations of these parser types will allow us to see how they differ both structurally and as well the differences in their error types. Given that the part-of-speech (POS) error distribution differs in each dependency technique, the ensemble weights can be learned from these distributions. To do this we will use fuzzy clustering of POS errors per dependency model to obtain our ensemble weights. While ensemble systems improve the overall performance through a combination of approaches, we examine further whether a classifier can determine the appropriate model to use on a per token level. To do this we implement an SVM classifier.

To address issues around under-resourced languages we examine the use of semi-supervised training methods with both dependency parsers and with our ensemble dependency parsers.

Contributions:

- We show that combining dependency parsers of different techniques in an ensemble framework, in particular constituency to dependency converted and traditional dependency techniques, leads to improved UAS for English dependency parsing.

- The same ensemble framework can be used for non-English languages for a similar improvement in most situations. These languages might lack the constituency to dependency conversion so in some cases the ensemble system needs to be augmented with additional models of the same dependency techniques.

- The ensemble framework not only improves UAS scores but also improves the overall dependency error rate for each individual part-of-speech

- Part-of-speech error distribution can successfully be used via fuzzy clustering to learn the weights of an ensemble system, leading to greater UAS scores and reduced dependency error rate for each part-of-speech.

- An SVM classifier can improve dependency parsing using model agreement features.
- Self-training is perhaps the most frequent use of semi-supervised training and one of the most successful. While this has been examined for parsing, the effects of using an ensemble parser or many models combined from different

Impact:
The benefit of these techniques do not simply stop with dependency parsing. A dependency parse tree is often an input into other natural language processes. While under-resources languages may not be examining an entire NLP pipeline, resource-rich languages that take advantage of these approaches should additionally measure success in applications further down the NLP pipeline. For instance, when UAS accuracy is reduced with a new parser, can machine translation accuracy be improved with statistical significance?

Semi-supervised training like previous techniques, can be used in the NLP pipeline. However when discussing under-resourced languages the issue becomes a little more cloudy. First, there may not be the resources available for the rest of the NLP pipeline. Parsing is part of the founding steps in a BLARK (Basic Language Resource Kit) (27). Second, for an under-resourced language the main goal is often to get more annotated resources. Semi-supervised parsing can be used to this effect as semi-supervised annotation. Every percentage point of UAS improvement is one more percentage point that a human annotator does not have to correct. For both reasons, semi-supervised parsing for under-resourced languages is a worthy goal and will have impact on the under-resourced community.

1.2 Dependency Parsing Down the Pipeline

Dependency parsing is typically evaluated by its labeled and unlabeled accuracy scores for a particular parse tree. The models are rarely used to evaluate performance on other NLP tasks. Although rarely evaluated, they are often used, as parsing is a main component of many NLP tasks. We will evaluate our ensemble models on one particular task, machine translation.

Problem: Dependency parsers are almost ubiquitously evaluated on their accuracy scores, these scores say nothing of the complexity and usefulness of the resulting structures. The structures may have more complexity due to the depth of their coordination or noun phrases. As dependency parses are a basic structures in which other systems are built upon, it would seem more reasonable to judge these parsers down the NLP pipeline. The types of parsing errors that cause significant problems in other NLP applications is currently an unknown.

Research Questions:
While a coordination error and a modifier error are treated as equal in severity in unlabeled accuracy scores, the effect on NLP systems may be greater. Two basic questions on annotation structures occur: First, can we minimize the types of errors that are most egregious when propagated to other NLP systems. Second, is there any relation between UAS accuracy and performance found in a translation system? While we have shown improvement with ensemble systems, the question still remains whether this has an overall positive increase in a machine translation system.

Research Approach:
To examine the effects of dependency parsing down the NLP pipeline, we now turn to a specific machine translation system. Our dependency models will be evaluated using the Treex system and TectoMT translation system (60). This system, as opposed to other popular machine translation systems, makes direct use of the dependency structure during the conversion from source to target languages via an interlingua-like approach with tectogrammatics. We will compare UAS accuracy to corresponding BLEU and NIST scores from the start to finish of the machine translation pipeline. Alternatively we examine the number of changes an average sentence goes through when our new ensemble models are used as opposed to a single dependency model from one technique.

To examine these behaviors we must first have gold standard data. To do this we manually annotated two sets of data. First we annotated gold translation data so that we have an overlap between the data used for dependency evaluation and data used to judge machine translation. Second we corrected dependency annotations on one section of Penn Treebank which is typically used to train dependency parsers. Using the annotated Penn Treebank section, we are able to determine the best conversion procedure to get near gold data to train our parsers with. Using these new parsers, we are able to evaluate the effect of dependencies more thoroughly.

Contributions:

- The first analysis of dependencies errors' effects on Machine Translation in TectoMT
- Results of ensemble dependency parsers when used in the machine translation pipeline
- Create the first gold standard data that ties both gold dependency structures with gold translation
- Empirically discover the most appropriate conversion methods from constituency to our new gold dependency structures
- Show the correlation between dependency structure and NIST and BLEU scores
- Results of new dependency models trained on gold data and gold converted data

Impact: Often new annotation or new parsing techniques are tried and abandoned if they do not give an immediate boost to UAS. It is our goal and the hopeful impact that parsing models will be evaluated on NLP systems other than just parsing. Increasing UAS scores is more equated to a learning problem while increasing the results of an NLP system are the result of adding additional detail and information to your early level NLP structures. The latter we find to be a more convincing argument for research.

A primary driver of much NLP research is the access to interesting and new training data. With the release of our dependency annotations on machine translation data, new work can be done on gaining more value out of dependency parsers and tools in a full NLP pipeline.

1.3 Structure of the Dissertation

Before delving into ensemble parsing and machine translation, we will first cover the basic terminology and background material in Chapter 2. Ensemble parsing will be covered in Chapter 3 along with semi-supervised techniques for under-resourced languages in Section 3.3. Using these parsing techniques, we will examine their overall effects on machine translation in Chapter 4. After the initial results, we will look at how we annotated gold dependency data for machine translation in Section 4.2. Following that, we will see how the changes correspond to NIST and BLEU scores. Chapter 5 will conclude.

2

Background and Terminology

2.1 The Machine Learning Cycle

Machine learning is ubiquitous in our everyday life. We observe machine learning in many ways, for example, every time shoppers purchase an item with their club discount cards, product data is collected to help a machine learn the shoppers' purchase habits. A machine trained on this information may then be able to predict which items should be located near each other in the store or which ad promotions a particular shopper may respond to. Machine learning is often a task dealing with images; for example, when passengers walk through security in an airport they may be being screened via facial recognition software. This software has most likely been trained on a large number of predetermined faces so that the software can recognize eyes, noses, and other facial parts. Perhaps, the largest section of machine learning deals with text analysis. This textual learning may be machine translation, email spam filtering, or named entity recognition. Advancements in machine learning affect our lives whether we are aware of its presence or not.

However, advancements in machine learning are largely hampered by the lack of data on which the machine can be trained to learn patterns of interest. This data in machine learning is commonly called training data. An example of training data may include sentences with tags assigned to words and word phrases relating to part-of-speech (POS) (35) or the training data may be sentence aligned translations, also referred to as parallel corpora (50).

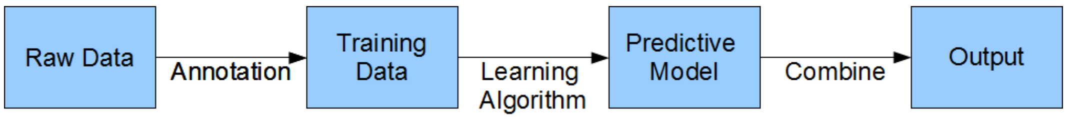


Figure 2.1: Overview of 1 iteration of a machine learning cycle

2.1.1 Typical Iterative Process Flow

Looking at machine learning from a bird-eye view will help us get some of the basic terms in order. The basic view can be seen in Figure 2.1. We start with raw data, generally speaking this data is not useful for higher end tools. Commonly raw data can be used to learn lexicons, spelling errors, and language models. To add information to this data we need to impose structure, more commonly referred to as annotations. These annotations may take many forms depending on the type of media, it may be ontological categorization, textual part-of-speech, or audio transcription. We focus particularly on syntactic annotation in the form of dependency relationships, later in the dissertation.

Once structure/annotation has been added, we define the data as “training data”. We use this training data as an input into machine learning algorithms which will learn patterns in the annotation and data. The goal of this process is to create a model which will be able to add structure/annotations to new unseen data without human interaction. Once we predict information about unseen data, we combine those predictions into a final output. If our annotation is part-of-speech, we may predict a POS for each word; if our annotation is ontologically based, we may predict a genre or class for the text, etc. Typically this process ends with the output stage. The output is then judged by a human or by some automatic metric. Although common, this is not always the case. A second option is available in which the training data is used to make several different models using different learning algorithms. Figure 2.2 demonstrates this process from a birds eye view.

2.1.1.1 Supervised

Taking Figure 2.1 as described is an example of supervised learning. This means that we know the annotation labels and structure ahead of time. This allows our output to use the same labels as the input. On the downside, this does not allow the system to generalize outside of its current knowledge. In particular, for supervised machine learning, we will be making use of Support Vector Machines (SVM). SVMs are a linear classifier that maximize the area between two sets of data points. SVMs can use hyperplanes for mapping to higher dimensional spaces as well (6).

2.1.1.2 Unsupervised

Alternatively, instead of having hand labeled annotations, there are a set of learning algorithms that are unsupervised. This means the labels are not known, so the annotation stage is skipped and the raw data is used for training. One instance of this is clustering. The output of the system gives each input a label, this label is normally not understandable though. This is simply a grouping in which someone must later look at to determine what it means. In almost all situations unsupervised learning performs worse than supervised. Even with lower performance, it is often an enticing option since under-resourced languages can use these techniques without manual annotation, which can be quite expensive.

2.1.1.3 Semi-supervised

Semi-supervised learning is a combination of supervised and unsupervised. It assumes some annotation has been done, but normally not as many as a typical supervised system. A model is trained off of these annotations and applied to unseen data. The results of the model can then be used to retrain the model. Additionally, data does not have to be used as direct training data. The additional data is often used to tune the parameters of existing supervised models, essentially being a form of domain adaptation. Since supervised performs better than unsupervised in most situations, the term semi-supervised might be misleading in its performance. Typically it does not perform in the middle of the two, but actually performs better than the supervised system. This is typically due to extra unannotated data used for parameter tuning or because using its own output allows for better domain adaptation through self-training.

2.1.1.4 Ensemble

We use ensemble methods for stacking and combining dependency parsers throughout this dissertation. The idea behind this is to take the best features of each predictive model and combine them with other models. For instance, one parser may perform well on a particular part-of-speech, while another does not. We would want to, on a node by node level, decide which parser is most likely correct. This may be for any number of factors including part-of-speech, sentence length, arc length, genre, n-grams, or one what we will use regularly in this document, model agreement features. For instance, if we make our decision based on part-of-speech as in Table 2.1, assume parser A predicts dependencies at 60% for Nouns (NN) and 30% for Verbs (VB) and vice versa for parser B, the optimal goal for an ensemble system would be to predict 60% across the board.

Model	POS	Accuracy
A	NN	60%
A	VB	30%
B	NN	30%
B	VB	60%

Table 2.1: Hypothetical ensemble scenario

The obvious drawback of this method is in the processing time. If not parallel, we increase our processing time linearly for each of the N parsers. So having an ensemble parser with 5 parsers will logically take much longer, so these methods are generally not a good idea for real time, consumer based systems. However, if accuracy is the most important, as it is in semi-supervised annotation, the extra processing time may be worth the effort. If time is an issue, closer attention should be paid to slowest system as well as making the process parallel.

2.1.2 Application Areas

Machine learning is widely spread throughout Natural Language Processing (NLP) and other disciplines. It is used from the earlies stages in tokenization, alignment, and spelling correction, all the way to the final stages of machine translation.

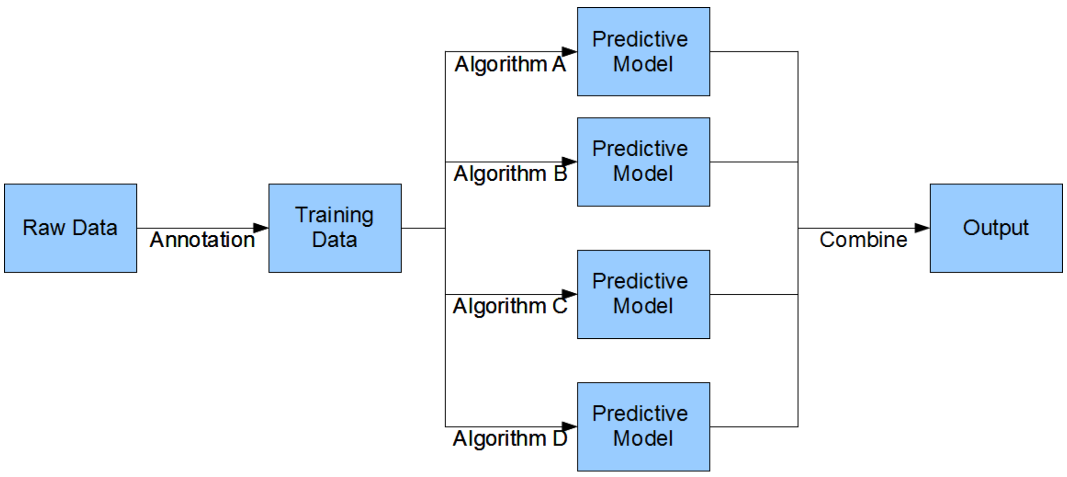


Figure 2.2: Overview of 1 iteration of a machine learning cycle with Ensemble Learning

However, it would be a fallacy to consider it solely for Natural Language Processing. Machine learning has been very successful in other disciplines such as Artificial Intelligence and Bioinformatics. There is an interesting cross over between NLP and Bioinformatics where analyzing the text of Biology papers using NLP can lead to discovering new gene and protein interactions.

Ultimately much of research is judged on the final machine translation score, however most of our work is done in the ensemble learning and annotations related to dependency parsing. While typically a "solved" problem for English, with an accuracy in the 90% range, dependency parsing is often as low as 60% for other languages. Because of this further work is needed in the field. We choose this area with the intent of improving these lower accuracy languages as well as increasing the top accuracy languages such as English.

Whichever field is chosen, the aim of much of this dissertation is to show two independent things not limited to the chosen field. One, ensemble methods can be useful with fields and data sources with little or no information available. Two, that no matter what field machine learning is applied to, great care should be taken when choosing an annotation scheme, as it can greatly affect your final product and products down the pipeline.

2.2 PDT

For the annotation of our gold data, we used the standard set by the Prague Dependency Treebank (PDT) (18). PDT is annotated on three levels, morphological, analytical, and tectogrammatical. Our focus will be primarily on the analytical trees.

2.2.1 Analytical Trees

This work mainly deals with dependency trees, however we do address constituent trees since we convert constituent data into dependency trees both for parsing and for training data collection. We define a dependency tree as a rooted directed tree which is acyclic and can only have a single head relation. The head is half of a dependency relation which is a connecting arc between two nodes, referred to as the head and the dependent. Determining the head/dependency relation can be based on a number of criteria such as the following (30):

- The head determines the syntactic category of a construction, and can sometimes replace the construction
- The head determines the semantic category of a construction, and the dependent gives the semantic specification
- The head is obligatory, the dependent is optional
- The head selects the dependent and determines whether the dependent is needed
- The form of the dependent depends on the head (agreement)
- The linear position of the dependent is specified with reference to the head

Constituent trees and dependency trees share many similarities but some important differences. Dependency trees are a much more efficient representation, not needing the extra nodes for phrasal categories, instead groupings can be considered subtrees. While for constituent trees, each phrase group gets its own node.

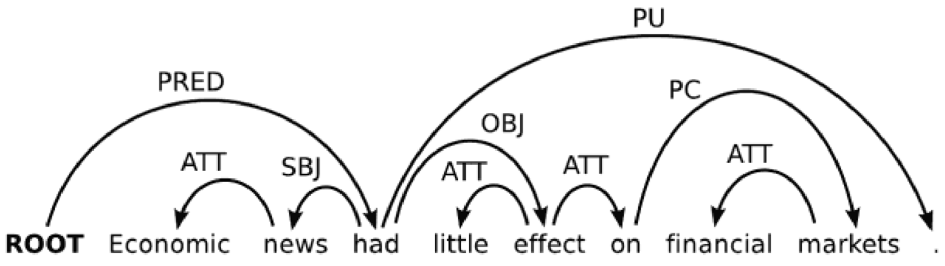


Figure 2.3: Example of a dependency parse. Figure taken from (30)

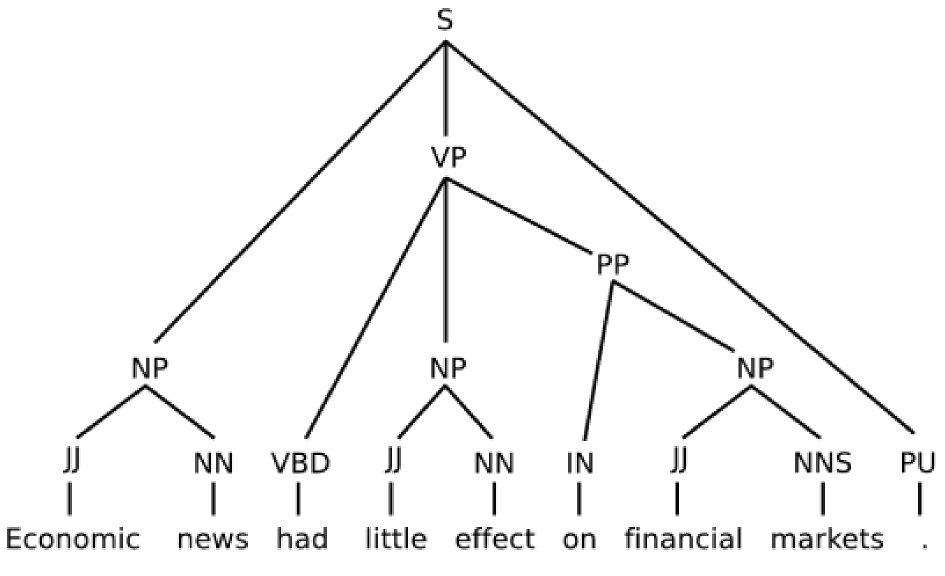

S
VP
PP
NP
NP
NP
JJ NN VBD JJ NN IN JJ NNS PU
Economic news had little effect on financial markets .

Figure 2.4: Example of a constituent parse. Figure taken from (30)

2.2.2 Annotation Standards and Differences

While there are many areas in which annotation can differ, we will mainly focus on coordination structure and noun phrase structure. Coordination structure is often dispute; some standards use the first component as the head, others use the coordination itself, and others use the last component. You can see that this can lead to many permutations. Two examples of coordination are shown in Figure 2.5.

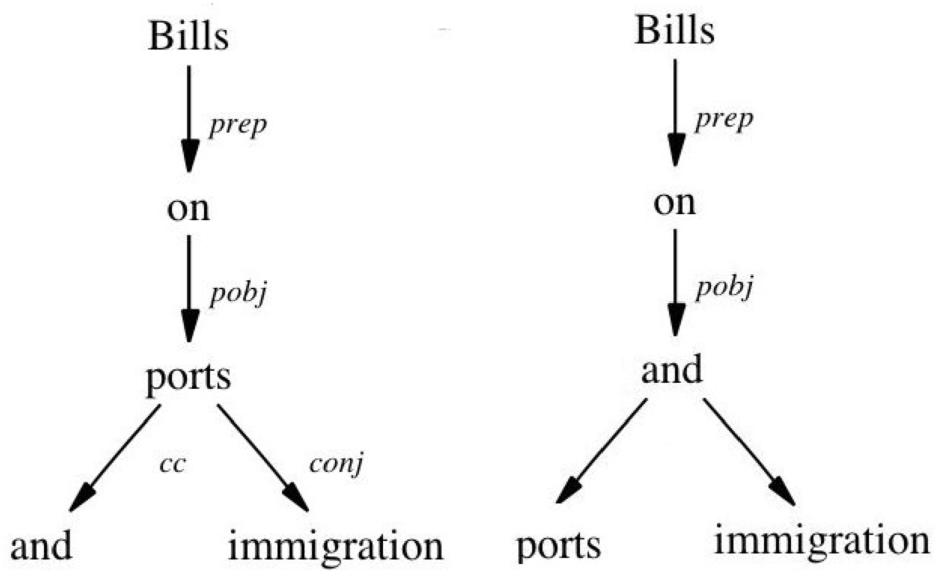


Figure 2.5: Example two coordinations annotations.

2.3 Dependency Parsing

Dependency parsing has been shown to be an important part of many NLP applications. Contrary to its counterpart constituency structure, dependency tree structure is often considered more useful in free word order languages. A common problem parsers have in these languages is the phenomenon of non-projectivity. This is when a subtree of a dependency graph is not contiguous, or visually cannot be drawn without intersecting lines (31). Dependency structures are better at showing agreement whereas constituency, or phrase based, trees typically show neighboring node groupings better due to the divide and conquer approach that context free grammars impose on sentences.

In (29), the authors confirm that two parsers, MSTParser and MaltParser, give similar accuracy results but with very different errors. MSTParser, a maximum spanning tree graph-based algorithm, has evenly distributed errors in terms of sentence length while MaltParser, a transition based parser, has errors on mainly longer sentences. This result comes from the approaches themselves. MSTParser is globally trained so the best mean solution should be found, this is why errors on the longer sentences are about the same as the shorter sentences. MaltParser on the other hand uses a greedy algorithm with a classifier that chooses a particular transition at each vertex. This leads to the possibility of the propagation of errors further in a sentence (37). Both these algorithms are discussed below along with a third technique, constituent transformation. It is important for all future empirical experiments to look at each kind of parser as the different types of errors may greatly change the resulting structures. Below is a brief overview of the type of dependency parsing techniques. This document will not included unsupervised dependency parsers, however we feel many of the approaches might be applicable to unsupervised parsing as well.

