Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Màster en Intel·ligència Artificial

Tesi de Màster

Joint Learning of Syntactic and Semantic Dependencies

per

Xavier Lluís Martorell

sota la direcció del doctor
Lluís Màrquez Villodre

Barcelona, Setembre de 2008

Abstract

In this master's thesis we designed, implemented and evaluated a novel joint syntactic and semantic parsing model.

Syntactic and semantic parsing have been and are still being addressed as sequence or pipeline of tasks. As far as we know, the only open domain exception to this pipeline approach was published by Musillo and Merlo (2006). The pipeline processing implies the undesirable and hard to recover effect of error propagation across components. Furthermore, syntax and semantics are assumed to interact between themselves at some degree and these interactions cannot be modeled by a pipeline system. Thus the main objective of this work is to design a joint model compare the pipeline and joint approaches. Also and in spite of a fair comparison, the computing costs are evaluated.

This master's thesis is concerned to the joint syntactic and semantic parsing under the machine learning paradigm. Thus a dataset must be available and evaluation measures provided. The CoNLL-2008 shared task (Surdeanu et al., 2008), devoted to joint parsing of syntactic and semantic dependencies, brings these elements and presents an excellent framework to train and evaluate our model. Shared task organizes merged together several data sources to provide training and testing datasets. Furthermore, the shared task evaluation framework is widely used and in addition our system can be compared to other shared task contributing teams.

Our proposed model for joint parsing relies on an on-line structured perceptron for learning (Collins, 2002) and on the Eisner algorithm (Eisner, 1996) for inference. The Eisner algorithm is a bottom-up parser previously successfully extended in the context of syntactic parsing (McDonald and Pereira, 2006; Carreras, 2007). Our proposal is one step further and, as far as we know, for the first time the Eisner algorithm is applied to jointly parse syntactic and semantic dependencies. At each algorithm step a syntactic dependency and some semantic dependencies

are simultaneously computed and finally a complete optimal syntactic-semantic structure is built.

Our implemented model shown to be feasible and efficient and presented encouraging results under the rich shared task framework. Unfortunately very few contributing teams presented other novel joint approaches. The overall results for the shared task were 78.11 global F_1, 85.84 LAS and 70.35 semantic F_1. These were moderate but encouraging results given the complexity achievable by a built from scratch system developed under the hard time constraints of the shared task.

A comparison of our model to an equivalent pipeline system, one of our first main concerns, showed that the joint system outperformed by 4.9 points the equivalent semantic pipeline system. The comparison between the joint system and the syntactic pipeline system presented similar results. We concluded that a joint parsing model is completely feasible and that some degree of syntactic and semantic interaction is exploitable.

Contents

Chapter 1

Introduction

Natural Language Understanding (NLU) is the set of tasks that deals with the exploitation of semantic knowledge present in natural language. NLU is of the major problems in Natural Language Processing (NLP). NLP is a field in Artificial Intelligence that deals with the automated generation and understanding of natural language.

NLU is a complex task informally considered, quoting Martin Kay, as *AI-complete* implying that the difficulty of solving this problem is as hard as solving other central Artificial Intelligence problems or also as making computers as intelligent as human beings.

The goals of NLU are still far from being reached. A major challenge for state-of-the-art NLP systems is The ambiguous nature of natural language. Despite this, up to date, recent research has contributed with significant progress in important subtasks of NLU such as dependency parsing and shallow semantic parsing.

A wide range of applications such as Question Answering, Automatic Summarization or Information Extraction could potentially benefit from NLU research.

The purpose of this master's thesis is to design a model to jointly perform syntactic dependency parsing and semantic role labeling. The model will be trained from a data source as we focus on the machine-learning approach to these two problems.

Dependency parsing is the task of deriving a syntactic dependency structure. In this structure each token has a link(dependency) to a *head* token, except the *root* token that has no head. The syntactic structure conforms a well defined tree.

Semantic role labeling is the annotation semantic arguments for each sentence predicate. The core arguments to annotate typically an-

swers the questions *who,when,what,why* regarding the predicate. Also *semantic role labelling* comprises the annotation of adjunct arguments (e.g., manner, location, negation).

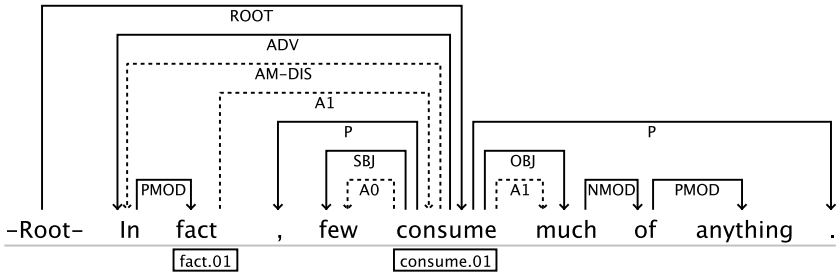

Figure 1.1: Syntactic and semantic dependencies

Figure 1.1 shows a sentence with syntactic dependencies (solid lines) and semantic dependencies (dotted lines). Semantic dependencies express semantic relations between predicates (boxes with the predicate lemma and sense) and tokens regarded as arguments. For instance, the token *of* is a syntactic dependent of *much* and acts as a noun modifier(NMOD). Token *in* is a semantic argument of *fact*, see the dotted line labeled *AM-DIS* standing for *adjunct argument of discourse*. Note that the syntactic structure conforms a tree.

Complex NLU problems are typically addressed by a pipeline architectures. The main advantage of this approach is the division of the complex problem into an abordable sequence of subtasks. Unfortunately propagated errors through the pipeline are hard to recover despite the use of backtracking strategies. Furthermore and concerning to the topics of this master's thesis, syntax and semantics are assumed to interact and these interactions cannot be modeled by sequential systems.

We considered after this brief discussion that two natural questions arise:

- *Can a joint system overcome the pipeline approach?*
- *Is it feasible to build a joint parsing system?*

Our interests in this work are in seeing whether or not a joint system can improve traditional pipeline systems and if it is possible to train our system at a reasonable cost. Thus the main *goals* of this master's thesis are:

- Compare the joint and pipeline approaches
- Design and implement a feasible joint learning architecture

The two previously stated questions will be recalled on the discussion chapter.

Master's thesis organization

The rest of this work is organized as follows:

- **Chapter 2 - Problem setting**
  gives a description of the problem to be addressed and defines the CoNLL-2008 shared task framework.
- **Chapter 3 - Background**
  details the machine learning and inference methods employed by state-of-the-art syntactic and semantic parsing systems, focusing to the applied in this work.
- **Chapter 4 - State of the art**
  outlines complete systems and and gives and overview of the state of the art of the concerned topics.
- **Chapter 5 - System design**
  describes in detail our joint model proposal and implementation.
- **Chapter 6 - Results**
  reports the experimentation done and the results accomplished by our system. Also includes a the CoNLL-2008 shared task final scores.
- **Chapter 7 - Final remarks**
  we present a deliberation about our work, the main conclusions are drawn and the future work is outlined.
- **Appendix A - Descriptive data analysis**
  includes an exploratory analysis of the training corpus data focusing on relevant aspects for a joint parsing.
- The remaining appendices contains detailed scores and some useful references.

Chapter 2

Problem setting

Contents

In this chapter we describe the problem to address. It comprises the definition of task to be addressed, the evaluation measures and the training and testing data. The CoNLL-2008 shared task brings in a convenient way all these required elements. The next section describes the shared task and the evaluation measures and outlines the training data. The analysis of the data is on appendix A, this analysis the first step towards a better understanding of the problem data, please refer to this appendix.

2.1 The CoNLL-2008 shared task

The CoNLL-2008 Shared Task (Surdeanu et al., 2008)[1] is devoted to the joint parsing of syntactic and semantic dependencies. Previous CoNLL-2007 and CoNLL-2006 shared tasks (Nivre et al., 2007a; Buchholz et al., 2006) were devoted to syntactic dependency parsing.

[1] See the official website `http://www.yr-bcn.es/conll2008/`

CoNLL-2005 and CoNLL-2004 (Carreras and Màrquez, 2004; Carreras and Màrquez, 2005) shared tasks aborded the semantic role labelling problem. Present year CoNLL-2008 shared task should not be regarded as a simple merging of previous years shared tasks. This shared task is differentiated by:

- A unified formalism to express syntactic dependencies, sense predicates and arguments. All are expressed by annotated links between tokens, see section 2.1.3.
- A richer set of syntactic and semantic information by using new algorithms to convert the original corpus formats to the new unified formalism.
- SRL for both nominal and verbal predicates.

This shared task posed new challenges:

- The semantic parsing was traditionally done base upon a constituent syntactic structure. This shared sets for semantic parsing a dependency structure. The potential advantages of this change are:
  - The dependency structure may be the appropriate structure for most NLP applications. Some of them (e.g., information extraction and information retrieval) have been relying for long on dependency structures. Constituent parsing is often unnecessary for a wide range of applications.
  - Dependency parsing algorithms are more efficient, linear parsers can be build.
- The dependency structure is one of the richest available to date, including several new syntactic labels.
- The tasks invites to a joint approach of syntactic and semantic dependency parsing.

The CoNLL-2008 Shared Task as previous shared tasks allows the teams to participate into two challenges:

closed challenge In this challenge the participating teams must strictly build their system using the information provided in the training files and tune using the development files. This constraint are aimed to provide a fair evaluation environment.

open challenge In this case teams can make use of any additional tools and resources. Also the organizers provides supplementary resources: tagged named entities, word net senses and a syntactic parse obtained from *MaltParser* (Nivre et al., 2007b).

The *closed challenge* is the most attended of the two challenges as it offers a fair framework for contributing teams. Up to date, the CoNLL-2008 shared task had became the shared task with the largest number of registered teams ever. More than 50 teams registered for participation, but finally only 23 of the teams presented a system.

2.1.1 Task overview

Conceptually the task can be divided in the three parts:

- Syntactic dependency parsing.
  A syntactic tree y must be computed from an input sentence x. The tree y is rooted on a special node x_0, added for this purpose. The arcs $(i, j) \in y$ are labeled arcs with syntactic tags l, also noted $i \xrightarrow{l} j$.

- Predicate identification and predicate sense disambiguation
  A set of sentence tokens are to be tagged as predicates. There is an exact correspondence between predicates and tokens. Furthermore the identified predicates must be labeled also with their sense, the senses are indexed and provided by a set of frames files.

- Semantic argument annotation
  The predicate arguments must be detected and linked their predicates. Note that not all sentence tokens are arguments nor predicates. The argument annotation task consists in generating a set of simple trees $t_1, \ldots, t_q$ one for each of the q identified predicates of the sentence x. Each graph t_i is a tree that does no necessarily connects to all sentence tokens. The root of the tree corresponds to the predicate and only direct sons are allowed. Each son is an argument. Also the tree is labeled with the set of semantic tags. Merging all trees will result in a Directed Acyclic Graph, but also not necessarily covering all tokens nor being connected.

A detailed definition and further discussion of the syntactic and semantic parsing tasks is to be found in the *background* chapter, sections 3.1,3.3.

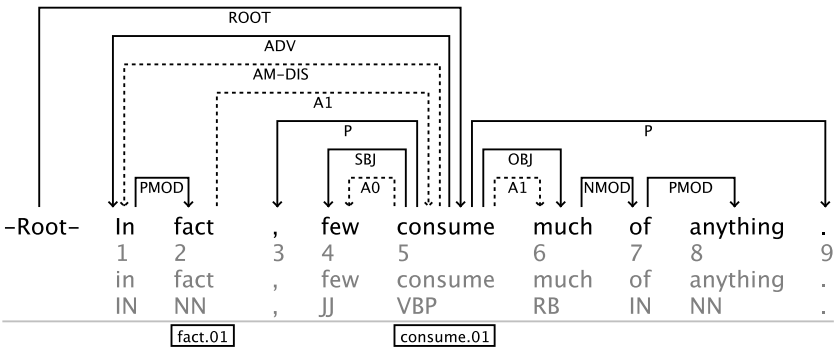

Figure 2.1: A sample sentence with lemma and POS

Figure 2.1 shows an input sentence with some information fields. The fields plotted are the sentence tokens, an index of tokens and the corresponding lemma and Part of Speech(POS) of each token. The output to generate for each sentence is the syntactic structure (solid lines and labels), the semantic structure (dotted lines and labels) and the predicate identification and sense disambiguation (boxes with predicate lemma and sense number).

2.1.2 Data overview

The shared task organizers made available resources to be used as the *only* data source for the closed challenge participating teams. The released data is:

- Training corpus file
  A set of ∼40k sentences completely annotated with syntactic dependencies, predicates and semantic arguments. The corpus is based on the WSJ corpus. It is built by merging various corpus, see section 2.1.3.
- Frames files
  Also named rolesets, i,e., the number of senses and allowed arguments for each predicate.
  - NomBank senses
    The nominal predicates and the list of allowed senses.
  - PropBank senses
    The verbal predicates and the list of allowed senses.

Data based on the *Wall Street Journal* corpus was provided for training, developing and testing. New data from the *Brown* corpus

was provided only for testing. The use of the *Brown* corpus data file only for testing is because it is intended to evaluate the system in an out-of-domain setting. Usually a significant drop in performance is observed when the testing data does not come from the training domain. This problem is commonly known as *domain adaptation* (Daumé III and Marcu, 2006).

On top of this corpus data various annotation projects labeled syntactic structure (PennTreebank 3 (Marcus et al., 1993), Brown University (Francis and Kucera, 1964)), semantic structure for verbal predicates (PropBank (Palmer et al., 2006)) and for nominal predicates (NomBank (Meyers et al., 2004)).

Details about the conversion process from the original constituent representation to the dependency structure are in Surdeanu et al (Surdeanu et al., 2008) an references therein.

A remarkable fact about the data sources is the definition of the argument labels. The arguments a predicate can accept (e.g., agent, patient) are neutrally labeled as A0,. . . ,A5. Adjunct arguments (e.g., temporal, manner) are not predicate-specific and are labeled as AM-TMP, AM-MNR, AM-LOC, AM-NEG, ... Although an effort by the annotators was made in this direction, the same A*X* argument can take different meanings across the set predicates (e.g. A3 for *go* is the start point but A3 for *do* is the instrument). Thus a classifier for an A*X* argument must in fact predict substantial different roles.

For an in depth data analysis please refer to A.

2.1.3 Corpus format

We describe[2] the data format used in the CoNLL-2008 Shared Task. The format was highly influenced by the formats used in the 2005, 2006, and 2007 shared tasks.

The data follows these general rules:

- The data files contain sentences separated by a blank line.
- A sentence consists of one or more tokens and the information for each token is represented on a separate line.
- A token consists of at least 11 fields, described in the table below. The fields are separated by one or more whitespace characters

[2]The contents of this section are mainly borrowed from the CoNLL-2008 shared task web site `http://www.yr-bcn.es/conll2008/`

(spaces or tabs). Whitespace characters are not allowed within fields.

Table 2.1 describes the fields stored for each token in the data sets.

Col	Field name	In/out	Description
1	ID	INPUT	Token counter, starting at 1 for each new sentence.
2	FORM	INPUT	Word form or punctuation symbol. The FORM field uses the original WSJ tokenization, i.e., hyphenated words such as "Atlanta-based" are not split.
3	LEMMA	INPUT	Predicted lemma of FORM, extracted from WordNet using the most common sense of the word. If FORM does not exist in WordNet, LEMMA is set to the lowercase version of FORM.
4	GPOS	INPUT (train/dev only)	Gold part-of-speech (POS) tag from the TreeBank. Note that, in order to have a realistic evaluation, this field is provided only for the training and development sets. For the testing sets this column will contain "_".
5	PPOS	INPUT	Predicted POS tag. These tags are predicted by a state-of-the-art POS tagger. To avoid overfitting the tagger on the training corpus the tags on the training set are generated through 10-fold cross-validation.

6	SPLIT_ FORM	INPUT	Tokens split at hyphens and slashes. Because some arguments, generally arguments of nominal predicates, may appear inside hyphenated words it is necessary to split hyphenated constructs in order to correctly annotate such arguments. For example, the TreeBank token "Atlanta-based" is split into three SPLIT_FORMs: "Atlanta", "-", and "based". To ensure the same number of rows for all columns corresponding to one sentence, the FORM, LEMMA, GPOS, and PPOS columns are padded with "_" fields for all split tokens.
7	SPLIT_ LEMMA	INPUT	Predicted lemma of SPLIT_FORM, extracted from WordNet using the most common sense of the word. If SPLIT_FORM does not exist in WordNet, SPLIT_LEMMA is set to the lower-case version of SPLIT_FORM.
8	PPOSS	INPUT	Predicted POS tags of the split forms. These tags are generated using the same state-of-the-art tagger and cross-validation process as PPOS.
9	HEAD	OUTPUT	Syntactic head of the current token, which is either a value of ID or zero ('0'). Note that both syntactic and semantic dependencies annotate the split-form tokens.
10	DEPREL	OUTPUT	Syntactic dependency relation to the HEAD. The syntactic dependency analysis is very similar to that used for the English data sets in the CoNLL 2007 shared task.

11	PRED	OUTPUT	Rolesets of the semantic predicates in this sentence. This includes both nominal and verbal predicates. The split-form tokens that are not semantic predicates must be marked with "_". We use the same roleset names as the PropBank and NomBank frames.
12+	ARG	OUTPUT	Columns with argument labels for the each semantic predicate following textual order, i.e., the first column corresponds to the first predicate in PRED, the second column to the second predicate, etc. Note that, because this algorithm uniquely identifies the ID of the corresponding predicate, it is sufficient to store the label of the argument here. The argument labels for verbal predicates follow the PropBank conventions. Labels of arguments to nominal predicates use NomBank conventions.

Table 2.1: CoNLL-2008 Shared Task datafileds

2.2 Evaluation

The evaluation of syntactic and semantic accuracy is widely preformed using the measures defined in the context of the CoNLL-2008 shared task (Surdeanu et al., 2008). The following sections details the syntactic and semantic scorings as well as well as the combined scores.

2.2.1 Syntactic scoring

Tree measures, also used in past CoNLL-2007 and CoNLL-2006 shared tasks (Nivre et al., 2007a; Buchholz et al., 2006), can be defined to asses the syntactic performance:

Labelled attachment score (LAS) measures the percentage of tokens with only a correct head (i.e., a correct dependency arc) *and* also a correct syntactic label.

Unlabelled attachment score (UAS) measures the percentage of tokens with a correct head (i.e., only a correct dependency arc).

Labelled accuracy score (LAC) measures the percentage of tokens with a correct syntactic label without regarding if the dependency points to an incorrect head.

The main measure to compare syntactics across systems is the *LAS*.

2.2.2 Semantic scoring

The semantic arguments are considered as semantic dependencies between the predicate and each argument, i.e., the predicate is considered as a root and its semantic arguments as its semantic dependents.

The semantic predicates are also converted to semantic dependencies. Each predicate is considered as a dependency from a virtual *root* node to the predicate. This new virtual dependency is labeled with the predicate sense as label. This is merely intended to provide a unified point of view for semantic scoring.

The semantic structures formed are always single-rooted connected graphs, but not necessary acyclic. This approach allows to define a unified scoring and a partial positive scoring when the predicate sense is incorrectly predicted.

We firstly review the *precision* and *recall* measures.

Precision is the fraction of correctly identified/tagged arguments with respect to the number of predicted arguments.

$$\text{precision} = \frac{|\text{correct predictions}|}{|\text{total predictions}|}$$

Recall is the fraction of correctly identified/tagged arguments with respect to the total number of arguments to predict.

$$\text{recall} = \frac{|\text{correct predictions}|}{|\text{total number of predictions to made}|}$$

$\mathbf{F}_1$ is the harmonic mean between *Precision* and *Recall*

$$\text{F}_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

We distinguish between labeled and unlabeled measures. Unlabeled measures do not regards semantic labels, only accounts for the identification of the semantic link between and argument and a predicate.

Unlabeled Precision the precision of predicted unannotated semantics links.

Unlabeled Recall the recall of predicted unannotated semantics links.

Unlabeled $\mathbf{F}_1$ the F_1 of the two previous measures.

Labeled Precision precision of the predicted semantic links and their labels.

Labeled Recall recall of the predicted semantic links and their labels.

Labeled $\mathbf{F}_1$ the F_1 of the two previous measures.

The labeled measures considers only a correct prediction when the argument is correctly identified *and* also is annotated with the correct label in the case of the arguments or with the correct sense for the predicates case.

Finally, another measure accounts for the complete correct semantic annotations.

Perfect Proposition $\mathbf{F}_1$ scores the entire semantic frame, is computed as the F_1 of the completely correct set of arguments and sense for each predicate. Note that it cannot be computed as the percentage of semantic propositions with all their arguments and sense correct as we can overpredict or underpredict the predicates that defines each semantic proposition

A semantic frame comprises a given predicate and all its arguments. Note that in other contexts a semantic frames also refers to the set of admissible arguments for a predicate.

The main measure to compare semantics across systems is the *Labelled F_1*.

Example 2.2.2.1 Semantic scoring
For example[3], if a correct semantic output is to identify a verb and 3 predicates, say:

```
verb.01: ARG0, ARG1, ARGM-TMP
```

And the system output is

[3]Example extracted from the CoNLL-2008 shared task website `http://www.yr-bcn.es/conll2008/`

`verb.02: ARG0, ARG1, ARGM-LOC`

The labeled precision score will be 2/4. The incorrect sense for the verb ('01' instead of '02') accounts for one error.

2.2.3 Global scoring

Several measures combines the syntactic and semantic scores.

Exact Match is the percentage of sentences that are completely correct, including syntactic dependencies, semantic dependencies and predicates.

Labeled Macro F_1 This measure is computed using the F_1 averaging of:

Labeled macro precision

$$\text{LabeledMacroPrecision} = \text{LabelSemanticPrecision} + \text{LAS}$$

Labeled macro recall

$$\text{LabeledMacroRecall} = \text{LabelSemanticRecall} + \text{LAS}$$

Micro F_1 this measure is computed considering all syntactic and semantic dependencies within the same bag, i.e., each prediction is considered an individual problem, precision and recall are computed and finally the F_1 score.

semantic labeled F_1 / syntactic LAS this measure is intended to measure the relative performance of the semantic component. The measure tries to capture the performance of the semantic component with respect to the syntactic parsing. As the syntactic parsing could significantly affect the semantic component this measure is aimed to give a more fair comparison of the semantic components of the systems.

The official evaluation criterion for the CoNLL-2008 shared task is the **Labeled macro F_1**. The systems are evaluated in a section of the *Wall Street Journal* and a section of the *Brown* corpus.

Chapter 3

Background

Contents

The purpose of this chapter is to review the approaches to syntactic dependency parsing and semantic parsing or semantic role labelling.

The *machine learning* section reviews the main learning algorithms *applied* to *state-of-the-art* syntactic and semantic parsing systems, and specially focusing on the applied in this work. But note that we are not intended to explore the not applied methods to the field nor to review the complete vast amount of machine learning literature.

3.1 Syntactic Dependency Parsing

The syntactic parsing is the is the task of deriving a syntactic structure. Two main approaches are used to describe the syntactic structure of a sentence.

- Constituent parsing
  The sentence is broken into a hierarchical set of constituents or phrases.
- Dependency parsing
  In this case the structure is represented by links between words or lexical items.

In this work we focus on dependency parsing. It is considered a more suitable approach for a wide variety of machine learning applications. Several efficient algorithms have been proposed for this task and are described in the following sections.

Formalization

We assume we parse a sentence x of length n with tokens $x_1, \ldots, x_n$. The syntactic dependency structure can be defined as a graph. All lexical items (or tokens) of x are linked by binary relations called *dependencies*. A dependency $d = \langle h, m \rangle$ is a head h and a modifier m (also called dependent). Usually, a syntactic dependency $d = \langle h, m, l \rangle$ is usually tagged with a syntactic label l. We represent a dependency $d = \langle h, m, l \rangle$ also as $h \xrightarrow{l} m$ and a path from a token or lexical unit i to j in the syntactic graph G as $i \xrightarrow{*} j$.

The dependency graph G can be represented as a weakly connected *Directed Acyclic Graph* (DAG). With a set of *labeled* arcs A and a set of ordered vertex V. The vertices of this graph are the sentence tokens.