2.3.1 Graph-Based

A dependency tree is a special case of a dependency graph that spawns from an artificial root and is acyclic. Because of this we can look at a large history of work in graph theory to address finding the best spanning tree for each dependency graph. The most common form of this type of dependency parsing is called arc-factored parsing and deals with the parameterization of the edge weights. The main drawback of these methods is that for non-projective trees, the worst case scenario for most methods is a complexity of $O(n^3)$ (8). However, for non-projective parsing Chu-Liu-Edmond's algorithm has a complexity of $O(n^2)$ (38). The most common tool for doing this is MSTParser, which is also used in the noun phrase bracketing experiments described later in Section 4.1.

2.3.2 Transition-Based

Transition-based parsing creates a dependency structure that is parameterized over the transitions used to create a dependency tree. This is closely related to the shift-reduce constituency parsing algorithms. Due to the notion of picking transitions in an abstract machine, the algorithms used for these systems tend to be greedy. The benefit of this is that the algorithms have a linear time complexity. However, due to the greedy algorithms, longer arc parses can cause error propagation across each transition (29). The standard tool for transition-based algorithms is MaltParser (42) which in the shared tasks was often tied with the best performing systems. Additionally we will use Zpar (63) which is based on MaltParser but with a different set of non-local features.

2.3.3 Constituent Transformation

While not a true dependency parser, one technique often applied is to take a state-of-the-art constituent parser and transform its phrase based output into dependency relations. This has been shown to also be state-of-the-art in accuracy for dependency parsing in English. This method has also been applied to the Czech language with Collin's parser (5). This path of research has mainly been applied for constituent treebanks that have been converted to dependency and have not thoroughly been tested for treebanks specifically annotated for dependency relations that might additionally have constituent annotations.

In most cases the models are built from the Penn Treebank, a constituent based treebank (35), using a phrase based parser. Then to parse a sentence into a dependency structure, the phrase based output is processed with a conversion tool e.g. PennConverter (22) or Stanford Converter (36). Versions of these converters were used in the CoNLL shared task to create dependency treebanks for a variety of the languages. For my experiments we will make use of Charniak's(4) and Stanford Parser (24) for constituent trees. For our experiments, we only make use of the PennConverter for the actual conversion.

2.3.4 Tamil Parsing

In later experiments, we will make use of two different treebanks for dependency parsing, Tamil and Indonesian. Previous parsing experiments in Tamil were done using a rule based approach which utilized morphological tagging and identification of clause boundaries to parse the sentences (48). The results were also reported for MaltParser and MSTParser. When the morphological tags were available during both training and testing, the rule based approach performed better than Malt and MST parsers. We will be examining the combination of both MST and Malt parsers.

2.3.5 Indonesian Parsing

There was research done on developing a rule based Indonesian constituency parser applying syntactic structure to Indonesian sentences. It uses a rule based approach by defining the grammar using PC-PATR (23). There was also research that applied the above constituency parser to create a probabilistic parser (16). To the best of our knowledge, no dependency parser has been created and publicly released for Indonesian.

2.3.6 Ensemble Parsing

Ensemble learning (7) has been used for a variety of machine learning tasks and recently has been applied to dependency parsing in various ways and with different levels of success. (17, 55) showed a successful combination of parse trees through a combination of trees with various weighting formulations. To keep their tree constraint, they applied Eisner's algorithm for reparsing (8).

Parser combination with dependency trees have been examined in terms of accuracy (52, 53, 61). However, the various techniques have generally examined similar parsers or parsers which have generated various different models. To the best of our knowledge, our experiments are the first to look at the accuracy and part-of-speech error distribution when combining constituent and dependency parsers of many different techniques together.

Other methods of parse combinations have shown to be successful such using one parser to generate features for another parser. This was shown in (43), in which MaltParser was used as a feature to MSTParser. The result was a successful combination of a transition-based and graph-based parser, but did not address adding other types of parsers into the framework. While this thesis focuses on statistical approaches to parsing, rule based parsing has been successfully used to correct parsers trained via machine learning, creating a hybrid approach (1).

2.3.7 Dependency Errors

In (29) the authors confirm that two parsers, MSTParser and MaltParser, give similar accuracy results but with very different errors. MSTParser, a maximum spanning tree graph-based algorithm, has evenly distributed errors across sentence lengths while MaltParser, a transition based parser, has errors on mainly longer sentences. This result comes from the approaches themselves. MSTParser is globally trained so the best mean solution should be found, this is why errors on the longer sentences are about the same as the shorter sentences. MaltParser on the other hand uses a greedy algorithm with a classifier that chooses a particular transition at each vertex. This leads to the possibility of the propagation of errors further in a sentence.

2.3.8 Data Sets

CoNLL Data
Much of the current progress in dependency parsing has been a result of the availability of common data sets in a variety of languages, made available through the CoNLL shared task (41). This data is in 13 languages and 7 language families. Later shared tasks also released data in other genres to allow for domain adaptation. The availability of standard competition gold level data has been an important factor in dependency based research.

Throughout this document we use the English CoNLL data. This data comes from the Wall Street Journal (WSJ) section of the Penn treebank (35). All parsers are trained on sections 02-21 of the WSJ except for the Stanford parser which uses sections 01-21. Charniak, Stanford and Zpar use pre-trained models *ec50spfinal*, *wsjPCFG.ser.gz*, *english.tar.gz* respectively. Additionally we make use of Italian and Japanese from the CoNLL datasets as well for our ensemble experiments. They were chosen due to their high baseline and different language families.

Tamil

Tamil belongs to Dravidian family of languages and is mainly spoken in southern India and also in parts of Sri Lanka, Malaysia and Singapore. Tamil is agglutinative and has a rich set of morphological suffixes. Tamil has nouns and verbs as two major word classes, and hundreds of word forms per lemma can be produced by the application of concatenative and derivational morphology. Tamil's rich morphology makes the language free word order except that it is strictly *head final*.

Only a few attempts were reported in the literature on the development of a treebank for Tamil. Our experiments are based on the openly available treebank (TamilTB) (49). Development of TamilTB is still in progress and the initial results for TamilTB appeared in (48).

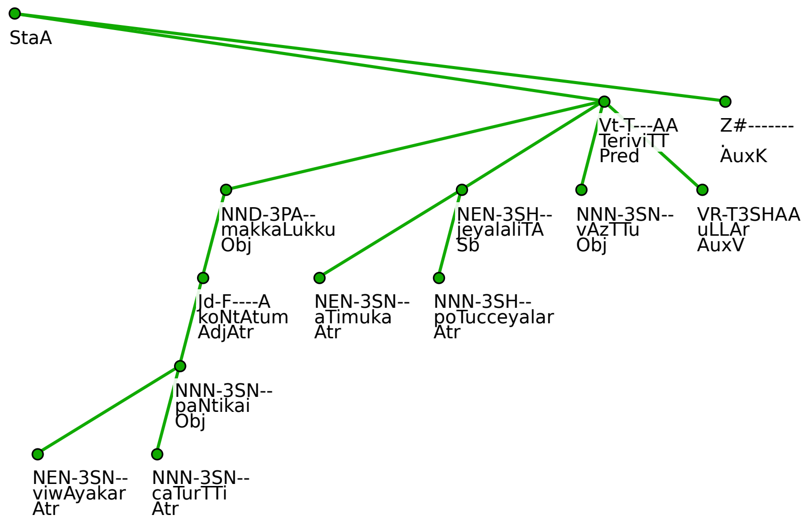
Table 2.2 shows the statistics of the TamilTB Treebank. The last two rows indicate how many word types have unique tags and how many have two tags. The table illustrates that most of the word types can be uniquely identified with single morphological tag and only around 120 word types take more than one morphological tag.

Description	**value**
#Sentences	600
#Words	9581
#Word types	3583
#Tagset size	234
#Types with unique tags	3461
#Types with 2 tags	112

Table 2.2: TamilTB: data statistics

Indonesian

The Indonesian treebank is a collection of manually constructed dependency trees.



Tamil:	விநாயகர்	சதுர்த்தி	பண்டிகை	கொண்டாடும்	மக்களுக்கு	அதிமுக	பொதுச்செயலாளர்	ஜெயலலிதா	வாழ்த்து	தெரிவித்த்	உள்ளார்	.
Tr:	viwAyakar	caTurTTi	paNtikai	**koNtAtum**	makkaLukku	aTimuka	poTucceyalALar	jeyalaliTA	vAzTTu	TeriviTT	uLLAr	.
Gloss:	Vinayakar	Chaturthi	festival	**celebrating**	people-DAT	ADMK	general-secretary	Jeyalalitha	wishes	express	AUX-PERF	.

English: ADMK General Secretary Jeyalalitha expressed her wishes to people who are celebrating Vinayakar Chaturthi festival .

Figure 2.6: Tamil dependency tree example for the sentence "ADMK General Secretary Jeyalalitha expressed her wishes to people who are celebrating Vinayakar Chaturthi festival ." Example taken from the Tamil Treebank (49)

It consists of 100 Indonesian sentences with 2705 tokens and a vocabulary size of 1015 unique tokens. It is taken from the IDENTIC corpus (32) of economic articles. The treebank is enriched with part-of-speech tags provided by MorphInd (33). Since the MorphInd output is ambiguous, the tags are disambiguated and corrected manually including the unknown POS tag. The annotation is done with the visual tree editor, TrEd (44) and stored in CoNLL format (2) for compatibility with several dependency parsers and other NLP tools. This treebank is publicly available as part of IDENTIC corpus (32). The distribution of the POS tags can be seen in Table 2.3.

Currently the annotation provided in this treebank is the unlabeled relationship between the head and its dependents. The annotation adheres to a general annotation guideline as follows:

- The main head node of the sentence is attached to the *ROOT* node.
- Similarly as the main head node, the sentence separator punctuation is also attached to the *ROOT* node.

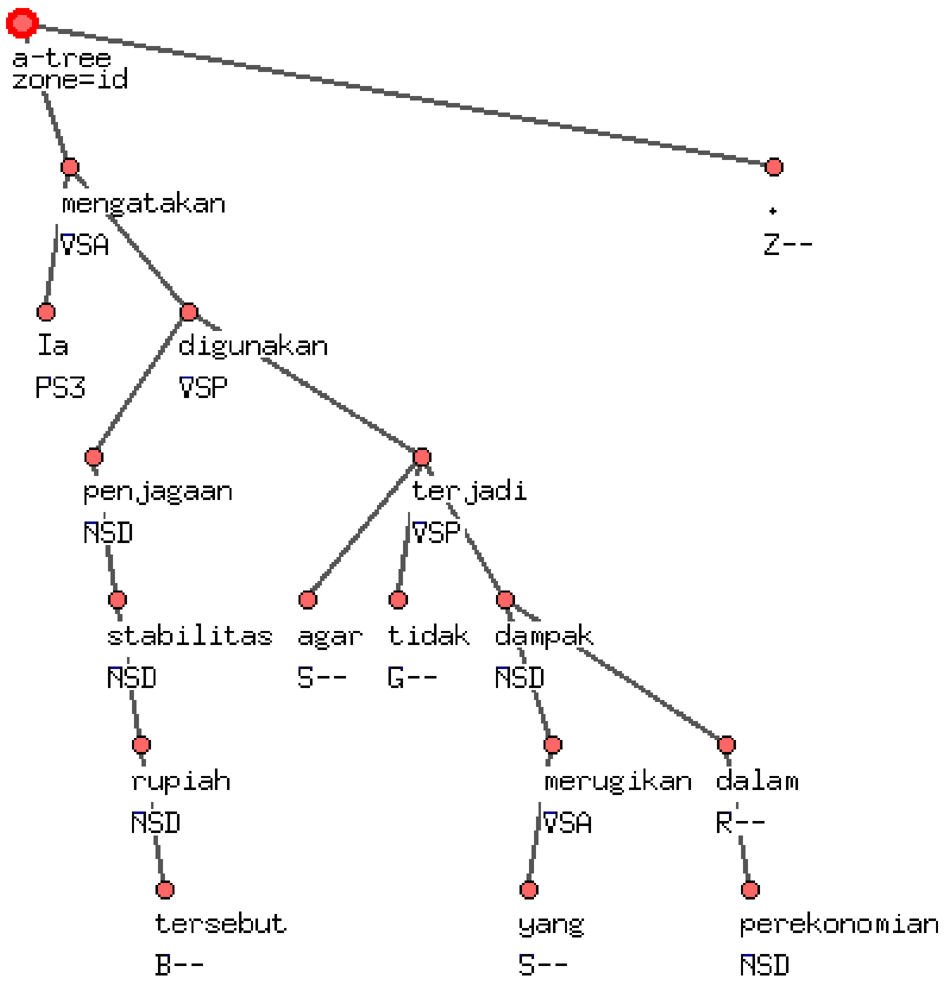


Figure 2.7: Indonesian dependency tree example for the sentence "He said that the rupiah stability protection is used so that there is no bad effect in economy."

POS tag	**Description**	**Freq**
NSD	Noun Singular	1037
Z–	Punctuation	278
VSA	Verb Singular Active	248
CC-	Cardinal Number	226
R–	Preposition	205
D–	Adverb	147
ASP	Adjective Singular Positive	127
S–	Subordinating Conjunction	104
VSP	Verb Singular Passiver	91
H–	Coordinating Conjunction	62
F–	Foreign Word	60
B–	Determiner	43
CO-	Ordinal Number	19
G–	Negation	17
PS3	Pronoun Singular 3rdPerson	12
W–	Question	7
O–	Copula	6
PP1	Pronoun Plural 1stPerson	6
ASS	Adjective Singular Superlative	4
PS1	Pronoun Singular 1stPerson	2
APP	Adjective Plural Positive	1
CD-	Colective Number	1
VPA	Verb Plural Active	1
VPP	Verb Plural Passive	1

Table 2.3: The distribution of the part-of-speech tag occurrences.

- The subordinate conjunction (with POS tag ‘S–’) nodes are attached to its subordinating clause head nodes. The subordinating clause head nodes are attached to its main clause head nodes.

- The coordination conjunctions (with POS tag ‘H–’) nodes, that connect between two phrases (using the conjunction or commas), are attached to the first phrase head node. The second phrase head nodes are attached to the conjunction node. It follows this manner when there are more than two phrases.

- The coordination conjunctions (with POS tag 'H–') nodes, that connect between two clauses (using the conjunction or commas), are attached to the first clause head node. The second clause head nodes are attached to the conjunction node. It follows this manner when there are more than two clauses.

- The prepositions nodes with the POS tag 'R–' are the head of Prepositional Phrases (PP).

- In quantitative numeral phrases such as "3 thousand", 'thousand' node will be the head and '3' node attached to 'thousand' node.

In general, the trees have the verb of the main clause as the head of the sentence where the Subject and the Object are attached to it. In most cases, the most left noun tokens are the noun phrase head, since most of Indonesian noun phrases are constructed in Head-Modifier construction.

2.3.9 Metrics

There are two standard metrics for comparing dependency parsing systems. *L*abeled *a*ttachment *s*core (LAS) and *u*nlabeled *a*ttachment *s*core (UAS). UAS studies the structure of a dependency tree and assesses whether the output has the correct head and dependency arcs. In addition to the structure score in UAS, LAS also measures the accuracy of the dependency labels on each arc. A third, but less common metric, is used to judge the percentage of sentences that are completely correct in regards to their LAS score. This score might be better used to judge how a dependency parser will affect other NLP tools that make use of the dependency parser output (2).

2.4 Machine Translation

There have been two dominant approaches to machine translation: a linguistic rule based approach and a statistical approach. In rule based machine translation, specific lexical rules are created for each language pair. This approach is not generalizable to multiple languages and thus can be time consuming when translations are required in more than one language pair. The major advantage of statistical machine translation (SMT) is that once the infrastructure of a system is built it can be trained for multiple languages using existing parallel text. SMT is currently the state-of-the-art approach and, in particular, phrase-based SMT systems, such as Moses (20, 21), are most commonly used. However, this requirement for parallel training data can be a disadvantage in languages where limited resources are available and large training data is the prime disadvantage of SMT.

Training data are collected in the form of parallel corpora, collections of texts translated into multiple languages and sentence aligned. Parallel corpora are extremely sparse when considering both the coverage of the world's 6000+ languages and the variety of textual genres. The large majority of parallel corpora exist only for parliamentary texts in major languages and are virtually nonexistent for minority languages. The process of creating new parallel corpora for a new language or new genre can be painstaking and time consuming, so SMT is not always a viable option for some under-resourced languages.

2.4.1 Phrase Based MT

Phrase based statistical machine translation is the most popular and among the most commonly used techniques for machine translation. This technique had high requirements for data, both parallel corpora and monolingual target side data. The most commonly used system is Moses (20, 21). Part of the success of Moses is due to the ease of installation which made it available to masses of MT researchers. Moses comes packaged with tools to create language models, word alignments, and evaluation tools. Yearly competitions are held with the Workshop for Machine Translation (WMT) in which systems compete against Moses or use a modified version of Moses to achieve the highest evaluation scores (see Section 2.4.6). To help with the adoption of phrase based techniques a yearly tutorial, the Machine Translation Marathon, is held in which participants are trained on the individual components and theory behind Moses. Additionally, participants often form groups and develop new components and features for Moses during this Marathon. Because of these reasons and the high automatic evaluation scores of the baseline Moses system, it is the standard choice for much of the research in statistical machine translation.

2.4.2 Rule Based MT

Rule based systems are a more classical approach to machine translation. They rely on linguistic information and rules constructed by a linguist. These systems often perform well on very restricted genres or domains but often fail to generalize to other uses. The main components needed are cross-lingual dictionaries, morphological analyzers, and an existing grammar. These systems get complicated quickly and therefore are hard to adapt to new situations. A common infrastructure to implement a rule based system is in Apertium (10). Apertium has been shown to be very useful for languages that are linguistically very similar.

2.4.3 Deep Transfer MT

Deep transfer machine translation is similar to the notion of translating to an interlingua language. The idea being if you can translate to an intermediate language that has very detailed information, you can then translate that language into your target language. This idea developed into using tectogrammatics as an abstraction of a language peculiarities as it generalizes certain aspects of a language while keep very details information about dependencies as well. For our study we use the TectoMT (47) system as it directly uses tectogramatics as well as an analytical layer in which we can directly manipulate the dependency trees.

TectoMT is a machine translation framework based on Praguian tectogrammatics (54) which represents four main layers: word layer, morphological layer, analytical layer, and tectogrammatical layer (47). This framework is primarily focused on the translation from English into Czech. Since much of dependency parsing work has been focused on Czech, this choice of machine translation framework logically follows as TectoMT makes direct use of the dependency relationships. The work in this section primarily addresses the noun phrase structure in the analytical layer (SEnglishA in Figure 2.8).

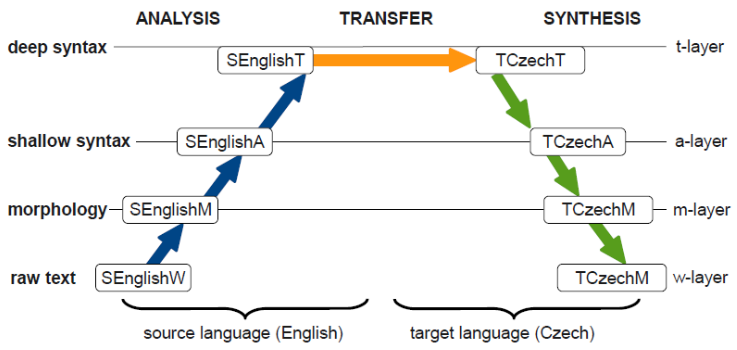


Figure 2.8: Translation Process in TectoMT in which the tectogrammatical layer is transfered from English to Czech (47).

2.4.4 Hybrid MT

Each of the above approaches has their pluses and minuses. A newer branch of MT has started which combines the different approaches into one system. This may be through a voting system as we will do with parsers. Machine translation output may be augmented with additional tools such as was done in DEPFIX with success (51). Or other techniques such as training two systems to translate from a middle language. For instance, we may translate Tamil to English using a statistical system and we call the output A. We can then train a system to learn how to translate the output A to English. Since A is already an "version" of the target language, in theory this is a much more simple task. Since A and English should be closely related, using a rule based system such as Apertium might make sense in this situation.

2.4.5 Data Sets

If experimenting in statistical machine translation, the most valuable resource is the parallel corpus. While many language may have one or two corpora, if your language is in Europe you have a much better chance at finding data. With the formation of the European Union, all participating countries have translations of the EU legislation in their own languages. This allows for one large multi-lingual parallel corpus. This corpus has been compiled at different intervals and has been released under the Europarl corpus (25).

For our experiments we will be using data released for the Workshop in Machine Translation (WMT). These workshops are a competition for teams to compete on the same data to create the best machine translation output. This is done for a handful of language pairs every year, so the data is continually being updated and improved. Since it is based around a shared task, this also gives us a good baseline for systems in the area. In particular we used the WMT shared task data for English to Czech for the years 2010, 2011, and 2012.

2.4.6 Metrics

The BLEU (*B*i*L*ingual *E*valuation *U*nderstudy) score is an automatic scoring mechanism for machine translation that is quick and can be reused as a benchmark across machine translation tasks. BLEU is calculated as the geometric mean of common n-grams percentages, multiplied by a brevity penalty, comparing a machine translation and a reference text (45).

NIST, from the *N*ational *I*nstitute of *S*tandards and *T*echnology, is based upon the BLEU n-gram approach however it is also weighted towards discovering more "informative" n-grams. The more rare an n-gram is, the higher the weight for a correct translation of it will be. This, in effect, lowers the importance of translating punctuation and common words such as articles.

2.5 Under-Resourced NLP

Under-resourced languages are those in which data, labor, or tools do not exist to fulfill need. A common metric for this is the availability of a BLARK, Basic

Language ToolKit (28). These toolkits should contain data for training or existing tools for tasks such as tokenization, morphological analysis, and syntactic parsing.

Under-resourced is often confused with under-populated. For instance, Icelandic only had slightly over 200,000 native speakers but due to government and university support, they have an existing BLARK (34) and well funded language programs. On the other hand most language in India are spoken by more people than then rest of the 6,000+ languages in the world, for instance Tamil with 60 million speakers, but they have little to no resources available. Some work has been done creating a dependency treebank (49), which we will use later with our ensemble methods for under-resourced languages (13).