The following properties are derived from this definition:

- G is weakly connected
  if we replace the arcs of G with undirected edges the resulting graph is connected.
- G is acyclic
  there is no non-empty directed path that starts and ends on the same node.

Usually tighter constrains are imposed.

- Single-head constraint
  every node has at most one head, i.e., if $i \rightarrow j$ then for all $k \neq i$ there is no $k \rightarrow j$

- Root node constraint
  a new node *root* x_0 not accepting incoming arcs or *heads* is added.

An important distinction on dependency graphs is by their *projectivity* property. Note that some parsing algorithms cannot deal with non-projective structures.

A graph G is projective:

$$\begin{aligned} \text{if } \; i \rightarrow j, \quad \text{then } \forall i' \; \text{ such that } i \leftarrow i', \\ i < i' < j \text{ or } j < i' < i \end{aligned} \tag{3.1}$$

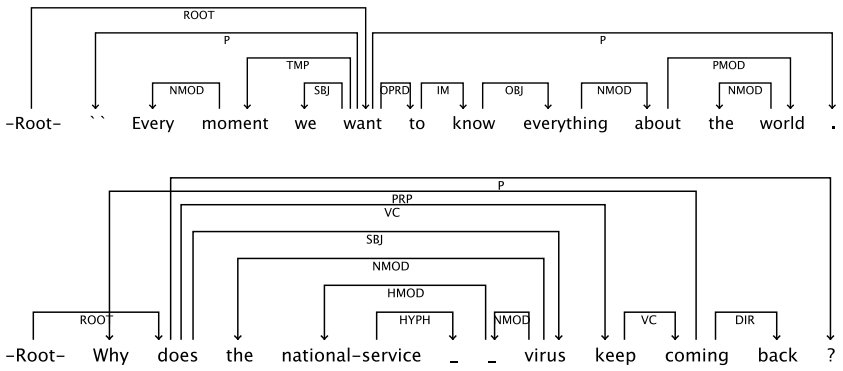

Figure 3.1: Projective and non-projective sentences

Figure 3.1 shows an example of a projective sentence(top) and a non-projective sentence(bottom), the last contains the non-projective links between *why-keep* and *does-?*.

The CoNLL-2008 shared task considers sentences to be single-rooted. Therefore the dependency graph is always a tree. All sentences of size n will have a dependency tree of $(n-1)$ arcs.

The dependencies define *head-modifier* relations. The intuitive idea behind these relations is (Hudson, 1987) [1]

- The *head* determines the syntactic category of the construction containing the *modifier*.

[1] Extracted from McDonal and Nivre *Introduction to dependency parsing* `http://dp.esslli07.googlepages.com/esslli1.pdf`

- The *head* determines the semantic category of the construction containing the *modifier*.
- The *head* is mandatory, the *modifier* is optional.
- The *head* selects the *modifier* a determines if *modifier* is obligatory.
- The form (by agreement or government) of *modifier* is defined by *head*.
- The position of *modifier* is specified by reference to *head*.

In some cases it is not clear which token is the head and which one the modifier. The following relations between tokens are ambiguous: [2]

- auxiliary-main verb and multi-word verbs
- subordination
- coordination
- prepositional phrases
- punctuation

This ambiguities are fixed by consistent rules across the corpus. For example, the punctuation marks are linked to the main predicate, coordination and subordination are linked by following the sentence order.

A general ambiguity in dependency parsing is the prepositional phrase attachment problem. It is the task of linking a prepositional phrase to its head (Ratnaparkhi et al., 1994)

Example 3.1.0.1 The PP attachement problem
In sentence 'I *saw* a *man* **with** *a telescope*'
the phrase introduced by *with* could be linked to the verb *saw* or to the noun *man*

Usually the preposition does not contains enough information to decide the head of the preposition (i.e., the head of the prepositional phrase). It is regarded as necessary to consider the relation between the head (in our case *saw* or *man*) and the grandson (*telescope*). Note that edge-factored models cannot handle grandson relations, see section 3.2.2.

[2] Extracted from McDonal and Nivre *Introduction to dependency parsing* `http://dp.esslli07.googlepages.com/esslli1.pdf`

Finally we remark some practical benefits of dependency parsing with respect to a constituent parsing :

- Dependency parsing algorithms comprise the MST algorithm ($O(n^2)$ time) and some versions and approximations of shift-reduce parsers ($O(n)$), both faster than constituent-based parsing algorithms.
- Non-projective dependency structures can better capture the structure of free order languages. A constituent parsing is a projective parsing.

3.2 Parsing algorithms

We describe state-of-the art syntactic dependency parsing algorithms. The algorithms presented in this sections assumes that the reader is familiar with the *structured learning framework*, for an introduction and further details see section 3.4.

Two main categories of syntactic parsers can be defined:

- Top-down parsers
  Left-left parsers
- Bottom-up parsers
  Shift-reduce or Left-right parsers and also CKY-based parsers (e.g., Eisner parser) are considered bottom-up parsers

Next sections describes parsing algorithms.

3.2.1 MST

The Maximum Spanning Tree (MST) is the name given to the problem of finding a tree in a graph with the highest score.

As we are dealing with dependency (tree) graphs, MST is a natural and elegant framework to generate the syntactic structure. The application of the MST algorithm to dependency parsing was firstly introduced by McDonald (2005b). One the algorithms to solve the MST problem is the Chu-Liu-Edmonds algorithm (Edmonds, 1967).

Algorithm 3.1 is an exact algorithm that computes the MST. The sketch of the algorithm is the following. It selects the highest scoring incoming edge for each vertex. If a tree results, it is the MST. Otherwise the graph contains a cycle. It removes every cycle by contracting the cycle to a single node, recalculates the incoming edges to the new node, and selects the new maximum scoring edges. Note that non-projective

Algorithm 3.1 Chu-Liu-Edmonds

Input: vertices of the graph (sentence tokens) and score$(\cdot,\cdot)$ function
$M \leftarrow$ the highest scoring incoming arcs for each vertex $\{G_M$ is the subgraph induced by M arcs$\}$
while cycles(G_M) **do**
  $K = \emptyset$
  for all $c \in$cycles(G_M) **do**
    $k =$contract(c)
    $K \cup k$
    for all (i, j) s.t. $j \in$vertex(C), $i \in$vertex$(G) \setminus$vertex(c) **do**
      score$(i, k) =$score$(i, j) - (score(I, j) - max_j(\text{score}(I, k))$, where I is all node in vertex$(G) \setminus$vertex(c)
    end for
  end for
  for all $k \in K$ **do**
    $I = \arg\max_{I'} score(I', k)$
    $a = (I, k) \in S$
    $S \leftarrow S \setminus a$
    $S \leftarrow S \cup (I, k)$
  end for
end while
return G_M

dependencies are captured. The computational cost is $O(n^3)$, however faster $O(n^2)$ implementations can be found (Tarjan, 1977).

Unfortunately, this framework has not been successfully extended. Much more attention had received Eisner-based algorithms (McDonald and Pereira, 2006) as allowed a greater degree of flexibility, see the following sections.

3.2.2 Eisner-based parsers

The Eisner (1996) designed a dynamic programming algorithm similar to CKY and applicable to dependency parsing. The work of McDonald (2005a) renewed the interest in the application of this algorithm with conjunction with discriminative learning to dependency parsing. The Eisner's algorithm, its extensions and the MST algorithms are considered (2005a) graph based models and conforming the *Maximum Spanning Tree framework* approach.

This algorithm cost is $O(n^3)$. An important factor to ensure the efficiency of the algorithm is the concept of *span*. A span is a dependency structure for a substring with the properties of no outgoing nor incoming edges from the rest of the input sentence. Furthermore for efficiency reasons the root of the span must be in one of the span ends. The algorithm computes the optimal projective tree.

We distinguish tree variants:

- First order model (Eisner, 1996).
- Second order model
  - McDonald (McDonald and Pereira, 2006).
  - Carreras (Carreras, 2007).

The following sections details the original Eisner algorithm (first order model) and its extensions (second order models).

First Order

We begin defining the first order model. A detailed description of the First order model with the concerning aspects for the design of the joint model of this work can be found of section 5.6.1.

An arc-based first order factorization is fragmentation of the score of a tree by sum (or product) of arc-scores.

$$\text{score}(y) = \sum_{f \in y} \text{score}(f) \tag{3.2}$$

A factor f is an arc of y.
y is the dependency tree
In first order models a *factor* is an arc of the model thus $f = \langle h, m, l \rangle$. Tokens h and m are the head and the modifier. l is the syntactic label.

Usually the score of a factor is a linear function

$$\text{score}(f) = \mathbf{w} \cdot \phi(f) \tag{3.3}$$

$\mathbf{w}$ is a weight vector parameterizing the model and
ϕ is a feature extraction function. Note that usually in addition to the factor, the input sentence is also considered to extract features, and passed to the function ϕ.

Dependency parsing is modeled as the computation of the best scoring tree among all trees for the input x:

$$\text{best\_tree} = \arg\max_{y \in \mathcal{Y}(x)} \text{score\_tree}(y, x, \mathbf{w}) \tag{3.4}$$

where $\mathcal{Y}(x)$ is the set of all dependency trees for x.

The Eisner algorithm performs the required inference to solve the model (i.e., computes the arg max). The inference is restricted to projective trees (i.e., $\mathcal{Y}(x)$ is the set of projective trees for x).

Algorithm 3.2 Eisner First Order

Input: sentence of length n and score$(\cdot, \cdot)$ function
$C[s][t][d][c] \leftarrow 0, \forall s, t, d, c$
for $k = 1, \ldots, n$ **do**
  for $s = 0, \ldots, n - k$ **do**
    $t \leftarrow s + k$
    $C[s][t][\leftarrow][0] = \max_{s \le r < t} C[s][r][\leftarrow][1] + C[r+1][t][\rightarrow][1] + \text{score}(t, s)$
    $C[s][t][\rightarrow][0] = \max_{s \le r < t} C[s][r][\leftarrow][1] + C[r+1][t][\rightarrow][1] + \text{score}(s, t)$
    $C[s][t][\leftarrow][1] = \max_{s \le r < t} C[s][r][\leftarrow][1] + C[r][t][\rightarrow][0]$
    $C[s][t][\rightarrow][1] = \max_{s < r \le t} C[s][r][\leftarrow][0] + C[r][t][\rightarrow][1]$
  end for
end for

Table $C[s][t][\leftrightarrow][0]$ represents the best span from start token s to end token t. As previously said a span has one of its end as a head or *partial* root. The arrows indicates which end. Finally the last index can take values $\{0, 1\}$ these values stands for open and closed spans:

- open spans
  the span can be extended in both ends.

- closed spans
  are finished structures but can be extended in only one of its ends.

The algorithm can be represented in a graphical form, see figure 3.2.

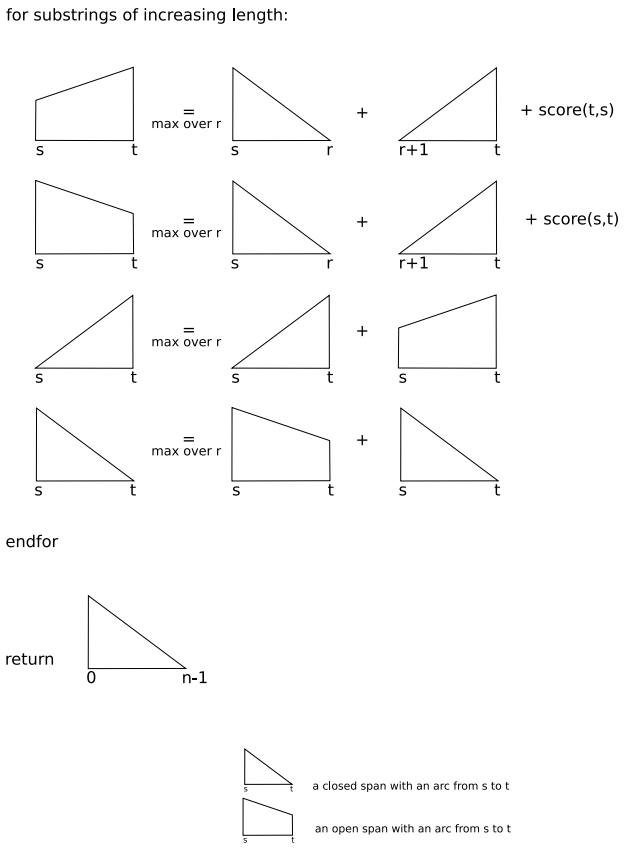

Figure 3.2: Eisner algorithm

The key point in algorithm 3.2 is that spans can be easily combined bottom up. Usually the first sentence token is the root. This token is added if not present. The best dependency tree is the span with the head on root that covers the complete sentence.

The computational cost is $O(n^3)$. Recall that it is a non-projective exact search algorithm. The arc-factorization to scoring is its main drawback, see section 3.1 and following sections for extensions.

Second Order McDonald

A second order model proposed by McDonald (McDonald and Pereira, 2006) alleviates some of the first order factorization limitations. The

model factorizes the tree in immediate adjacent dependencies:

$$\text{score\_tree}(y) = \sum_{\langle h,s,m,l\rangle \in y} \text{score}(h, s, m, l) \tag{3.5}$$

Now a factor $\langle h, s, m, l\rangle$ comprises a dependency $\langle h, m, l\rangle$ and the nearest dependency $\langle h, s, l_2\rangle$. The nearest dependency should not cross a span (i.e., it is only considered if it is inside the span being processed).

Algorithm 3.3 shows the Eisner inference implementation for this second order model.

Algorithm 3.3 Eisner Second Order McDonald

Input: sentence of length n and score$(\cdot, \cdot)$ function
$C[s][t][d][c] \leftarrow 0, \forall s, t, d, c$
for $k = 1, \ldots, n$ **do**
  for $s = 0, \ldots, n - k$ **do**
    $t \leftarrow s + k$
    {sibling construction}
    $C[s][t][\leftrightarrow][2] = \max_{s \le r < t} C[s][r][\leftarrow][1] + C[r + 1][t][\rightarrow][1]$
    {building of a dependency without sibling}
    $C[s][t][\leftarrow][0] = C[s][t - 1][\leftarrow][1] + C[t][t][\rightarrow][1] + \text{score}(t, -, s)$
    $C[s][t][\rightarrow][0] = C[s][s][\leftarrow][1] + C[s + 1][t][\rightarrow][1] + \text{score}(s, -, t)$
    {building of a dependency with sibling}
    $C[s][t][\leftarrow][0] = \max_{s \le r < t} \{C[s][t][\leftarrow][0],$
      $\max_{s \le r < t} C[s][r][\leftrightarrow][2] + C[r][t][\rightarrow][0] + \text{score}(t, r, s)\}$
    $C[s][t][\rightarrow][0] = \max_{s \le r < t} \{C[s][r][\leftarrow][0],$
      $\max_{s < r \le t} C[r][t][\leftarrow][0] + C[r][t][\leftrightarrow][2] + \text{score}(s, r, t)\}$
    {building of complete spans}
    $C[s][t][\leftarrow][1] = \max_{s \le r < t} C[s][r][\leftarrow][1] + C[r][t][\rightarrow][0]$
    $C[s][t][\rightarrow][1] = \max_{s < r \le t} C[s][r][\leftarrow][0] + C[r][t][\rightarrow][1]$
  end for
end for

The algorithm is very similar to the first order algorithm. The changes are in the scoring and labelling of a dependency. In first order models two spans of lesser size are combined. In this case we chose between combining two lesser size spans (i.e. a dependency without siblings) or the two lesser size spans a sibling span (i.e., dependency with siblings). Note the new index "2" of the dynamic programming table that accounts for sibling structures.

The algorithm runs in $O(n^3)$. McDonald and Satta (2007) proved that the non-projective version of this model is NP-hard to solve. Although a modest enhancement of the first order model significant im-

provement are reported (McDonald and Pereira, 2006).

Second order carreras

Carreras (2007) improved the previous second order model. The new model is intended to deal with the PP-attachment problem, see section3.1, capturing some grandson relations.

A factor now is defined as a $f = \langle h, m, c_h, c_{mi}, c_{mo} \rangle$.
where,
h is the head,
m the modifier
c_h is the closest child of h
$c_m i$ is the furthest child of m inside the actual span
$c_m o$ is the furthest child of m from the span to be combined.
As in the previous second order extension this relations are bounded by spans. In the case of c_h it only can be considered if it is inside the span from h to m.

The algorithm 3.4 also visits bottom-up the span table C. The main difference is that now the table entries are indexed by labels and grandsons.

- closed structures
  Are indexed by the start of the span s, the end token t, and in this case also by the syntactic label l.

- open structures
  Are indexed by the start of the span s, the end token t, and in addition the a child m. m is the closet child to the head of the span s or t.

Thus a space of $O(n^2L + n^3)$ is required. Note that in algorithm 3.4 we used a combined table $O(n^4L)$ only for clarity reasons.

Again significant improvements were reported (Carreras, 2007) at a expense of a higher algorithmic cost $O(n^4)$. Note that both second order extensions only regards the adjacent sibling and grandsons relations to the head, restricted to the span concept.

3.2.3 Shift-reduce parsers

A shift-reduce parser, bottom-up parser or LR parser can be defined as an algorithm that parses a sentence from left to right and uses a stack

Algorithm 3.4 Eisner Second Order Carreras

Input: sentence of length n and score$(\cdot,\cdot)$ function
$C[s][t][l][m][d][c] \leftarrow 0, \forall s, t, l, m, d, c$
for $k = 1, \ldots, n$ **do**
  for $s = 0, \ldots, n-k$ **do**
    $t \leftarrow s + k$
    {open structures.}
    {Note $\max_{r,c_h,c_{mi}}$ is $O(n^2)$ as c_h and c_{mi} are independent}
    $\forall l \quad C[s][t][l][-][\leftarrow][0] = \max_{r,c_h,c_{mi}} C[s][r][-][c_h][\leftarrow][1] + C[r+1][t][-][c_{mi}][\rightarrow][1] + \text{score}(t, s)$
    $\forall l \quad C[s][t][l][-][\rightarrow][0] = \max_{r,c_h,c_{mi}} C[s][r][-][c_h][\leftarrow][1] + C[r+1][t][-][c_{mi}][\rightarrow][1] + \text{score}(s, t)$
    {closed structures}
    $\forall r \quad C[s][t][-][r][\leftarrow][1] = \max_{c_{mo},l} C[s][r][l][-][\leftarrow][1] + C[r][t][-][c_{mo}][\rightarrow][0]$
    $\forall r \quad C[s][t][-][r][\rightarrow][1] = \max_{c_{mo},l} C[s][r][-][c_{mo}][\leftarrow][0] + C[r][t][l][-][\rightarrow][1]$
  end for
end for

of symbols. Note that this abstract definition of the algorithm is not directly applicable to bottom-up Eisner parsers.

Let $\{\alpha, w\}$ the configuration of the parser in a given time.
α is a stack of symbols or partially processed items.
w the remaining input sequence.

Two basic processing rules defines a bottom up parser:

- shift
  $\langle \alpha, aw \rangle \rightarrow \langle \alpha a, w \rangle$
- reduce
  $\langle \beta\alpha, w \rangle \rightarrow \langle \beta A, w \rangle$
  for some grammar rule or transformation $A \longrightarrow \alpha$.

Note that the algorithm is inderministic and searches for any path from $\langle \lambda, w \rangle$ resulting in a configuration of the form $\langle S, \lambda \rangle$. Where λ is the empty string symbol, and S the start symbol of a grammar.

3.2.4 MaltParser

Shift-reduce parsers are applied in the context of data-driven paring by Nivre et al. (2003) and Covington (2001). These parsers extend

the generic shift-reduce parsing previously defined. They can be fully described by T and C as following.

- T is the set of transitions with transition functions $t \in T$.
- C is the set of configurations, with initial and terminal states.

A configuration $c = \langle \alpha, w, A \rangle$ is:

- α is a stack of tokens $i \geq m, m \geq n$
- w is the remaining part of the word.
- A is a set of labeled dependency arcs, (i.e., $A = \{i \xrightarrow{l} j\}$) . The induced graph by A, $G_A(\text{vertices} = \{0, \ldots, n\}, \text{arcs} = A)$ is a dependency graph.

Given a configuration we define a transition. A *transition* is a function $t : C \rightarrow C$. It is a mapping from configurations to new configurations, it is not necessary defined for all configurations in C. $S = \langle C, T, c_s, C_t \rangle$

Stack operations noted by $\alpha|i$ represents that i is on top of the stack and α is the remaining part of the stack. String operations noted by $j|w$ represents that j is the first token of the concatenated sequence jw. The initial configuration of the parser for an input string x is $C(\alpha, w, A) = \{\lambda, x, \emptyset\}$.

We define the set of transitions between states for the arc-eager improved Nivre's variant:

- left arc for label l
  $\langle \alpha|i, j|w, A \rangle \rightarrow \left\langle \alpha, j|w, A \cup (j \xrightarrow{k} i, k) \right\rangle$
  if i is not the root node
  and there is no previous assigned head for token j (one head constraint).
- right arc for label l
  $\langle \alpha|i, j|w, A \rangle \rightarrow \left\langle \alpha|i|j, w, A \cup (i \xrightarrow{k} j, k) \right\rangle$ if there is no previous assigned head for token j.
- reduce
  $\langle \alpha|i, w, A \rangle \rightarrow \langle \alpha, w, A \rangle$
  if i has yet an assigned head.
- shift
  $\langle \alpha, i|w, A \rangle \rightarrow \langle \alpha|i, w, A \rangle$

The previous parsing model definition is non-deterministic, i.e., the set of transitions to parse the input is not defined. Usually (Nivre et al., 2007b) we isolate the non-deterministic component of the parser in an *oracle* function, later this function can be approximated by a deterministic function.

Algorithm 3.5 Oracle shift-reduce parsing

```
initial configuration c = {λ, x, ∅}
while is_not_terminal(c) do
  if α = λ then
    c ← shift(c)
  else
    t ←oracle(c)
    c ←t(c)
  end if
end while
return graph(c)
```

Algorithm 3.5 shows the parsing algorithm using the generic oracle function. Note that the algorithm is linear if the *oracle* function is constant-time.

The parser must perform a sequence of decisions or transitions defined by the *oracle* function. These decisions can be conditioned on the previous history. A classifier can be trained receiving as features the history but also features from the tokens on top of the stack, the first token of the remaining input and from dependencies in the partially constructed graph G_A. SVM and MBL(Memory-Based Learning e.g., k-nearest neighbour)(Daelemans and van den Bosch, 2005) are widely chosen machine learning methods for this task, partially because are the implemented methods in *MaltParser*

Nivre arc-eager parser is a linear time parser. Although a $O(n)$ algorithm the training is costly. The constraints imposed by the admissible transitions restricts the set of parseable dependency trees to the projective trees and even in this case it is not an exact search algorithm.

3.2.5 Projectivization algorithms

The non-projective dependencies are the crossing links in dependency graphs, see section 3.1. Widely used algorithms for syntactic parsing are unable to generate them. This is a problem not only because errors will be produced but also because the corrections performed by

learning algorithms could cause further misclassifications. Next subsections describes common approaches to transform or projectivize a non-projective dependency graph.

McDonald's deprojectivization algorithm

McDonald and Pereira (2006) proposed a simple algorithm as a postprocessing to transform a predicted projective graph into a non-projective using the scoring functions trained by the learning algorithm.

Algorithm 3.6 Pseudoprojective McDonald's technique

```
Input: sentence of length n and scoreTree(·) function
while ¬ max iterations do
  bestIncreaseScore ← 0
  increaseScoreFound ← FALSE
  for newModifier← 1, ... n and newHead← 0, ..., n do
    if (newHead, newModifier) ∉ y then
      y' = y − (actualHead(newModifier, y), newModifier)
      y = y ∪ (newHead, modifier)
      if isAWellFormedTree(y') then
        increaseScore ← scoreTree(y')−scoreTree(y)
        if increaseScore > bestIncreaseScore then
          bestIncreaseScore ← increaseScore
          increaseScoreFound ← TRUE
          bestHead ← newHead
          bestModifier ← newModifier
        end if
      end if
    end if
  end for
  if increaseScoreFound then
    {// delete the old head and modifier}
    y ← y ∪ (head(bestModifier), bestModifier)
    {//add the new one}
    y ← y ∪ (bestHead, bestModifier)
  end if
end while
```

Algorithm 3.6 simply tries to substitute a predicted arc for another arc with a higher score.

Nivre and Nilsson pseudo-projectivization

Nivre and Nilsson (Nivre and Nilsson, 2005) proposed an algorithm to transform the input non-projective dependency tree to a projective one, and latter recover the original structure. The algorithm consists in lifting the non-projective arcs until they become projective. A special tagset is added to allow a posterior reconstruction.

Algorithm 3.7 Pseudoprojective Nivre and Nilsson generic algorithm

```
Input: parse tree y
while isNonProjective(y) do
    a ← the smallest non-projective arc of y)
    y = y − a∪ lift(a, A)
end while
return y
```

Algorithm 3.7 defines the procedure. The *smallest non projective arc* is the non-projective arc with the shortest distance from the head to the dependent. It can be proved that all non-projective trees can be projectivized by a finite set of lift operations. The *lift* operation substitutes the head of the modifier by the grandfather of the modifier.

$$\text{lift}(h \to m) = \begin{cases} g \to m & \text{if } g \to h \\ \text{undefined} & \text{otherwise} \end{cases} \tag{3.6}$$

The lifted arc are relabeled. Is it possible to include in the label the precise information to reconstruct the original path, but this will increase excessively the number of labels and so the training time. Empirical results (Nivre and Nilsson, 2005) showed that a good strategy is the *head+path* strategy.

In the *head+path* strategy is defined as follows. Let $h \overset{d_{hm}}{\to} m$ be the non projective arc from h to m with label d_{hm}. And let the grandfather of h be g. Thus we use the $g \overset{d_{gh}}{\to} h$ to lift the non-projective dependency. The arc $h \overset{d_{hm}}{\to} m$ is substituted by $g \overset{d_{hm}\downarrow d_{gh}}{\to} m$ and the $g \overset{d_{gh}}{\to} h$ arc is relabeled as $g \overset{d_{gh}\downarrow}{\to} h$.

Using this strategy, the reconstruction of the original non-projective tree is not always possible. A variety of approaches can be employed, including sophisticated machine learning techniques.

A simple strategy is: If we found an arc $g \overset{d_{hm}\uparrow d_{gh}}{\to} m$ on output, we search the descendants of g top-down and left-right arc $g \overset{d_{gh}\downarrow}{\to} h$. If

no node $g \overset{d_{gh}\downarrow}{\rightarrow} h$ was found we search the descendants of g without constraining the search to the nodes with a path tagged label. If no node was found we remove the tags of the arc and we do not move the dependency.

The algorithm allows us present to learning/inference method only projective sentences. The sentences are transformed before and after they are parsed.

3.3 Semantic Parsing

Shallow semantic parsing or Semantic Role Labelling (SRL) is the task of identifying the *who,when,what,why* of the sentence predicates. It comprises also the annotation of adjunct arguments of the predicates (e.g, locative, temporal, manner and cause arguments). These semantic dependencies express semantic links between predicates and arguments and represents relations between entities and events in text.

Noun and verbs can behave as predicates and receive semantic arguments. The arguments a predicate can take are defined in frames and are also called *roles*. The most frequent roles are *agent*, *patient*, *instrument* but also the *locative, temporal, manner* and *cause* adjuncts.

Usually the semantic dependency task was only focused in verbal predicates. Most of published work is based on SRL for verb predicates and assumes a constituent structure as starting point. The CoNLL-2008 shared task introduced a new framework. Noun and verb predicates are processed and semantic arguments are regarded as semantic *dependencies*. See section 2.1.2 for a details. The introduction of the dependency formalism in SRL does not modifies the consideration of SRL simply as a classification problem.

The semantic dependency graphs are simple graphs, only relating the predicate with all its arguments by single links, i.e., the graph has one head and a set of semantic dependents. There is no required connection between the dependencies across the sentence predicates. If we merge the semantic graphs from all predicates a DAG is conformed, without any further constraint.

Figure 3.3 shows a sample sentence with nominal and verbal predicate. Note that the last predicate *ends* is a noun.

SRL is mainly approached by a four step architecture (Màrquez et al., 2006). The architecture is implemented in these phases: pruning, local scoring, identification and classification and finally joint scoring.

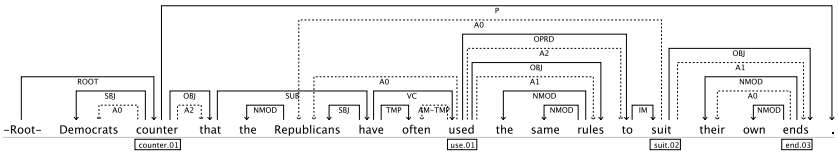

Figure 3.3: Nouminal and verbal predicates

Some systems apply small variations. Other approaches are the BIO tagging (Màrquez et al., 2005) and the tree CRF (Cohn and Blunsom, 2005).

3.3.1 Mainstream approach

We describe each one of the architecture steps of this approach. A syntactic structure and the identification of sentence predicates is assumed.

- Pruning or filtering.
  A set of simple rules introduced by Xue and Palmer (2004) are commonly used to discard improbable token candidates. There is a huge amount of non-argument tokens with a non-balanced distribution with respect to argument tokens (i.e., most tokens are not arguments). This large number of non-arguments and the uneven distribution degrades the performance and increases the cost of machine learning algorithms. The filter step alleviates this problem with almost no reduction in the classification upper bounds for the subsequent phases.
- Local scoring.
  Each argument candidate is scored or a computed probability is assigned. We note this scoring as *local* because information about other candidates is not used (i.e., the scoring does not depends on the scoring of other candidates).
- Argument identification and classification
  Typically a first classifier is applied to discard candidates and a second classifier assigns the best scoring semantic label to each selected candidate.
- Global scoring or postprocessing
  The classified arguments are globally combined in this phase. Domain constraints (e.g., a predicate cannot take repeated core arguments) can be enforced.

One of the most interesting point of this architecture is the global scoring step. The step can be performed by a wide variety of techniques.

- ILP.
  Integer Linear Programming (ILP) is used to ensure domain constraints (Punyakanok et al., 2004), see also 3.5.5.
- Reranking.
  Rereanking selects the best solution among the pool of candidates generated (Toutanova et al., 2007).
- Probabilistic models:
  - generative models (Thompson et al., 2003)
  - sequence tagging (Màrquez et al., 2005)
  - CRF on tree structures (Cohn and Blunsom, 2005)

Finally a frequently used technique is to combine the output of various SRL system. There is not a clear distinction between *global scoring* and *ensemble* or *combination* of outputs. The global scoring techniques are useful to combine system and in reverse, a global scoring could benefit from a richer set of inputs.

3.3.2 BIO tagging

The semantic annotation can be performed by a sequential labeling of chunks (Pradhan et al., 2005; Màrquez et al., 2005). A generic BIO tagging architecture can be regarded as:

- Sequentialization.
  Clause boundaries and other syntactic information is used to define a chunking of the sentence in suitable fragments.
- Scoring.
  Nodes are labelled with BIO tags.
- Global inference.
  Conflicting previous labellings can be treated.

The scoring step tags each chunk with one or more of the following labels:

- (B) Beginning of an argument.
- (I) Inside an argument.
- (O) Outside an argument.

Some systems uses separate BIO labels for each argument. The previous history can be used as a feature to predict the current BIO label. Some constrains can be enforced at this stage, e.g., a correct BIO structure, to not cross boundary clauses (an algorithm cannot span across sentence clauses), and the no assignation of non accepted arguments for the given predicate.

Finally, and specially if multiple labels were assigned to the same token (i.e., a inside and outside label for the same argument but different probabilities), postprocessing or inference must be applied to select the best combination.

3.4 Structured learning framework

Machine learning tasks are being increasingly applied to complex problems. In this work a central topic is the prediction of a syntactic structure, a structured learning task.

We begin reviewing the classical framework for supervised machine learning. Let S be training set with pairs of instances (x, y) where $x \in X$ is the feature representation of the instance, $y \in Y$ is class (a structure in structured learning)

We assume that the collection of examples (x, y) is drawn i.i.d from an unknown distribution D over $X \times Y$.
A loss function $l(y, \hat{y})$ is the cost of proposing $\hat{y}$ when the correct values was y. The goal of a learn algorithm is to find a classifier h that minimizes the loss function and usually also a regularization function.

$$h : X \to Y \tag{3.7}$$

The structured learning problem is defined in an analogous way. In this case y is an structured output. Table 3.1 summarizes the differences carries by the use of structures.

classification	**classes** Y	$\|Y\|$	**enumeration of** $\|Y\|$	**loss**
Binary	$\{+, -\}$	1	not required	0-1 (usually)
Multiclass	$c_1, \ldots, c_n$	n	exhaustive	0-1 (usually)
Structured	all structures (trees,..)	exponential	intractable	precision/recall (usually)

Table 3.1: Binary vs. structured learning

A general form for a structured classifier h is:

$$h(x) = \arg\max_{\hat{y} \in \mathcal{Y}} \text{score}(x, \hat{y}) \tag{3.8}$$

Note that the classification function cost is determined by the argmax (i.e., the size of $\mathcal{Y}(x)$) and the scoring function. Thus scores must be computed for all possible structures $\mathcal{Y}(x)$. $\mathcal{Y}(x)$ is exponential on the size of x. In our case, the learning of tree structures, there are $n!$ possible parse trees for a sentence of length n.

The overcome this problem the key idea is to decompose the structure to predict into fragments and combine these fragments to build the output incrementally. Note also that this factorization will probably carry an approximate search of the best output solution $\hat{y}$.

Thus simple decisions are learnt by a classifier and the chained by an inference algorithm, therefore the name *learning-inference* paradigm (Punyakanok et al., 2005). All discriminative learning algorithms and a wide range of inference algorithms are available. Among the inference algorithms are: BIO tagging, Open-close tagging, Shift-reduce, Dynamic programming, viterbi decoders, beam search. Some statistical models (e.g., HMM, CRF, PCFG) also assumes a factorization of the generative prediction task. Simpler probabilities or decisions are estimated or learn and combined.

The use of discriminate learning algorithms offers some advantages over probabilistic approaches. Probabilistic approaches assumes strong independence assumptions, and usually suffer sparsity problems and high computing cost. In the use of discriminative learning is a much more flexible tool, not only local classifiers for single decisions can be learn but also some global learning is achievable. Global training is intended to correct local decisions only when combined are performed incorrectly.

The *learning-inference* paradigm is the framework we use to accomplish the structure prediction task. Next sections will review *inference* algorithms and *learning* methods used in conjunction with the formers. Note that in some cases we distinct the actual *inference model* from the implemented *inference* algorithm, as sometimes the inference algorithm performs an approximate search over a space easily defined as a *model*.

3.5 Machine learning methods

In this section we cover machine learning algorithms used in state-of-the-art syntactic and semantic dependency parsing, focusing in the applied to this work. Integer linear programming, is also covered in this section

3.5.1 Structured Perceptron

The *Structured Perceptron* (Collins, 2002) is a variant of the well-known *Perceptron* (Rosenblatt, 1958).

The main benefit of this algorithm is its low cost in time and memory, as it is a simple algorithm. The memory requirements in its primal form are independent of the size of the training set. This is an important factor in the context of the large corpus of the shared task.

Algorithm 3.8 Structured Perceptron

```
Input: examples (x, y) from the training set
w ← 0
for t ← 1 to numEpochs do
  for all (x, y) ∈ training set do
    ŷ =inference_algorithm(x, w)
    for all factor ∈ y \ ŷ do
      w^(l) ← w^(l) + φ(factor, x, ŷ)
    end for
    for all factor ∈ ŷ \ y do
      w^(l) ← w^(l) − φ(factor, x, ŷ)
    end for
  end for
end for
```

In a typical dependency parsing setting the *inference_algorithm* will be the parser (e.g., Eisner) that generates the dependency tree $\hat{y}$ and a *factor* will be a dependency of that tree (e.g., $factor = \langle h, m, l \rangle$).

For each misclassified instance, we update the weights of missing(i.e., factor $\in y \setminus \hat{y}$) and overpredicted factors (i.e., factor $\in \hat{y} \setminus y$). An alternative is to score possible factors for a given input an ensure that the correct factors are scored higher than all the rest.

In our framework, a dual version of the perceptron is infeasible. The memory requirements will be the following: the number of instances is about 40k, the number of features per dependency is about 200k, therefore we will need to have in memory about $40k \cdot$ num_of_dependencies $\cdot 200k$ floating point numbers for each label.

3.5.2 Reranking perceptron

An extension of the perceptron algorithm to perform reranking of solutions was presented by Collins (2002) and extended by Shen et al. (2005).

Algorithm 3.9 is very similar to algorithm 3.8. The *GEN* function generates all the predicted structured outputs. Note that $|\text{GEN}(x)|$ is exponential in the case of tree predictions for x, thus this function must be constrained.

Algorithm 3.9 Reranking Perceptron

Input: examples (x, y) from the training set and *GEN* function
$w \leftarrow 0$
for $t \leftarrow 1$ to numEpochs **do**
  for all $(x, y) \in$ training set **do**
    $\hat{\mathcal{Y}} = \text{GEN}(x)$
    $\hat{y} = \arg\max_{\hat{y}' \in \hat{\mathcal{Y}}} \text{score}(x, y', \mathbf{w})$
    for all factor $\in y \setminus \hat{y}$ **do**
      $w^{(l)} \leftarrow w^{(l)} + \phi(\text{factor}, x, \hat{y})$
    end for
    for all factor $\in \hat{y} \setminus y$ **do**
      $w^{(l)} \leftarrow w^{(l)} - \phi(\text{factor}, x, \hat{y})$
    end for
  end for
end for

The main difference with respect to the previous algorithm is the treatment of the generated solutions. In the first case (structured perceptron), the inference algorithm outputs the best solution x. In the second case, we first generate using *GEN* the n best outputs for x, we constrain the number to n to avoid the exponential number of outputs. Then each one of the generated solutions are scored. The key point is that the score is a global function over the complete output and input, i.e., features about the whole solution $\hat{y}$ are used, these are not restricted to local features or features about factors. Global features are regarded as necessary to capture properties of the complete solutions.

It is fast algorithm usually applied to combine different systems output.

3.5.3 Passive-aggressive perceptron and MIRA

Crammer et al. (2006) presented the passive-aggressive perceptron family, a set of improved perceptron-inspired algorithms. The Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003) can be regarded as a passive-aggressive perceptron, described below.

The main difference from the perceptron algorithm is that we are now trying to perform the correct prediction with a minimum margin. Note that this is not the maximum-margin setting, furthermore the algorithm is an on-line approximate algorithm.

Passive-aggressive perceptron brings new update rules for the weight vector of perceptron algorithm. The new rule is based on the following equation.

$$\mathbf{w}_{t+1} = \arg\min_{w} \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 \quad \text{s.t.} \quad l(\mathbf{w}, (x, y)) = 0 \tag{3.9}$$

where $\mathbf{w}_t$ and $\mathbf{w}_{t+1}$ are the weight vectors for epoch t and $t+1$
$l(\mathbf{w}, (x, y))$ is a loss function with a minimum classification margin of 1.

$$l(\mathbf{w}, (x, y)) = \begin{cases} 0 & \text{if } |y - w \cdot x| \leq 1 \\ 1 - y(w \cdot x) & \text{otherwise} \end{cases} \tag{3.10}$$

Thus we are interested in the minimum update to the weight vector (i.e, $\arg\min_w \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2$ such that the new weight vector correctly performs the classification with a margin 1. This rule will force significative updates to achieve a 0 loss and no updates when the loss is still 0, therefore the name of *passive-aggressive* perceptron.

Applying Lagrangian multipliers it can be proved (Crammer et al., 2006) that the equation 3.9 is translated to the following perceptron update rule

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \tau_t y x \tag{3.11}$$

where (x, y) is the training sample for epoch t,
τ_t is defined as:

$$\tau_t = \frac{l(\mathbf{w}, (x, y))}{\|x\|^2} \tag{3.12}$$

Thus the difference with respect the perceptron is that we compute τ_t instead of a fixed learning rate for the perceptron update rule.

Note that the of zero loss constraint is too hard for noisy or non-separable data. Few variants are intended to deal with this problem. The first simply set a parameter C to limit the maximum update rate τ_t:

$$\tau_t = \min\left\{C, \frac{l(\mathbf{w}, (x, y))}{\|x\|^2}\right\} \tag{3.13}$$

The update rule in equation 3.13 is named*PA-I*.

The second option is a little bit more sophisticated:

$$\tau_t = \frac{l(\mathbf{w}, (x, y))}{\|x\|^2 + \frac{1}{2C}} \tag{3.14}$$

The update rule in equation 3.14 is named*PA-II*.

The two equations 3.13 and 3.14 can be justified again by the solution of an optimization problem using Lagrange multiplies, the proof can be found in Crammer et al. (2006).

The PA-I associated optimization problem to derive the update rule is:

$$\mathbf{w}_{t+1} = \arg\min_w \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 + C\xi \quad \text{s.t.} \quad l(\mathbf{w}, (x, y)) \leq \xi, \xi \geq 0 \tag{3.15}$$

In this case we introduced the parameter C and slack variables ξ. The slack variables contains the value of the errors made. In this case we jointly minimize

the weight update and the errors, the lasts determined by a the parameter C. No strict zero loss constraint appears, the loss is introduced as the slack variables.

The PA-II associated optimization problem to derive the update rule is:

$$\mathbf{w}_{t+1} = \arg\min_{w} \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 + C\xi^2 \quad \text{s.t.} \quad l(\mathbf{w}, (x, y)) \leq \xi \tag{3.16}$$

It is an alternative to the previous equation with quadratic slack variables.

The application of passive-aggressive perceptron to structured output classification is performed in the same way as for the perceptron algorithm.

A proof of the convergence properties is detailed in Crammer et al. (2006). The simple update rules for all variants are fast to compute leading only to a slight increase in computation time with respect to the perceptron algorithm. This makes the algorithm suitable for large-data tasks. Note the algorithm solves one optimization problem for each one of the training samples. The optimization only regards the current instance and weight vector.

3.5.4 SVM

The SVM is a linear classifier (Boser et al., 1992),

$$y = \langle w, x \rangle + b \tag{3.17}$$

The training algorithm computes the parameters w and b that minimizes:

$$\min \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \quad \text{subject to} \quad y(\{w, x\} + b) \geq 1 - \xi_i \tag{3.18}$$

where $\xi = \max(0, 1 - \gamma_i)$
γ_i is the value of the margin for instance i
C is a parameter that penalizes incorrectly classified instances. Note that to maintain a consistent notation the instances x and classes y are not indexed by i.
This optimization problem leads to the maximum margin separation of the two classes.

The previous formulation is the *soft-margin* formulation of SVMs that allows us to deal with non-separable classes. All dot products $\langle x, y \rangle$ can be efficiently computed by a kernel function $k(x, y) = \langle x, y \rangle$ that can represent a dot product in a projected space. Kernel functions are widely used to transform the linear classifier into a much complex classifier. The most used kernel functions are:

- linear kernel $k(x, y) = x \cdot y$
- polynomial d-degree kernel $k(x, y) = (x \cdot y + c)^d$

- radial kernel $k(x, y) = \frac{\|x-y\|^2}{2\sigma^2}$

In our case, the problem to optimize in SVM training could be impractical to be solved due to the large amount of training data.

3.5.5 Integer Linear Programming

In Integer Linear Programming (ILP) we want to maximize (or minimize) a linear function (Schrijver, 1998):

$$\begin{aligned} &\text{maximize } f(x_1, \ldots, x_n) \\ &\text{subject to } x_i + \ldots + x_j < b_k \end{aligned} \tag{3.19}$$

ILP problems can be solved by the simplex algorithm (Schrijver, 1998). ILP solving is a NP-hard problem (Karp, 1972). Long executions must be cut but most times the result can be computed in polynomial time, implemented ILP solvers use many optimization tricks to alleviate this problem. The application of this technique to SRL can be found in section 5.9.

3.6 Feature extraction and selection

A large amount of features is used for syntactic and semantic parsing. Features are commonly counted in millions and represented by binary vectors.

Typically the training examples are the dependencies $d = \langle h, m, l \rangle$ of each input sentence. The features are not just extracted from dependencies d but also from the rest of the input sentence x. Thus the feature extraction function ϕ can be noted as $\phi(\langle h, m, l \rangle, x) = v$, where $v \in \{0, 1\}^n$ is the feature representation of the dependency d.

The feature extraction process is usually implemented as follows. Strings representing features are extracted from $(\langle h, m, l \rangle, x)$, each string can be a concatenation of extracted properties (e.g POS(h)·POS(m)·POS(h+1)·POS(h+2)). The the strings are indexed and ϕ sets the corresponding component of the binary feature vector v to *true*.

The feature sets widely used (Màrquez et al., 2006) for syntactic parsing and semantic role labelling are defined in McDonal (2005b), Carreras et al. (2006), Xue and Palmer (Xue and Palmer, 2004) and Surdeanu (Surdeanu et al., 2007). A detailed coverage of these features and few new features is to be found in section 5.10.

As millions of features are typically extracted a feature selection method is required. Two techniques, frequency threshold filtering and backward selection, are the choice of state-of-the-art systems (Carreras and Màrquez, 2005; Nivre et al., 2007a). Despite of the simplicity of these two methods consistent high results are obtained.

The well known backward selection method is typically applied to groups of features. In each step not a feature is removed but a whole group of them. The group to remove is selected by training the system with all feature groups except one and selecting the removed group that mostly degrade the system performance.

As we are concerned on the feasibility of our complex system, backward selection is not a suitable approach. The huge amount of extracted features (millions) difficult the application of any other feature selection method but clearly this topic should receive more attention. We limited the scope of this work to feature selection methods applied in the context of the state-of-the-art SRL and dependency parsing systems.

Chapter 4

State of the art

Contents

This chapter outlines the state of the art of dependency parsing and shallow semantic parsing, the two main task to be performed by the CoNLL-2008 shared task participating teams. The following sections contains the main aspects of the concerned state of the art and briefly reviews remarkable complete systems. For important details and a discussion of referenced algorithms, methods and techniques please refer to the *background* section.

4.1 Syntactic dependency parsing

The CoNLL-2006 and CoNLL-2007 shared tasks (Buchholz et al., 2006; Nivre et al., 2007a) were devoted to syntactic dependency parsing. These tasks boosted the research on dependency parsing. Several presented system contributed with novel and competitive methods.

Team	English LAS score (%)
Carreras	89.61
Sagae	89.01
Nakagawa	88.41
Titov	88.39
Nilsson	88.11

Table 4.1: Best CoNLL-2007 systems for the English language

Table 4.1 shows[1] the top 5 among the 23 presented systems at the CoNLL-2007 shared task closed challenge. Note the shared task evaluated 10 languages and we are only focusing on the English language. We briefly review the top performing systems.

Team	**prep**	**model**	**inference**	**learning**	**comb**
Carreras	no	Eisner-Carreras	Eisner-Carreras	perceptron	—
Sagae	yes	Shift-reduce	LR greedy best-first, left-right	SVM	—
Nakagawa	no	probabilistic model + classification	MST + classification	Gibbs sampling + SVM	—
Titov	yes	shift-reduce	probabilistic, beam of sequences	ISBN	—
Nilsson	yes	shift-reduce	arc-eager	SVM	CLE

Table 4.2: Summary of top performing CoNLL-2007 shared task systems

Table 4.2 summarizes the architecture of these 5 systems. Combination and projectivization tricks are widely employed. The systems marked with a preprocessing step use the Nivre and Nilsson pseudo-projective algorithm, see section 3.2.5. Most systems uses algorithms that build the annotated syntactic tree in one step. An exception is the Nakagawa two-step process. The first step is the dependency graph generation and the second the annotation of each dependency with a syntactic label. We represented by "+" the concatenation of the two processes. ISBN (Titov and Henderson, 2007) are Bayesian probabilistic graphical models and Gibbs sampling (Geman and Geman, 1984) is a method for estimating probabilities.

Unfortunately most participants did not reported the consumed resourced for training but it is suspected that top performing teams invested huge amounts of computational resources. We can see that shift-reduce and Eisner-based models are the mainstream approaches. Some top performing system employed ISBN networks and probabilistic models but these techniques are not widely used by other participants (Nivre et al., 2007a). Eisner-based models and shift reduce parsers offers simplicity, reduced training times and a competitive performance as advantages with respect to these other approaches.