With 6,000+ spoken languages in the world, we can only hope to make a small dent. For our part in the situation we focus on ensemble and self-training methods for under-resourced languages in dependency parsing. For such language it is usually too early to get consistent and useful machine translation results so we only focus on this early building block. By improving dependency parsing, future NLP tools can be built and more data can be automatically or semi-automatically annotated to help expand each languages' reach. While not a complete NLP pipeline, we think addressing an early BLARK module is important and any improvement that can be added for under-resourced languages can reduce the cost and time needed to build future systems.

2.5.1 Data Mining

For under-resourced languages, data mining is often a key ingredient to a successful system. Both statistical machine translation and dependency parsing require a large amount of training data. It is often the case that additional data will have to be "mined" from the web or from other sources. This data is likely not annotated or aligned to any other language. Since this is a common technique we address this issue by looking at self-training as a way to use this data to assist what little annotations and alignments a language might have. In the self-training section of this dissertation, we examine Indonesian, which has few annotations, but a large corpus of monolingual unannotated data. Using this, we attempt to improve our dependency parsing results while adding some annotations to new data with some confidence.

2.5.2 Languages Used

We use the following languages in different sections of this document. For self-training we need a somewhat small dataset, in Indonesian (32), so we can test many iterations as well as addressing issues surrounding under-resourced languages. To test our ability to improve the state-of-the-art, we do most of our tests on English. When determining the best ensemble techniques, we also examined additional language types such as Italian and Japanese. Japanese was of particular interest since it has such a high baseline score. To test our meta-classifier we also used Tamil, an Indian language, to test an under-resourced language with a full treebank (49) as well as Indonesian.

3

Improving current models: Use what you have

In the first of the two empirical chapters we will start with the optimum situation from the viewpoint of the parser developer. In this situation, we have the following:

1. Ample training, tuning, and testing datasets. For dependency parsing we limit ourselves to whether or not the CoNLL shared task has curated data for a language. While data certainly exists for languages that have not been involved in CoNLL, for the most part, the data described in Section 2.3.8 is what will be used.

2. Established predictive models with relatively high accuracy for a variety of techniques. For dependency parsing this includes both dependency techniques along with constituency techniques.

3. Standard metrics by which to judge improvement. For dependency parsing, we mainly focus on UAS which is outlined in Section 2.3.9.

3.1 Ensemble Learning

Ensemble methods are often used when we want models trained on different data. We will look at ways to combine models trained on the same or similar data.

When looking at similar or identical data, we hope that the different models give us complimentary views of the same data. Ideally you do not want the models to be "good" at the same things. It is equivalent to building strong development team, you need a designer, a coder, and a tester. They may have some overlap but you need an expert in each area to make the final decision. To get started with ensemble parsing, we have created an ensemble class in Treex that collects all analytical trees present and combines their structure into an edge matrix. An edge matrix is a simple structure to store a directed graph. Each edge is assigned some "weight". In the end, we have to generate a parse tree out of this matrix.

To generate a single ensemble parse tree, our system takes N parse trees as input. The inputs are from a variety of parsers as described in 2.3. We will call these parsers our **Base Parsers**. All edges in these parse trees are combined into a graph structure. This graph structure accepts weighted edges via **Graph Edge Weighting Algorithms**. So if more than one parse tree contains the same tree edge, the graph will be weighted appropriately according to a chosen weighting algorithm. One could imagine many way of combining edges through additive and multiplicative methods but our specific weighting algorithms used in our experiments are described in Section 3.1.1.

Once the system has a weighted graph, the system then uses an algorithm to find a corresponding tree structure by a selected **Tree Algorithm** so there are no cycles. In this set of experiments, we constructed a tree by finding the minimum spanning tree using ChuLiu/Edmonds' optimization algorithm, which is a standard choice for MST tasks. The result should be our final **Ensemble Parse**. Figure 3.1 graphically shows the decisions one needs to make in this framework to create an ensemble parse.

3.1.1 Maximum Spanning Tree Combination

To start with, we will apply a maximum spanning tree algorithm, ChuLiu/Edmonds' algorithm. For supervised parsing this may return an "average" parse. Since these algorithms work best with many inputs we can use many "baseline" parsers as well as the same parsers retrained with different data sets. The theory being, the

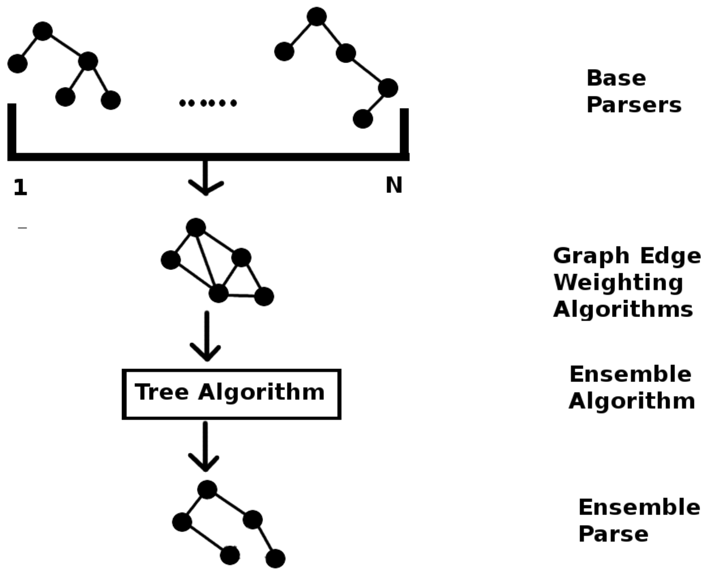


Figure 3.1: General flow to create an Ensemble parse tree

more parsers and the more different types of parsers used, the better chance we have to get an accurate ensemble parse.

With a maximum spanning tree combination, we have two options. We can have an complete graph or a incomplete graph. A complete graph means that in the edge matrix we can have a default value for every edge. In this case if one edge has a very high probability but you have to pay a large cost to get to that edge, the algorithm may create a shortcut to the edge using these nonexistent edges that were inserted by default. In the incomplete graph we only add values into the edge matrix that exist in our base parsers. This means no new edge can be added that was not in our base parsers. All of our experiments in this document deal with incomplete graphs.

3.1.1.1 Parsers

For English, we use 5 of the most commonly used parsers which enables us to have a wide scope for ensemble learning. They range from graph-based approaches to transition-based approaches to constituent parsers. Constituency output is converted to dependency structures using PennConverter (22). All parsers are integrated in the Treex framework (46, 60) using the publicly released parsers from the respective authors but with Perl wrappers to allow them to work on a common tree structure. For testing we use section 23 of the WSJ for comparability reasons with other papers. This test data contains 56,684 tokens. For tuning we use section 22. This data is used for determining the weighting features for the POS error distribution in Section 3.1.2.

For Italian and Japanese, we do not have access to a constituency to dependency transformation which limits use to too few parsers. So for these languages we only use MaltParser and MSTParser but we use different training parameters to create various parsing models. For MaltParser, we use 7 models and for MSTParser, we use 2 models, these are enumerated in Table 3.1. For tuning data we remove the last 500 sentence of the training data. As with English, this is used for calculating the POS error distribution used in our weighting schemes in Section 3.1.2.

Model Name	**Dependency System**
nonproj	MST
proj	MST
nivreeager	Malt
nivrestandard	Malt
2planar	Malt
planar	Malt
stackeager	Malt
stacklazy	Malt
stackproj	Malt

Table 3.1: Enumerated baseline parsing algorithms

In addition to the UAS score of the enumerated parsers, we also report the accuracy of an Oracle Parser. This parser is simply the best possible parse composed only of edges offered by the individual dependency parsers. If the reference, gold standard, tree has an edge that any of the parsers contain, we include that edge in the Oracle parse. Initially all nodes of the tree are connected to an artificial root. Since only edges that exist in a reference tree are added, the Oracle Parser maintains the acyclic constraint.

Parser	UAS
Charniak	92.08
Stanford	87.88
MST	86.49
Malt	84.51
Zpar	76.06

Table 3.2: Our baseline parsers and corresponding UAS used in our ensemble experiments

For English we ran 2^5 model combinations but only report on combinations of three or more models. We conducted 2^9 combinations of models for each of three weighting schemes for both Japanese and Italian. To consolidate the results we only display the top ten results for Italian and Japanese.

3.1.1.2 Weighting Schemes

Currently we are applying three weighting algorithms to the graph structure. All three of these are simple weighting techniques but even in their simplicity we can see the benefit of this type of combination.

- **Uniform Weights**: an edge in the graph gets incremented +1 weight for each matching edge in each parser. If an edge occurs in 4 parsers, the weight is 4.
- **UAS Weighted**: Each edge in the graph gets incremented by the value of its parsers individual accuracy. So based on the UAS baseline parsing results from Table 3.3, an edge in Charniak's tree gets .92 added while MST gets .86 added to every edge they share with the resulting graph. This weighting should allow us to add poor parsers with very little harm to the overall score.
- **Plural Voting Weights**: In Plural Voting, the parsers are rated and each gets a "vote" based on their quality. With N parsers the best parser gets N votes while the last place parser gets 1 vote. In this experiment, Charniak received 5 votes, Stanford received 4 votes, MSTParser received 3 votes,

MaltParser received 2 votes, and Zpar received 1 vote. Votes in this case are added to each edge as a weight.

- **UAS^{10}**: For this weighting scheme we took each UAS value to the 10th power. This gave us the desired effect of making the differences in accuracy more apparent and giving more distance from the best to worst parser. This exponent was empirically found and the results are shown in Table 3.4.

3.1.1.3 Results

Table 3.3 contains the results of different parser combinations of the 5 parsers in Table 3.2. The results seem to indicate that using two parsers will give you an "average" score. Ensemble learning seems to start to have a benefit around 3 parsers with a few combinations having a better UAS score than any of the baseline parsers, these cases are in bold throughout the table. When we add a 4th parser to the mix almost all configurations lead to an improved score when the edges are not weighted uniformly. The only case in which this does not occur is when Stanford's Parser is not used. When all five parsers are used together with Plural Voting, the ensemble parser improves over the highest individual parser's UAS score. For UAS^{10} voting, the 5 parser combination gives the second highest accuracy score. The top overall score is when we use UAS^{10} weighting with the 4 top individual parsers. For parser combinations that do not feature Charniak's parser, we also find an increase in overall accuracy score compared to each individual parser, although never beating Charniak's individual score.

To see the maximum accuracy an ensemble system can achieve, we include an Oracle Ensemble Parser in Table 3.3. As we can see in Table 3.3, the ceiling of ensemble learning is 97.41% accuracy. Because of this high value, ensemble learning should be a very prosperous area for dependency parsing research.

To discover the best exponential value in UAS^X we looked at our combining all parsers at different exponential values. We empirically test different values on our tuning data. UAS^{10} is the top scoring weight for English. The results are in Table 3.4. We only discover this weight using the "all" parser setting and only on English. If this setup was used in production it would be wise to relearn this

System	Uniform Weighting	UAS Weighted	Plural Voting	UAS^{10} Weighted	Oracle UAS
Charniak-Stanford	89.84	92.08	92.08	92.08	94.85
Charniak-Mst	89.14	92.08	92.08	92.08	95.33
Charniak-Malt	88.15	92.08	92.08	92.08	95.4
Charniak-Zpar	84.10	92.08	92.08	92.08	94.49
Stanford-Mst	86.92	86.49	87.88	86.49	94.29
Stanford-Malt	86.05	87.88	87.88	87.88	94.09
Stanford-Zpar	81.86	87.88	87.88	87.88	93.02
Mst-Malt	85.54	86.49	86.49	86.49	90.38
Mst-Zpar	81.19	86.49	86.49	86.49	92.03
Malt-Zpar	80.07	84.51	84.51	84.51	91.46
Charniak-Stanford-Mst	91.86	**92.27**	**92.28**	**92.25**	96.48
Charniak-Stanford-Malt	91.77	**92.28**	**92.3**	92.08	96.49
Charniak-Stanford-Zpar	91.22	91.99	92.02	92.08	95.94
Charniak-Mst-Malt	88.80	89.55	90.77	92.08	96.3
Charniak-Mst-Zpar	90.44	91.59	92.08	92.08	96.16
Charniak-Malt-Zpar	88.61	91.3	92.08	92.08	96.21
Stanford-Mst-Malt	87.84	**88.28**	**88.26**	**88.28**	95.62
Stanford-Mst-Zpar	**89.12**	**89.88**	**88.84**	**89.91**	95.57
Stanford-Malt-Zpar	**88.61**	**89.57**	87.88	87.88	95.47
Mst-Malt-Zpar	**86.99**	**87.34**	**86.82**	86.49	93.79
Charniak-Stanford-Mst-Malt	90.45	**92.09**	**92.34**	**92.56**	97.09
Charniak-Stanford-Mst-Zpar	91.57	**92.24**	**92.27**	**92.26**	96.97
Charniak-Stanford-Malt-Zpar	91.31	**92.14**	**92.4**	**92.42**	97.03
Charniak-Mst-Malt-Zpar	89.60	89.48	91.71	92.08	96.79
Stanford-Mst-Malt-Zpar	**88.76**	**88.45**	**88.95**	**88.44**	96.36
All	91.43	91.77	**92.44**	**92.58**	97.41

Table 3.3: Initial Results of the minimum spanning tree algorithm on a combined edge graph. Scores are in **bold** when the ensemble system increased the UAS score over all individual systems.

exponential value through new tuning data for the model combination choice and particular language each time.

X	UAS^X
0.5	91.77
2	91.84
4	91.98
6	92.44
8	92.47
9	92.52
10	**92.58**
11	92.57
12	92.57
16	92.43

Table 3.4: UAS scores of our ensemble parser with all parsers included at different exponential values (UAS^x)

3.1.1.4 POS Errors

Given the prior research on differences in dependency errors seen in Section 2.3.7, we look at the error distribution for all five parsers along with our best ensemble parser configuration. Much like the previous work we expect different types of errors, given that our parsers are from 3 different parsing techniques. To examine if the ensemble parser is substantially changing the parse tree or is just taking the best parse tree and substituting a few edges, we examine the part-of-speech errors in Table 3.5.

As we can see the range of POS errors varies dramatically depending on which parser we examine. For instance for *CC*, Charniak has 83% while MST is only 71% accurate. There are also POS errors that are almost always universally bad such as the left parenthesis *(*. Given the large difference in POS errors, weighting an ensemble system by POS is a logical choice, which we will address further in Section 3.1.2. As we can see in Figure 3.2, the varying POS accuracies indicate that the parsing techniques we have incorporated into our ensemble parser, are significantly different. In almost every case in Table 3.5, our ensemble parser achieves the best dependency accuracy for each POS, while reducing the average relative error rate by 8.64%.

The current weighting systems don't simply default to the best parser or to an average of all errors. In the majority of cases our ensemble parser obtains the top accuracy. The ability of the ensemble system to use maximum spanning tree on an edge graph allows the ensemble parser to connect previously unconnected nodes for an overall gain, which is preferable to techniques which only select the best model for a particular tree. In all cases, our ensemble parser is never the worst parser. In cases where the POS is less frequent, our ensemble parser seems to average out the error distribution.

We have shown the benefits of using a maximum spanning tree algorithm in ensemble learning for dependency parsing. Not only do we combine dependency parsers, we show the effectiveness of combining constituent parsers with other dependency parsing techniques. This ensemble method shows improvements over the current state-of-the-art for each individual parser. We also show a theoretical maximum oracle parser which indicates that much more work in this field can take place to improve dependency parsing accuracy toward the oracle score of 97.41%.

We demonstrated that using parsers of different techniques, especially including transformed constituent parsers, can lead to the best accuracy within this ensemble framework. The improvements in accuracy are not simply due to a few edge changes but can be seen to improve the accuracy of the majority of POS tags over all individual systems.

To amplify the effect of POS error reduction further, we will look to learn the ensemble weights though our POS error distribution. To do this we will cluster the POS accuracies of our parsers and combine in a similar fashion. These experiments are detailed in Section 3.1.2.

3.1.1.5 Iteratively combining ensemble systems

In the previous section, we saw improvement reaching that of the best model. If the new system creates an even better model it would seem logical to iteratively

POS	**Charniak**	**Stanford**	**MST**	**Malt**	**Zpar**	**Best Ensemble**	**Relative Error Reduction**
PDT	88.890	77.78	83.33	88.89	77.78	88.89	0.00
CC	83.540	74.73	71.16	65.84	20.39	**84.63**	6.64
NNP	94.590	92.16	88.04	87.17	73.67	**95.02**	7.81
,	84.450	78.02	63.13	60.12	65.64	**85.08**	3.99
WP$	90.480	71.43	85.71	90.48	0.00	90.48	0.00
VBN	91.720	89.81	90.35	89.17	88.26	**93.81**	25.27
WP	83.780	80.18	80.18	82.88	2.70	81.08	-16.67
RBR	77.680	62.50	75.00	76.79	68.75	**78.57**	4.00
CD	94.910	92.67	85.19	84.46	82.64	**94.96**	1.02
RP	96.150	95.05	97.25	95.60	94.51	**97.80**	42.86
JJ	95.410	92.99	94.47	93.90	89.45	95.85	0.00
PRP	97.820	96.21	96.68	95.64	95.45	**98.39**	26.09
TO	94.520	89.44	91.29	90.73	88.63	94.35	-2.94
EX	96.490	98.25	100.00	100.00	96.49	**98.25**	50.00
WRB	63.910	60.90	68.42	73.68	4.51	63.91	0.00
RB	86.260	79.88	81.49	81.44	80.61	**87.19**	6.74
FW	55.000	45.00	60.00	25.00	35.00	55.00	0.00
WDT	97.140	95.36	96.43	95.00	9.29	**97.50**	12.50
VBP	91.400	83.29	80.92	75.81	50.87	91.27	-1.45
JJR	88.380	80.81	74.75	70.20	68.18	87.37	-8.70
VBZ	91.970	87.35	83.86	80.78	57.91	**92.46**	6.06
NNPS	97.620	95.24	100.00	95.24	69.05	100.00	100.00
(	73.610	75.00	54.17	58.33	15.28	73.61	0.00
UH	87.500	62.50	75.00	37.50	37.50	87.50	0.00
POS	98.180	96.54	98.54	98.72	0.18	**98.36**	10.00
$	82.930	80.00	67.47	66.40	52.27	**84.27**	7.81
“	83.990	79.66	76.08	58.95	74.01	**84.37**	2.35
:	77.160	72.53	45.99	44.44	53.70	**79.63**	10.81
JJS	96.060	90.55	88.19	86.61	82.68	93.70	-60.00
LS	75.000	50.00	100.00	75.00	75.00	75.00	0.00
.	96.060	93.48	91.07	84.89	87.56	**97.08**	25.81
VB	93.040	88.48	91.33	90.95	84.37	**94.24**	17.27
MD	89.550	82.02	83.05	78.77	51.54	**89.90**	3.28
NNS	93.100	89.51	90.68	88.65	78.93	**93.67**	8.26
NN	93.620	90.29	88.45	86.98	83.84	**94.00**	6.00
VBD	93.250	87.20	86.27	82.73	64.32	**93.52**	4.03
DT	97.610	96.47	97.30	97.01	92.19	**97.97**	14.78
#	100.000	80.00	0.00	0.00	0.00	100.00	0.00
’	88.280	83.79	81.84	69.92	79.88	**90.04**	15.00
RBS	90.000	76.67	93.33	93.33	86.67	90.00	0.00
IN	87.800	78.66	83.45	80.78	73.08	87.48	-2.66
SYM	100.000	100.00	100.00	0.00	0.00	100.00	0.00
PRP$	97.640	96.07	47.22	96.86	93.12	97.45	-8.33
)	70.830	77.78	96.46	55.56	12.50	**72.22**	4.76
VBG	85.190	82.13	82.74	82.25	81.27	**89.35**	28.10
Average							**7.79**

Table 3.5: POS errors for each of our systems that are used in the ensemble system. We also our best ensemble system which is the combination of all parsers using UAS^{10}. All POS errors are calculated using the testing data provided by section 23 of the WST. The ensemble system that generated these errors was parameterized on tuning data, section 22 of the WSJ.

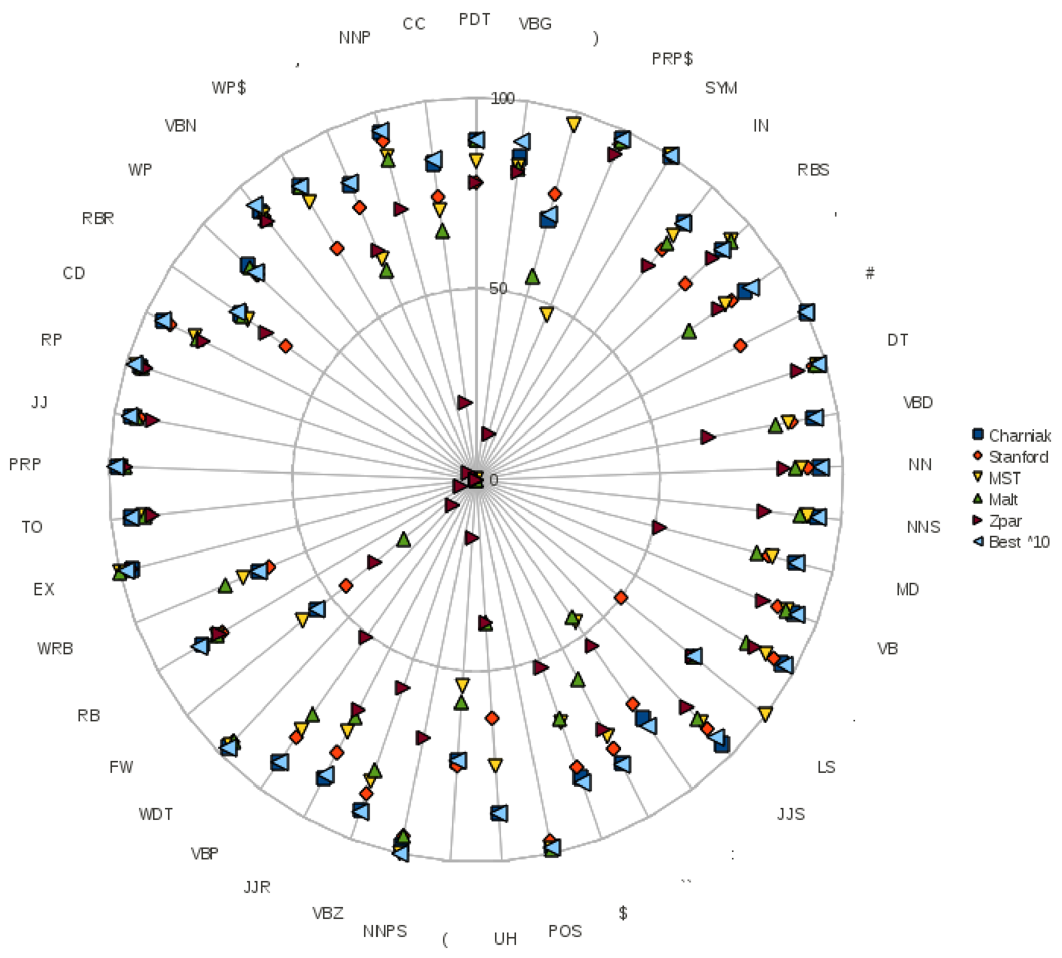


Figure 3.2: Parsers dependency error rates by part-of-speech inclusing the best ensemble system

include this system in the same process cycle. One would expect an initial gain that would quickly reduce. To test whether we can use the new ensemble parse as an input to the same ensemble system we use the following process:

1. Run the baseline system and create an ensemble parse
2. Remove worst performing model
3. Add ensemble parse and rerun
4. Remove previous ensemble parse and lowest performing system

5. Add new ensemble system
6. Repeat this process until there are 3 systems left (each iteration removes 1 model)

We remove the previous ensemble parse along with the lowest performing system in order to try to not bias the structure toward the previous ensemble structure. We assume each iteration the new ensemble parse will gather some of that previous structure since it is an input to the system. If we were to keep each ensemble system as input into each iteration, giving our additive weighting scheme, our system will never converge to an answer different than the first ensemble parse.