Classification tasks, if tractable, are usually performed by SVM classifiers. In Eisner-based models it is only practical to employ the perceptron algorithm, or at a higher cost, MIRA, see the algorithms descriptions in section 3.5.

As we just seen, state-of-the-art syntactic dependency parsing is based

[1] See `http://depparse.uvt.nl/depparse-wiki/AllScores` for detailed scores

mainly on shift-reduce parsers and Eisner-based algorithms. The features defined by McDonald et al. (2005b) and Carreras et al. (Carreras et al., 2006) are widely used with almost no modifications.

A central concept in Eisner-based parsers is the edge factorization. Shift-reduce parsers are a more complex approach. In fact the scoring of an dependency can also be viewed as and edge-factored scoring using history features but taking into account that previous to the dependency scoring a chain of decisions based also on the parser history features were made.

The edge-factorization implies that the score *(probability)* of a dependency is not related *(conditionally independent)* with respect to other dependencies. This is an unrealistic simplification and one of the major drawbacks of this factorization. A well known problem in dependency parsing is the ambiguity on the attachment of prepositional phrases. In this case the *head*, the preposition and the son form a chain of 2 edges. It is believed that the main information is carried by the son of the preposition. Thus edge-factored models will be unable to exploit this information.

Despite this factorization, we have to note that context information, not at syntactic level but a surface level (i.e., the surrounding words of a dependency) is widely used as a feature. Higher order extensions are intended to capture more context information at a expense of higher computational cost, see section 3.2.2. But this methods can only consider a limited set of second order dependencies (i.e., just an immediately adjacent dependency not crossing the position of the head). Another strategy to widen the limited scope of the edge factorization is to use reranking or systems combination, see section 3.9. Global properties of the output can be used as features. All this strategies contributes with significant performance improvements. And recall that the edge factorization is intended to render parsing tractable.

Just few weeks before the completion of this work a remarkable research work inspired in the TAG formalism was presented (Carreras et al., 2008). The main advantage of this method is its the ability to use features from dependency trigrams.

Out of domain parsing, multi-lingual parsing and the adequacy of a given parsing strategy to a language are not well understood topics and presented little or non-significant progress in the last years.

4.2 Semantic role labeling

The CoNLL-2005 and CoNLL-2004 shared tasks were devoted to semantic role labeling (SRL). The shared tasks evaluated SRL under the framework of all predicates identified and the syntactic constituent parse trees as a basis. One of the interesting points of the CoNLL-2005 shared task is the addition of chunk parse output to the syntactic available information.

Team	Precision (%)	Recall (%)	F_1
punyakanok	81.18	74.92	77.92
pradhan	81.87	73.21	77.30
haghighi	78.34	75.78	77.04
màrquez	78.44	74.83	76.59
surdeanu	79.35	71.17	75.04

Table 4.3: Best CoNLL-2005 systems

Table 4.3[2] shows the top 5 systems on the CoNLL-2005 shared task (19 submitted results).

Team	ML	synt	pre	arch	glob	post	comb
punyakanok	SNoW	n-best	prun	i+c	yes	no	yes
haghighi	ME	n-best	?	i+c	yes	no	yes
màrquez	AB	2 parses	no	BIO	no	no	yes
pradhan	SVM	3 parses	?	BIO	no	no	yes
surdeanu	AB	1	prun	c	no	yes	no

Table 4.4: Summary of top performing CoNLL-2005 shared task systems

Table 4.4[3][4] contains a summary of the architecture of participating teams. The column *ML* contains the machine learning methods. The *synt* column indicates if multiple syntactic trees were used to perform SRL. Next column indicates if prepropcessing was performed. *Arch* column contains if the architecture performs identification and classification, only classification or it is a BIO tagging. *Glob* column notes if global information is used through the annotation process. *Postprocessing* contains if the system applied a simple postprocessing, usually a set of rules. The final column *comb* indicates if system combination was performed.

Maximum entropy models(8 of 19 teams) and SVM(6 of 19) were the most used classifiers in the CoNLL-2005 (Carreras and Màrquez, 2005). We can observe in SRL the use of much more complex machine learning methods (e.g., SVM, AdaBoost) with respect to the syntactic dependency parsing. The reason is that SRL is considered a classification task and does not suffer from the overhead caused by inference methods in syntactic parsing. Two

[2]See `http://www.lsi.upc.edu/~srlconll/st05/st05.html` for detailed scores

[3]Extracted from Carreras and Màrquez (2005)

[4]ME: maximum entrophy, AB: AdaBoost, prun: pruning, i+c: identification and classification

main approaches are applied to SRL, *BIO tagging* and *i+c* classification, see section 3.3 for further details.

The introduction of the ILP postprocessing overcome other types of simple postprocessing. System combination was a widely used technique to improve results. The intensive use of syntactic structures and the subsequent pipeline approaches pointed out the importance and the propagated errors due to syntax.

As previously noted SRL is widely approached by the 4 step process of filtering/pruning, local scoring, argument identification and classification and global scoring. Again we refer the reader to the *background* description on section 3.3.

The final global scoring step step is an interesting point to exploit. In this step we can consider global features and dependencies and constraints between the classified arguments. A classifier or reranking method can be trained to improve the final performance by combining a set of previously generated candidates. The global step can capture interactions between the set of arguments for a predicate or even between arguments from different predicates. The possibilities of this phase are limited by the previous candidate generation.

The filtering or pruning task is widely performed as a simple set of rules that discards most of the improbable candidates to argument. ILP postprocessing is also an easy to implement technique that a greater number of SRL system are including.

An interesting alternative approach is the CRF on trees applied by Cohn and Blunsom (2005), in this case the syntactic structure of the tree is used as the CRF graphical model. One of the main advantages of this approach is the exact computation of the optimal output, in this case the optimal semantic annotation. Note also that the training algorithm for this CRF can only be approximate. Unfortunately the training costs of this model are nearly prohibitive and it had little impact in the SRL community.

PropBank was extensively exploited in SRL research. Regarding nominal SRL there is less published work, one of the reasons is the relatively new release of NomBank. A point take into account is that this corpus are tagged with neutral labels that are not completely consistent across different predicates.

The learning is usually performed using the set of features defined in Xue and Palmer (2004) and Surdeanu (2007).

SRL systems strongly relies on the input syntax. This pipeline approach accuses the errors generated by the previous syntactic parser. This problem is alleviated by using a combination of different syntactic-parser inputs.

A joint system could overcome these limitations allowing to interact the syntactic and semantic layers.

SRL is still not regarded as a structured prediction problem, with the notable exception of Cohn and Blunsom (2005). Out of domain adaption is

again a far from being solved task.

4.3 Joint syntactic and semantic dependency parsing

The only known open domain work in joint parsing is by Musillo and Merlo (2006).

In this case, the prediction and learning of syntactic and semantic annotations is done by considering them just as one single annotation. The system concatenates the syntactic and semantic labels in one label. (e.g., a constituent "SUBJ'´ that also is "A0" for some predicate will be labelled as "SBJA0") With this new set of tags a classifier is trained and then the sentences are annotated. A postprocess is required to recover the separate labels and link the semantic label to the corresponding predicate. The training of this model carries the serious problem of requiring examples to cover all label combinations. In addition a large number of classifier must be trained for each label combination.

The joint parsing of syntactic and semantic dependencies is at the date of this master's thesis start an undeveloped field. Plenty of research directions can be taken.

As a result of the CoNLL-2008 shared task few joint models (Surdeanu et al., 2008), including the developed during this work, presented novel approaches to the joint parsing of syntactic and semantic dependencies, see section 6.2.1.

Chapter 5

System design

Contents

5.1 Introduction

The main goal of this work is to build a joint learning architecture for syntactic and semantic parsing and to test whether the syntactic and semantic layers can benefit each other from the global training and inference.

All the components of our system are build from scratch. Due to strong time constraints, our design decisions are biased towards constructing a simple and feasible system.

As we seen in chapter 4, the semantic parsing is mainly a classification task and the syntactic dependency parsing is widely performed using two main approaches: shift-reduce parsing and Eisner-based parsing. We decided to extend one of this two approaches to jointly annotated syntactic and semantic dependencies. In any extension the cost of the base algorithm will probably be increased. Thus the flexibility to extend the models and the cost are the main criteria considered.

Shift-reduce parsers are asymptotically more efficient($O(n)$) but at each step they carry an expensive learning part. Usually these parsers are much more slower than Eisner $O(n^3)$ parsers. The Eisner bottom-up parsing is implemented by dynamic programming and it is a flexible framework to be extended.

Furthermore, systems based on Eisner algorithm (Carreras et al., 2006; Carreras, 2007) showed a competitive performance in the syntactic parsing of the English language in past CoNLL shared tasks (Buchholz et al., 2006; Nivre et al., 2007a). We believe that extending the Eisner algorithm to jointly parse syntactic and semantic dependencies it is a natural step to follow. The Eisner algorithm was previously successfully extended to consider higher order dependencies (McDonald and Pereira, 2006; Carreras, 2007).

Given these reasons we decided in favour of an Eisner-based system. As we are increasing the computational cost of the inference algorithm we require a critically efficient learning method, the perceptron and its variants are the choice for this task, see chapter 4 for further discussion.

Our proposal is to define a first order linear model to jointly parse syntactic and semantic dependencies, that relies on an on-line averaged structured perceptron for learning (Collins, 2002) and an extended Eisner algorithm (Eisner, 1996) for joint parsing inference.

5.2 Baseline

As we are building a system from scratch we need some reference systems and results to evaluate the performance of our system during the development process.

The last CoNLL-2006 and CoNLL-2007 (Buchholz et al., 2006; Nivre et al., 2007a) were devoted to syntactic dependency parsing. Unfortunately the

evaluation of this task was done on different corpus built different conversion algorithms.

For this current shared task organizers provided the output of *Maltparser* therefore we used this output as a reference for the syntactic parsing task.

The identification of verb predicates can be easily accomplished by a set of few rules. The task of predicate sense disambiguation can be adorded with the simple baseline of assigning the most frequent sense.

The previous CoNLL-2004 and CoNLL-2005 (Carreras and Màrquez, 2004; Carreras and Màrquez, 2005) made available the output of a baseline system consisting on few simple rules, as to tag the first noun phrase of the predicate as A0 and the next noun phrase as A1. These rules apply only to verb predicates and do not gives an enough competitive baseline. No comparable system for the semantic part can be found as our setting brings a new enriched dependency structure to perform SRL also it includes noun and verb predicates and a simultaneous identification of them. We will regard as reference the previous work on SRL system. We expect to achieve $\sim$80F_1 in A0 and A1 classification and to found poor performance for AM-LOC and AM-TMP as are often easily confused (Carreras and Màrquez, 2004).

5.3 Difficulties

The two tasks to be jointly performed are the syntactic and semantic parsing These tasks are related but not identical. Syntactic and semantic dependencies represents different relations. A semantic relation between two tokens does not imply a syntactic relation of the same two. Semantic dependencies can take place between words loosely related by the syntax. In addition a semantic dependency can occur between the same word in noun predicates (Meyers et al., 2004), see predicate *maker* of figure 5.1.

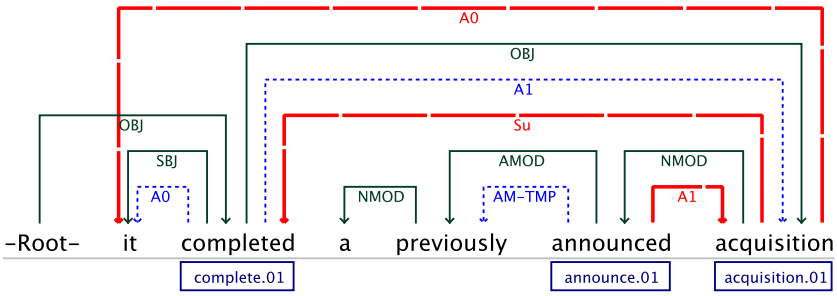

Figure 5.1: Self semantic dependencies.

Next figure 5.2 shows another sample sentence with annotated syntactic and semantic dependencies. Note that the semantic dependencies (dotted and dashed lines) do not always overlaps with syntactic dependencies (solid lines). Note also that some semantic dependencies are in fact *reversed* with

respect to the syntax, i.e., relating the same tokens but pointing towards different *heads* (e.g., the A1 relation of *announced-acquisition* is NMOD for *acquisition-announced*). We assume that all predicates are already identified (i.e, the joint model tasks are restricted to syntactic and semantic parsing).

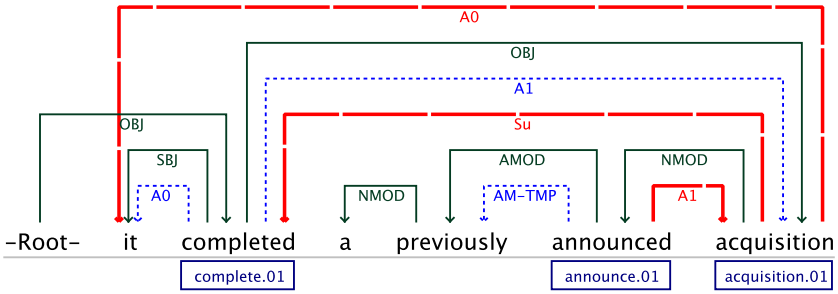

Figure 5.2: Syntactic and semantic dependencies.

The Eisner chosen parsing algorithm is a bottom-up parser that follows a tree structure. That structure is the syntactic structure. Note that the semantic structure is not complete (i.e., the semantic dependencies do not form a tree). The exactly overlapping dependencies are easy to be simultaneously predicted when we are parsing a syntactic dependency, see dotted lines of figure 5.3. In this case we can annotate a semantic dependency at the same time than we are scoring a syntactic dependency.

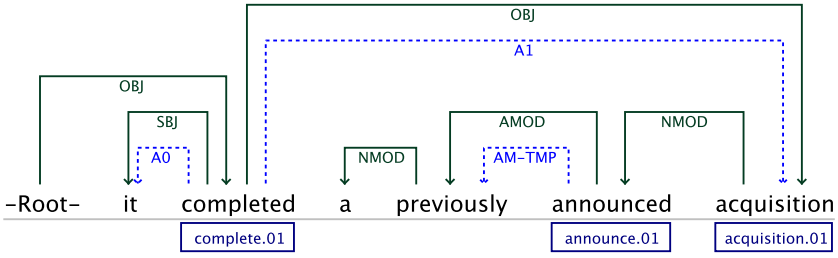

Figure 5.3: Semantic overlapping dependencies.

To deal with the non-overlapping semantic dependencies (bold dashed lines of figure 5.2) our solution is to assign a semantic label whenever we encounter just the token that receives the semantic dependency. For example, in figure 5.4 the two semantic A0 dependencies will be predicted at the encounter of the receiving token *it*. The semantic dependencies will be predicted at the same time than we are parsing the syntactic dependency labeled *SBJ*. The result is that we can consider that the syntactic dependency *SBJ* is extended with semantic labels (*SBJ,A0,_,A1*), one for each predicate, as seen in figure 5.5. In this case the first A0 corresponds to the first predicate, the "_" is a null label indicating that the second predicate has no semantic relation with the

token *it* and the last A0 tags the semantic dependency to the third predicate.

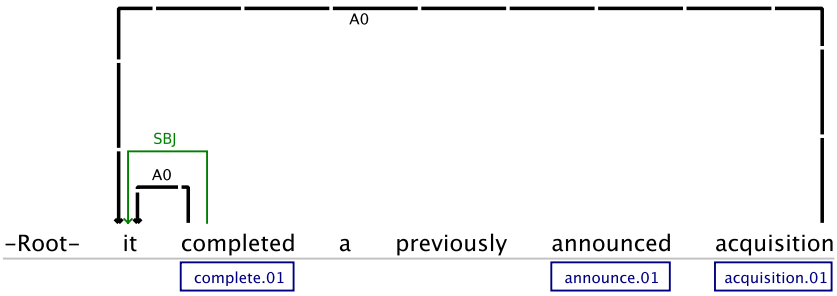

Figure 5.4: Semantic non-overlapping dependencies.

Thus, even if the head (i.e., predicate) of a semantic dependency is not the same token as the syntactic head we will be able to classify the semantic dependency. The result of applying this procedure to the whole sentence is show in figure 5.6.

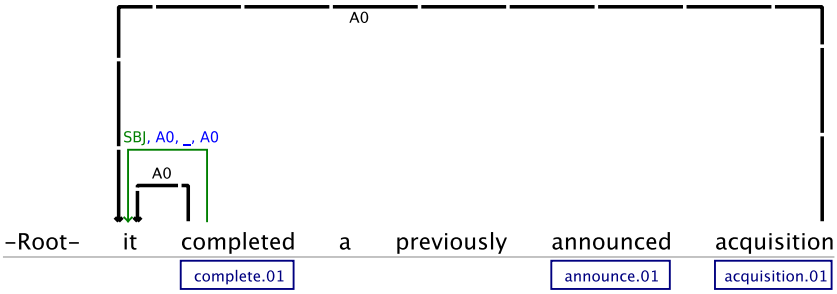

Figure 5.5: Extended syntactic and semantic dependencies.

Another difficulty is that state of the art SRL systems (Surdeanu et al., 2007) strongly rely on features extracted from the *complete* syntactic tree. The joint model grows syntactic and semantic structures at the same time, so features extracted from the syntactic tree (e.g., a syntactic path between a modifier and a distant predicate) are not available or expensive to compute within the joint Eisner parsing. Looking back to the figure 5.5 it could be possible that we are in fact assigning a semantic dependency between a token that is parsed and a token that it is not already parsed. Therefore the exact syntactic relation between these two token will remain unknown at that time.

We overcome this problem again with a very simple (though not elegant) solution, consisting of introducing a previous syntactic parsing step. In this previous parsing we will precompute just the minimum syntactic path features that will unavailable at joint parsing time. Note that most of the features will be extracted from the joint tree. Other features (e.g., paths) although fixed by the previous parse could be partially adapted at joint time. For example, a path between a modifier and a predicate could be formed by the path from the

modifier to its syntactic head know at joint time plus the precomputed path from the head to the distant predicate.

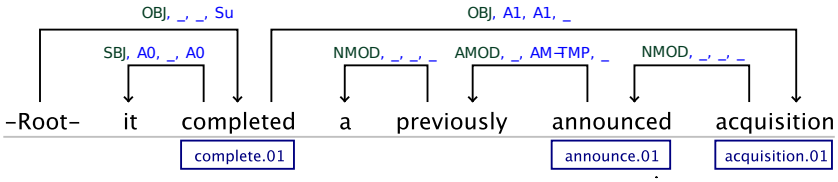

Figure 5.6: Syntactic and semantic dependencies.

The final syntactic and semantic trees will be completely and solely built at the joint parsing step. The formalization of this intuitive solutions is described in the following sections.

5.4 Architecture

The previous discussion on the difficulties of the join model concluded with the solution of a syntactic pre-parser to the unavailable features problem. Therefore in our architecture a *predicate identification* and *syntactic parse* phases must be included before to the *joint parsing* phase.

The predicate sense disambiguation is a also to be performed. We will implement this task on the latter postprocessing step. At this stage the joint parser had already assigned arguments for each identified predicate. This information will be useful to disambiguate the predicate sense.

This section outlines the main components of our system:

1. Preprocessing.
   In *preprocessing*, the training corpus is traversed and feature extraction performed. Main features are borrowed from pre-existing well-known systems (see next subsection).

2. Syntactic parsing.
   The initial *syntactic parsing* is based on an Eisner parser trained with perceptron and it is merely intended to allow the extraction of syntactic-based features for all the following phases (which share exactly the same feature set extracted from these parse trees).

3. Predicate identification.
   Predicate identification recognizes predicates by applying SVM classifiers[1] and a set of simple heuristic rules.

4. Joint syntactic-semantic parsing.
   The *joint syntactic-semantic parsing* phase is the core module of this

[1]We used SVM-light (see *www.joachims.org* for details).

work. It simultaneously derives the syntactic and semantic dependencies by using a first order Eisner model, extended with semantic labels and trained with an averaged perceptron.

5. Postprocessing.
   Finally, *postprocessing* selects the most frequent sense for each predicate and applies ILP to enforce domain constraints in the output.

5.5 Preprocessing

All features in our system are computed in the preprocessing phase. We use the features described in McDonald et al. (2005a) and Carreras et al. (2006) as input for the syntactic parsing phase. The joint syntactic-semantic parser uses all the previous features and also specific features for semantic parsing from Xue and Palmer (2004) and Surdeanu et al. (2007). The features originally designed for a constituent syntactic representation have been straightforwardly adapted to the dependency structure used in this shared task, by substituting any reference to a syntactic constituent by the head of that constituent. About 5M features were extracted from the training corpus. The number of features was reduced to ∼222K using a frequency threshold filter. A detailed description of the feature set can be found at the next section 5.10

5.6 Syntactic parsing

We use the Eisner algorithm combined with an on-line averaged Pereceptron. The Eisner algorithm has a $O(n^3)$ cost. In practice, there is also a relevant multiplying constant, i.e., the number of dependency labels. To improve the efficiency and performance, the labels that are considered for a candidate dependency are filtered according to the POS of the head and modifier words. The filters discard all labels not previously seen in the training corpus between words with the same POS. This strategy allowed us to significantly improve efficiency without any loss in accuracy, see the 5.11

5.6.1 Basic model

The basic model used for parsing which is also the starting point for the joint model outlined in 5.6.1. We now give a detailed description. Let L be the set of syntactic labels, $x = x_1, \ldots, x_n$ a sentence with n words, and $\mathcal{Y}(x)$ the set of all possible projective dependency trees for x.

A dependency tree $y \in \mathcal{Y}(x)$ is a labeled tree with arcs of the form $\langle h, m, l \rangle$ that is rooted on an artificial node, 0, added for this purpose. The head, h, and modifier, m, for a dependency index words in the sentence and can take values in $0 \leq h \leq n$ and $1 \leq m \leq n$. $l \in L$ is the label of the dependency.

The dependency parser (dp) finds the best scored tree for a given sentence x:

$$\text{dp}(x, \mathbf{w}) = \arg\max_{y \in \mathcal{Y}(x)} \text{score\_tree}(y, x, \mathbf{w}) \tag{5.1}$$

Using an arc-based first order factorization, the function score_tree$(y, x, \mathbf{w})$ is defined as the summation of scores of the dependencies in y:

$$\text{score\_tree}(y, x, \mathbf{w}) = \sum_{\langle h,m,l \rangle \in y} \text{score}(\langle h, m, l \rangle, x, \mathbf{w}), \tag{5.2}$$

where $\mathbf{w}$ is the weight vector of the parser, computed using an on-line perceptron. The weight vector $\mathbf{w}$ can be seen as a concatenation of $|L|$ weight vectors of d components, one for each of the labels: $\mathbf{w} = (\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(l)}, \ldots, \mathbf{w}^{(|L|)})$. A function ϕ is assumed to extract features from a dependency $\langle h, m, l \rangle$ and from the whole sentence x. This function represents the extracted features as a d-dimensional vector.

With all these elements, the score of a dependency $\langle h, m, l \rangle$ is computed as a linear function:

$$\text{score}(\langle h, m, l \rangle, x, \mathbf{w}) = \phi(\langle h, m, l \rangle, x) \cdot \mathbf{w}^{(l)} \tag{5.3}$$

5.7 Predicate identification

We identified as verb predicates all verbs excluding the auxiliaries and the verb *to be*. These simple rules based on the POS and lemma of the tokens are enough to correctly identify almost all verb predicates.

With regard to noun predicates, we directly identified as predicates the lemmas which appeared always as predicates with a minimum frequency of 5 times in the training corpus. The remaining noun predicates were identified by a 2-degree polynomial SVM. This classifier was trained with the same features used in subsequent phases, but excluding those requiring identified predicates.

5.8 Joint parsing

The joint parser extends the Eisner algorithm to jointly assign syntactic and semantic labels. An important concern in this stage is the feasibility of the system.

Again, the joint parser as the syntactic parser relies on filters to improve its performance. In addition to the filter by the POS-POS combination, a new filter built using information from the frames files is used. For a discussion about these filters see section 5.11.

We train different classifier for noun and verb arguments. Usually features to train verb argument classifiers are combination of other features intended

to reproduce the benefits of kernelization (Xue and Palmer, 2004). In the same direction we considered the nominal and verbal predicate identification tasks not enough similar and we trained separate classifiers.

The joint parser outputs the best joint tree but also the second best is generated. This data is intended to be exploited and combined at the post-processing phase.

5.8.1 Joint Model

The previously described basic parsing model will be extended to jointly assign semantic dependency labels. Let S be the set of semantic labels. Note that at this point, a sentence x has a set of q words already identified as predicates. We will refer to them as $p_1, \ldots, p_q$, where $p_i \in \{1, \ldots, n\}$. We consider that each dependency has a set of semantic tags $l_{sem\,p_1}, \ldots, l_{sem\,p_q}$ one for each sentence predicate p_i. Also, we consider an extra *no-argument* label in the set of semantic labels S. Thus, an extended dependency d_s is defined as:

$$d_s = \left\langle h, m, l_{syn}, l_{sem\,p_1}, \ldots, l_{sem\,p_q} \right\rangle , \tag{5.4}$$

where l_{syn} denotes the syntactic label for the dependency.

Again, the best parse tree is that maximizing the score of a first order factorization:

$$\mathrm{dp}(x, \mathbf{w}, y') = \arg\max_{y \in \mathcal{Y}(x)} \mathrm{score\_tree}(y, x, \mathbf{w}, y') \tag{5.5}$$

$$\begin{aligned} &\mathrm{score\_tree}(y, x, \mathbf{w}, y') = \\ &= \sum_{\langle h,m,\mathbf{l}\rangle \in y} \mathrm{score}(\langle h, m, \mathbf{l}\rangle , x, \mathbf{w}, y') , \end{aligned} \tag{5.6}$$

where the dependency label is now extended to $\mathbf{l} = \langle l_{syn}, l_{sem\,p_1}, \ldots, l_{sem\,p_q}\rangle$ and y' denotes the precomputed syntax tree. The score of a syntactic-semantic dependency is:

$$\mathrm{score}\left(\langle h, m, \mathbf{l}\rangle , x, \mathbf{w}, y'\right) = \tag{5.7}$$

$$\mathrm{syntactic\_score}\left(h, m, l_{syn}, x, \mathbf{w}\right) + \tag{5.8}$$

$$\mathrm{sem\_score}\left(h, m, l_{sem\,p_1}, \ldots, l_{sem\,p_q}, x, \mathbf{w}, y'\right) \tag{5.9}$$

The syntactic score is computed as described in the basic model. Finally, the semantic scoring function computes the semantic score as the sum of the semantic scores for each predicate semantic label:

$$\begin{aligned} &\mathrm{sem\_score}\left(h, m, l_{sem\,p_1}, \ldots, l_{sem\,p_q}, x, \mathbf{w}, y'\right) = \\ &\sum_{l_{sem\,p_i}} \frac{\phi_{sem}\left(\left\langle h, m, l_{sem\,p_i}\right\rangle , x, p_i, y'\right) \cdot \mathbf{w}^{(l_{sem\,p_i})}}{q} \end{aligned} \tag{5.10}$$

Note that each sentence x has a different number of predicates q. To avoid an excessive weight of the semantic component in the global score and a bias towards sentences with many predicates, the score is normalized by the number of predicates in the sentence. If a sentence has no predicates the joint model will behave identically as the syntactic model.

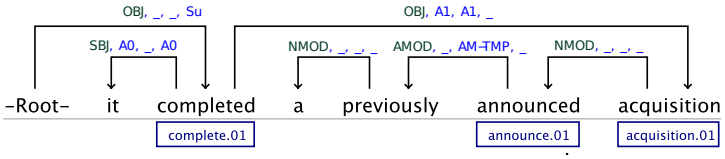

Figure 5.7: Syntactic and semantic dependencies.

Figure 5.7 recalls the previous example of a sentence fragment with syntactic and semantic dependencies. The three predicates of the sentence are already identified: $\{p_1 = \text{completed}, p_2 = \text{announced}, p_3 = \text{acquisition}\}$. All dependencies are of the form $d = \langle h, m, l_{syn}, l_{sem\,p_1}, l_{sem\,p_2}, l_{sem\,p_3} \rangle$. The new semantic labels express semantic relations between a modifier and a predicate that can be anywhere in the sentence. In this example, the correct output for the dependency *previously-announced* is $h = \text{announced}$, $m = \text{previously}$, $l_{syn} = \text{AMOD}$, $l_{sem\,p_1} = \text{null}$, $l_{sem\,p_2} = \text{AM-TMP}$, $l_{sem\,p_3} = \text{null}$.

The above described factorization allows the parser to simultaneously assign syntactic and semantic labels and also to maximize a joint syntactic-semantic score of the tree. Note that the semantic scoring function ϕ_{sem} extracts features from the modifier, the head and the predicate of the parsed dependencies. The proposed model allows to capture interactions between syntax and semantics not only because the syntactic and semantic scores are combined but also because the semantic scoring function relies on features extracted from the head-modifier-predicate relations. Thus, the semantic scoring function depends on the syntactic dependency being built, and, in reverse, the semantic score can modify the dependency chosen.

Regarding implementation issues, note that we compute $|L|+|S|\cdot q$ scores to assign $q+1$ labels to a given dependency. The scores are computed independently for each label. Otherwise, interactions among these labels, would raise the number of possible combined labels to an exponential number, $|L| \cdot |S|^q$, making the exhaustive evaluation infeasible in practice.

5.9 Postprocessing

This phase has the purpose of disambiguate the predicate sense. We postponed this task until postprocess to be able to extract information from the assigned arguments by the joint parser. Two different strategies were implemented.

- Most frequent sense.
  This is a simple postprocess that assigns the most frequent sense to each identified predicate. The frequencies were extracted from the training corpus. corpus.

- Integer linear programming.
  ILP (see section 3.5.5) can be applied to perform inference and enforce domain constraints on the output.

The combination of different system outputs and constraint enforcement was successfully previously done using ILP (Punyakanok et al., 2004; Surdeanu et al., 2007).

We will combine the best and second best output of our system by ILP. Several domain constraint can be enforce, although Surdeanu et al. (2007) showed that only two constraints are really useful:

1. There are no repeating core arguments

2. There are no embedding arguments

Note that the overlapping is not possible in the same tree structure. Only and embedding could occur.

The ILP variables represent candidate arguments weighted by the score of the parser output. Therefore, the objective function F maximizes the sum of arguments weights under some domain constraints.

Let a_k be a binary variable (i.e., it can take values 0 or 1). We have one a_k for each one of the arguments generated by the parser, from the best and second best outputs. We will consider that the following functions are available (i.e., the binary variable a_k contains more information that its value).

- isCore(a_k) wether or not the argument associated to a_k is a core argument.

- modifier(a_k) the token labeled with a_k.

- score(a_k) is the score output of the parser for the argument represented by a_k.

We want to compute:

$$\arg\max_{\{a_k\}} F(\{a_k\}) = \sum \text{score}(a_k) a_k \tag{5.11}$$

We add the following constraints:

1. Only one core argument,

$$a_i + a_j \leq 1 \quad \textbf{if } \text{isCore}(a_i) \text{ and } \text{isCore}(a_j) \text{ and } \text{argument}(a_i) = \text{argument}(a_j) \tag{5.12}$$

2. Non embedding,

$$a_i + a_j \leq 1 \quad \textbf{if } \text{embedding}\left(\text{modifier}(a_i), \text{modifier}(a_j)\right) \tag{5.13}$$

Note that best output scores are always higher than the second best score. The second best output will only be selected in cases of some constraint is violated. For example if there are two A0 arguments for a predictate we will discard one of them and select instead the second best output.

An important drawback when we use the ILP approach is its tendency to include as much arguments as possible. As we are maximizing a function, all non-zero scored arguments will be added to the final result unless they violate some constraint. To overcome this undesirable consequence Koomen et al. (2005) introduced a penalty factor to the objective function. The parameter simply sets a threshold for the coefficients (e.g., probabilities, scores or number predictions from combined system) of the argument variables. Note that our system output will not significantly suffer from this problem because when the first (best) output is a null argument for a given token, the second best output will not assign any argument for the same token, thus the second output does not adds arguments for different tokens. Also the tagging of the tree structure avoids assigning overlapping arguments, only embedding arguments could be assigned. Finally, the Kommen et al. proposal was not implemented.

In addition to the enforcing of the domain constraints the information contained in the frames of the noun and verb predicates is exploited to assist the predicate sense disambiguation. The frames contains the allowed arguments for each predicate sense.

We jointly compute the best predicate sense at this phase. For this reason, we add new variables to the objective function. The new variables represents each predicate sense weighted by its training corpus frequency. The maximization process will select the most frequent sense if no constraints were violated. The following new constraints were added:

1. We can only select one sense per predicate.

2. If a core argument is not valid for a sense, the sense and the argument cannot be simultaneously selected.

The second constraint contains the information extracted from the training frames files provided with the corpus. The predicates identified that are not present in the training corpus are assigned to a default sense "01". The output of the ILP postprocess selects the arguments and the sense for each predicate.

We will define a new set of variables. p_i is a binary variable representing the selected sense for a given predicate p. i can take values from 1 to the number of senses.

freq(p_i) is the frequency of the sense represented by p_i, the frequency is computed by counting from the training corpus.

$$\arg\max_{\{a_k\},\{p_i\}} F(\{a_k\},\{p_i\}) = \left[\sum \text{score}(a_k)a_k\right] + \beta\left[\sum \text{freq}(p_i)p_i\right] \quad (5.14)$$

The parameter β sets the trade-off between selecting the most frequent sense or selecting all arguments. A high value of β will force to select the most frequent sense and delete all forbidden arguments for that sense if any. A low value of β will force to select a sense that accepts all predicted arguments.

The constraints are translated to

1. Only one sense per predicate,

$$p_1 + \ldots + p_z \leq 1 \quad (5.15)$$

   where z is the last sense

2. If a core argument is not valid for a sense, the sense and the argument cannot be simultaneously selected,

$$a_i + p_j \leq 1 \quad \textbf{if} \quad \text{isInvalid}\left(\text{argument}(a_i), \text{roleset}(p_j)\right) \quad (5.16)$$

As we seen in the section A.2 some predicate lemmas are incorrectly predicted. Therefore we could discard correct arguments for a predicate because we are regarding an incorrect frame file. To overcome this problem we collected a list of all the lemmas that were incorrectly assigned for each true predicate lemma. When we found one of these lemmas the constraints will be softened to allow all the arguments allowed by each one of the similar lemmas of the list.

See the 3.5.5 for further about the ILP framework. There are other possibilities to combine systems output. Machine learning can be used (Surdeanu et al., 2007) to exploit a richer set of features, although in this case it is harder to enforce domain constrains.

5.10 Features

In this section we describe the features used by our system. McDonald et al. (2005a) and Carreras et al. (2006) defined a rich set of syntactic features widely used for syntactic dependency parsing. Xue and Palmer (2004) and Surdeanu et al. (2007) published widely used features for SRL, but applied to

a constituent syntax tree. The features that refers constituents were adapted to consider the head of the constituent.

Most of the features are detailed in these works but minor implementation details are often left. In the following section we will describe in detail each implemented feature.

All features are prefixed by a string indicating their feature type to prevent the generation of identical features from different groups of features. The operator "·" represents a concatenation of the strings representing the features.

Recall that we assume that we are always extracting features from a dependency d:

$$d = \langle h, m, l \rangle \tag{5.17}$$

with head h, modifier m and syntactic label l. The feature extraction function $\phi(d, x) = v$ can extract features not just from the dependency but also from the rest of the sentence. The feature representation is the binary vector $v \in \{0, 1\}^n$. In the case of the dynamic features, the partially computed output $\hat{y}$ of the parser is also visible for $\phi(d, x, \hat{y}) = v$, thus some features are extracted on-line while parsing the sentence.

The semantic feature extraction function ϕ_{sem} for an extended dependency d_s:

$$d_s = \langle \text{head}, \text{modifier}, l_{syn}, l_{sem\,p_1}, \ldots, l_{sem\,p_q} \rangle \tag{5.18}$$

Will take the form:

$$\phi_{sem}\left(\left\langle h, m, l_{sem\,p_i} \right\rangle, x, p_i, y'\right) \tag{5.19}$$

where p_i is the predicate and y' the pre-parsed syntactic tree. Note that we extract features for each possible combination of head-modifier-predicate.

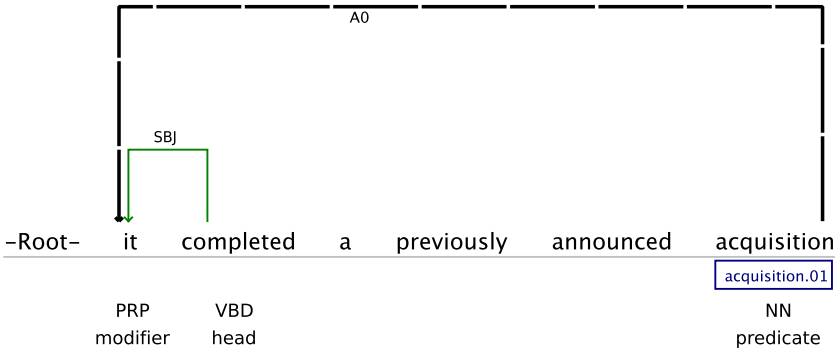

Figure 5.8: A sample feature extraction

Figure 5.8 shows a sample sentence. In this case, if we are interested in the features of the semantic A0 dependency *it-acquisition* we will extract features

from the the modifier and predicate but also from the head. For instance, some extracted features will be PRP·VBD, PRP·NN and VBD·NN.

We assume that we can retrieve from each token its POS, lemma, splitForm and other fields by functions as **splitLemma**, **splitForm**, etc. The coarse part of speech was computed extracting the first two characters that identifies a fine POS. We consider fine POS the set of POS defined for the penn treebank. See the section 2.1.3 for a description of the data fields available.

5.10.1 Syntactic Features

McDonald et al. (2005a; 2005b) extracted features from corpus and filtered the number of features by a frequency threshold. and proposed back-off of features consisting in taking the first n-grams of the combination features and the first characters of the tokens to reduce the low frequency of some features. We also substituted the token back-off by the the token lemma. Also McDonald et al. (2006) used morphological features that are not directly available in our training corpus. Thus this features are discarded.

We designed few new features described in the subsection *New Features and in the* Grandson Dynamic Features.

Token Features

In this case we assume that the token t represents any input token. This feature extraction function will be called passing as t the *head* and the *modifier*. Also this feature extraction function can be called for other tokens if required by any other of the feature extraction functions.

The features extracted are:

- **splitLemma**(t)
- **splitForm**(t)
- **coarsePOS**(t)
- **finePOS**(t)
- **splitForm**(t) · **finePOS**(t)
- **splitLemma**(t) · **finePOS**(t)

Context Features

As context features we call the previous *Token Features* extraction function for the tokens $-2, -1, +1, +2$ with respect to the given token of the sentence.

In addition we consider Context Features:

- **finePOS**(t) · **finePOS**$(t-1)$

- **finePOS**(t) · **finePOS**($t-1$) · **finePOS**($t-2$)
- **finePOS**(t) · **finePOS**($t+1$)
- **finePOS**(t) · **finePOS**($t+1$) · **finePOS**($t+2$)

When the tokens at positions $-2, -1, +1, +2$ are not available (e.g. at the beginning or end of the sentence) a special feature was extracted indicating this condition.

Dependency Features

- **splitForm**(head) · **finePOS**(head) · **splitForm**(mod) · **finePOS**(mod)
- **finePOS**(head) · **splitForm**(mod) · **finePOS**(mod)
- **splitForm**(head) · **splitForm**(mod) · **finePOS**(mod)
- **splitForm**(head) · **finePOS**(head) · **splitForm**(mod)
- **splitForm**(head) · **finePOS**(head) · **finePOS**(mod)
- **splitForm**(head) · **splitForm**(mod)
- **finePOS**(head) · **finePOS**(mod)

The same features were extracted but substituting the *form* where present by the *lemma* of the token.

- **splitLemma**(head) · **finePOS**(head) · **splitLemma**(mod) · **finePOS**(mod)
- **finePOS**(head) · **splitLemma**(mod) · **finePOS**(mod)
- **splitLemma**(head) · **splitLemma**(mod) · **finePOS**(mod)
- **splitLemma**(head) · **finePOS**(head) · **splitLemma**(mod)
- **splitLemma**(head) · **finePOS**(head) · **finePOS**(mod)
- **splitLemma**(head) · **splitLemma**(mod)

Dependency Context Features

- **finePOS**(head) · **finePOS**(head+1) · **finePOS**(mod−1) · **finePOS**(mod)
- **finePOS**(head) · **finePOS**(head+1) · **finePOS**(mod) · **finePOS**(mod+1)
- **finePOS**(head−1) · **finePOS**(head) · **finePOS**(mod) · **finePOS**(mod+1)
- **finePOS**(head) · **finePOS**(head−1) · **finePOS**(mod−1) · **finePOS**(mod)

Distance Features

- **finePOS**(head) · **distance**(head, mod) · **finePOS**(mod)
- **finePOS**(head) · **finePOS**(k) · **finePOS**(mod) for each token k placed between the head and the modifier.
- **distance**(head, mod) as the number of token between the head and the modifier in the sentence word order.
- **numVerbs**(head, mod) the number of tokens with a coarse POS indicating a verb between head and modifier.
- **numCoords**(head, mod) the number of tokens with a coarse POS indicating that is a coordination word (e.g., *and*, *or*) between head and modifier.
- **numPunc**(head, mod) the number of tokens with a coarse POS indicating that is a punctuation word (e.g., ";") between head and modifier.
- **whether** $head < mod$ **or not** with respect to the sentence word order.

Note that we are always working with binary features. When we deal with numeric attributes (e.g., distances) we will have to add a new feature for each possible value. In addition large values Thus the distance number is binned into a set of intervals.

New Features

We added the following small feature variations, not present in the previous cited works:

- **sentence size** the binned size of the whole sentence.
- **relative position** the binned relative position of head and modifier inside the sentence.

We pretended to include this information as it is possible that the previous measures that computes the distance between head and modifier will benefit from taking into account the whole sentence size.

Dynamic Features

This features are newly introduced in Carreras et al. (2006) intended to overcome the limitations of the First Order Projective parsing model, see 3.2.2.

This features can exploit the span or subtree being build. The features are based on the previous syntactic labels assigned by the bottom-up parser.

- **finePOS**(head) · **finePOS**(mod) · **synLabel**(k) we add a feature for each possible value of k, where k are the sons of the *head*.
- **finePOS**(head) · **finePOS**(mod) · **concatenatedSynLabel**(**sons**(head)) we add a feature for each son k of *head*. Each features has for each son of *head* the syntactic label of the son and also concatenated all the previous syntactic labels.
- **finePOS**(head) · **finePOS**(mod) · **numberOfSons**(head)) the *numberOfSons* function can take values only from {no sons, 1 to 4 sons, ¿4 sons }

Grandson Dynamic Features

Following the same concept of the *Dynamic Features* we defined a new set of features capturing grandson attributes, see 3.2.2.

- **finePOS**(head) · **finePOS**(mod) · **synLabel**(k) we add a feature for each possible value of k, where k are all the sons of the sons of *head* (i.e., the grandsons).
- **finePOS**(head) · **finePOS**(mod) · **concatenatedSynLabel** (grandsons(head)) we add a feature for each grandson k of *head*. Each features has for each grandson of *head* its syntactic label and also concatenated all the previous syntactic labels for the rest of the grandsons.
- **numberOfGrandSons**(head)) the *numberOfGrandSons* function can only take values from the set {no grandsons, 1 to 4 grandsons, >4 grandsons }
- **numberOfSons**(mod) the *numberOfSons*(mod) function extracts the number of sons of the current modifier. These are the subset of head grandson corresponding to the modifier.

5.10.2 Semantic Features

We tried to adapt the features originally designed for a constituent dependency representation. For example we substituted the first and last word of a constituent by the first and last child.

We also include 2,3,4,5-grams of the features that contains sequences (e.g., the sequence of POS of sons of a given modifier).

We refer to sons of a given token regarding the syntactic structure.

Animacy Features (Joanis, 2002) are features used for verb sense disambiguation. We included them to allow the use of the same set of features for a predicate disambiguation classifier.

In addition a set of words that are tagged as temporal modifiers (i.e, tomorrow, later) in the training corpus is extracted and also a set of bag of words. The precomputed set of bag words limits the maximum number of words that will be extracted as features as this feature can generate a huge set of feature vector entries.

Constituent Features

- **constituent Type and head** in our case the constituent features are adapted to consider the head of the modifier.
  - **coarsePOS**(modifier)
  - **synLabel**(modifier)
  - **finePOS**(modifier)
  - **splitForm**(modifier)
  - **splitLemma**(modifier)
  - **coarsePOS**(modifier) · **coarsePOS**(head)
  - **finePOS**(modifier) · **finePOS**(head)
- **firstAndLastChild**
  - **finePOS**(**firstSon**(modifier)) *firstSon(*mod*)* is the first son of the *modifier* with respect to the word sentence order.
  - **finePOS**(**firstSon**(modifier)) · **finePOS**(*lastSon*(modifier)) *lastSon(*mod*)* is the last son of the *modifier* with respect to the word sentence order.
  - **finePOS**(**firstSon**(modifier)) · **before**(**firstSon**(modifier), modifier) *before*(*firstSon*(modifier), modifier) indicates if the position that occupies the first son of the *modifier* is before the *modifier* itself.
  - **numSons = 0** this is a binary feature indicating that the modifier has no sons.
  - **numSons = 1**
- **posSequenceChild**
  - **posSequence**(**sons**(modifier)) the sequence of POS of each son of *modifier*.
  - **posSequenceAndNum**(**sons**(modifier)) the sequence of POS of each son of *modifier* and the number of sons (i.e., grandsons of the modifier) for each one.
- **bag of words** The bag of words of each sentence constrained to the words that are adjectives, nouns or verbs.

- **topSequenceChild** The POS of the direct sons of the modifier.
- **governingCategory** the phrase of the modifier, in our case adapted to the POS of the head of the modifier.
  - **finePOS**(head(modifier))
  - **coarsePOS**(head(mod))
- **named entity** Our named entity recognizer is very simple, it considers as named entities all words with the first letter in upper case.
- **temporal keyword** We add this feature if the *modifier* appears in a list of temporal keywords, see 5.5.
- **animacy Features** whether or not the modifier is a personal pronoun.

Context Constituent Features

As *Context Constituent Features* we call the previous *Constituent Features* extraction function for the tokens $-2, -1, +1, +2$ with respect to the *modifier* token.

In addition we compute the

- **coarsePOS**(modifier) · **coarsePOS**(modifier $-$ 1)
- **coarsePOS**(modifier) · **coarsePOS**(modifier $-$ 2)
- **coarsePOS**(modifier) · **coarsePOS**(modifier $+$ 1)
- **coarsePOS**(modifier) · **coarsePOS**(modifier $+$ 2)
- **whether or not** $modifier$ **is the first/last word**

Predicate Features

- **predicateSyntacticFeatures**
  - **coarsePOS**(predicate)
  - **finePOS**(predicate)
  - **coarsePOS**(predicate) · **coarsePOS**(mod)
  - **finePOS**(predicate) · **finePOS**(mod)
  - **splitForm**(predicate)
  - **splitLemma**(predicate)
- **cardinalityFeatures** This features tries to capture if the predicate is a multi-word verb.

  - **isVerb**(predicate) **and isVerb**(predicate − 1) e.g., stopped/VBD trading/VBG
  - **isVerb**(predicate) **and isVerb**(predicate − 2) e.g., has/VBZ already/RB begun/VBN
  - **isVerb**(predicate) **and isModal**(predicate−1) e.g., could/MD watch/VB
  - **isVerb**(predicate) **and isModal**(predicate−2) e.g., could/MD not/RB handle/VB
  - **coarsePOS**(head(predicate))

- **voice** Features indicating if the predicate is a passive voice verbal form.
  - **isVerb**(predicate) **and isVerbToBe**(predicate − 1)
  - **isVerb**(predicate) **and isAdverb**(predicate−1) **and isVerbToBe**(predicate−2)
  - **isVerb**(predicate) **and isVerbToBe**(predicate−1) · **isBefore**(mod, predicate) This is a very useful feature for correctly tagging A0 and A1 arguments. For predicates in active voice A0 almost always precedes the predicate and A1 is the fist argument following the verb. For passive voice predicates the situation is reversed.
  - **isVerb**(predicate) **and isAdverb**(predicate − 1) **and isVerbToBe**(predicate − 2) · **isBefore**(mod, predicate)

- **subcategorizationRule**
  - **are all the predicate sons at his left/right**
  - **synLabels**(sons(predicate))
  - **synLabels**(sons(head(modifier))) This is a new feature to allow to capture the sense of the subcategorization rule feature for arguments that have not the predicate as their direct head.