Growth was slow to non-existent in comparison to basic UAS weighting formula over the first iteration, this can be seen in Figure 3.3. Counter to the hypothesis, the system increases a bit on the later iterations. Overall the results, only giving a mild increase and leveling off quickly, are not significant enough to look at much further but in some situations they may be helpful.

Exponent	Iteration 1	Iteration 2	Iteration 3	Iteration 4
1	91.83%	91.83%	92.37%	92.41%
2	91.83%	91.83%	92.37%	92.41%
6	92.45%	92.45%	92.45%	92.47%
10	92.55%	92.55%	92.55%	**92.55%**

Table 3.6: UAS scores across iterative iterations. Exponent is the weighting parameter of the weighting scheme UAS^X

Table 3.6 shows that the system did not change much from exponential 6 onwards. This is due to the additive system favoring the exaggerated weight of the ensemble system. Since the ensemble system already contains a likely MST parse of the remaining systems, it is hard to get this number to shift. Although the results are lower, we think the system with no exponential (exponent=1) is more interesting. Without introducing any artificial weighting schema such as exponentials, we are able to increase the overall score by 0.58. We stopped testing at $X = 10$ so as to keep it consistent with our previous experiments.

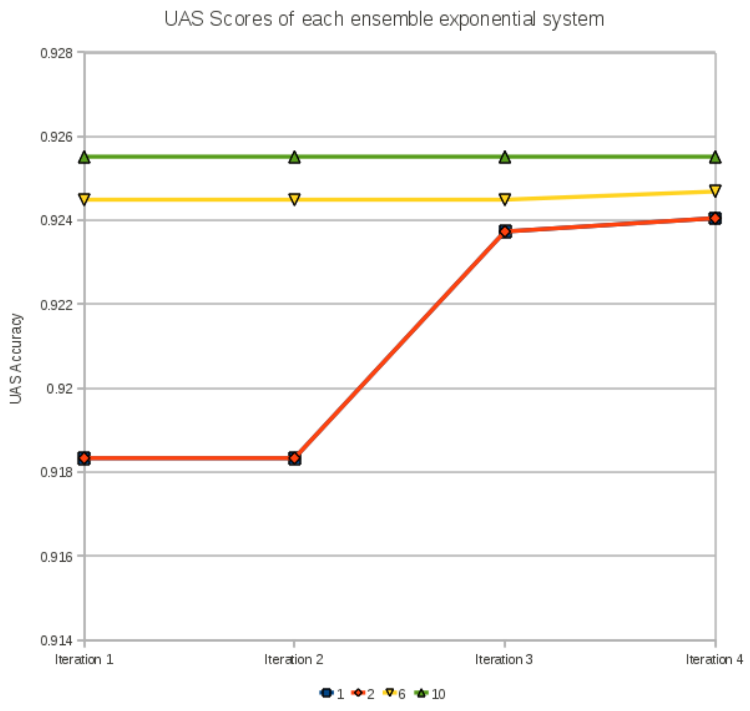


Figure 3.3: UAS scores of each exponential ensemble system across iterations

3.1.2 Fuzzy Clustering

Each dependency parsing technique described thus far achieves state-of-the-art results, however each technique achieves this success via different error distribution. To minimize these errors and to increase state-of-the-art parsing accuracy, we now examine ensemble techniques that weight graph edges based on part-of-speech errors. To do this, we cluster all dependency parsing models on part-of-speech error counts. This leads us to have a different weighting scheme between dependency parsers for each individual part-of-speech.

3.1.2.1 Weighting Schemes

We apply three weighting algorithms to the graph structure. First we give each parser uniform weight. Second we weight each particular edge by a combination of models weights determined by the part-of-speech error distribution. Finally we apply exponential scaling to the POS weights (POS^{10}) to amplify the differences between models.

$$Weight_{edge} = \sum_{i=1}^{N} c_i \sum_{j=1}^{M} w_j$$

Figure 3.4: Equation for calculating the weight of 1 edge across N POS clusters each with their own weight c and M models each with their own weight w, where each M predicts the edge

- **Uniform**: This is the same as our previous fixed weight experiments with each edge in the graph getting incremented +1 weight for each matching edge in each parser.
- **POS**: Each edge of the graph is weighted by a combination of weighting schemes determined by the particular part-of-speech. This is described in more detail in Section 3.1.2.2.
- **POS^{10}**: For this weighting scheme, we took each POS model score from the previous weighting scheme and raised it to the 10th power at run time which was empirically chosen. This was once again an opportunity to exaggerate the differences in each parser when it came to each part-of-speech.

3.1.2.2 Determining Part-of-Speech Clustering Weights

To automatically learn the weights of our models, we turn to part-of-speech error analysis. We obtain a POS error distribution from our tuning data. Using fuzzy clustering with the cosine distance metric over 20 iterations we find 3 clusters. For a particular part-of-speech we get a weight corresponding to each cluster that sum to 1. In N clusters, we have M weights corresponding to each M models. So for a particular edge, we get its weight by summing each cluster multiplied by all model weights as seen in Figure 3.4. If a part-of-speech did not occur in the tuning data, the weights are equally split across all clusters.

Our clustering algorithm is based on Fuzzy-Cmeans algorithm. This algorithm allows for a “data point” to exist partially in many clusters. The cluster centroid is iteratively calculated. For our data points we will use a count of correctly predicted POS’s tags for each parser so for one entry we would have NOUN⇒Parser1⇒10, Parser2⇒20 Parser3⇒5, Parser4⇒6. The clusters will then specify the centroids of different clusters of these data points. We use 3 clusters which gives use 3 combinations of model weights.

- Determine clusters for a POS
- Multiply each corresponding cluster weight against each model that predicts the edge
- Sum the result
- Repeat for each cluster
- Sum results from all cluster

Models	**Cluster 1**	**Cluster 2**	**Cluster 3**
Charniak	21.48%	26.46%	31.68%
Stanford	20.47%	24.91%	27.62%
Mst	20.29%	24.25%	21.57%
Malt	19.38%	20.47%	10.43%
Zpar	18.38%	3.91%	8.70%

Table 3.7: Cluster weights for each model when averaged based across centroids for our English models

For instance for Table 3.7 and Table 3.8 we will look at a node which has the POS VBZ. Let us assume that the edge we are looking at is only predicted by ZPar and Charniak. Step one we see that Cluster 1 has a weight of 0.971. We would then sum the weights of the models with the predicted edge $(21.48*0.971)+(18.38*0.971) = 38.7$. We then repeat this for Cluster 2 ($(26.46*0.025)+(3.91*0.025) = 0.75$) and Cluster 3 ($(31.68*0.0036)+(8.70*0.0036) = 0.14$). For the final weight we combine these weights $38.7+0.75+0.14 = 39.59$. As you can see Cluster 2 and Cluster 3 provide little weight to the final result. This is because as was seen in Figure 3.2, Zpar did relatively well on this POS compared to its poor results on others. Cluster 1 gives a higher score to ZPar in this situation.

POS	**Cluster 1**	**Cluster 2**	**Cluster 3**
(	0.007	0.9928	0.0007
)	0.043	0.9508	0.0067
CC	0.008	0.9904	0.0019
JJ	1.000	0.0001	0
MD	0.804	0.182	0.0136
NN	1.000	0	0
NNP	1.000	0.0001	0
NNPS	0.007	0.9917	0.0013
PDT	0.210	0.7277	0.0618
PRP	1.000	0.0001	0.0001
VB	1.000	0	0
VBD	0.981	0.0153	0.0036
VBZ	0.971	0.025	0.0041
WDT	0.003	0.9963	0.0005

Table 3.8: The weights of each cluster for selected POS tags.

3.1.2.3 Results

This weighting system models the POS tag in a fashion similar to the POS error distribution. For instance the POS tag "CC" has high weights for Cluster 1. Cluster 1 gives very little weight to Zpar. If we examine the POS errors in Table 3.5, Zpar did very poorly on these tags. Overall, it does appear that the clusters tend towards a more balanced weighting scheme while only pointing out true outliers.

Table 3.9 shows the results of the run on our testing data in which the fuzzy clusters were determined on our tuning data. The accuracies are higher but comparable to what is seen with a basic uniform weighting scheme. The weights were a combination of the fuzzy clustering weights based on POS errors shown in Table 3.8. This score, **92.54%**, which occurred when all parsers were used, is the best accuracy of all our ensemble techniques and model combinations with English.

Parser	**Uniform**	**POS**	**POS^{10}**	**Oracle**
Charniak-Stanford-Mst	91.86	**92.27**	92.08	96.48
Charniak-Stanford-Malt	91.77	**92.28**	92.08	96.49
Charniak-Stanford-Zpar	91.22	92.00	92.08	95.94
Charniak-Mst-Malt	88.80	89.55	92.08	96.3
Charniak-Mst-Zpar	90.44	91.59	92.08	96.16
Charnial-Malt-Zpar	88.61	91.30	92.08	96.21
Stanford-Mst-Malt	87.84	**87.94**	87.88	95.62
Stanford-Mst-Zpar	**89.12**	**89.89**	87.88	95.57
Stanford-Malt-Zpar	**88.61**	**89.60**	**89.58**	95.47
Mst-Malt-Zpar	**86.99**	**87.34**	86.49	93.79
Charniak-Stanford-Mst-Malt	90.45	**92.09**	**92.45**	97.09
Charniak-Stanford-Mst-Zpar	91.57	**92.24**	**92.49**	96.97
Charniak-Stanford-Malt-Zpar	91.31	**92.15**	92.08	97.03
Charniak-Mst-Malt-Zpar	89.60	89.53	92.08	96.79
Stanford-Mst-Malt-Zpar	**88.76**	**88.40**	87.88	96.36
All	91.43	91.84	**92.54**	97.41

Table 3.9: UAS scores of our ensemble parser using POS fuzzy clustering weights for English. Values are bolded wherever the result is greater than any individual model within the ensemble system.

3.1.2.4 Ensemble with various training parameters

Models	IT-UAS	JA-UAS
mstnonproj	72.89%	78.65%
mstproj	76.03%	84.04%
nivreeager	82.08%	92.99%
nivrestandard	81.11%	92.87%
2planar	82.58%	90.01%
planar	81.89%	90.81%
stackeager	81.44%	93.25%
stacklazy	81.27%	93.43%
stackproj	81.57%	91.87%

Table 3.10: Our baseline parsers and corresponding UAS used in our ensemble experiments for Italian and Japanese

The previous English experiments tested five parsers with a single model each. For some languages, we do not have the needed constituent conversions or a variety of dependency parsers with a high average. Because of this we also wanted to test using two parsers with various training parameters to simulate different models. We show this on Italian and Japanese data.

Table 3.11 shows the scores for Japanese. The scores are taken from the top 10 performing systems for POS^{10} weighting scheme. All of the top 10 systems performed at or better than the best performing individual model. This is a promising result since Japanese already has a relatively high baseline compared to other languages. Extending the results from the English experiments, we might see even greater improvement given more diversity of models instead of only MaltParser and MSTParser.

Table 3.12 shows the scores for Italian. The scores are taken from the top 10 performing systems for POS^{10} weighting scheme as well. Overall none of the combinations were able to achieve as high of a score as the best individual model which was the *2planar* trained MaltParser. Of all the weighting schemes, POS^{10} performed the best with an average accuracy of 77.38% and a max accuracy of 81.34%. Although this did not beat *planar2* the average POS error reduction,

Model Combos	**Uniform**	**POS**	**POS^{10}**	**Oracle**
mstnonproj-2planar-nivrestandard nivreeager-stacklazy-stackeager	92.96%	93.00%	93.43%	97.51%
mstnonproj-nivrestandard nivreeager-stackeager	93.28%	93.43%	93.43%	97.08%
mstnonproj-nivrestandard nivreeager-stacklazy	93.35%	93.43%	93.43%	97.04%
mstnonproj-planar nivreeager-stacklazy	92.63%	93.45%	93.43%	97.11%
mstnonproj-planar-nivrestandard nivreeager-stacklazy	92.73%	93.12%	93.43%	97.32%
mstnonproj-2planar nivreeager-stacklazy	93.24%	93.43%	93.45%	97.22%
mstnonproj-nivrestandard-nivreeager	93.29%	93.45%	93.45%	96.59%
mstnonproj-planar-nivrestandard nivreeager-stacklazy-stackeager	93.38%	93.45%	93.45%	97.46%
mstnonproj-2planar-nivrestandard nivreeager-stacklazy	92.84%	93.08%	93.50%	97.41%
mstnonproj-2planar nivrestandard-nivreeager	93.26%	93.59%	93.59%	97.11%
Average over all Combos	92.41%	92.65%	92.63%	96.84%

Table 3.11: UAS scores of our ensemble parser using POS fuzzy clustering weights for Japanese

described in Section 3.1.2.5, shows us another story on how this ensemble system may be used.

Model Combos	Uniform	POS	POS^{10}	Oracle
mstnonproj-2planar-nivrestandard nivreeager-stacklazy-stackeager	80.77%	80.57%	81.04%	89.82%
mstnonproj-nivrestandard nivreeager-stackeager	81.06%	81.10%	81.04%	90.36%
mstnonproj-nivrestandard nivreeager-stacklazy	80.61%	80.77%	81.08%	89.78%
mstnonproj-planar nivreeager-stacklazy	80.75%	80.89%	81.10%	90.72%
mstnonproj-planar-nivrestandard nivreeager-stacklazy	80.61%	80.75%	81.10%	89.87%
mstnonproj-2planar nivreeager-stacklazy	80.57%	80.77%	81.14%	89.68%
mstnonproj-nivrestandard-nivreeager	80.69%	80.99%	81.16%	90.62%
mstnonproj-planar-nivrestandard nivreeager-stacklazy-stackeager	80.75%	81.04%	81.24%	90.58%
mstnonproj-2planar-nivrestandard nivreeager-stacklazy	80.95%	81.14%	81.32%	87.40%
mstnonproj-2planar nivrestandard-nivreeager	80.34%	80.93%	81.34%	88.30%
Average over all Combos	77.33%	77.36%	77.38%	87.98%

Table 3.12: UAS scores of our ensemble parser using POS fuzzy clustering weights for Italian

3.1.2.5 POS error reduction

Figure 3.5 shows visually how the best ensemble system for Italian is at or better than all other parsers in terms of POS errors. This shows that the ensemble system is not just an averaging of errors but actually does reduce error for each individual POS. Similar results can be seen in English as well.

POS	**Charniak**	**Stanford**	**MST**	**Malt**	**Zpar**	**Best Ensemble**	**Relative Error Reduction**
PDT	88.890	77.78	83.33	88.89	77.78	88.89	0.00
CC	83.540	74.73	71.16	65.84	20.39	**84.63**	6.64
NNP	94.590	92.16	88.04	87.17	73.67	**95.02**	7.81
VBN	91.720	89.81	90.35	89.17	88.26	**93.81**	25.27
JJ	95.410	92.99	94.47	93.90	89.45	95.85	0.00
PRP	97.820	96.21	96.68	95.64	95.45	**98.39**	26.09
TO	94.520	89.44	91.29	90.73	88.63	94.35	-2.94
RB	86.260	79.88	81.49	81.44	80.61	**87.19**	6.74
FW	55.000	45.00	60.00	25.00	35.00	55.00	0.00
WDT	97.140	95.36	96.43	95.00	9.29	**97.50**	12.50
VB	93.040	88.48	91.33	90.95	84.37	**94.24**	17.27
MD	89.550	82.02	83.05	78.77	51.54	**89.90**	3.28
NNS	93.100	89.51	90.68	88.65	78.93	**93.67**	8.26
NN	93.620	90.29	88.45	86.98	83.84	**94.00**	6.00
DT	97.610	96.47	97.30	97.01	92.19	**97.97**	14.78
Average							**7.79**

Table 3.13: POS errors for each of our systems that are used in the ensemble system for English. We also include the POS error distribution for our best ensemble system. All POS errors are calculated using the testing data, section 23 of the WST. The ensemble system that generated these errors was parameterized on tuning data, section 22 of the WSJ. We only display a reduced set of POS tags for space but the Average is over all POS tags including those not shown.

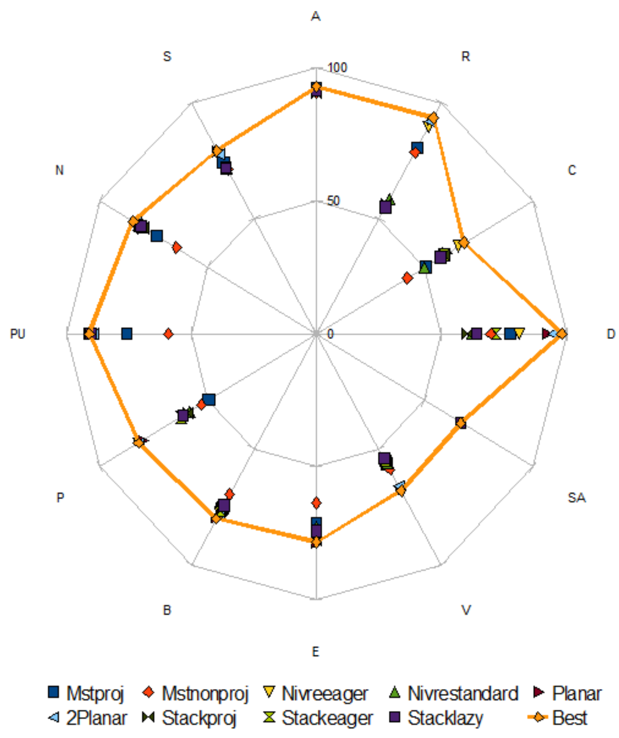


Figure 3.5: We can visually see how the ensemble system reduces POS errors across each POS. The line connects the best ensemble system for Italian on each of its POS tags

POS	**Mst proj**	**Mstnon proj**	**Nivre eager**	**Nivre standard**	**Planar**	**2Planar**	**Stack proj**	**Stack eager**	**Stack lazy**	**Best ensemble**	**Relative Error Reduction**
A	92.33	90.67	91.67	91.67	92	92	91	91.67	91.67	93	8.7%
S	74.55	70.98	78.62	71.56	79.13	78.11	72	72.36	72	79.56	2.09%
N	73.78	64.63	82.32	80.49	81.71	83.54	79.88	82.32	81.1	84.76	7.41%
PU	75.68	59.1	90.63	89.67	90.08	89.13	90.63	90.22	90.22	90.9	2.9%
P	49.76	53.17	81.95	58.54	80	81.95	59.51	62.44	61.46	81.95	0%
B	75.63	69.54	77.66	76.14	79.7	78.17	75.13	77.16	74.11	80.2	2.5%
E	71.32	63.38	78.1	72.47	78.23	77.46	76.18	74.9	74.14	78.23	0%
V	56.45	58.87	67.8	56.31	67.66	66.1	55.04	55.18	54.18	67.8	0%
SA	66.67	66.67	66.67	66.67	66.67	66.67	66.67	66.67	66.67	66.67	0%
D	77.36	69.81	81.13	62.26	92.45	94.34	60.38	71.7	64.15	98.11	66.67%
C	50.3	42.01	65.68	49.7	60.36	58.58	59.17	59.17	57.4	68.05	6.9%
R	80.79	78.82	89.66	58.37	92.86	91.87	56.16	54.93	54.93	93.6	10.34%
Avg	70.38	65.64	79.32	69.49	80.07	79.83	70.14	71.56	70.17	81.9	**8.96%**

Table 3.14: POS errors for Italian and its relative error reduction

Next in Table 3.14 we look at the relative POS error reduction rate and its average across all parts-of-speech. Table 3.14 indicates that while the POS clustering ensemble system did not perform better than the best overall system, it did reduce error on an edge by edge level in terms of POS error. This indicates that locally the system makes better decisions but the overall structure of the parse tree is incorrect. To correct this we must look at combining POS clustering with an ensemble method that will favor an overall structure.

We have shown a means to model different parsers' error distribution based on part-of-speech through fuzzy clustering. Using this distribution, we have weighted our ensemble parser to distribute an edge's weight across clusters and across models. This has reduced the dependency errors on a part-of-speech level however, our scores are still a good amount lower than the possible oracle. We think this is primarily because this method makes correct decision on a node by node level but fails to achieve the larger structure. This makes sense since we are only modeling errors on the parts-of-speech tag. To see if can get a better overall structure, we now look at a more describe ensemble system in which only one parser determines the edge.

3.2 Model Classification

Here we will use a meta-classifier to select which model will choose each node's parent. A dependency tree in this situation may be made up of many different parsers but each node will only be determined by one parser each. This is contrary to the previous section where each node took input and weights from each individual parser. Such a meta-classifier can be based on a number of features such as:

- POS: Certain parsers are better at certain POS. More importantly some parsers are particularly bad at certain POS tags.
- Index in Sentence: Some models maybe be better at predicting tokens at the beginning of the sentences (low index number) such as MaltParser which uses a greedy algorithm. So not to create sparse features, we should break this into buckets. For instance the index is in the first quartile, second quartile and so on in the sentence. Either that or we could use discrete buckets for example the token is in 0:3 or 4:7. Both would reduce sparsity. Possibly these should be different features since one is a discrete position and one is a relative position.
- Sentence Length: Like in the previous case this is based on some parsers being better on shorter sentences due to error propagation caused by greedy algorithms. However, the previous one bases it on a token by token decision. This will decide based on the entire sentence length, not just the current token index.
- Model Agreement: We use a binary feature for each possible model combination on whether or not they agree on a dependency edge. This feature will be our main focus as it is language independent.

We have the immediate problem in this section of the data being multi-label in nature. For instance for a particular feature set (POS, Sentence length) it is very likely that multiple parsers have the correct edge in the predicted SVM training data. For this reason we need to use Multi-label classifier techniques.

There are algorithms and software sets to address this problem. There are also techniques to transform a multi-label training set into a single label set.

This section should bridge between using ensemble (little pieces of each parser) to discrete choices (letting one parser select the head of a node). The ensemble selection should work similarly to the section above in selecting the final tree from a graph structure. However, using a discrete choice brings new complications. If we use a classifier on each node and this allows each node to be selected by a different parser possibly. It is very likely that we will have cycles and more importantly we will have a disconnected graph. In this case, we will need to augment how we select the final tree and look at techniques such as finding random forests.

3.2.1 SVM Based Parser Classification

Morphologically rich languages are often short on training data or require much higher amounts of training data due to the increased size of their lexicon. This section examines a new approach for addressing morphologically rich languages with little training data to start.

Using Tamil as our test language, we create 9 dependency parse models with a limited amount of training data. Using these models we train an SVM classifier using only the model agreements as features. We use this SVM classifier on an edge by edge decision to form an ensemble parse tree. Using only model agreements as features allows this method to remain language independent and applicable to a wide range of morphologically rich languages.

We show a statistically significant 5.44% relative improvement over the average dependency model and a statistically significant 0.52% relative improvement over the best individual system.

3.2.1.1 Process Flow

When dealing with small data sizes, it is often not enough to show a simple accuracy increase. This increase can be very reliant on the training/tuning/testing data splits as well as the sampling of those sets. For this reason our experiments are conducted over 8 training/tuning/testing data split configurations. For each

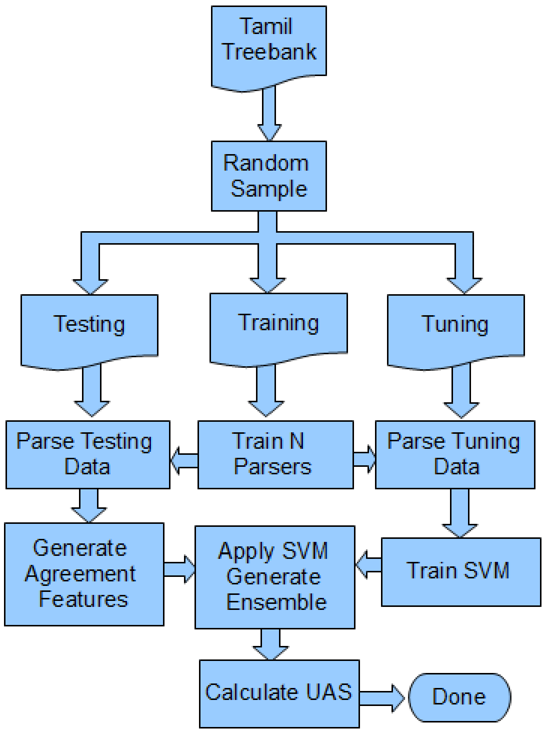


Figure 3.6: Process Flow for one run of our SVM ensemble system. This Process in its entirety was run 100 times for each of the 8 data set splits.

configuration we randomly sample without replacement the training/tuning/testing data and rerun the experiment 100 times. These 800 runs, each on different samples, allow us to better show the overall effect on the accuracy metric as well as the statistically significant changes as described in Section 3.2.1.4. Figure 3.6 shows this process flow for one run of these experiments.

3.2.1.2 Parsers

For this section, we generate two models using MSTParser (38), one projective and one non-projective to use in our ensemble system. Additionally we generate many transition-based parsers. We make use of MaltParser (42), which in the CoNLL shared tasks was often tied with the best performing systems. For this parser, we generate 7 different models using different training parameters and use them as input into our ensemble system along with the two graph-based models described above. These parsing algorithms are enumerated in Table 3.1

3.2.1.3 Ensemble SVM System

We train our SVM classifier using only model agreement features. Using our tuning set, for each predicted dependency edge, we create $\binom{N}{2}$ features where N is the number of parsing models. We do this for each model which predicted the correct edge in the tuning data. So for $N = 3$ the first feature would be a 1 if model 1 and model 2 agreed, feature 2 would be a 1 if model 1 and model 3 agreed, and so on. This feature set is novel and widely applicable to many languages since it does not use any additional linguistic tools.

Model	Head Prediction
1	Node 3
2	Node 4
3	Node 2
4	Node 3
5	Node 2

Table 3.15: Node predictions for SVM based meta-classifier

In Table 3.15, we can see an example of 5 parsers and their prediction for a head of a node. Let us say in this example that Model 2 is correct and the correct head is Node 4. We would create a feature sets based on Table 3.16 in which our features would describe a few scenarios. First (10010) would say that when Model 1 and Model 4 agree, Model 2 is correct. Second (01000) when Model 2 disagrees with everyone, Model 2 is correct. Third (00101) when Model 3 and Model 5 agree, Model 2 is correct. These are all feature sets that are used in our SVM. For each edge in the ensemble graph, we use our classifier to predict which model should be correct, by first creating the model agreement feature set for the current edge of the unknown test data. The SVM predicts which model should be correct and this model then decides to which head the current node is attached. At the end of all the tokens in a sentence, the graph may not be connected and will likely have cycles. Using a Perl implementation of minimum spanning tree in which each edge has a uniform weight, we obtain a minimum spanning forest, where each subgraph is then connected and cycles are eliminated in order to achieve

	Model 1	Model 2	Model 3	Model 4	Model 5
Model 1	1	0	0	1	0
Model 2	0	1	0	0	0
Model 3	0	0	1	0	1
Model 4	1	0	0	1	0
Model 5	0	0	1	0	1

Table 3.16: SVM Agreement Matrix

a well formed dependency structure. Figure 3.7 gives a graphical representation of how the SVM decision and maximum spanning tree algorithm create a final ensemble parse tree which is similar to the construction used in (14, 19).

3.2.1.4 Evaluation

To test statistical significant, we use Wilcoxon paired signed-rank test. For each data split we have 100 iterations each with different sampling. Each model is compared against the same samples so a paired test is appropriate in this case. We report statistical significance values for $p < 0.01$ and $p < 0.05$.

3.2.1.5 Results and Discussion

For each of the data splits, Table 3.17 shows the percent increase in our SVM system over both the average of the 9 individual models and over the best individual model. As the Table 3.17 shows, our approach seems to decrease in value along with the decrease in tuning data. In both cases when we only used 5% tuning data we did not get any improvement in our average UAS scores. Examining Table 3.18, shows that the decrease in the 90-5-5 split is not statistically significant however the decrease in 85-5-10 is a statistically significant drop. However, the increases in all data splits are statistically significant except for the 60-20-20 data split.

It appears that the size of the tuning and training data matter more than the size of the test data. Given that the TamilTB is relatively small (see Table 2.2) when compared to other CoNLL treebanks, we expect that this ratio may shift

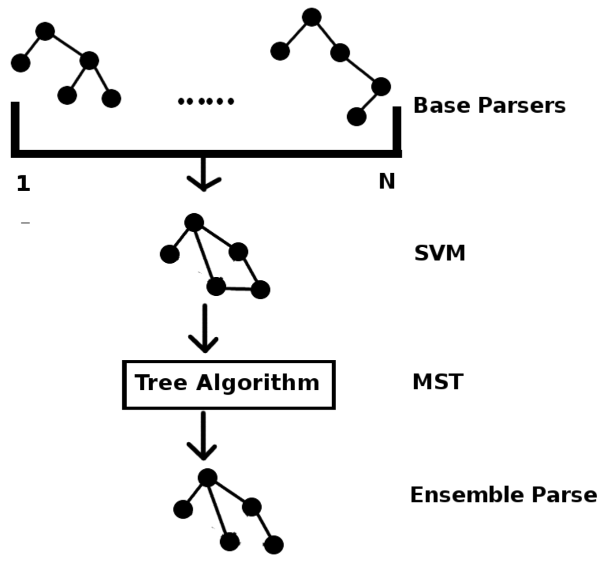


Figure 3.7: General flow to create an Ensemble parse tree for a discrete SVM selection

Data Split	Average SVM UAS	% Increase over Avg	% Increase over Best
70-20-10	76.50%	5.13%	0.52%
60-20-20	76.36%	5.68%	0.72%
60-30-10	75.42%	5.44%	0.52%
60-10-30	75.66%	4.83%	0.10%
85-5-10	75.33%	3.10%	-1.21%
90-5-5	75.42%	3.19%	-1.10%
80-10-10	76.44%	4.84%	0.48%

Table 3.17: Average increases and decreases in UAS score for different Training-Tuning-Test samples. The average was calculated over all 9 models while the best was selected for each data split

more when additional data is supplied since the amount of out of vocabulary, OOV, words will decrease as well. As OOV words decrease, we expect the use of additional test data to have less of an effect.

The traditional approach of using as much data as possible for the training does not seem to be as effective as partitioning more data for tuning an SVM. For instance, the high test training percentage we use is 90% applied to training with 5% for tuning and testing each. In this case the best individual model had a UAS score 76.25% and the SVM had a UAS of 75.42%. One might think using 90% of the data would achieve a higher overall UAS score than using less training data. On the contrary, we achieve a better UAS score on average using only 60%, 70%, 80%, and 85% of the data towards training. This additional data spent for tuning appears to be worth the cost.

Model	70-20-10	60-20-20	60-30-10	60-10-30	85-5-10	90-5-5	80-10-10
2planar	*	*	*	*	*	*	**
mstnonproj	*	*	*	*	*	*	**
mstproj	*	*	*	*	*	*	**
nivreeager	*	*	*	*	**	x	*
nivrestandard	*	*	**	x	*	*	*
planar	*	*	*	*	*	*	**
stackeager	*	*	*	x	*	**	*
stacklazy	*	*	*	x	*	**	*
stackproj	**	*	*	x	**	**	**

Table 3.18: Statistical Significance Table for different Training-Tuning-Test samples. Each experiment was sampled 100 times and Wilcoxon Statistical Significance was calculated for our SVM model's increase/decrease over each individual model. $* = p < 0.01$, $* * p =< 0.05$, $x = p \geq 0.05$

To further examine the tuning/training data trade off, we turn to a new language to see if the results are replicated. For this we will look at Indonesian.

For each of the data splits, Table 3.19 shows the percent increase in our SVM system over both the average of the 7 individual models and over the best individual model. As the Table 3.19 shows, we obtain above average UAS scores

Data Split	Average SVM UAS	% Increase over Average	% Increase over Best	Statistical Significant
50-40-10	60.01%	10.65%	4.34%	Y
60-30-10	60.28%	10.35%	4.41%	Y
70-20-10	62.25%	10.10 %	3.70%	Y
80-10-10	60.88%	8.42%	1.94%	Y
50-30-20	61.37%	9.73%	4.58%	Y
60-20-20	62.39%	9.62%	3.55%	Y
70-10-20	62.48%	7.50%	1.90%	Y
50-20-30	61.71%	9.48%	4.22%	Y
60-10-30	62.57%	7.89%	2.47%	Y
90-5-5	60.85%	0.56%	0.56%	N
85-10-5	61.15%	0.56%	0.56%	Y
80-15-5	59.23%	0.54%	0.54%	Y
75-20-5	60.32%	0.54%	0.54%	Y
70-25-5	59.54%	0.54%	0.54%	Y
65-30-5	59.76%	0.54%	0.54%	Y
60-35-5	59.31%	0.53%	0.53%	Y
55-40-5	57.27%	0.50%	0.50%	Y
50-45-5	57.72%	0.51%	0.51%	Y

Table 3.19: Average increases and decreases in UAS score for different Training-Tuning-Test samples. The average was calculated over all 7 models while the best was selected for each data split. Each experiment was sampled 100 times and Wilcoxon Statistical Significance was calculated for our SVM model's increase/decrease over each individual model. $Y = p < 0.01$ and $N = p \geq 0.01$ for all models in the data split

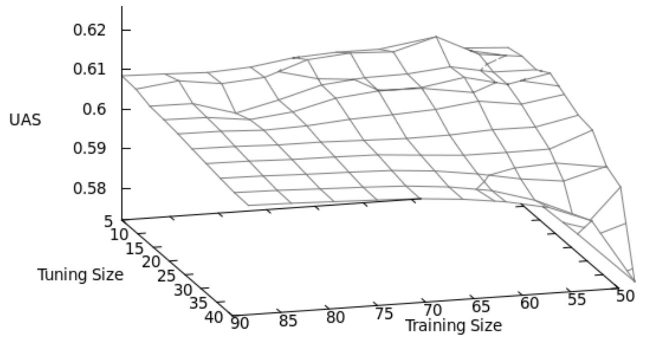


Figure 3.8: Surface plot of the UAS score for the tuning and training data split.

in every data split. The increase is statistical significant in all data splits except one, the 90-5-5 split. This seems to be logical since this data split has the least difference in training data between systems, with only 5% tuning data. Our highest average UAS score was with the 70-20-10 split with a UAS of 62.48%. The use of 20% tuning data is of interest since it was significantly better than models with 10%-25% more training data as seen in Figure 3.8. This additional data spent for tuning appears to be worth the cost.

The selection of the test data seems to have caused a difference in our results. While all our ensemble SVM parsing systems have better UAS scores, it is a lower increase when we only use 5% for testing. Which in our treebank means we are only using 5 sentences randomly selected per experiment. This does not seem to be enough to judge the improvement.

We have shown a new SVM based ensemble parser that uses only dependency model agreement features. The ability to use only model agreements allows us to keep this approach language independent and applicable to a wide range of morphologically rich languages. We show a statistically significant 5.44% improvement over the average dependency model and a statistically significant 0.52% improvement over the best individual system for Tamil. Additionally we reproduce the results on Indonesian with an improvement on individual accuracy of 4.92% on average.

We believe this methodology to be an improvement over the fixed weight ensemble system as it allows under-resourced languages to quickly retrain and change dependency parsers when they might not know which algorithm is best. Additionally since it is a discrete choice it is far less prone to one parser taking over the weighting, unless that parser was the only accurate parser on the tuning data. Next we will further show its use for under-resourced languages by examining the used of the ensemble SVM system when combined with self-training.

3.3 Self Training

Manual dependency annotation is very time consuming and costly. While these annotations exist for some of the larger treebanks, the cost of annotation is prohibitive for under-resourced languages. For this reason, semi-supervised approaches are an appropriate direction. Researchers can use models trained on less amounts of data to annotate unseen data. The cost of fixing some errors in the resulting parse, is in most cases, less costly then starting from scratch.

When using a parsing model as a pre-processing tool, any improvement in accuracy, reduces the work needed to be done by the annotator. We examine the use of an ensemble dependency parser, which uses a variety of trained models, to create a more accurate parse. This is done with the use of an SVM classifier that requires no linguistic information about the particular language, it only needs to know which models agree with each other. This makes it a prime candidate for use in under-resourced languages, where the linguistic tools may not be available.

3.3.1 Methodology

The following methodology was run 12 independent times. Each time new testing/tuning/training data sets were randomly selected without replacement. In each iteration the SVM classifier and dependency models were retrained using self-training. Also for each of the 12 experiments, new random self-training datasets were selected from the larger corpus. The results in the next section are averaged amongst these 12 independent runs. Figure 3.9 shows this process flow for one run of these experiments.

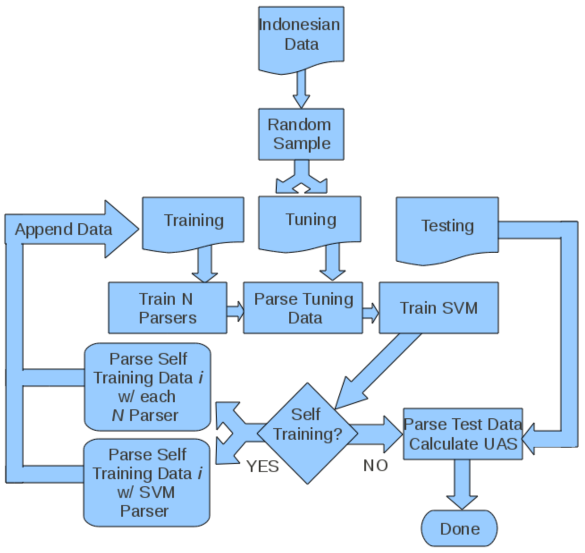


Figure 3.9: Process Flow for one run of our self-training system. There is one alternative scenario in which the system either does self-training with each N parser or with the ensemble SVM parser. These constitute two different experiments. For all experiments i=10 and N=7

3.3.2 Parsers

For Indonesian we do not have access to a constituency to dependency transformation which limits us from using established constituent to dependency transformation parsing techniques. We showed the result using two parsing techniques with the Tamil data so for this we artificially limit ourselves to one. Because of this, we only use MaltParser but we use different training parameters to create various parsing models. For MaltParser we use a total of 7 model variations which are enumerated in Table 3.1.

3.3.2.1 Self-training Scenarios

For self-training we will mainly be testing our ensemble system. We have chosen to use the SVM ensemble system, as we feel is it best for under-resourced languages. We train our SVM classifier using only model agreement features in a method similar to Section 3.2.1.

The data for self-training is also taken from IDENTIC and it consists of

45,000 sentences. The data does not contain any dependency relation information but it is enriched with POS tags. It is processed with the same morphology tools as the training data described in section 2.3.8 but without the manual disambiguation and correction. This data and its annotation information are available on IDENTIC homepage[1].

For self-training we present two scenarios. First, all parsing models are retrained with their own predicted output. Second, all parsing models are retrained with the output of our SVM ensemble parser. Self-training in both cases is done over 10 iterations of 20 sentences. Sentences are chosen at random from unannotated data of size 45,000 sentences. This allows us to examine self-training to a training data size of twice the original set.

3.3.3 Results

Without self-training, the SVM ensemble model outperformed all other baseline models achieving an average score of 62.3%. The other models average 57.4%. The ability to use an SVM agreement model as opposed to the best base model, has the advantage of allowing the SVM models to weight the parsers in each iteration. This eliminates the problem of languages where the annotated resources are very scarce and it is hard to determine the best model from the first 100 sentences. The iterative process allows for the best model to change and for the SVM model to take advantage of this new and prior information.

As can be seen in Figure 3.10, the base models did better when trained with additional data that was parsed by our SVM ensemble system. The higher UAS accuracy seems to of had a better effect than receiving dependency structures of a similar nature to the current model. We show the 2Planar model in Figure 3.10 but this was the case for each of the 7 models. On an interesting note, the SVM system had least improvement, 0.60%, when the component base models were trained on its own output. This seems warranted as other parser combination papers have shown that ensemble systems prefer models which differ more so that a clearer decision can be made (14, 19). The improvements when self-training on our SVM output over the individual parsers' output can be seen in Table

[1]http://ufal.mff.cuni.cz/~larasati/identic/index.html

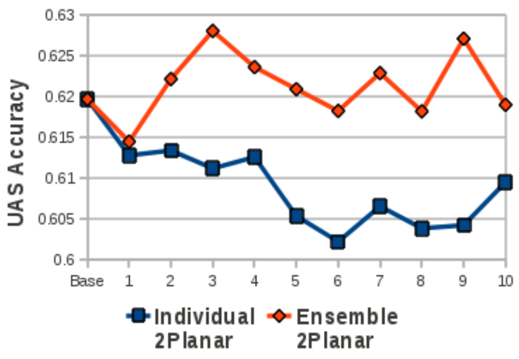


Figure 3.10: We can see that the self-trained 2Planar model that is trained with the ensemble output consistently outperforms the self-trained model that uses its own output. Results are graphed over the 10 self-training iterations

3.20. Again these are averages over 12 runs of the system, each run containing 10 self-training loops of 20 additional sentences.

Model	% Improvement %
2planar	1.10%
nivreeager	0.40%
nivrestandard	1.62%
planar	0.87%
stackeager	2.28%
stacklazy	2.20%
stackproj	1.95%
svm	0.60%

Table 3.20: The % Improvement of all our parsing models including our ensemble svm algorithm over 12 complete iterations of the experiment.

We have shown a successful implementation of self-training for dependency parsing on an under-resourced language. Self-training in order to improve our parsing accuracy can be used to help semi-supervised annotation of additional data. We show this for an initial data set of 100 sentences and an additional self-trained data set of 200 sentences. We introduce and show a collaborative SVM classifier that creates an ensemble parse tree from the predicted annotations and improves individual accuracy on average of approximately 5%. This additional accuracy can release some of the burden on annotators for under-resourced language annotation who would use a dependency parser as a pre-annotation tool. Using these semi-supervised annotation techniques should be applicable to many languages since the SVM classifier is essentially blind to the language and only considers the models' agreement. Most importsntly we show that self-training with multiple models is better when the self-training data is the result of the ensemble system as opposed to self-training with each model's own data.

4

The Pipeline: Dependency Parsing's Effect on Machine Translation

The previous chapter showed various useful combinations of parsers. However, we feel the evaluation should not end there. In this chapter we aim to show a motivation for further evaluation of dependency parsing. To do this we will show that noun phrase structure in a parser will effect a syntax-based machine translation system. Following that we see how the structures from our ensemble parser help or hurt machine translation. Much like we had an oracle score for our ensemble parsing experiment, we need a gold standard for machine translation. To fully test the effects if dependency trees on syntax-based machine translation, we present experiments with hand annotated machine translation data. Using these hand annotations we will examine whether we can improve our previous baseline and ensemble systems.