- **extraSemantic** Initially were not considered the concatenation features where added
  - **splitLemma**(predicate)
  - **splitLemma**(head)
  - **distance**(predicate, modifier)
  - **splitLemma**(predicate) · **distance**(predicate, modifier)
  - **splitLemma**(predicate) · **splitLemma**(modifier)
  - **isPredicate**(modifier)
  - **isPredicate**(head)
  - **is** predicate = modifier**?**
  - **is** predicate = head**?**

Predicate-Constituent Features

- **relativePosition**
  - **position**(predicate, modifier) whether the *predicate* is before, same word or after the *modifier*
  - **embeddingLevel**(predicate, modifier) captures if the predicate is a descendant, ancestor or has another syntactic relation.
- **constituentPath**
  - syntacticPath(modifier, predicate) the concatenated syntactic labels from *modifier* to *predicate*
  - syntacticDistance(modifier, predicate) the distance from *modifier* to *predicate* with respect to the syntactic structure
  - syntacticPathArrowed(modifier, predicate) the concatenated syntactic labels from *modifier* to *predicate*, indicating when we traversing the nodes if we are going by an upward or downward syntactic edge (i.e., following the parent or the son direction).
- **syntacticFrame**
  - **synFrame**(predicate, modifier) the coarse POS of each of *predicate* sons tagging the position occupied by the *modifier* or the corresponding ascendant if the *modifier* is inside the subtree spanned by *predicate*.
  - **synFrame**(head(modifier), modifier) tagging the position occupied by the *modifier* and the *predicate* or the corresponding ascendant if the *predicate* is inside the subtree spanned by the *head*.

5.10.3 Dynamic semantic features

This features could capture analogous features as the *Dynamic Syntactic Features*. As it is reasonable to think that the previous sons of a given head will contribute with significant information, the previously assigned arguments for a given predicate could also do. As we are building the joint tree based on the syntactic structure it is possible that some previously assigned arguments are not visible (i.e., outside the actual sentence span), see 7.1 for further details.

The use *Dynamic Syntactic Features* produced discouraging results, see 6.1. Thus finally we decided to do not implement the *Dynamic Semantic Features*.

5.10.4 Semantic features for predicates

All the previous semantic features, except the ones that requires or assumes identified predicates are used by the predicate identification classifier.

5.10.5 Feature selection

The whole training corpus is processed and each one of the previously described features is extracted. As the number of combinations lemma, form and POS could be huge we extracted the first 5M features and then computed their frequency.

From this 5M features we discarded using a threshold filter all the features bellow a frequency of 5000 for the syntactic features and 25000 for the semantic features. We lowered this threshold by a factor of 1/3 for certain semantic features, the ones which are concatenated with the lemma of the predicate and the *relative position* syntactic features.

5.11 Efficiency

The efficiency is a critical point in our system design. Some techniques could make the difference between a feasible and unfeasible systems. For example a naive implementation of the dot product without taking into account that we are working with sparse vectors could multiply the time consumed by score function by several orders of magnitude.

To achieve we written the software in the *C++* language. Important efficiency included the use of filters and the preallocation of memory and reuse of objects, and to work with structures and algorithms adapted to sparse vector. We will review only the filters used.

5.11.1 Filters

The Eisner algorithm has an $O(n^3)$ cost. But there is a hidden constant, the number of labels to assign multiplies the asymptotic bound. Thus the number of syntactic and semantic labels plays a critical role in the algorithm performance. Various filters were implemented to reduce the number of labels to be scored.

For a review of the data files managed by our system and other implementation issues related to efficiency, please see section B.

Filters by frequency

Labels that rarely appears in the training corpus, say labels with a frequency ¡ 5 will be near impossible to be learned in out setting, see section A.1. Probably will be never predicted at test time. We discarded the syntactic and semantic labels with very low frequency. These filters shown an improvement in the parsing efficiency. Also the length of the weight vector was diminished, as there are no weights for these labels. This improved the performance of the vector averaging computation due to the shorter vector length.

Filters by POS-POS

A powerful method for filtering is to restrict the labels to assign between two token by their Part of Speech (POS). Conceptually if we are parsing a possible dependency between to tokens t_1 and t_2, the syntactic POS-POS filter is a map:

$$\text{syn\_filter}(\text{POS}(t_1), \text{POS}(t_2)) = l_1, \ldots, l_n \tag{5.20}$$

where $l_1, \ldots, l_n$ are the admissible labels between t_1 and t_2.

The syntactic filter by POS-POS reduced the computation time by a factor of 5 without any loss in accuracy, see the 6.1.2 section.

The filter is intended to be used at each $O(n^2)$ Eisner loop. Thus accessing must be very fast, otherwise the advantages of the filter use will be surpassed by the filter cost.

Each POS of each token is converted to an integer in the preprocessing phase. The POS of the head and the POS of the modifier are directly translated to a vector access of $O(1)$ cost by combining the two integers. The list of labels returned by the filter is a single linked list that it is directly used to iterate over the set of assignable labels.

Filter by Lemma

The frames files contains exploitable information as the arguments that each predicate can take. In this filter we regard only the POS of the modifier token and the lemma of the predicate. Note that an argument annotation is a relation between the *predicate* and the *modifier* and the predicate not necessarily is the same token as the *head*.

This filter is queried by the POS of the *modifier* and the lemma of the *predicate*. In this case the building process of the filter is different from the previous ones. The corpus is traversed to collect all admissible argument for each POS. We collect independent lists for arguments referring to noun predicates and verb predicates.

These lists are copied as admissible for each predicate lemma, but each one of these lists are filtered with the information found in the frames files (i.e., the admissible arguments). We removed the arguments that are not in any roleset of the predicate.

5.12 Design of the equivalent pipeline system

To evaluate the ability of our joint system to overcome a pipeline approach we built an equivalent pipeline system. The pipeline system is as similar as possible to our joint system.

5.12.1 Syntactic component

The syntactic equivalent system was easily and simply built just by disconnecting the semantic components of our final joint system. We simply have to not score any semantic label when we are parsing a sentence.

As we previously seen the joint model behaves identically than the syntactic model in sentences with no predicates.

5.12.2 Semantic component

In this case it not possible to just disconnect the syntactic component as the parse algorithm cannot produce a tree structure from the unconnected semantic dependencies.

The semantic pipeline system is implemented to perform a classification token by token and for each predicate. For each sentence head-modifier pair is called the same scoring function than in our joint model. Also the learning algorithm remains unchanged.

Chapter 6

Results

Contents

6.1 Experimentation and results

A large number experiments were performed through the development process. As we stated at the start of this work, our main point of interest is to compare the pipeline and joint approaches and see if the last can overcome the former. Another important point regards the feasibility of the system. See section 5.11 for a detailed explanation about the efficiency improvements. In addition frequent design decision were made during the project development, and when possible the decision were supported by the included experiments.

We reran some experiments with the latest system configuration to facilitate a comparison across experiments. Some experiments are really expensive and were done using a reduced corpus or they are just performed once on the corpus versions available at the time of the run, in this case it will be noted.

Each comparison is set for a system with exact parameter configurations except for the variable that is being experimented. The results, unless otherwise noted, are for 1 epoch training on the whole training corpus with a learn

rate of 0.01 and using the 176k most frequent features for syntactic parsing and the same 176k plus 46k new features for semantic parsing. All scores are referring to the *development* corpus, recall that the *test* corpus must only be used for the final scoring.

Until the release of the training corpus, the small trial dataset provided by the shared task organizers was used to perform experiments and test the system. We do not report results from this corpus.

Below each experiment data there is brief discussion about the obtained results.

6.1.1 Preprocess

Number and groups of features

The first step of our system extracts and sets the features used by the subsequent phases.

Feature group	extracted features	selected features
Token features	41240	3812
Context features	186492	18996
Dependency features	3522869	3254
Dependency context features	148495	1760
Distance features	57392	6902
Extra features	90	85
Dynamic features	43422	824
Total features	4M	35633

Table 6.1: Extracted and selected features

Table 6.1 shows the number of extracted features and selected features. Note a lower number of selected features (35k) with respect to the final configuration (176k). We are concerned about an overgeneration of frequent features of some group that will prevent a fair selection of features among different groups. Table 6.1 shows that we are not discarding any group of features by using this simple method.

The feature group with the largest number of features is the *context dependency features* group. It contains concatenations of words and POS from the previous and following tokens. Thus a lot of combinations are generated but also as each combination is relatively infrequent and most of these features are discarded.

The optimal number of features was selected trough some empirical results using the 10% of the corpus. The threshold selection method forces us to firstly fix a threshold and then expect a reasonable number of filtered features.

The alternative method of sorting the frequency of the extracted features (4M or 5M) and selecting the top scoring ones is almost unfeasible due to cost of the sorting algorithm. The precomputation of syntactic features produced and acceptable file of 1.2GB.

The extraction of semantic features carried some space problems. The semantic threshold was set to 25000, we selected 46k features and filled a space of 71GB, (9GB compressed). In this case a major contributing factor in the selection of the semantic number of extracted features was the disc space and not just the final performance. Recall that syntactic features are extracted for each *modifier-head* combination but semantic features requires combinations of *modifier-head-predicate*.

The selection by threshold is a very efficient method that achieved good results, see the following sections. Anyway, more sophisticated features selection methods could improve the final performance. Some shared task participants, see section 6.2.1, employed the backward selection method, most of them not reporting the consumed time in this process. As we are designing a joint system we expect an increased computational cost of the core of the system, a wrapper feature selection is not a proper method for our case.

All the subsequent experiments are performed using 188k syntactic features and 46k semantic features.

6.1.2 Syntax

Use of syntactic tags vs. one tags

McDonald and Pereira (2006) reported better results when the Eisner parser not just assigns dependencies between words but also assigns the corresponding syntactic label. We compare these two approaches.

Scoring metric	w/o syn tags (%)	with syn tags (%)
Labeled attachment score:	7.18053	83.5171
Unlabeled attachment score:	51.0609	87.1434
Labeled accuracy score:	11.8467	89.5499

Table 6.2: Use of syntactic tags

The relevant score in table 6.2 is the *Unlabeled attachment score*, labeled scores are not applicable for a parser that does not produce annotations. Our results strongly confirm our expectations. One may think that predicting just the structure should be an easier task. But for a perceptron it remains as a difficult task. Recall that it is a linear classifier, and in our case infeasible to be kernelized. The use of a set of perceptrons, one specialized for each syntactic label, in fact renders the problem less difficult.

Use of a generic dependency label vs. all labels

The usual setting for a dependency parsing is the following: As we receive each pair of tokens we score all syntactic labels and chose the best. In fact we are performing two task simultaneously: linking the tokens and labelling that link. We considered that it could make sense to change the scoring of a dependency and not just compute the syntactic label score but also a generic likeliness of the unlabeled dependency.

Scoring metric	with tag (%)	w/o tag (%)
Labeled attachment score:	82.3603	83.5171
Unlabeled attachment score:	87.0355	87.1434
Label accuracy score:	88.3032	89.5499

Table 6.3: Use of a generic dependency tag

The addition of a generic dependency score failed to outperform the usual setting. Recalling the discussion of the previous experiment it is apparently difficult to learn a good scorer for all kinds of dependencies. The addition of this low performing classifier probably degraded the final results.

Averaged perceptron

Several works (Collins, 2002) reported a significant increase in final performance due to the perceptron averaging.

Scoring metric	non-averaged (%)	averaged (%)
Labeled attachment score:	77.8141	83.5171
Unlabeled attachment score:	81.9558	87.1434
Label accuracy score:	86.2413	89.5499
Epoch time (s)	1396	3389
Time per instance (s)	0.0355415	0.0862824

Table 6.4: Averaged Perceptron

Although we tried to implement an efficient averaged perceptron the computation times are still high. In fact it is possible to compute the averaged perceptron recording only the number of modifications to each feature with a minimum overhead.

Note that there is a difference in the latter reported time for the syntactic system and this reported times. The reported times on this table 6.4 are for the joint system with some semantic components disconnected. As we did not disconnected all semantic components we can appreciate some overhead.

The results of table 6.4 supports the use of the averaged perceptron even at the expense of the higher computing time.

Dynamic Features

Carreras et al. (2006) defined a new set of features computed on-line during the parsing process and obtained a significant and positive impact on the system results, for a detailed discussion see 3.2.2.

Scoring metric	**dyn feats (%)**	**standard features (%)**
Labeled attachment score:	83.5531	83.5171
Unlabeled attachment score:	87.1164	87.1434
Label accuracy score:	89.6278	89.5499
Epoch time (s)	16954	3389
Time per instance (s)	0.431641	0.0862824

Table 6.5: Dynamic features

The dynamic features are extracted inside the parser steps. At that time a large quantity of dynamic features is generated. It is possible to select the admissible features on-line or have them precomputed.

Our first implementation precomputed all dynamic features, a second one, ran the Eisner algorithm a couple of iterations to extract and precompute the features.

Table 6.5 shows the result of the second implemented strategy that runs the Eisner algorithm to preselect the dynamic features. Note that the increase in computational time is substantial and the performance gain (0.036) is not significative.

The results obtained with different selection algorithms pointed out the selection strategy plays an important role. We believe that dynamic features shouldn't be preselected to reproduce the Carreras et al. (2006) results.

Our next planned experiment was to also include features about grandsons and not just sons of the processed head. Due to this discouraging results and also the associated expensive computational cost we do not present further experimentation regarding this topic.

Projectivization techniques

The Eisner algorithm is unable to parse *crossing* or *non-projective* links. As some sentences (7.6%) contains one or more non-projective links a strategy to deal with this links should be used. Otherwise each one of this sentence will force a error correction by the learning algorithm. The correction is unnecessary and will modify weight vectors associated to labels that in fact could have

been correctly adjusted. McDonald and Pereira (2006) used a simple trick to deal with non-projective links, see section 3.2.5.

Scoring metric	**result (%)**
baseline	
Labeled attachment score:	83.5171
Unlabeled attachment score:	87.1434
Label accuracy score:	89.5499
No threshold, limited to 1 change	
Labeled attachment score:	81.9108
Unlabeled attachment score:	85.4561
Labeled accuracy score:	89.4630
1.05 threshold, limited to 1 change	
Labeled attachment score:	83.5141
Unlabeled attachment score:	87.1434
Labeled accuracy score:	89.5499

Table 6.6: McDonald's projectivization trick

Table 6.6 shows some of the extensive experimentation that was done producing discouraging and similar. Several configurations were tried, all failing to improve the non-projective baseline. The tested configurations include:

- Limiting the number of non-projective created links to 1,2,3 or no limit.
- Setting a threshold of 2%, 5%, 10% of the total tree score
- Setting a fixed threshold
- Including the non-projectivization inside the training and not just on the testing
- Some combinations of the above

The McDonald projectivization algorithm is, as far as we know, only applied to unlabeled trees. As we are applying this technique to labeled trees we could not expect similar results.

We believe that the algorithm could work well in unlabeled trees due to the fact that the score of a dependency is only related to the dependency probabilities of being selected. As in our case there are several labels competing for a dependency choosing it is possible that we couldn't use the label score only to change the dependencies selected.

The final system is completely non-projective, see sections 3.1 and 3.2.5 for further discussion.

Maltparser (reference parser) vs our parser

The organizers provided the output of *Maltparser* (Nivre et al., 2007b) to be used for the participants in the shared task *open* challenge. In fact, some participants at the CoNLL-2008 shared task, see section 6.2.1, used the *Maltparser* as their only dependency parser.

Scoring metric	result (%)
MaltParser reference	
Labeled attachment score:	84.1285
Unlabeled attachment score:	87.4461
Labeled accuracy score:	89.2412
Our parser, 1 epoch	
Labeled attachment score:	84.0176
Unlabeled attachment score:	87.6648
Labeled accuracy score:	89.7896
Our parser, 2 epoch	
Labeled attachment score:	84.5900
Unlabeled attachment score:	88.0934
Labeled accuracy score:	90.2451

Table 6.7: Maltparser performance

The *Maltparser* output had been useful to asses the system performance while we were in the development process. The experiment in table 6.7 was early done, it uses a different version of the corpus[1] and less features (85k instead of 176k). We are confident that the small changes that the corpus suffered will not significantly affect these results. This experiment confirms us that the parser is well implemented and the features and the learning algorithm are producing the expected results.

Syntactic filter

The Eisner algorithm cost contains a hidden constant depending on the size of the syntactic labels set. The use of a filter does not only improves the cost but also it could benefit the performance preventing the use of unlikely labels for a given dependency.

Recall that the number of syntactic labels is 70. The filter allows us to deal with a mean of 8.4 labels instead. In addition the computational cost was reduced by a factor of 5. The number of discarded correct labels (0.2%) is

[1] During the development, few releases of the corpus with minor corrections were provided.

negligible, therefore the upper bound of the syntactic parser (99.8%) remains almost unchanged.

Syntactic filter performance (development corpus)	
Number of incorrectly discarded labels	0.203788 %
Mean syntactic labels returned per query	8.36858

Table 6.8: Performance and upper bounds of the syntactic filter

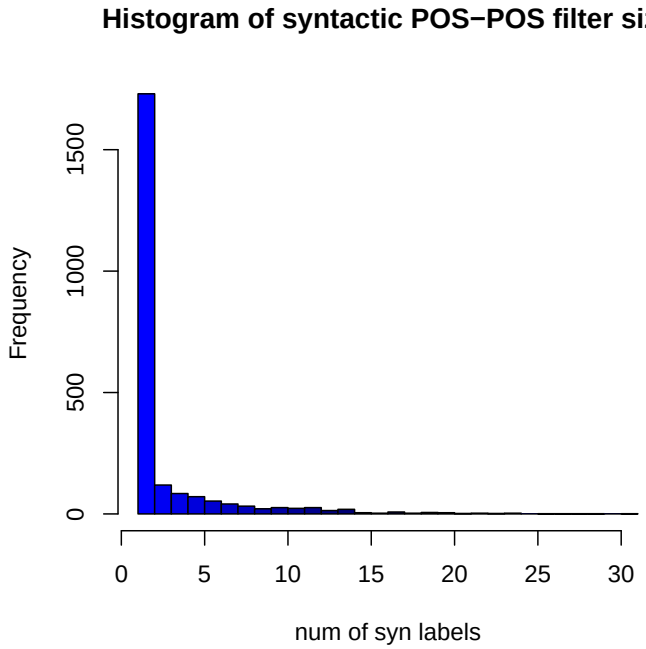

Figure 6.1: Histogram of syntactic filter size

The histogram in figure 6.1 shows that most combinations of POS-POS gives a filtered number of label much lower than 70. Most of the filter entries are just filled with one label. Note that if the POS-POS combination is not seen in the training corpus we always add a null label, as in every dependency and combination we must produce a score and a tag.

Passive-aggressive perceptron

The passive-aggressive perceptron algorithm modifies the update rule of the regular perceptron, see section 3.5.3. We expected to observe and improved performance.

Scoring metric	result (%)
baseline perceptron	
Labeled attachment score:	73.07
Unlabeled attachment score:	78.1707
Labeled accuracy score:	82.684
PA-I	
Labeled attachment score:	72.4526
Unlabeled attachment score:	77.6942
Labeled accuracy score:	81.9917
PA-II	
Labeled attachment score:	71.4217
Unlabeled attachment score:	76.8221
Labeled accuracy score:	81.2725

Table 6.9: Performance of the passive-aggressive perceptron

In this case the experiments were performed using a reduced set of 24k features and just 10% of the corpus. In addition the perceptron was not averaged.

We expected to see clearer improvements from the passive aggressive algorithm when the corpus size is small as the perceptron and the passive-aggressive perceptron algorithms could converge to close solutions after a large set of examples. We believe that the averaging could smooth the differences between the two algorithms.

We consider that the poor results are due to the processing of non-projective sentences. As non-projective sentences always generate errors in parsing, an aggressive update is forced, these updates will not allow the parser to correctly analyze a non-projective sense but they could change previous good weight vectors.

Final syntactic score

As we designed a joint system we will evaluate the final syntactic performance in the next *Joint parsing* section.

6.1.3 Predicate identification

To review statistics from the predicate distribution in the corpus see A.2. As we previously discussed we can correctly identify most of the verbal predicates by simply using few rules, see table 6.10.

	Precision (%)	Recall (%)	F_1
verb pred id	99.87	94.43	97.08

Table 6.10: Performance of verbal predicate identification

Although high results the rules made few errors. We will see in detail some of these errors.

Example 6.1.3.1 Incorrect rule classification
A non-predicate token *tailed/VBN* incorrectly identified as predicate

```
18 pony-tailed pony-tailed JJ JJ pony pony NN 20 HMOD _ _ _
19 _ _ _ _ --HYPH 18 HYPH _ _ _
20 _ _ _ _ tailed tail VBN 17 OBJ _ _ A1
```

Recall that corpus tokens as *made-for-TV* are split by hyphens. Our classification rules always tags a VBN token that is not in an auxiliary verb form as predicate.

The rules does not capture a very few multi-word verb constructions. In addition very few tokens inside a split word are labeled as verbs and therefore incorrectly tagged as predicates. As the performance of the rules is 97.08% F_1 there is a small room for improvement at at risk of causing overfitting. We invested our resources in the improvement of other components.

An SVM classifier with a list of nouns is used to identify nominal predicates. Table 6.11 shows some results for different SVM configurations.

	Precision (%)	Recall (%)	F_1
linear kernel, C=1	77.9731	94.0106	85.2441
linear kernel, C=2	77.2204	93.9035	84.7487
2-deg poly, C=1	82.2143	95.0744	88.1779
2-deg poly, C=2	82.2143	95.0744	88.1779
2-deg poly, C=1, all corpus	83.41	95.8689	89.2065

Table 6.11: Performance of the SVM for nominal predicate identification

Note that each one of the trainings consumes a few hours and was performed using the 10% of the training corpus. The final selected configuration was a 2-degree polynomial kernel with a parameter $C = 1$.

Finally and for this configuration we detailed results in table 6.12. Note that this table includes nominal and verbal predicates.

PPOSS	gold	correct	system	recall (%)	precision (%)	F_1
CD	1	0	0	0.00	NaN	NaN
JJ	16	0	0	0.00	NaN	NaN
NN	2181	1721	2024	78.91	85.03	81.86
NNP	5	0	0	0.00	NaN	NaN
NNS	1021	848	1038	83.06	81.70	82.37
PRF	1	0	0	0.00	NaN	NaN
RB	1	0	0	0.00	NaN	NaN
RP	3	1	1	33.33	100.00	50.00
VB	796	793	804	99.62	98.63	99.12
VBD	710	710	738	100.00	96.21	98.07
VBG	445	445	492	100.00	90.45	94.99
VBN	689	688	718	99.85	95.82	97.79
VBP	203	203	245	100.00	82.86	90.63
VBZ	317	317	354	100.00	89.55	94.49
WP	1	0	0	0.00	NaN	NaN
overall	6390	5726	6414	89.61	89.27	89.44

Table 6.12: Precision and recall for predicate identification

6.1.4 Joint Parsing

We will firstly review the semantic results. Most of the experiments are focused in the semantic component of the system but as we are working with a joint system there are interactions reflected on the syntactic scores. The most relevant scores for the shared task evaluation are marked in tables in *italics*. Note that all experiments, except the presented at the end of this subsection, are performed using the default predicate disambiguation. This disambiguation does not interacts with the rest of the system performance as it is the last postprocessing step. Final results are significantly higher due to the use of the most frequent sense tagging at postprocessing.

Hard updates vs. soft updates

Our joint parser produces a syntactic and a semantic structure simultaneously. Therefore the output can contain both kind of errors. A token correctly semantically annotated could in fact be linked to a syntactically incorrect head. We investigated if its is convenient to do not consider these errors also as semantic errors.