4.1 Deep Noun Phrase Structure in MT

Flat noun phrase structure was, up until recently, the standard in annotation for the Penn Treebank. With the recent addition of internal noun phrase annotation, dependency parsing and applications down the NLP pipeline are likely affected.

Some machine translation systems, such as TectoMT, use deep syntax as a language transfer layer. It is proposed that changes to the noun phrase dependency parse will have a cascading effect down the NLP pipeline and in the end, improve machine translation output, even with a reduction in parser accuracy that the noun phrase structure might cause. This section examines this noun phrase structure's effect on dependency parsing, in English, with a maximum spanning tree parser and shows a 2.43%, 0.23 BLEU score, improvement for English to Czech machine translation.

4.1.1 Introduction

Noun phrase structure in the Penn Treebank has up until recently been only considered, due to underspecification, a flat structure. Due to the annotation and work of Vadas and Curran (56, 57, 59), we are now able to create Natural Language Processing (NLP) systems that take advantage of the internal structure of noun phrases in the Penn Treebank. This extra internal structure introduces additional complications in NLP applications such as parsing.

Dependency parsing made many improvements due to the CoNLL X shared task (3). However, in most cases, these systems were trained with a flat noun phrase structure in the Penn Treebank. Vadas' internal noun phrase structure has been used in previous work on constituent parsing using Collin's parser (58), but has yet to be analyzed for its effects on dependency parsing.

Parsing is very early in the NLP pipeline. Therefore, improvements in parsing output could have an improvement on other areas of NLP in many cases, such as machine translation. At the same time, any errors in parsing will tend to propagate down the NLP pipeline. One would expect parsing accuracy to be reduced when the complexity of the parse is increased, such as adding noun phrase structure. But, for a machine translation system that is reliant on parsing, the new noun phrase structure, even with reduced parser accuracy, may yield improvements due to a more detailed grammatical structure. This is particularly of interest for dependency relations, as it may aid in finding the correct head of a term in a complex noun phrase.

This section examines the results and errors in parsing and machine translation of dependency parsers, trained with annotated noun phrase structure, against those with a flat noun phrase structure. The results also serve as motivation for our further research into dependency annotations and machine translation. These results are compared with two systems: a Baseline Parser with no internally annotated noun phrases and a Gold NP Parser trained with data which contains gold standard internal noun phrase structure annotation. Additionally, we analyze the effect of these improvements and errors in parsing down the NLP pipeline on the TectoMT machine translation system (60).

4.1.1.1 Dependency Parsers

As noted in Section 2.3.7, we expect each parser to have different errors handling internal noun phrase structure, but for this experiment we will only be examining the globally trained MSTParser. This will allow us to examine one set of machine translation experiments with one variable.

4.1.1.2 TectoMT

As described in Section 2.4.3 TectoMT is a modular framework built in Perl. This allows great ease in adding the two different parsers into the framework since each experiment can be run as a separate "Scenario" comprised of different parsing "Blocks". This allows a simple comparison of two machine translation system in which everything remains constant except the dependency parser.

Scenario for Baseline Parser	Scenario Gold NP Parser
Penn_To_Dep	Vadas NP Script
Sentence_Split	Penn_To_Dep_Gold_NP
Tokenize	Sentence_Split
MSTParser	Tokenize
	MSTParser

Table 4.1: Example of Scenarios with different Blocks (47)

4.1.1.3 Noun Phrase Structure

The Penn Treebank is one of the most well known English language treebanks (35), consisting of annotated portions of the Wall Street Journal. Much of the annotation task is painstakingly done by annotators in great detail. Some structures are not dealt with in detail, such as noun phrase structure. Not having this information makes it difficult to tell the dependencies on phrases such as "crude oil prices" (58). Without internal annotation it is ambiguous whether the phrase is stating "crude prices" (crude (oil prices)) or "crude oil" ((crude oil) prices).

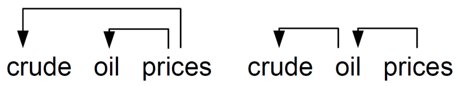


Figure 4.1: Ambiguous dependency caused by internal noun phrase structure.

Manual annotation of these phrases would be quite time consuming and as seen in the example above, sometimes ambiguous and therefore prone to poor inter-annotator agreement. Vadas and Curran have constructed a Gold standard version Penn treebank with these structures. They were also able to train supervised learners to an F-score of 91.44% (56, 57, 59). The additional complexity of noun phrase structure has been shown to reduce parser accuracy in Collin's parser but no similar evaluation has been conducted for dependency parsers. The internal noun phrase structure has been used in experiments prior but without evaluation with respect to the noun phrases (11).

4.1.2 Methodology

The Noun Phrase Bracketing experiments consist of a comparison two systems.

1. The Baseline system is McDonald's MSTParser trained on the Penn Treebank in English converted to dependencies without any extra noun phrase bracketing.

2. The Gold NP Parser is McDonald's MSTParser trained on the Penn Treebank in English with gold standard noun phrase structure annotations (56).

4.1.2.1 Data Sets

To maintain a consistent dataset to compare to previous work we use the Wall Street Journal (WSJ) section of the Penn Treebank since it was used in the CoNLL X shared task on dependency parsing (3). Using the same common breakdown of datasets, we use WST section 02-21 for training and section 22 for testing, which allows us to have comparable results to previous works. To test the effects of the noun phrase structure on machine translation, ACL 2008's Workshop on Statistical Machine translation's (WMT) data are used.

4.1.2.2 Process Flow

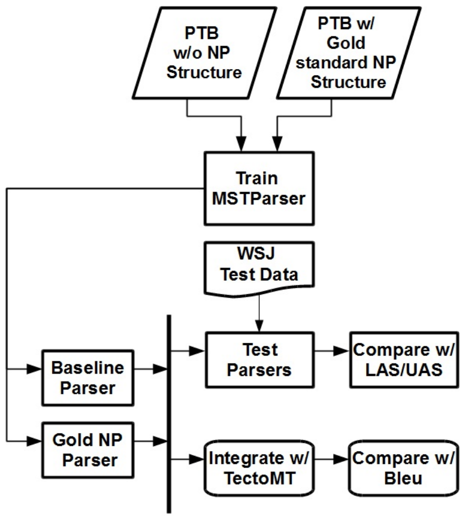


Figure 4.2: Experiment Process Flow. PTB (Penn Tree Bank), NP (Noun Phrase Structure), LAS (Labeled Accuracy Score), UAS (Unlabeled Accuracy Score), Wall Street Journal (WSJ)

We begin the experiments by constructing two data sets:

1. The Penn Treebank with no internal noun phrase structure (PTB w/o NP structure).
2. The Penn Treebank with gold standard noun phrase annotations provided by Vadas and Curran (PTB w/ gold standard NP structure) and then converted to dependencies using the PennConverter.

From these datasets we construct two separate parsers. These parsers are trained using McDonald's Maximum Spanning Tree Algorithm (MSTParser) (39).

Both of the parsers are then tested on a subset of the WSJ corpus, section 22, of the Penn Treebank and the UAS and LAS scores are generated. Errors generated by each of these systems are then compared to discover where the internal noun phrase structure affects the output. Parser accuracy is not necessarily the most important aspect of this work. The effect of this noun phrase structure down the NLP pipeline is also crucial. For this, the parsers are inserted into the TectoMT system.

4.1.2.3 Metrics

This experiment compares the two parsing systems against each other using both UAS and BLEU scores. In both cases the test set data is sampled 1,000 times without replacement to calculate statistical significance using a pairwise comparison.

4.1.3 Results and Discussion

When applied, the gold standard annotations changed approximately 1.5% of the edges in the training data. Once trained, both parsers were tested against section 22 of their respective annotated corpora. As Table 4.2 shows, the Baseline Parser obtained near identical LAS and UAS scores. This was expected given the additional complexity of predicting the noun phrase structure and the previous work on noun phrase bracketing's effect on Collin's parser.

Systems	LAS	UAS
Baseline Parser	88.12%	91.11%
Gold NP Parser	88.10%	91.10%

Table 4.2: Parsing results for the Baseline and Gold NP Parsers. Each is trained on Section 02-21 of the WSJ and tested on Section 22

While possibly more error prone, the 1.5% change in edges in the training data did appear to add more useful syntactic structure to the resulting parses as can be seen in Table 4.3. With the additional noun phrase bracketing, the resulting BLEU score increased 0.23 points or a 2.43%. The improvement is statistically significant with 95% confidence using pairwise bootstrapping of 1,000 test sets randomly sampled with replacement (26, 62). In Figure 4.3 we can see that the difference between each of the 1,000 samples was above 0, meaning the Gold NP Parser performed consistently better in each sample as shown in Table 4.3.

Systems	BLEU
Baseline Parser	9.47
Gold NP Parser	**9.70**

Table 4.3: TectoMT results of a complete system run with both the Baseline Parser and Gold NP Parser. Both are tested on WMT08 data. Results are an average of 1,000 bootstrapped test sets with replacement.

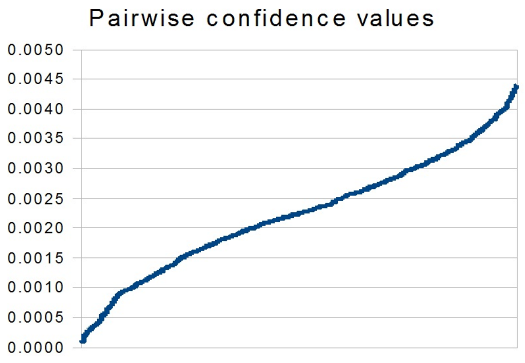


Figure 4.3: The Gold NP Parser shows statistically significant improvement with 95% confidence. The difference in BLEU score is represented on the Y-axis and the bootstrap iteration is displayed on the X-axis. The samples were sorted by the difference in BLEU score.

Visually, changes can be seen in the English side parse that affect the overall translation quality. Sentences that contained an incorrect noun phrase structure such as “The second vice-president and Economy minister, Pedro Solbes” as seen in Figure 4.4 and Figure 4.5 were more correctly parsed in the Gold NP Parser. In Figure 4.4 “and” is incorrectly assigned to the bottom of a noun phrase and does not connect any segments together in the output of the Baseline Parser,

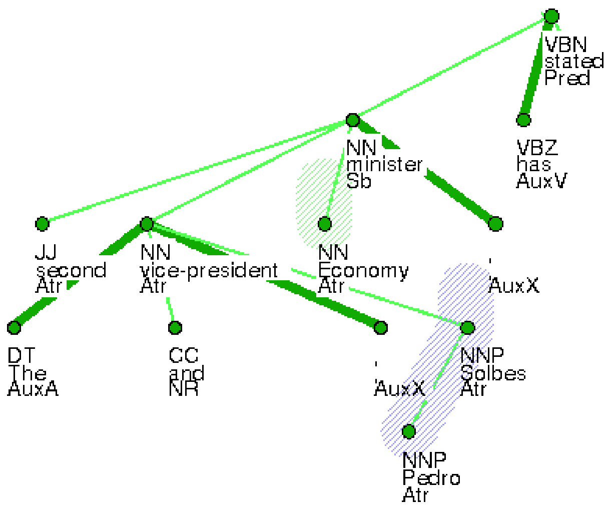


Figure 4.4: The parse created with the data with flat structures does not appear to handle noun phrases with more depth, in this case the 'and' does not properly connect the two components.

while it connects two phrases in Figure 4.5 which is the output of the Gold NP Parser. This shift in bracketing also allows the proper noun, which is shaded, to be assigned to the correct head, the rightmost noun in the phrase.

This section has demonstrated the benefit of additional noun phrase bracketing in training data for use in dependency parsing and machine translation. Using the additional structure, the dependency parser's accuracy was minimally reduced. Despite this reduction, machine translation, much further down the NLP pipeline, obtained a 2.43% jump in BLEU score and is statistically significant with 95% confidence. Future work should examine similar experiments with MaltParser and other machine translation systems.

This section gives additional motivation for further examination of annotation structure. With the success of noun structure, we now look at whether the constructions formed by our ensemble parsers will help or hurt machine translation.

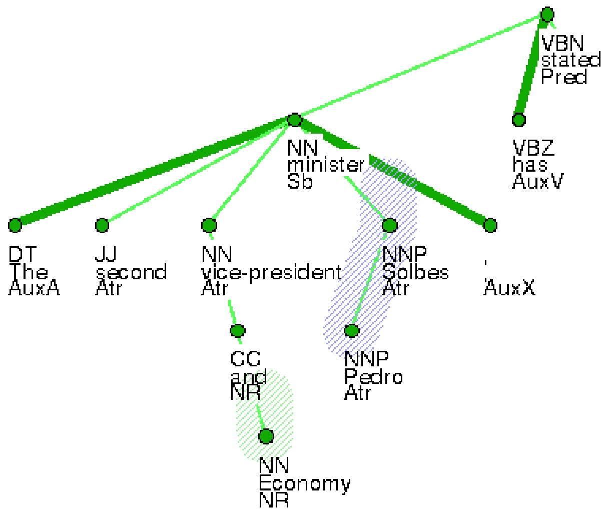


Figure 4.5: With the addition of noun phrase structure in parser, the complicated noun phrase appears to be better structured. The "and" connects two components instead of improperly being a leaf node.

4.2 The Effects of Hybrid Ensemble Parsers on MT

Given the success for using noun phrase structures, we now want to look at two things. One, how our ensemble structures effect machine translation and two, how a gold standard parse will effect the translation result.

4.2.1 Annotation Style

To find the maximum effect that dependency parsing can have on the NLP pipeline, we annotated English dependency trees to form a gold standard. Annotation was done with two annotators using a tree editor, TrEd (44), on data that was preprocessed using MSTParser. For the annotation of our gold data, we used the standard annotation standards described in the Prague Dependency Treebank (PDT) (18). PDT is annotated on three levels, morphological, analytical, and tectogrammatical. For our gold data, we do not touch the morphological layer, we only correct the analytical layer (i.e. labeled dependency trees). For machine translation experiments later in the chapter, we allow the system to automatically generate a new tectogrammatical layer based on our new analytical layer annotation. Because the Treex machine translation system uses a tectogrammatical layer, when in doubt, ambiguity was left to the tectogrammatical (t-layer in Figure 2.8) to handle.

4.2.2 Data Sets

Evaluation Set

For the annotation experiments, we use text provided by the 2012 Workshop for Machine Translation (WMT2012). The data consists of 3,003 sentences. We automatically tokenized, tagged, and parsed these sentences. This data set was also chosen since it is disjoint from the usual dependency training data, allowing researchers to use it as a out-of-domain testing set. The parser used is an implementation of MSTParser. We then hand corrected the analytical trees to have a "Gold" standard dependency structure. Analytical trees were annotated on the PDT standard. Most manual corrections involved coordination construction along with prepositional phrase attachment.

Having only two annotators has limited us to evaluating our annotation only through spot checking and through comparison with other baselines. Annotation happened sequentially one after another. Possible errors were additionally detected through a set of automatic tests. As a comparison we will evaluate our gold data set versus other parsers in respect to their performance on previous data sets, namely the Wall Street Journal (WSJ) section 23.

Training Set

All the parsers were trained on sections 02-21 of the WSJ converted to dependencies using the PennConverter, except the Stanford parser which also uses section 01. We retrained MST and Malt parsers and used pre-trained models for the other parsers. Machine translation data was used from WMT 2010, 2011, and 2012. Using our gold standard we are able to evaluate the effectiveness of different parser types from graph-base, transition-based, constituent conversion to ensemble approaches on the 2012 data while finding data trends using previous years data.

4.2.3 Translation Components

To examine the effects of dependency parsing down the NLP pipeline, we now turn to syntax based machine translation. Our dependency models will be evaluated using the TectoMT translation system (47). This system, as opposed to other popular machine translation systems, makes direct use of the dependency structure during the conversion from source to target languages via a tectogrammatical tree translation approach.

We use the different parsers in separate translation runs each time in the same Treex parsing block. So each translation scenario only differs in the parser used and nothing else. As can be seen in Figure 2.8, we are directly manipulating the Analytical portion of Treex. As seen in previous Ensemble papers (9, 12, 13, 14, 61) and in the previous chapter, parsing accuracy can be improved by combining parsers' outputs for a variety of languages. We apply a few of these systems to English using models trained for both dependencies and constituents. The parsers used are as follows:

- **MST**: Implementation of Ryan McDonald's Minimum spanning tree parser (39)
- **MST with chunking**: Same implementation as above but we parse the sentences based on chunks and not full sentences. For instance this could mean separating parentheticals or separating appositions (46)
- **Malt**: Implementation of Nivre's MaltParser trained on the Penn Treebank (40)
- **Malt with chunking**: Same implementation as above but with chunked parsing
- **ZPar**: Yue Zhang's statistical parser. We used the pretrained English model (english.tar.gz) available on the ZPar website for all tests (63)
- **Charniak**: A constituent based parser (ec50spfinal model) in which we transform the results using the PennConverter (22)

- **Stanford**: Another constituent based parser (24) whose output is converted using PennConverter as well (wsjPCFG.ser.gz model)
- **Fixed Weight Ensemble**: A stacked ensemble system combining five of the parsers above (MST, Malt, ZPar, Charniak, Stanford). The weights for each tree are assigned based on UAS score (14)
- **Fuzzy Cluster**: A stacked ensemble system as well but weights are determined by a cluster analysis of POS errors (15)
- **SVM**: An ensemble system in which each individual edge is picked by a meta classifier from the same 5 parsers as the other ensemble systems (12, 13).

4.2.4 Evaluation

For machine translation, we report two automatic evaluation scores, BLEU and NIST. We examine parser accuracy using UAS. We compare a machine translation system, integrating 10 different parsing systems, against each other using these metrics. We report UAS scores for each parser on section 23 of the WST and BLEU and NIST scores for the WMT test set in Table 4.4.

4.2.5 Results and Discussion

Type of Changes in WMT Annotation

Since our gold annotated data was preprocessed with MSTParser, our baseline system at the time, we started with a decent baseline and only had to change 9% of the dependency arcs in the data. These 9% of changes roughly increases the BLEU score by 7%.

Parsers vs our Gold Standard

On average, our gold data differed in head agreement from our base parser 14.77% of the time. When our base parsers were tested on the WSJ section 23 data they had an average error rate of 12.17% which is roughly comparable to the difference with our gold data set which indicates overall our annotations are close to the accepted standard from the community. The slight difference in percentage fits into what is expect in annotator error and in the errors in the conversion process of the WSJ by PennConverter.

MT Results in WMT with Ensemble Parsers

- **WMT 2010**: As seen in Table 4.4, the highest resulting BLEU score for the 2010 data set is from the fixed weight ensemble system. The other two ensemble systems are beaten by one component system, Charniak. However, this changes when comparing NIST scores. Two of the ensemble method have higher NIST scores than Charniak, similar to their UAS scores.

- **WMT 2011**: The 2011 data corresponded the best with UAS scores. While the BLEU score increases for all the ensemble systems, the order of systems by UAS scores corresponds exactly to the systems ordered by NIST score and corelates strongly (Table 4.5). Unlike the 2010 data, the MSTParser was the highest base parser.

- **WMT 2012**: The ensemble increases are statistically significant for both the SVM and the Fixed Weight system over the MSTParser with 99% confidence, our previous baseline and best scoring base system from 2011. We examine our data versus MST instead of Charniak since we have preprocessed our gold data set with MST, allowing us a direct comparison in improvements. The fuzzy cluster system achieves a higher BLEU evaluation score than MST, but is not significant. In pairwise tests, it wins approximately 78% of the time. This is the first dataset we have looked at where the BLEU score is higher for a component parser and not an ensemble system, although the NIST score is still higher for the ensemble systems.

Human Manual Evaluation: SVM vs the Baseline System

We selected 200 sentences at random from our annotations and they were given to 7 native Czech speakers. 77 times the reviewers preferred the SVM system,

Parser	**UAS**	**NIST(10/11/12)**	**BLEU(10/11/12)**
MST	86.49	5.4038/5.5898/5.1956	12.99/13.58/11.54
MST w chunking	86.57	5.4364/5.6346/5.2364	13.43/14.00/11.96
Malt	84.51	5.3747/5.5702/5.1484	12.90/13.48/11.27
Malt w chunking	87.01	5.4110/5.6025/5.1904	13.39/13.80/11.73
ZPar	76.06	5.2676/5.4635/5.0846	11.91/12.48/10.53
Charniak	92.08	5.4750/5.6561/5.2816	13.49/13.95/**12.26**
Stanford	87.88	5.4000/5.5970/5.1892	13.23/13.63/11.74
Fixed Weight	92.58	**5.4911**/5.6831/**5.2902**	**13.53**/14.04/12.23
Fuzzy Cluster	92.54	5.4730/5.6820/5.2672	13.47/14.06/12.06
SVM	92.60	5.4846/**5.6837**/5.2891	13.45/**14.11**/12.22

Table 4.4: Scores for each machine translation run for each dataset (WMT 2010, 2011 and 2012 results are given in both columns)

	NIST	BLEU
2010	0.98	0.93
2011	0.98	0.94
2012	0.95	0.97

Table 4.5: Pearson correlation coefficients for each year and each metric when measured against UAS. Overall NIST has a stronger correlation to UAS scores, however both show a strong relationship.

48 times they preferred the MST system, and 57 times they said there was no difference between the quality of the sentences. On average each reviewer looked at 26 sentences with a median of 30 sentences. Reviewers were allowed three options: sentence 1 is better, sentence 2 is better, both sentences are of equal quality.

	+	=	-
+	12	12	0
=		3	7
-			7

Table 4.6: Pairwise agreement between annotators for our SVM and baseline systems. (-,-) all annotators agreed the baseline was better, (+,+) SVM was better, (+,-) annotators disagreed

Table 4.6 indicates that the SVM system was widely preferred. When removing annotations marked as equal, we see that the SVM system was preferred 24 times to the Baseline's 14.

Although a small sample, this shows that using the ensemble parser will at worse give you equal results and at best a much improved result.

MT Results with Gold Data

In the perfect situation of having gold standard dependency trees, we obtained a NIST of 5.3003 and a BLEU of 12.39. For our gold standard system run, the parsing component was removed and replaced with our hand annotated data. These are the highest NIST and BLEU scores we have obtained including using all base parsers or any combinations of parsers.

We have shown that ensemble parsing techniques have an influence on syntax-based machine translation both in manual and automatic evaluation. Furthermore we have shown a stronger correlation between parser accuracy and NIST rather than the more commonly used BLEU metric. We have also introduce a gold set of English dependency trees based on the WMT 2012 machine translation task data, which shows a larger increase in both BLEU and NIST. While on some datasets it is inconclusive whether using an ensemble parser with better accuracy has a large enough effect, we do show that practically you will not do worse using one and in many cases do much better.

4.3 Annotation of the Penn Treebank

The previous sections show that using a different annotation set has a positive change to our translation results. We used Gold data that was hand annotated. This process obviously cannot be expect to be widely used as it is very time consuming. Instead of writing extra post parsing conversion scripts, that may introduce new errors, we decided to retrain our parsings on new data.

Since most parsing experiments are trained on the Penn Treebank, we did the same. Typically the parsers are trained on section 02 through section 21. This turned out to be far too much, and very time expensive to annotate, so instead we focused on one section. We decided to annotate section 23 of the Penn Treebank which is typically used as the test set. With this data complete our intent is to find the best automatic conversion to approximate this annotation scheme and then to apply it to the full Penn Treebank.

4.3.1 Annotation

To annotate section 23 of the Penn Treebank, we followed a similar process as to the WMT data. First, based on previous experiments, we applied Vadas and Curran's deep noun phrase structure corrections. Second, we converted the constituent trees to dependencies using the "-2007" conversion options on the PennConverter, these were the options used in previous ensemble experiments.

Once converted, we annotated all sentences into our Gold annotation standard. A second annotator hand checked various files. Because of the lack of annotators on the project, no kappa or inter-annotator scores were computed. The same style was used as we did in the WMT data, so we expect the same amount of accuracy.

Overall we changed 7.40% of the edges when compared to our standard *conll2007* conversion option. Simular to WMT data most of the changes came from coordination structure, punctuation changes, and multi word expressions

4.3.2 Finding the best conversion from constituent to PDT style

We ran different permutations of the PennConverter's possible options, as seen in Table 4.7, on section 23. Each permutation we compared against our hand annotated corpus. We found that only one flag needed to be set to get closest to our annotations and that was the coordination flag set to Prague style.

As can be seen in Table 4.7, many options gave us improvements but only the coordination had the highest effect. We then ran each of the options in combination with the Prague coordination flag, and at best we received the same accuracy and in many cases the score dropped.

Additionally we had some differences in punctuation style, for this we post processed the data. This mainly consisted of adjusting the end punctuation to match our style. With these changes we can convert new constituent trees at 95% accuracy using PennConverter. We measured this against Section 23 of the Wall Street Journal which contains all our hand annotated annotations.

As we can see from Table 4.11, the PennConverter with the new settings can match our annotations at a much higher accuracy than using the existing parsers. As expected all parsers dropped in accuracy since the annotation standard was changed. The biggest change was in the constituent parsers which make direct use of the PennConverter. Due to this we retrained the parsers, as well, using the new conversion from PennConverter. Using these retrained models we study their effect on machine translation.

Using the retrained models we see we get closer to our standard but our discovered PennConverter options and post processing steps still give us better accuracy. The retrained models in some cases, such as Charniak, have a higher UAS accuracy than the previous base parser (the one without gold annotations). So in the end, we feel this adds more linguistic knowledge to our parses and increases the overall accuracy.

Parser	UAS
baseline	93.05
coordStructure=prague	95.10
posAsHead=true	91.24
prepAsHead=false	75.23
subAsHead=false	90.83
whAsHead=true	91.54
imAsHead=false	90.76
splitSmallClauses=false	92.86
rootLabels=true	93.05
advFuncs=false	93.05
labelCoords=true	93.05
splitSlash=false	7.40
ddtGapping=false	93.02
conll2008clf=false	93.05
conll2008exp=false	93.05
iobj=true	93.05
relinkCyclicPRN=false	92.90
name=false	93.05
suffix=false	93.05
title=false	93.05
posthon=false	91.76
appo=false	90.55
clr=true	93.05
deepenQP=true	92.77
qmod=true	93.05
noPennTags=true	91.47
noPennTags=true	91.47
rightBranching=false	92.98
-2007	92.59

Table 4.7: Conversion options for each of the PennConverter's available options through one level

Parser	UAS
coordStructure=prague -posAsHead=true	93.26
coordStructure=prague -prepAsHead=false	76.93
coordStructure=prague -subAsHead=false	92.89
coordStructure=prague -whAsHead=true	93.60
coordStructure=prague -imAsHead=false	92.79
coordStructure=prague -splitSmallClauses=false	94.90
coordStructure=prague -rootLabels=true	95.10
coordStructure=prague -advFuncs=false	95.10
coordStructure=prague -labelCoords=true	95.10
coordStructure=prague -splitSlash=false	7.11
coordStructure=prague -ddtGapping=false	95.07
coordStructure=prague -conll2008clf=false	95.10
coordStructure=prague -conll2008exp=false	95.10
coordStructure=prague -iobj=true	95.10
coordStructure=prague -relinkCyclicPRN=false	94.95
coordStructure=prague -name=false	95.10
coordStructure=prague -suffix=false	95.10
coordStructure=prague -title=false	95.10
coordStructure=prague -posthon=false	93.74
coordStructure=prague -appo=false	92.49
coordStructure=prague -clr=true	95.10
coordStructure=prague -deepenQP=true	94.82
coordStructure=prague -qmod=true	95.10
coordStructure=prague -noPennTags=true	93.49
coordStructure=prague -noPennTags=true	93.49
coordStructure=prague -rightBranching=false	95.05

Table 4.8: Conversion options for each of the PennConverter's available options through two levels

Parser	**UAS**
MST	82.20
Malt	79.42
Zpar	71.5
Charniak	82.59
Stanford	79.31
PennConverter	95.00%

Table 4.9: UAS scores of each parser measured on out gold data section 23 of PTB

Parser	**UAS**
MST	85.22
Malt	82.57
Zpar	71.57
Charniak	92.22
Stanford	88.07

Table 4.10: UAS scores of each parser measured on our gold data section 23 of PTB after retraining

4.3.3 Effects on Parsers

MST and MaltParser were both retrained using the entire Penn Treebank, sections 02-21. We reconverted the Penn Treebank using the settings and post processing blocks discovered and created from the previous section using PennConverter. We evaluate the newly trained parsers on both our Gold data and the previous standard data set. Parsing models created for both Malt and MST will be tested with chunking varieties as well.

4.3.4 Effects on Machine Translation

We ran the base Treex setup for MST, MaltParser, MST with Chunking, Malt with Chunking, and our Ensemble Methods. Note that Zpar and our constituent parsers are not retrained here so only a portion of the ensemble parser has changed. Due to the change, our SVM also needed to change.

Both Stanford and Charniak parsers have changed results, this is not due to retraining but because we use different PennConverter settings. Charniak and Stanford results are incorporated into our Ensemble SVM scores.

Parser	**NIST (WMT 10/11/12)**	**BLEU (WMT 10/11/12)**
MST	5.40/5.63/5.24	13.33/13.86/11.83
MST with chunk	5.44/5.67/5.27	13.76/14.17/12.27
Malt	5.26/5.50/5.13	13.24/13.65/11.84
Malt with chunk	5.33/5.54/5.17	13.55/13.77/12.03
SVM	5.48/5.71/5.30	13.81/14.35/12.49

Table 4.11: NIST and BLEU using retrained MST and Malt models. Additionally Charniak and Stanford make use of the new conversion parameters

Table 4.11 shows the results for each retrained system and the results of using these new models in our SVM ensemble system. Overall the retraining had a positive effect, when compared to Table 4.4, with a greater effect on MST than on MaltParser. In only one instance did our BLEU score decrease. This was with MaltParser with chunking on 2011 data. We saw that with using WMT gold data in the previous experiments, we obtained a positive increase in NIST scores. Contrary to the Gold data, our retrained parsers only noticeably improved the BLEU score. The NIST score seemed overall unaffected.

To test our SVM system, we used both retrained models and kept in chunking. For all constituent parsers we used the most current discovered tuning parameters, the same MST and Malt are trained on. The SVM parameters were then retrained on Section 22 of the Wall Street Journal, as was done in the previous set of experiments. The results on the three WMT data sets was a noticeable increase. Given that we use three different datasets with different BLEU baseline we describe the changes in percent change, seen in Table 4.12.

Table 4.12 shows percent increases against the previous baseline for that parser. So MST is the percent change vs the previous MST experiment, SVM vs the previous SVM experiment, etc. Because of this SVM and MST are not directly comparable. MST has a slightly larger percent increase, but SVM has a higher overall BLEU score.

Parser	**NIST (WMT 10/11/12)**	**BLEU (WMT 10/11/12)**
MST	-0.07%/0.72%/0.85%	2.62%/2.06%/2.51%
MST with chunk	0.07%/0.63%/0.64%	2.46%/1.21%/2.59%
Malt	-2.13%/-1.26%/-0.36%	2.64%/1.26%/5.06%
Malt with chunk	-1.5%/-1.12%/-0.39%	1.19%/-0.22%/2.56%
SVM	0.00%/0.53%/0.38%	3.45%/1.7%/ 2.21%

Table 4.12: NIST and BLEU using retrained MST and Malt models. Additionally Charniak and Stanford make use of the new conversion parameters

To show the effects of dependency structures on syntax-based machine translations we needed types of gold annotation data. First, we needed gold dependency trees for data with a parallel translation. Only with this can we test the oracle situation with limited errors. Second, to step away from the oracle situation, we need better training data for our existing parsers. To solved the first problem we annotated data from the WMT 2012 shares task, which contains English to Czech translations. For the second issue we looked at traditional training data and hand annotated a section of the Penn Treebank. This hand annotated data was used to discover the best constituent conversion so that we could expand our training data without hand annotating it all. Using this data we showed the optimal parameter for the PennConverter. Applying this conversion and post editing scripts, we obtained approximately 95% accuracy post conversion without hand annotated results.

We have shown results for new parsing models that make use of our gold dependency annotations, accomplished with the Penn Treebank annotations. Satisfied with a 95% constituent conversion accuracy, we applied the conversion to the entire Penn Treebank. The parsers, retrained on this data, showed an increase in accuracy. When applied to our machine translation scenaio we saw an additional increase in the final BLEU score of our translation system. These increases were also apparent in our ensemble SVM system, which returned our highest BLEU score to date.

Both sets of data, but in particular the WMT annotations, should have a positive effect on the field. Having gold for both an early process and the final process in an NLP pipeline will help researchers concentrate on particular tasks in machine translation without wondering if the errors were created by an earlier process in the pipeline.

5 Conclusion

We have shown improvements to dependency parsing through three means. First, we have combined parse trees with an ensemble approach, employing a variety of voting schemes for English. Second, we have modeled errors in terms of dependency errors per part-of-speech using fuzzy clustering to help weight our ensemble parser for Italian and Japanese. Third, we have used an SVM meta-classifier on each node to determine which model selects the parent for Tamil and Indonesian. All three approaches have been successful. We feel the biggest contribution of the three would be the third parser since it is linguistically independent. It achieves state-of-the-art results by only using model agreement features, which know nothing about the contributing languages. This makes the approach applicable for small and large languages alike. To help with under-resourced languages, we also showed the usefulness of using our ensemble approach with self-training to create additional training data.

We have shown that these ensemble parsers are useful down a syntax-based machine translation pipeline. We have measured this success with the BLEU and NIST automatic metrics along with a small number of human evaluators for an English to Czech translation. This indicated that the results of the ensemble parsers truly did change the parse tree in a useful manner. To fully test our theory that the dependency parse correctness is an important component of the pipeline, we moved next to examine a gold standard.

To the best of our knowledge, there is no gold data sets that combine both machine translation parallel texts along with gold standard dependency trees. To fill the void in this area, we hand annotated the English side of the English-Czech parallel dataset of the WMT 2012 shared task. This allowed us to fully see the impact of gold dependency trees on machine translation. As an additional perk to the dependency parsing community, this data set is a valuable out of domain test set as well. On this new data set we showed the highest BLEU scores obtained on the evaluation set by the TectoMT system. One cannot always expect gold level dependency trees so we next turned to improving the typical parser training data used by the community.

When trying to improve the training data used for dependency parsing, we first gathered evidence and motivation that the annotation decisions truly effected the final output in the NLP pipeline. To do this we first examined the depth of noun phrases. While parsers are typically trained with flat noun phrases, we showed improvement to the TectoMT system by adding the existing noun phrase annotations.

With the success of noun phrase structure, we decided to aim for gold level training data for dependency parsing for English. We first annotated section 23 of the Penn Treebank by hand so that we had a reference set. Next we showed the optimal PennConverter options that would approximate these annotations. After enumerating and testing all options, we have a post conversion accuracy of 95%. Using this conversion, we converted all of the Penn Treebank and retrained our parsers. The result gave us higher UAS scores for our baseline parsers as well as our ensemble parsers and additionally improved the BLEU scores in our machine translation pipeline.

With high oracle parsing scores still in the distance, we feel more work in ensemble parsing could accomplish even better results in syntax-based machine translation. While 95% accuracy with our constituent conversion is a large improvement over past training data, more analysis of the errors may lead to greater gains. We feel that our data set, combining machine translations with gold dependency annotations, will leave a lasting impact on future research in this area.

References

[1] Eckhard Bick. **Hybrid Ways to Improve Domain Independence in an ML Dependency Parser**. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 1119–1123. 20

[2] Sabine Buchholz and Erwin Marsi. **CoNLL-X shared task on multilingual dependency parsing**. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 149–164, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. 22, 25

[3] Sabine Buchholz and Erwin Marsi. **CoNLL-X shared task on multilingual dependency parsing**. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 149–164, Morristown, NJ, USA, 2006. Association for Computational Linguistics. 72, 75

[4] Eugene Charniak and Mark Johnson. **Coarse-to-fine n-best parsing and MaxEnt discriminative reranking**. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 173–180, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. 18

[5] Michael Collins, Lance Ramshaw, Jan Hajic, and Christoph Tillmann. **A Statistical Parser for Czech**, 1999. 18

[6] Corinna Cortes and Vladimir Vapnik. **Support-Vector Networks**. *Mach. Learn.*, **20**(3):273–297, September 1995. 11

[7] Thomas G. Dietterich. **Ensemble Methods in Machine Learning**. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, pages 1–15, London, UK, 2000. Springer-Verlag. 19

[8] Jason Eisner. **Three New Probabilistic Models for Dependency Parsing: An Exploration**. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhagen, August 1996. Association for Computational Linguistics. 17, 19

[9] Richárd Farkas and Bernd Bohnet. **Stacking of Dependency and Phrase Structure Parsers**. In *Proceedings of COLING 2012*, pages 849–866, Mumbai, India, December 2012. The COLING 2012 Organizing Committee. 81

[10] MikelL. Forcada, Mireia Ginest-Rosell, Jacob Nordfalk, Jim ORegan, Sergio Ortiz-Rojas, JuanAntonio Prez-Ortiz, Felipe Snchez-Martnez, Gema Ramrez-Snchez, and FrancisM. Tyers. **Apertium: a free/open-source platform for rule-based machine translation**. *Machine Translation*, **25**(2):127–144, 2011. 27

[11] Michel Galley and Christopher D. Manning. **Quadratic-Time Dependency Parsing for Machine Translation**. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 773–781, Suntec, Singapore, August 2009. Association for Computational Linguistics. 74

[12] Nathan Green, Septina Dian Larasati, and Zdeněk Žabokrtský. **Indonesian Dependency Treebank: Annotation and Parsing**. In *Proceedings of the 26th Pacific Asia Conference on Language, Information, and Computation*, pages 137–145, Bali,Indonesia, November 2012. Faculty of Computer Science, Universitas Indonesia. 81, 82

[13] Nathan Green, Loganathan Ramasamy, and Zdeněk Žabokrtský. **Using an SVM Ensemble System for Improved Tamil Dependency Parsing**. In *Proceedings of the ACL 2012 Joint Workshop on Statistical*

Parsing and Semantic Processing of Morphologically Rich Languages, pages 72–77, Jeju, Republic of Korea, July 12 2012. Association for Computational Linguistics. 30, 81, 82

[14] Nathan Green and Zdeněk Žabokrtský. **Hybrid Combination of Constituency and Dependency Trees into an Ensemble Dependency Parser**. In *Proceedings of the EACL 2012 Workshop on Innovative hybrid approaches to the processing of textual data*, Avignon, France, 2012. 60, 67, 81, 82

[15] Nathan Green and Zdeněk Žabokrtský. **Ensemble Parsing and its Effect on Machine Translation**. Technical Report 48, 2012. 82

[16] Ria Hari Gusmita and Ruli Manurung. **Some initial experiments with Indonesian probabilistic parsing**. In *Proceedings of the 2nd International MALINDO Workshop*, 2008. 19

[17] Gholamreza Haffari, Marzieh Razavi, and Anoop Sarkar. **An Ensemble Model that Combines Syntactic and Semantic Clustering for Discriminative Dependency Parsing**. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 710–714, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. 19

[18] Jan Hajič. **Building a Syntactically Annotated Corpus: The Prague Dependency Treebank**. In Eva Hajičová, editor, *Issues of Valency and Meaning. Studies in Honor of Jarmila Panevová*, pages 12–19. Prague Karolinum, Charles University Press, 1998. 14, 79

[19] Johan Hall, Jens Nilsson, Joakim Nivre, Gülsen Eryigit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. **Single Malt or Blended? A Study in Multilingual Parser Optimization**. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 933–939, 2007. 60, 67

[20] Hieu Hoang, Alexandra Birch, Chris Callison-burch, Richard Zens, Rwth Aachen, Alexandra Constantin, Marcello Federico, Nicola Bertoldi, Chris Dyer, Brooke Cowan, Wade Shen, Christine Moran, and Ondrej Bojar. **Moses: Open source toolkit for statistical machine translation**. pages 177–180, 2007. 26

[21] Hieu Hoang and Philipp Koehn. **Design of the Moses Decoder for Statistical Machine Translation**. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 58–65, Columbus, Ohio, June 2008. Association for Computational Linguistics. 26

[22] Richard Johansson and Pierre Nugues. **Extended Constituent-to-dependency Conversion for English**. In *Proceedings of NODALIDA 2007*, pages 105–112, Tartu, Estonia, May 25-26 2007. 18, 35, 81

[23] Joice. *Pengembangan lanjut pengurai struktur kalimat bahasa indonesia yang menggunakan constraint-based formalism. Undergraduate thesis.* Master's thesis, Faculty of Computer Science, University of Indonesia, 2002. 19

[24] Dan Klein and Christopher D. Manning. **Accurate unlexicalized parsing**. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 423–430, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. 18, 82

[25] Philipp Koehn. **Europarl: A Parallel Corpus for Statistical Machine Translation**. 29

[26] Philipp Koehn. **Statistical Significance Tests for Machine Translation Evaluation**. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 388–395, Barcelona, Spain, July 2004. Association for Computational Linguistics. 77

[27] Steven Krauwer. **The Basic Language Resource Kit (BLARK) as the First Milestone for the Language Resources Roadmap**. In *Proceedings of the 2003 International Workshop Speech and Computer (SPECOM 2003)*, pages 8–15. Moscow State Linguistic University, 2003. 4

[28] Steven Krauwer. **The Basic Language Resource Kit (BLARK) as the First Milestone for the Language Resources Roadmap**. In *Proceedings of the 2003 International Workshop Speech and Computer (SPECOM 2003)*, pages 8–15. Moscow State Linguistic University, 2003. 30

[29] Sandra Kübler, R. McDonald, and J. Nivre. *Dependency parsing*. Synthesis lectures on human language technologies. Morgan & Claypool, US, 2009. 17, 18, 20

[30] Sandra Kübler, Ryan McDonald, and Joakim Nivre. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2009. xiii, 14, 15

[31] Marco Kuhlmann and Giorgio Satta. **Treebank Grammar Techniques for Non-Projective Dependency Parsing**. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 478–486, Athens, Greece, March 2009. Association for Computational Linguistics. 16

[32] Septina Dian Larasati. **IDENTIC Corpus:Morphologically Enriched Indonesian-English Parallel Corpus**. 2012. 22, 31

[33] Septina Dian Larasati, Vladislav Kuboň, and Daniel Zeman. **Indonesian Morphology Tool (MorphInd): Towards an Indonesian Corpus**. *Systems and Frameworks for Computational Morphology*, pages 119–129, 2011. 22

[34] Hrafn Loftsson and Eirkur Rgnvaldsson. **IceNLP: a natural language processing toolkit for icelandic**. In *INTERSPEECH 2007, 8th Annual Conference of the International Speech Communication Association, Antwerp, Belgium, August 27-31, 2007*, pages 1533–1536. ISCA, 2007. 30

[35] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. **Building a large annotated corpus of English: the Penn Treebank**. *Comput. Linguist.*, **19**:313–330, June 1993. 9, 18, 20, 74

[36] Marie-Catherine De Marneffe, Bill Maccartney, and Christopher D. Manning. **Generating typed dependency parses from phrase structure parses**. In *In LREC 2006*, 2006. 18

[37] Ryan McDonald and Joakim Nivre. **Characterizing the Errors of Data-Driven Dependency Parsing Models**. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131, 2007. 17

[38] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. **Non-Projective Dependency Parsing using Spanning Tree Algorithms**. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics. 17, 58

[39] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. **Non-projective dependency parsing using spanning tree algorithms**. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 523–530, Morristown, NJ, USA, 2005. Association for Computational Linguistics. 76, 81

[40] Joakim Nivre. **An Efficient Algorithm for Projective Dependency Parsing**. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*, pages 149–160, 2003. 81

[41] Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. **The CoNLL 2007 Shared Task on Dependency Parsing**. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic, June 2007. Association for Computational Linguistics. 20

[42] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gulsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. **MaltParser: A language-independent system for data-driven dependency parsing**. *Natural Language Engineering*, **13**(2):95–135, 2007. 18, 58