Table 6.13 shows that to consider syntactic errors (hard policy) as requiring also a semantic correction degrades the performance. We believe that a semantic annotation is a link between the *predicate* and the *modifier*. A strict

update policy will penalize semantic dependencies that do not require a correct *head* to be identified.

Scoring metric	hard update (%)	soft update (%)
SYNTACTIC SCORE		
Labeled attachment score:	*83.34*	*84.07*
Unlabeled attachment score:	86.90	87.65
Label accuracy score:	89.63	89.59
Exact syntactic match:	14.53	16.03
SEMANTIC SCORE		
Labeled precision:	65.07	69.99
Labeled recall:	64.49	68.71
Labeled F1:	*64.78*	*69.35*
Unlabeled precision:	77.18	82.71
Unlabeled recall:	76.49	81.20
Unlabeled F1:	76.83	81.95
Proposition precision:	24.04	33
Proposition recall:	24.13	33.51
Proposition F1:	24.09	33.44
Exact semantic match:	5.32	6.14
OVERALL MACRO SCORES (Wsem = 0.50)		
Labeled macro precision:	74.20	77.03
Labeled macro recall:	73.91	76.39
Labeled macro F1:	*74.06*	*76.71*
Unlabeled macro precision:	82.04	85.18
Unlabeled macro recall:	81.70	84.42
Unlabeled macro F1:	81.87	84.80
Exact overall match:	3.07	3.52

Table 6.13: Perceptron update policies

Semantic filters

The semantic filters are analogously implemented and used as the syntactic filters. We are interested in the resulting upper bounds of applying this filters.

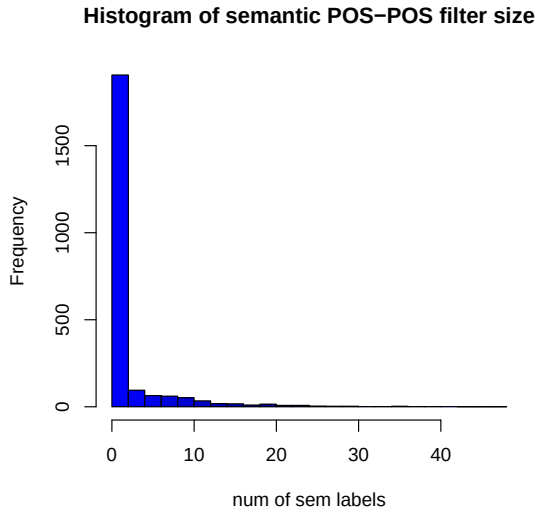

Figure 6.2: Histogram of semantic POS-POS filter size

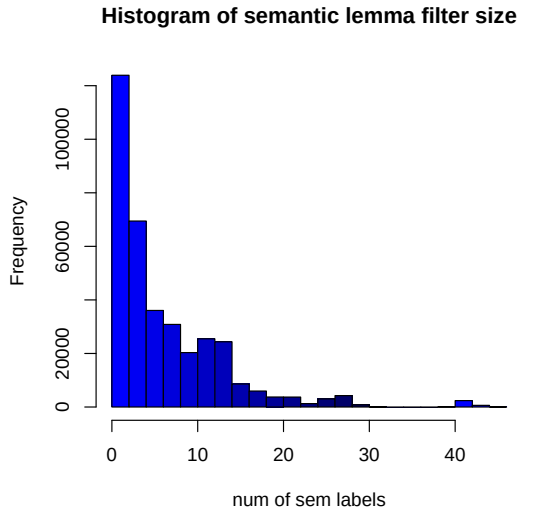

Figure 6.3: Histogram of semantic lemma filter size

Figure 6.2 shows the number of semantic labels returned by each filter entry. Most entries, or POS-POS combinations contains no semantic labels. Table 6.14 complements this information. Most of the POS-POS combinations will rarely accept any semantic label, but it could be possible that more plausible POS-POS combinations could be labeled with a large set of semantic labels.

The histogram in figure 6.3 shows the distribution of the size of the filter entries. In this case, the number of entries is larger but as we expected the are less filtered labels. The use of the POS-POS filter is a simpler and effective method, see table 6.14.

In our final implementation, we combined the to filters. The lemma filter contributes discarding 0.18% (25.0052-24.8203%) of the dependencies in addition to the 24.8203% discarded by the POS-POS filter. Although it is only a slight contribution we continued the use of the filter as the filter queries have a negligible computational cost.

Semantic POSPOS filter performance	
Entries	2304
Number of incorrectly discarded labels	0.44%
Mean semantic labels returned per query	11.2005
Directly discarded arguments	24.8203 %
Query returned just 1 label	7.41502 %
Semantic lemma filter performance	
Entries	364992
Number of incorrectly discarded labels	0%
Mean semantic labels returned per query	17.4408
Directly discarded arguments	4.3521 %
Query returned just 1 label	1.33266 %
Combined filter discarded args	25.0052 %

Table 6.14: Performance and upper bounds of the semantic filters

Both semantic filters significantly improved the system performance reducing the number of semantic labels from 80 to a mean of 11.2, see table 6.14. In addition the filters allowed us to directly discard argument annotation in 25% of the dependencies considered by the Eisner algorithm. In our architecture the argument identification step is set *before* the filter queries. If filters discard some tokens as arguments that tokens will not be considered to train the argument identification classifier. This scheme reduces the number of non-argument training examples passed to the argument identification step. Recall that a more balanced class distribution improves the perceptron perfor-

mance, see section 3.5.1 and the following experiments for further discussion about this topic.

Joint argument identification

The argument identification is a widely employed step before the argument classification (Màrquez et al., 2006). In this case we will perform the identification as a completely independent classification (i.e., outside the Eisner inference algorithm).

Scoring metric	no joint id (%)	joint id (%)
SYNTACTIC SCORE		
Labeled attachment score:	*77.49*	*84.07*
Unlabeled attachment score:	80.72	87.65
Label accuracy score:	85.84	89.59
Exact syntactic match:	5.77	16.03
SEMANTIC SCORE		
Labeled precision:	71.41	69.99
Labeled recall:	61.64	68.71
Labeled F1:	*66.16*	*69.35*
Unlabeled precision:	84.36	82.71
Unlabeled recall:	72.82	81.20
Unlabeled F1:	78.16	81.95
Proposition precision:	25.85	33
Proposition recall:	25.95	33.51
Proposition F1:	25.90	33.44
Exact semantic match:	4.04	6.14
OVERALL MACRO SCORES (Wsem = 0.50)		
Labeled macro precision:	74.45	77.03
Labeled macro recall:	69.57	76.39
Labeled macro F1:	*71.93*	*76.71*
Unlabeled macro precision:	82.54	85.18
Unlabeled macro recall:	76.77	84.42
Unlabeled macro F1:	79.55	84.80
Exact overall match:	1.87	3.52

Table 6.15: Joint argument identification

The system performed better when the argument identification is not a previous independent step, see table 6.15.

As in all experiments we provide the same setting to fairly compare the two strategies. Note that less elegant solutions for argument identification can be

used. A separated argument identification gives us a lot more design flexibility. We can train a wide variety of classifiers. More complex classifiers can be employed without the implementation issues that represents the inclusion of a learning component inside the Eisner inference algorithm.

The argument identification step is merely intended to improve the learning algorithm performance. There is a tendency on perceptron classifiers to overpredict the most frequent class. As the number of examples from the most frequent class is higher, more corrections are applied for this class examples. Thus, the corrections will ensure a higher absolute number of correct prediction for the majority class. The consequence in our case is a higher precision and lower recall of the argument identification. The low recall is associated with an overprediction of a non-argument class, the most frequent class.

An alternative solution is to change the learning algorithm. For example, SVM classifiers (see section 3.5.4) are much less affected by an uneven class distribution. But an SVM classifier for this task will be impractical to apply due to the large amount of data. Another alternative is to use a BIO tagging (see section 3.3.2) but this labelling is incompatible with our setting. BIO tagging requires a sequential processing of the arguments, in our setting the arguments are independently annotated and combined by the Eisner algorithm.

Predicate normalization

The scoring function of a dependency is a newly designed function for this work. Several parameters can be tuned. We expect improved results normalizing the semantic score by the number of sentence predicates. Note that if a predicate produced a null semantic label for the given dependency, the score of this null label will be zero. Thus the number of null labels are not counted in the normalization, i.e., we normalize by the number of predicates with arguments.

Table 6.16 reports a slight improvement by using the normalization strategy in the important scores for the task evaluation, as the *Labeled macro* F_1 score. The variable number of predicates in each sentence will probably produce an uneven semantic weight across the corpus sentences. This increased semantic weight slightly benefits semantic scores but it degrades the syntactic performance. Thus the best strategy for our joint system is to normalize by the number of predicates.

Final joint system results: syntax

The final system configuration is now evaluated in more detail. The previous experiments pointed out the chosen final configuration. We use 188k syntactic features plus 46k semantic, no dynamic features, soft semantic update policy, joint argument identification, predicate normalization, all filters, the obtained parameters for the SVM classifier.

Scoring metric	No-normalize (%)	normalize (%)
SYNTACTIC SCORE		
Labeled attachment score:	*83.82*	*84.07*
Unlabeled attachment score:	87.50	87.65
Label accuracy score:	89.78	89.59
Exact syntactic match:	15.36	16.03
SEMANTIC SCORE		
Labeled precision:	70.18	69.99
Labeled recall:	68.68	68.71
Labeled F1:	*69.42*	*69.35*
Unlabeled precision:	82.83	82.71
Unlabeled recall:	81.06	81.20
Unlabeled F1:	81.94	81.95
Proposition precision:	33.38	33
Proposition recall:	33.51	33.51
Proposition F1:	33.44	33.44
Exact semantic match:	6.22	6.14
OVERALL MACRO SCORES (Wsem = 0.50)		
Labeled macro precision:	77.00	77.03
Labeled macro recall:	76.25	76.39
Labeled macro F1:	*76.62*	*76.71*
Unlabeled macro precision:	85.16	85.18
Unlabeled macro recall:	84.28	84.42
Unlabeled macro F1:	84.72	84.80
Exact overall match:	3.52	3.52

Table 6.16: Score normalization by number of predicates

Evaluation metric	detail	result
SYNTACTIC SCORES:		
Labeled attachment score:	28203 / 33368	84.52 %
Unlabeled attachment score:	29371 / 33368	88.02 %
Label accuracy score:	30112 / 33368	90.24 %
Exact syntactic match:	223 / 1335	16.70 %

Table 6.17: Final syntactic scores

Table 6.17 shows the final syntactic scores, a detailed view of the results for each syntactic label is found in table 6.18.

Syn label	gold	correct	system	recall (%)	precision (%)
ADV	1256	879	1338	69.98	65.70
AMOD	536	334	452	62.31	73.89
APPO	444	299	439	67.34	68.11
BNF	1	0	1	0.00	0.00
CONJ	706	612	730	86.69	83.84
COORD	915	597	842	65.25	70.90
DEP	772	591	724	76.55	81.63
DEP-GAP	2	0	0	0.00	NaN
DIR	119	68	97	57.14	70.10
DIR-GAP	1	0	0	0.00	NaN
DIR-PRD	1	0	0	0.00	NaN
DTV	14	12	13	85.71	92.31
EXT	52	40	47	76.92	85.11
EXTR	15	9	15	60.00	60.00
GAP-LOC	2	0	0	0.00	NaN
GAP-LOC-PRD	1	0	0	0.00	NaN
GAP-OBJ	2	0	1	0.00	0.00
GAP-SBJ	5	0	1	0.00	0.00
GAP-TMP	4	0	1	0.00	0.00
GAP-VC	2	0	2	0.00	0.00
HMOD	423	403	433	95.27	93.07
HYPH	421	421	437	100.00	96.34
IM	512	507	510	99.02	99.41
LGS	93	80	95	86.02	84.21
LOC	556	327	549	58.81	59.56
LOC-OPRD	4	0	0	0.00	NaN
LOC-PRD	26	15	30	57.69	50.00
MNR	109	54	110	49.54	49.09
NAME	1138	837	1024	73.55	81.74
NMOD	8922	8077	9048	90.53	89.27
OBJ	1728	1550	1774	89.70	87.37
OPRD	373	307	359	82.31	85.52
P	3760	2892	3771	76.91	76.69
PMOD	3263	3016	3334	92.43	90.46
POSTHON	125	109	127	87.20	85.83
PRD	509	416	487	81.73	85.42
PRD-TMP	1	0	1	0.00	0.00
PRN	52	9	42	17.31	21.43
PRP	105	65	108	61.90	60.19
PRT	97	80	99	82.47	80.81
PUT	11	8	10	72.73	80.00

ROOT	1334	1244	1334	93.25	93.25
SBJ	2407	2207	2471	91.69	89.32
SUB	371	328	376	88.41	87.23
SUFFIX	295	289	301	97.97	96.01
TITLE	173	138	140	79.77	98.57
TMP	755	460	707	60.93	65.06
VC	953	923	988	96.85	93.42
VOC	2	0	0	0.00	NaN

Table 6.18: Precision and recall for syntactic dependencies

Table 6.18 shows detailed scores. Infrequent labels are hard to predict due to the low number of training examples but their misclassifications have a small impact on the final score. Some infrequent labels are in fact combinations (e.g., LOC-PRD) of more frequent ones. The learned classifiers for the single labels could improve the prediction of combined labels but at the expense of a increased design complexity. *LOC*, *TMP* and *MNR* are frequently mispredicted due to the similarities they present . The syntactic results are comparable to other well known state of the art syntactic parsing system, see experiment in section 3.2.4. A detailed analysis of individual prediction errors and the fine tuning of the system is outside the scope of this work as our main propose is to evaluate the whole joint learning architecture.

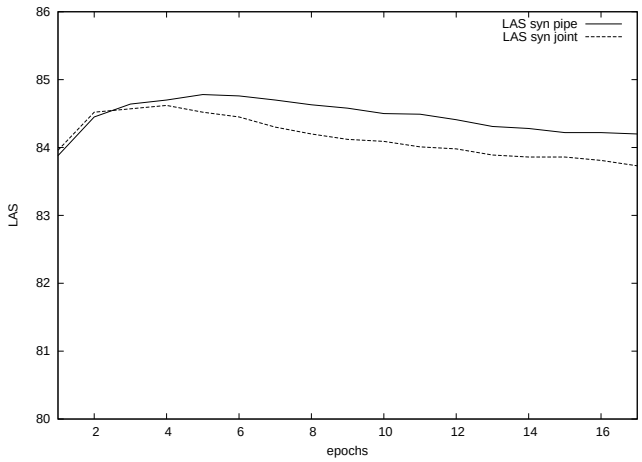

Figure 6.4: Learning curves for the syntactic-only and joint parsers.

Figure 6.4 shows the learning curves from epoch 1 to 17 for the joint and the equivalent pipeline systems. More specifically, it includes the LAS performance on syntactic parsing. We can observe that the syntactic LAS scores for the syntactic and joint parsers are very similar, showing that there is no loss in syntactic performance when using the joint syntactic-semantic

strategy.

Final joint system results (semantics)

Setting the system with the final configuration we extract detailed semantic results.

Table 6.19: Final semantic scores

Evaluation metric	detail	result
SEMANTIC SCORES:		
Labeled precision:	(8585 + 5099) / (11994 + 6414)	74.34 %
Labeled recall:	(8585 + 5099) / (13865 + 6390)	67.56 %
Labeled F1:		70.79
Unlabeled precision:	(9922 + 5726) / (11994 + 6414)	85.01 %
Unlabeled recall:	(9922 + 5726) / (13865 + 6390)	77.25 %
Unlabeled F1:		80.95
Proposition precision:	2132 / 6414	33.24 %
Proposition recall:	2132 / 6390	33.36 %
Proposition F1:		33.30
Exact semantic match:	77 / 1335	5.77 %

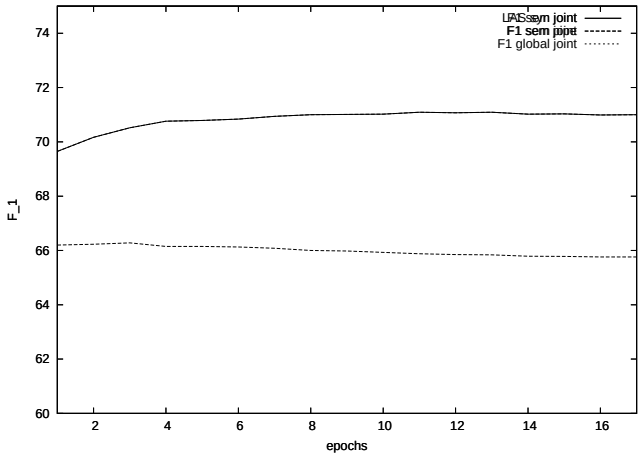

Figure 6.5: Learning curves for the semantic-only and joint parsers.

The learning curves of figure 6.5 represents the F_1 labeled semantic scores for both the joint and the only-semantic parser. In this case a clear improvement between 3 and 5 points is observed.

PPOSS+ARG	gold	correct	system	recall (%)	precision (%)	F_1

CD* + A1	1	0	0	0.00	NaN	NaN
JJ* + A0	5	0	0	0.00	NaN	NaN
JJ* + A1	15	0	0	0.00	NaN	NaN
JJ* + A2	1	0	0	0.00	NaN	NaN
JJ* + AM-EXT	1	0	0	0.00	NaN	NaN
JJ* + AM-MNR	1	0	0	0.00	NaN	NaN
JJ* + AM-MOD	1	0	0	0.00	NaN	NaN
JJ* + AM-TMP	2	0	0	0.00	NaN	NaN
NN* + A0	1419	542	785	38.20	69.04	49.19
NN* + A1	2278	1282	1961	56.28	65.37	60.49
NN* + A2	934	347	537	37.15	64.62	47.18
NN* + A3	192	87	135	45.31	64.44	53.21
NN* + A4	19	3	6	15.79	50.00	24.00
NN* + A5	1	0	0	0.00	NaN	NaN
NN* + AM-ADV	17	0	5	0.00	0.00	NaN
NN* + AM-CAU	3	0	0	0.00	NaN	NaN
NN* + AM-DIR	3	0	0	0.00	NaN	NaN
NN* + AM-DIS	5	0	0	0.00	NaN	NaN
NN* + AM-EXT	18	6	13	33.33	46.15	38.71
NN* + AM-LOC	154	64	139	41.56	46.04	43.69
NN* + AM-MNR	198	75	146	37.88	51.37	43.61
NN* + AM-MOD	4	0	0	0.00	NaN	NaN
NN* + AM-NEG	19	6	6	31.58	100.00	48.00
NN* + AM-TMP	290	162	228	55.86	71.05	62.55
NN* + R-A0	2	0	0	0.00	NaN	NaN
NN* + R-A1	2	0	0	0.00	NaN	NaN
NN* + R-AM-TMP	2	0	0	0.00	NaN	NaN
PR* + A0	1	0	0	0.00	NaN	NaN
PR* + A1	1	0	0	0.00	NaN	NaN
RB* + A0	1	0	0	0.00	NaN	NaN
RB* + A1	1	0	0	0.00	NaN	NaN
RB* + AM-DIS	1	0	0	0.00	NaN	NaN
RP* + A1	2	1	1	50.00	100.00	66.67
RP* + AM-TMP	1	0	0	0.00	NaN	NaN
VB* + A0	2019	1498	1941	74.20	77.18	75.66
VB* + A1	2924	2459	3043	84.10	80.81	82.42
VB* + A2	660	397	607	60.15	65.40	62.67
VB* + A3	110	54	95	49.09	56.84	52.68
VB* + A4	65	45	60	69.23	75.00	72.00
VB* + A5	2	0	0	0.00	NaN	NaN
VB* + AA	1	0	0	0.00	NaN	NaN
VB* + AM-ADV	270	125	256	46.30	48.83	47.53
VB* + AM-CAU	43	30	36	69.77	83.33	75.95
VB* + AM-DIR	31	10	22	32.26	45.45	37.74
VB* + AM-DIS	196	118	171	60.20	69.01	64.30
VB* + AM-EXT	28	11	20	39.29	55.00	45.84
VB* + AM-LOC	187	119	225	63.64	52.89	57.77
VB* + AM-MNR	230	100	195	43.48	51.28	47.06
VB* + AM-MOD	309	271	283	87.70	95.76	91.55
VB* + AM-NEG	104	89	125	85.58	71.20	77.73
VB* + AM-PNC	80	36	59	45.00	61.02	51.80
VB* + AM-PRD	3	0	1	0.00	0.00	NaN
VB* + AM-TMP	585	383	541	65.47	70.79	68.03
VB* + C-A1	139	87	128	62.59	67.97	65.17
VB* + C-A2	3	0	0	0.00	NaN	NaN
VB* + C-AM-CAU	1	0	0	0.00	NaN	NaN
VB* + C-AM-DIR	1	0	0	0.00	NaN	NaN
VB* + C-AM-DIS	0	0	2	NaN	0.00	NaN

VB* + C-AM-EXT	0	0	1	NaN	0.00	NaN
VB* + C-AM-MNR	2	0	1	0.00	0.00	NaN
VB* + R-A0	142	118	133	83.10	88.72	85.82
VB* + R-A1	79	47	62	59.49	75.81	66.67
VB* + R-A2	5	2	2	40.00	100.00	57.14
VB* + R-AM-CAU	3	0	1	0.00	0.00	NaN
VB* + R-AM-EXT	1	0	1	0.00	0.00	NaN
VB* + R-AM-LOC	9	1	2	11.11	50.00	18.18
VB* + R-AM-MNR	6	0	1	0.00	0.00	NaN
VB* + R-AM-TMP	29	10	18	34.48	55.56	42.55
WP* + A0	1	0	0	0.00	NaN	NaN
WP* + AM-MOD	1	0	0	0.00	NaN	NaN
WP* + R-A1	1	0	0	0.00	NaN	NaN

Table 6.20: Final semantic performance of the joint system

The detailed results for the joint system are shown in table 6.20. Most of the arguments are the core arguments A0 and A1, usually representing the agent and the patient of a predicate. Reasonable high results are achieved in the classification of A0 and A1 for verb predicates. AM-NEG argument classification performed surprisingly poor, usually the AM-NEG argument can be easily identified by a negation word (e.g, *no*, *never*). We did not found for this problem a clear error pattern. As we can see the final performance of the system is severely hurt by the nominal predicate classification (i.e., the arguments associated to NN and JJ part of speech). To point out the causes of this low performance further research is required in the uncommonly studied problem of nominal argument classification.

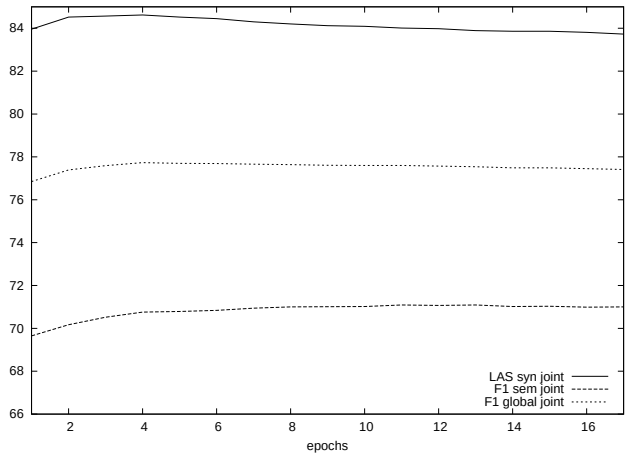

Figure 6.6: Learning curves for the joint parser.

Overall results are quite stable from epoch 4 (syntax slightly decreases but semantics slightly increases). This results points that the semantic prediction could hurt the syntactic tree. Another possibility is that the learning of semantic dependencies requeries a larger number of epochs or a higher learning

rate, note that there are less semantic dependencies that syntactic ones. It could be possible that the syntactic part requires less epochs and overfitting occurs while the semantic part is not yet suffering it. This hypothesis is partially contradicted by the fact that the equivalent pipeline system does not improves as the number of epochs increases, recall figure 6.5. This is not a strong statement as the pipeline and joint system could differently behave, new experiments should be performed. For further discussion about the syntactic and semantic interactions see the chapter 7. Given the learning curve results the presented system at the CoNLL-2008 shared task was trained for 4 epochs.

6.1.5 Predicate disambiguation

The last postprocessing step is to disambiguate the predicate sense. There are 6930 predicates, as disambiguation is scored as a semantic dependency task, the predicates represent the 31.54% of the semantic score.