[43] Joakim Nivre and Ryan McDonald. **Integrating Graph-Based and Transition-Based Dependency Parsers**. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio, June 2008. Association for Computational Linguistics. 20

[44] Petr Pajas and Peter Fabian. **TrEd 2.0 - newly refactored tree editor**. http://ufal.mff.cuni.cz/tred/, Institute of Formal and Applied Linguistics, MFF UK, 2011. 22, 79

[45] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. **BLEU: a method for automatic evaluation of machine translation**. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Morristown, NJ, USA, 2002. Association for Computational Linguistics. 29

[46] Martin Popel, David Mareček, Nathan Green, and Zdeněk Žabokrtský. **Influence of Parser Choice on Dependency-Based MT**. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 433–439, Edinburgh, Scotland, July 2011. Association for Computational Linguistics. 35, 81

[47] Martin Popel, Zdeněk Žabokrtský, and Jan Ptáček. **TectoMT: Modular NLP Framework**. In *IceTAL*, pages 293–304, 2010. xiii, xix, 27, 28, 73, 81

[48] Loganathan Ramasamy and Zdeněk Žabokrtský. **Tamil dependency parsing: results using rule based and corpus based approaches**. In *Proceedings of the 12th international conference on Computational linguistics and intelligent text processing - Volume Part I*, CICLing'11, pages 82–95, Berlin, Heidelberg, 2011. 19, 21

[49] Loganathan Ramasamy and Zdeněk Žabokrtský. **Prague Dependency Style Treebank for Tamil**. In *Proceedings of LREC 2012*, İstanbul, Turkey, 2012. xiii, 21, 22, 30, 31

[50] Philip Resnik. **Mining the Web for Bilingual Text**. In *In Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 527–534, 1999. 9

[51] Rudolf Rosa, David Mareček, and Ondej Dušek. **DEPFIX: A System for Automatic Correction of Czech MT Outputs**. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 362–368, Montréal, Canada, June 2012. Association for Computational Linguistics. 28

[52] Kenji Sagae and Alon Lavie. **Parser Combination by Reparsing**. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132, New York City, USA, June 2006. Association for Computational Linguistics. 19

[53] Kenji Sagae and Jun'ichi Tsujii. **Dependency Parsing and Domain Adaptation with LR Models and Parser Ensembles**. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 1044–1050, Prague, Czech Republic, June 2007. Association for Computational Linguistics. 19

[54] Petr Sgall. *Generativní popis jazyka a česká deklinace*. Academia, Prague, Czech Republic, 1967. 27

[55] Mihai Surdeanu and Christopher D. Manning. **Ensemble models for dependency parsing: cheap and good?** In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 649–652, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. 19

[56] David Vadas and James Curran. **Adding Noun Phrase Structure to the Penn Treebank**. In *Proceedings of the 45th Annual Meeting of*

the Association of Computational Linguistics, pages 240–247, Prague, Czech Republic, June 2007. Association for Computational Linguistics. 72, 74

[57] David Vadas and James R. Curran. **Large-Scale Supervised Models for Noun Phrase Bracketing**. In *Conference of the Pacific Association for Computational Linguistics (PACLING)*, pages 104–112, Melbourne, Australia, September 2007. 72, 74

[58] David Vadas and James R. Curran. **Parsing Internal Noun Phrase Structure with Collins' Models**. In *Proceedings of the Australasian Language Technology Workshop 2007*, pages 109–116, Melbourne, Australia, December 2007. 72, 74

[59] David Vadas and James R. Curran. **Parsing Noun Phrase Structure with CCG**. In *Proceedings of ACL-08: HLT*, pages 335–343, Columbus, Ohio, June 2008. Association for Computational Linguistics. 72, 74

[60] Zdeněk Žabokrtský, Jan Ptáček, and Petr Pajas. **TectoMT: Highly Modular MT System with Tectogrammatics Used as Transfer Layer**. In *Proceedings of the 3rd Workshop on Statistical Machine Translation, ACL*, pages 167–170, 2008. 5, 35, 73

[61] Daniel Zeman and Zdeněk Žabokrtský. **Improving Parsing Accuracy by Combining Diverse Dependency Parsers**. In *In: Proceedings of the 9th International Workshop on Parsing Technologies*, 2005. 19, 81

[62] Ying Zhang, Stephan Vogel, and Alex Waibel. **Interpreting BLEU/NIST scores: How much improvement do we need to have a better system**. In *In Proceedings of Proceedings of Language Resources and Evaluation (LREC-2004*, pages 2051–2054, 2004. 77

[63] Yue Zhang and Stephen Clark. **Syntactic Processing Using the Generalized Perceptron and Beam Search**. *Computational Linguistics*, **37**(1):105–151, 2011. 18, 81

Appendix

Here are a few examples of trees from our WMT gold annotations including the automatic parse followed by our gold annotation.

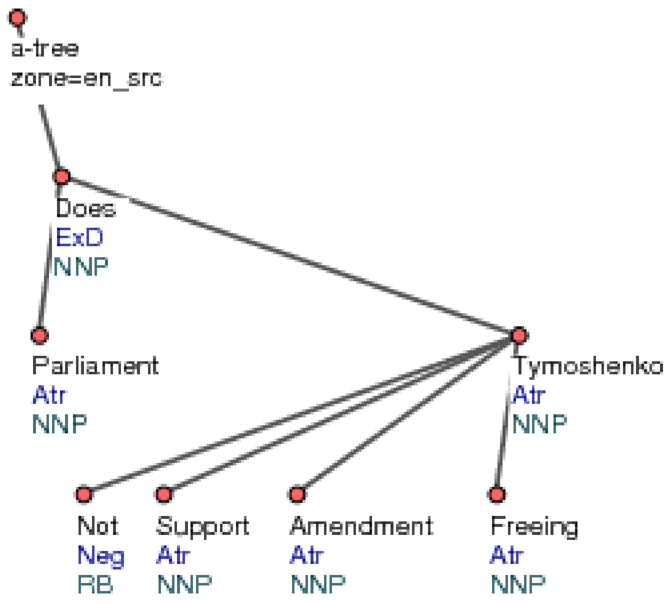


Figure A.1: Original incorrect parse. Errors most likely cause by incorrect tagging due to the capitalization of the title.

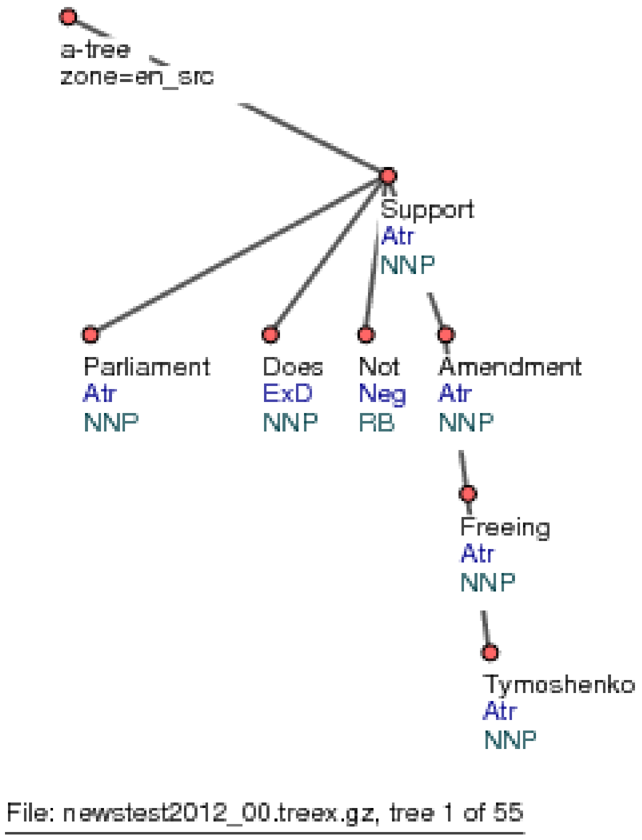


Figure A.2: Correction of the structure. Tags in this situation were not always corrected. Further processing of the data is needed.

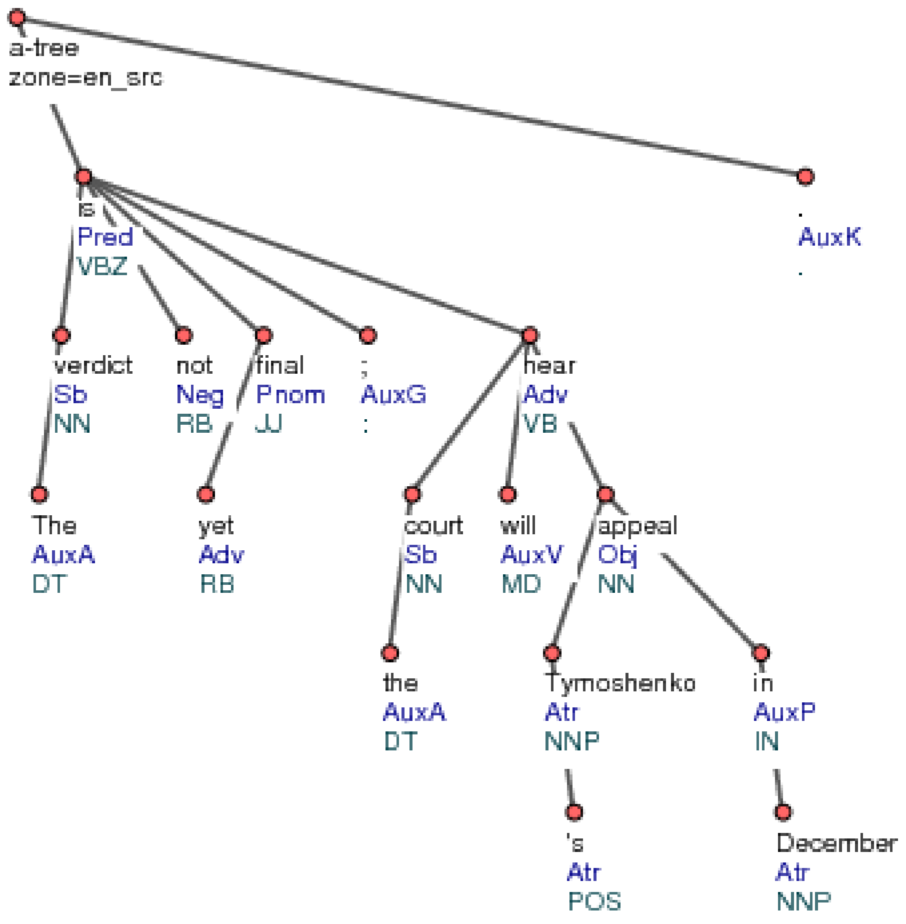


Figure A.3: Often sentences were not segmented where we would like. For instance when two sentences are connected with a semicolon. In these situations we made the semicolon the root of the tree.

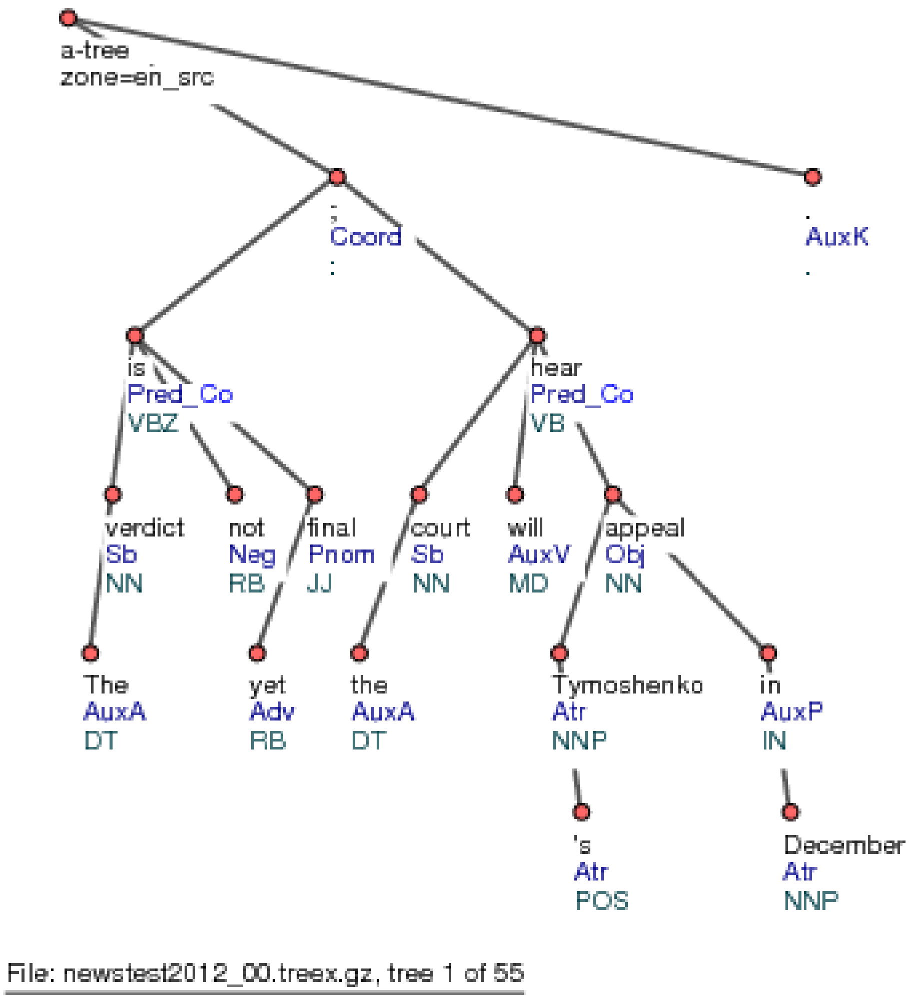


Figure A.4: Fix for issues with sentence segmentation and semicolons.

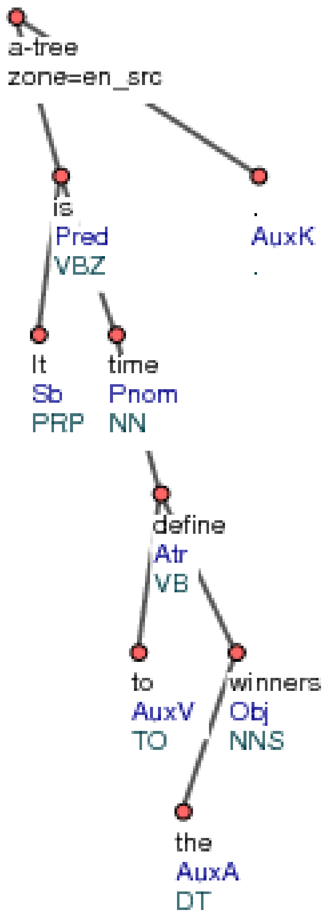


Figure A.5: We made the decision to structures such as "to define" up one level.

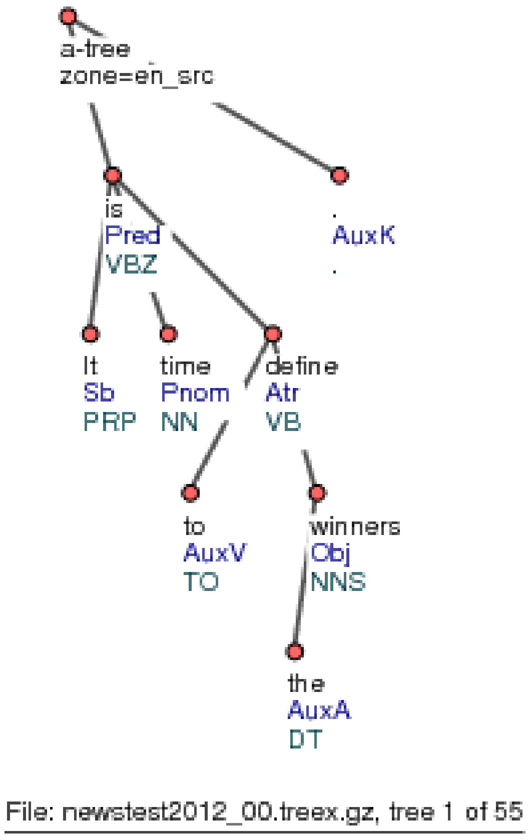


Figure A.6: Correct version of the “to define” structure.

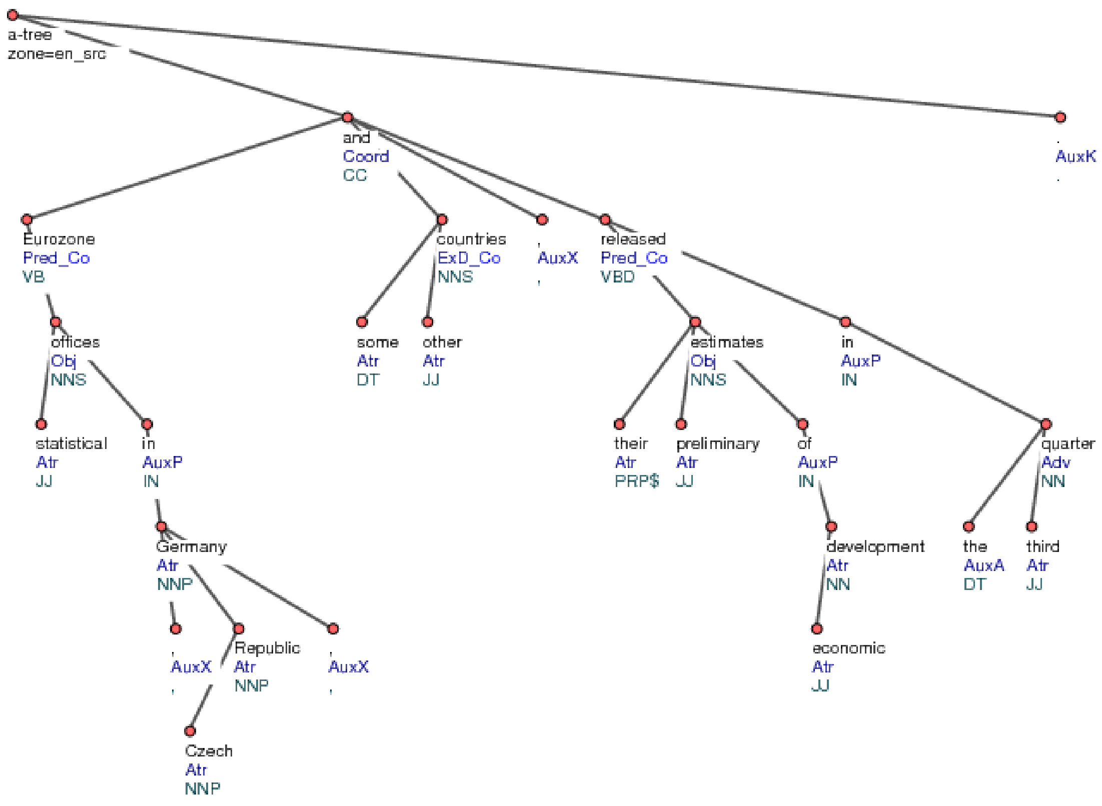


Figure A.7: Coordination is often incorrect with the automatic parses. In this case the last clause is placed with the false root.

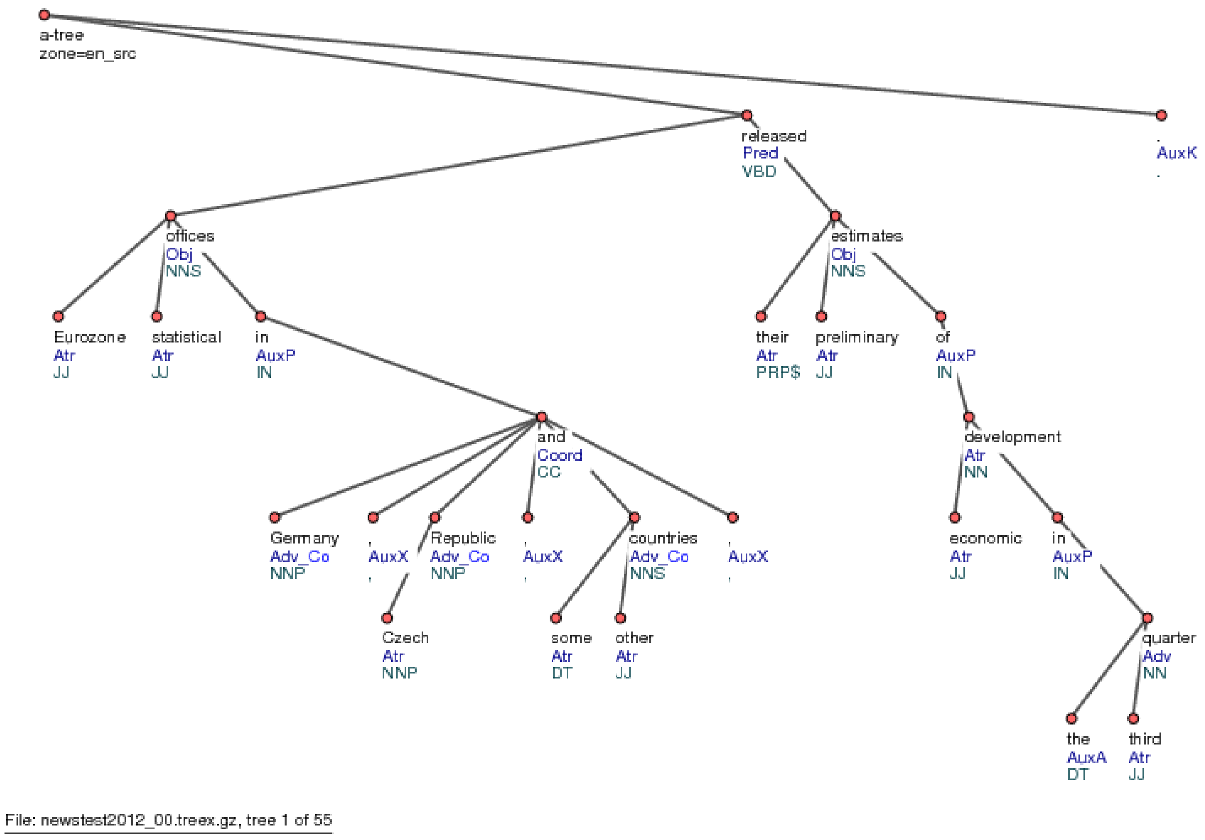


Figure A.8: Corrected coordination structure.