The baseline strategy to perform the task is to assign a default sense *.01* to each identified predicate. Table 6.21 shows the results of this simple postprocess.

PPOSS	gold	correct	system	recall (%)	precision (%)	F_1
CD	1	0	0	0.00	NaN	NaN
JJ	16	0	0	0.00	NaN	NaN
NN	2181	1515	2024	69.46	74.85	72.05
NNP	5	0	0	0.00	NaN	NaN
NNS	1021	774	1038	75.81	74.57	75.18
PRF	1	0	0	0.00	NaN	NaN
RB	1	0	0	0.00	NaN	NaN
RP	3	1	1	33.33	100.00	50.00
VB	796	621	804	78.02	77.24	77.63
VBD	710	603	738	84.93	81.71	83.29
VBG	445	370	492	83.15	75.20	78.98
VBN	689	591	718	85.78	82.31	84.01
VBP	203	166	245	81.77	67.76	74.11
VBZ	317	268	354	84.54	75.71	79.88
WP	1	0	0	0.00	NaN	NaN
overall	6390	4909	6414	76.82	76.54	76.68

Table 6.21: Precision and recall for predicate disambiguation (default sense .01)

Also a simple strategy is to assign the most frequent sense to each identified predicate.

Scoring metric	result (%)
default .01 sense	
Labeled precision:	76.54
Labeled recall:	76.82
Labeled F1	76.68
most frequent sense	
Labeled precision:	79.50
Labeled recall:	79.80
Labeled F1:	79.65

Table 6.22: Performance of the most frequent sense postprocess

As expected table 6.22 shows that this simple postprocess is enough to achieve a significative performance gain. We also include detailed results in table 6.23.

PPOSS	gold	correct	system	recall (%)	precision (%)	F_1
CD	1	0	0	0.00	NaN	NaN
JJ	16	0	0	0.00	NaN	NaN
NN	2181	1549	2024	71.02	76.53	73.67
NNP	5	0	0	0.00	NaN	NaN
NNS	1021	783	1038	76.69	75.43	76.05
PRF	1	0	0	0.00	NaN	NaN
RB	1	0	0	0.00	NaN	NaN
RP	3	1	1	33.33	100.00	50.00
VB	796	675	804	84.80	83.96	84.38
VBD	710	629	738	88.59	85.23	86.88
VBG	445	382	492	85.84	77.64	81.53
VBN	689	611	718	88.68	85.10	86.85
VBP	203	182	245	89.66	74.29	81.25
VBZ	317	287	354	90.54	81.07	85.54
WP	1	0	0	0.00	NaN	NaN

Table 6.23: Precision and recall for predicate disambiguation (most frequent sense)

An alternative to this two previous strategies would be to train classifier. But the low number of examples per verb sense could severely hurt the classifier performance. To overcome the problems caused by the low number of examples a remarkable algorithm applied to word sense disambiguation by

Ando (2007) exploits a common subset of selected features across different classifiers.

In addition, some more work is published in predicate sense disambiguation, and substantial more about word sense disambiguation (Agirre and Soroa, 2007). A detailed discussion about these topics is outside the scope of this master's thesis.

The semantic arguments that the predicate takes could provide information to improve the predicate sense disambiguation. This is the reason behind applying as the last step the sense disambiguation. Note on the contrary that the previous argument classification could benefit form knowing which is the predicate sense. The predicate sense disambiguation is an interesting task to include in a joint architecture in the future.

Integer Linear Programming

ILP offers an alternative to classifier training for predicate sense disambiguation. In addition domain constraints and information from the predicate rolesets could be exploited.

Score metric	results (%)
best output	
labeled precision	81.4733
labeled recall	66.2892
labeled F_1	73.1011
second best output	
labeled precision	3.57238
labeled recall	2.9066
labeled F_1	3.20528
combined oracle	
labeled precision	84.6554
labeled recall	69.1958
labeled F_1	76.1489

Table 6.24: Performance of the best and second best output

Recall that the second best output of the joint parser is available. Table 6.24 shows the performance of the best and second best output. In this case we use gold syntax, gold predicates and predicted arguments to perform the experiment. The combined oracle entry in the table accounts for the performance of a system that selects always the correct argument from the available best and second best outputs.

baseline	
selected best (1)	100
selected second best (2)	0
selected none (0)	0
selected 1 and correct	97.3801
selected 2 and correct	nan
selected 0 and correct	nan
selected 1 and better to select 2	0.20008
selected 2 and better to select 1	nan
selected 0 and better to select 1	nan
selected 0 and better to select 2	nan
Labeled semantic precision	84.44
Labeled semantic recall	73.67
F_1	78.68

Table 6.25: Baseline performance for the ILP system

one core constraint	
selected best (1)	99.8421
selected second best (2)	0.14795
selected none (0)	0.00992955
selected 1 and correct	97.4848
selected 2 and correct	34.5638
selected 0 and correct	25
selected 1 and better to select 2	0.189954
selected 2 and better to select 1	31.2081
selected 0 and better to select 1	30
selected 0 and better to select 2	0
Labeled semantic precision	85.15
Labeled semantic recall	73.28
F_1	78.77

Table 6.26: Performance for the ILP system with the one core constraint

one core and forbidden argument constraints	
selected best (1)	99.8377
selected second best (2)	0.150929
selected none (0)	0.011419
selected 1 and correct	97.4872
selected 2 and correct	35.1974
selected 0 and correct	21.7391
selected 1 and better to select 2	0.187974
selected 2 and better to select 1	31.5789
selected 0 and better to select 1	30.4348
selected 0 and better to select 2	0
Labeled semantic precision	85.14
Labeled semantic recall	73.26
F_1	78.75

Table 6.27: Performance for the ILP system with the one core constraint and sense disambiguation

Tables 6.25, 6.26 and 6.27 show the system performance for different combinations of constraints by the ILP system. The percentages appearing in tables denote global accuracy for argument classification (e.g., 97.4872% of accuracy includes the non argument labels that represents most of the semantic labels but are not accounted in the usual precision-recall measures).

The second best output is clearly a very bad output, see table 6.24. The few arguments that the ILP system selects from the second output are mostly incorrect. Thus we cannot expect improved results by the output combination.

The sense selection using the predicate frames files performed poorly. The sense selection is based on the forbidden arguments for each predicate. Usually most predicate senses accept A0,...,A3 arguments. Thus the selection of sense finally relies on the infrequent arguments A4,A5. As this arguments are rare is not possible to disambiguate most predicates using the roleset information.

6.1.6 Efficiency

During the development phase, the time per sentence evolved from the first prototype of about 2min and to 0.2s for the syntactic only final system. The extraction of dynamic features accounted for about 23 seconds per sentence

at the first implementation and was reduced to 0.2 seconds on the last system, although finally not included.

system	time(s)	memory(GB)
Syntactic pipeline only system	0.2	0.7
Semantic pipeline only system	0.2	0.4
Joint parser	0.3	1.5

Table 6.28: Summary of system performance

Table 6.28 summarizes the system performance. The syntactic parser can be trained at 0.2 s/sentence, and the joint parser at 0.3 s/sentence. Efficiency at test time is only slightly better. These times are for a single processor of an *amd64 Athlon x2 5000+*.

Regarding efficiency, the proposed architecture is really feasible. About 0.7GB of memory is required for the syntactic parser and 1.5GB for the joint parser. Most of these memory needs are due to the filters used. The semantic extension of the Eisner algorithm requires only a new table with back-pointers for each predicate.

The filters reduced the computational cost by a factor of 5 with no loss in accuracy. These filters have almost no effect on the theoretical upper bound discarding the correct labels for only 0.2% of the syntactic dependencies and 0.44% of the semantic arguments in the development corpus, see the previous sections for details.

Function	Function calls	Time (%)
Averaged perceptron	9547	66.5
Compute Syntactic and Semantic scores	7,501,544	20.0
Dot product computation	217,507,753	18.0
Read and decompress files	20,000	2.9
Protected vector access	3,466,886,628	1.3
Assign Labels and sum subtree scores	447,000,004	0.6

Table 6.29: Profiling of a system run

We did not included results for all functions. Note that the scoring function (20% of time) includes all the dot product computations (18% of time). Protected vector access checks that we are retrieving a valid position. The results are extracted for a run with a reduced corpus of 10K sentences. These results confirms that the system is efficiently implemented, except for the averaged perceptron computation. The inner loop tagging of labels is fast (0.6%) as the scores are previously computed by the corresponding function called a lesser number of times.

6.2 The CoNLL-2008 shared task evaluation

The CoNLL-2008 shared task was the task with the largest number of registered teams ever, more than 50 teams registered for participation. But only 23 of the participants finally presented a system. Our system was one of the very few systems that contributed with a joint paring architecture.

6.2.1 Shared task results

Our system was presented in the CoNLL-2008 shared task.

Group	Name	WSJ + Brown	WSJ	Brown
Lund (*)	Johansson (*)	85.49	86.61	76.34
Yahoo! (*)	Ciaramita (*)	82.69	83.83	73.51
HIT-IR	Che	82.66	83.78	73.57
Hong Kong (*)	Zhao (*)	82.24	83.41	72.70
Geneva (*)	Henderson (*)	80.48	81.53	71.93
Koc	Yuret	79.84	80.97	70.55
GSLT ML2	Samuelsson	79.79	80.92	70.49
DFKI 2	Zhang	79.32	80.41	70.48
NAIST	Watanabe	79.1	80.3	69.29
Antwerp	Morante	78.43	79.52	69.55
HIT-ICR	Li	78.35	79.38	70.01
UPC (*)	**Lluís (*)**	**78.11**	**79.16**	**69.84**
UT Austin	Baldridge	77.49	78.57	68.53
Koc	Yatbaz	77.45	78.43	69.61
USTC	Chen	77	77.95	69.23
Korea	Lee	76.9	77.96	68.34
Peking	Sun	76.28	77.1	69.58
Colorado	Choi	71.23	72.22	63.44
UAIC	Trandabat	63.45	64.21	57.41
DFKI 1	Neumann	19.93	20.13	18.14

Table 6.30: Published results from the CoNLL-2008 shared task web site

The highlighted results in table 6.30 corresponds to the results published in our CoNLL-2008 description paper (Lluís and Màrquez, 2008) and the shared task web site. The overall results on the test set (78.11 global F_1, 85.84 LAS, 70.35 semantic F_1) were computed by using 5 epochs of training, the optimal on the development set, see the learning curve on the previous section 6.1.4

The global F_1 result on the WSJ test corpus is 79.16, but these results drop 9.32 F_1 points on the out-of-domain Brown corpus. Also, a significant

performance drop is observed when moving from verb argument classification (74.58 F_1, WSJ test) to noun argument classification (56.65 F_1, WSJ test). Note that the same features were used for training noun and verb argument classifiers. These results point out that there is room for improvement on noun argument classification. We regard these results as encouraging given that the system is built from scratch under the hard time constraints imposed by the shared task.

Few teams contributed with a joint parsing system. We consider the only comparable presented system, as it performed syntactic and semantic parsing completely simultaneous the Henderson et al. (2008) system. A detailed discussion about each recently presented system is outside the scope of this work and the reader is referred to the CoNLL-2008 shared task proceedings, see appendix E.

Chapter 7

Final remarks

Contents

7.1 Discussion

We presented a novel model for syntactic and semantic parsing by extending Eisner's first order model. Syntactic and semantic interactions are captured by globally optimizing a combined score. The computational cost of the associated new algorithm is admissible in practice, leading to fairly efficient parsers, both in time and memory requirements.

The system was trained on a substantially large corpus ($\sim$1M tokens). For the first time a corpus with syntactic dependencies and nominal and verbal predicate arguments was made available due to the effort of the organizers and corpus annotators. This corpus is fundamental to perform research on this topic. As in any other corpus there are still few inconsistencies and errors.

Results obtained with the presented joint approach are promising though not outstanding in the context of the CoNLL-2008 shared task. We believe that there is room for substantial improvement since many of the current system components are fairly simple. In addition, higher order extensions to the Eisner algorithm and well-known techniques for dealing with non-projective structures can be incorporated in our model. Also, we are planing to incorporate other subtasks such as predicate identification and the syntactic pre-parse.

One of the potential drawbacks of our current approach is the need for a syntactic parsing preceding the joint model. This previous parse is simply intended to allow the extraction of syntax-based features for the semantic classification. The alternatives to avoid this pre-parse are complex. The bottom-up

nature of the parser restricts the available information to the current parsed subtree. Thus all syntactic features could be dynamically computed only in the cases where the predicate coincides with the head of the current modifier. These cases account for the 63.6% of the training corpus arguments.

On the other hand, if a predicate is located outside the subtree (i.e., in a sibling subtree), the syntactic relation required will be completely unknown to the parser at that time. Furthermore, another possibility is that the predicate can be located at a lower level within the current subtree. These two previous cases would require to recompute again the score of the current subtree when all features are available (i.e., when the required syntactic path is finally formed).

The resulting cost of this approach would be prohibitive and approximate search needed. Our previous parsing phase is just an efficient and simple solution to the feature extraction problem in the joint model.

Regarding the last postporcessing step we can realize that information contained in frames files remains unexploited for our system. The postprocessing step achieved poor results. In part due to the mostly incorrect second best parser output. But also due to the high degree of overlapping between the allowed arguments of each predicate sense, preventing the ILP postprocessing to improve the sense selection. It is not clear whether or not we can exploit with simple methods the frames files. The next option to consider is to train a classifier to disambiguate predicate senses.

Most of the experiments were intended to support implementation decisions with a significant impact on the final system performance. Some these experiments failed to produce the expected results. One of the remarkable failures concerns the passive-aggressive perceptron as it did not outperform the regular perceptron algorithm. We believe that other configurations of this algorithm and the use of projectivization techniques will allow us to fully exploit this simple but powerful learning algorithm. A detailed discussion about each experiment can be found in the previous chapter 6.

7.2 Conclusions

The emphasis of this work is not on machine learning methods but in the design and implementation of a joint model.

An important factor in practical system design is the simplicity of the design. The objective of constructing a joint system moved us away from this simplicity principle. But recall that our goal was exploit syntactic and semantic interactions in a joint architecture.

The result of this work is a novel joint system that also is one of the most efficient presented systems with reasonably high and promising results. The produced results gave us some light to begin to answer the questions that we stated at the beginning of this work. Regarding the comparison of the

joint and pipeline approaches, as previously seen, the joint model showed a similar syntactic performance and a clearly better semantic performance than the equivalent pipeline system. Thus showing that some degree of syntactic-semantic interaction is exploitable.

The effects of the syntactic and semantic overlapping remain as an open question. There is only a moderate degree (63.6%) of direct overlapping between the syntactic *head-modifier* and semantic *predicate-modifier* relations. We believe that the overlapping problem could affect the dependency scores. If the semantic score is highly dependent on a correct head the resulting increased score could benefit the choosing of a correct dependency. Otherwise, joint scores can introduce a significant amount of noise to the final scoring function.

In addition this overlapping problem points to the serious question of where is the upper bound of the model. Another novel joint approach presented on CoNLL-2008 by Henderson et al. (2008) showed similar findings concerning the pipeline vs. joint approaches. Henderson et al. reported an improvement of the syntactic component by 0.3 points and the semantic component by 3.5 points with respect to the pipeline system.

Unfortunately, no other completely joint system have been presented in the CoNLL-2008. But the broad range of parsing algorithms remains far from being fully exploited and extended, see section 3.1.

All of this master's thesis goals, including the design, implementation and evaluation of the joint syntactic and semantic model have been successfully accomplished. Furthermore, this research work resulted in a publication in the proceeding of the CoNLL-2008 (Lluís and Màrquez, 2008). The published article was one of the five selected for an oral presentation in the CoNLL-2008 due to its originality.

All the possible extensions and a complete analysis of our joint model proposal are still far from being completed. All in all, further research is required in this direction.

7.3 Future work

In this section we sketch the main tasks that we are intended to develop as a continuation of this research work.

1. Higher degree of joint processing.
   The predicate identification and the syntactic pre-parse are two previous phases that could be exploited inside the joint model framework. The inclusion of these phases in the joint model seems a fairly complex task, see section 7.1, but we are already working in promising approximate solutions to address these problems.

2. Extension to higher order dependencies.
   Higher order extensions allows us to overcome some of the arc-factored approach limitations. Higher order models by McDonald and Pereira (2006) and Carreras (2007) described also in section 3.2.2, showed a significant performance improvement, at a expense of higher cost.

3. Improvement of the semantic classifier component.
   The semantic classifier component did not achieved top performance. It is a fairly simple component as simple features are shared across different semantic tasks: argument identification, argument classification and predicate identification. Improved performance can be achieved by using different set of features for each task.

4. Identification of the upper bounds of the model.
   A complete exploitation of syntactic and semantic interactions may not be possible due to model limitations. A comparison to an improved semantic baseline system is one the first steps towards a more detailed evaluation of our joint model.

5. Projectivization techniques.
   The model is unable to parse non-projective dependency links. These links always generate classification errors and force learning updates. Nivre and Nilsson (2005) algorithm is regarded as a simple and effective solution. In this case a rerun of the passive-aggressive perceptron experiment with the new configuration could improve its results.

6. Feature engineering and system tuning.
   Some top performing system employed very simple feature selection techniques that can be improved. Feature engineering, including the definition of new features, is a task that should receive more attention. Also a lot system parameters are to be fine-tuned to increase the system final performance.

7. Extended assessment of alternative joint models.
   There is a large number of parsing algorithms and models that are not exploited and remain available to be extended. Plenty of research directions can be taken.

References

Agirre, Eneko and Aitor Soroa. 2007. Semeval-2007 task 02: Evaluating word sense induction and discrimination systems. In *Procceedings of ACL 2007*.

Ando, Rie Kubota. 2007. Applying alternating structure optimization to word sense disambiguation. In *Procceedings of CoNLL 2007*.

Boser, Bernhard, Isabelle Guyon, and Vladimir Vapnik. 1992. An training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*.

Buchholz, Sabine, Erwin Marsi, Amit Dubey, and Yuval Krymolowski. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-2006)*.

Carreras, Xavier and Lluís Màrquez. 2004. Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL-2004)*.

Carreras, Xavier and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL-2005)*.

Carreras, Xavier, Mihai Surdeanu, and Lluís Màrquez. 2006. Projective dependency parsing with perceptron. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-2006)*.

Carreras, Xavier, Michael Collins, and Terry Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of CoNLL-2008*.

Carreras, Xavier. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the 11th Conference on Computational Natural Language Learning (CoNLL-2007)*.

Cohn, Trevor and Philip Blunsom. 2005. Semantic role labelling with tree conditional random fields. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*.

Collins, Michael. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*.

Covington, Michael A. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*.

Crammer, Koby and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*.

Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*.

Daelemans, Walter and Antal van den Bosch. 2005. *Memory-Based Language Processing*. Cambridge University Press.

Daumé III, Hal and Daniel Marcu. 2006. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*.

Edmonds, Jack. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*.

Eisner, Jason M. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*.

Francis, W. Nelson and Hernry Kucera, 1964. *Brown Corpus Manual. A Standard Corpus of Present-Day Edited American English, for use with Digital Computers*.

Geman, Stuart and Donald Geman. 1984. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *EEE Transactions on Pattern Analysis and Machine Intelligence*.

Henderson, James, Paola Merlo, Gabrielle Musillo, and Ivan Titov. 2008. A latent variable model of synchronous parsing for syntactic and semantic dependencies. In *Proceedings of CoNLL-2008 Shared Task*.

Hudson, Richard. 1987. Zwicky on heads. *Journal of Linguistics*.

Joanis, Eric. 2002. Automatic verb classification using a general feature space. Master's thesis, University of Toronto.

Karp, Richard M. 1972. *Reducibility Among Combinatorial Problems*. Complexity of Computer Computations.

Koomen, Peter, Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2005. Generalized inference with multiple semantic role labeling systems. In *Proceedings of CoNLL-2005 Shared Task*.

Lluís, Xavier and Lluís Màrquez. 2008. A joint model for parsing syntactic and semantic dependencies. In *Proceedings of CoNLL-2008 Shared Task*.

Marcus, Mitchell P., Beattrice Santorini, and Mary Ann Mracinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*.

Màrquez, L., P. R. Comas, J. Giménez, and N. Català. 2005. Semantic role labeling as sequential tagging. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*.

Màrquez, Lluís, Kenneth C. Litkowsky, Suzanne Stevenson, and Xavier Carreras. 2006. Special issue on semantic role labeling. *Computational Linguistics*.

McDonald, Ryan and Fernando Lerman, Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Conference on Natural Language Learning (CoNLL-2006)*.

McDonald, Ryan and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Empirical Methods in Natural Language Processing and Natural Language Learning (EMNLP-CoNLL 2007)*.

McDonald, Ryan and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)*.

McDonald, Ryan and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *International Conference on Parsing Technologies (IWPT 2007)*.

McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-2005)*.

McDonald, Ryan, Kiril Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorith. In *Human Language Technologies and Empirical Methods in Natural Language Processing HLT-EMNLP 2005*.

Meyers, Adam, Ruth Reeves, Catherine Macleod, Rachel Szekely, Veronica Zielinska, Brian Young, and Ralph Grishman. 2004. The NomBank project: An interim report. In *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*.

Musillo, Gabriele and Merlo Paola. 2006. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Europen Chapter of the Association for Computational Lingustics Workshop: Robust Methods in Analysis of Natural Language Data (EACL-2006)*.

Nivre, Joakim and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*.

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007a. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the 11th Conference on Computational Natural Language Learning (CoNLL-2007)*.

Nivre, Joakim, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007b. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*.

Nivre, Joakim. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*.

Palmer, Martha, Paul Kingsburry, and Daniel Gildea. 2006. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*.

Pradhan, Sameer, Kadri Hacioglu, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2005. Semantic role chunking combining complementary syntactic views. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*.

Punyakanok, Vasin, Dan Roth, Wen-tau Yih, and Dav Zimak. 2004. Semantic role labeling via integer linear programming inference. In *Proceedings of Coling 2004*.

Punyakanok, Vasin, Dan Roth, Wen-tau Yih, and Dav Zimak. 2005. Learning and inference over constrained output. In *In Proc. of the International Joint Conference on Artificial Intelligence (IJCAI-2005)*.

Ratnaparkhi, Adwait, Jeff Reynar, and Salim Roukos. 1994. A maximum entropy model for prepositional phrase attachment. *Journal of Linguistics*.

Rosenblatt, Frank. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*.

Schrijver, Alexander. 1998. *Theory of Linear and Integer Programming*. John Wiley & sons.

Shen, Libin and Aravind K. Joshi. 2005. Ranking and reranking with perceptron. *Machine Learning*.

Surdeanu, Mihai, Lluís Màrquez, Xavier Carreras, and Pere R. Comas. 2007. Combination strategies for semantic role labeling. *Journal of Artificial Intelligence Research*.

Surdeanu, Mihai, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the 12th Conference on Computational Natural Language Learning (CoNLL-2008)*.

Tarjan, Robert E. 1977. Finding optimum branchings. *Networks*.

Thompson, Cynthia A., Roger Levy, and Christopher D. Manning. 2003. A generative model for semantic role labeling. In *Proceedings of 14th European Conference on Machine Learning (ECML)*.

Titov, Ivan and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the International Conference on Parsing Technologies (IWPT-07)*.

Toutanova, Kristina, Aria Haghighi, and Christopher Manning. 2007. Joint learning improves semantic role labeling. In *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL-2005)*.

Xue, Nianwen and Martha Palmer. 2004. Calibrating features for semantic role labeling. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP-2004)*.

Appendix A

Descriptive data analysis

In this section we describe the data through some statistics. We will focus on relevant aspects for a joint processing point of view, as syntactic and semantic relations. This step will allow us to better understand the problem that we are intended to address. Only the *train corpus* and the *development corpus* are analyzed. The *test corpus* must only be used for a final evaluation, note that any information exploited from the test corpus will dismiss a fair evaluation and could biase our decisions towards an artificially higher score. The *development corpus* is intended for the tasks of system tuning and testing during the development phase. When applicable, statistics are drawn for both corpus to asses an even distribution across both of them.

Corpus size

An overview of the corpus is outilned in table A.1. The corpus is extracted from the WSJ and contains mainly financial information.

measure	**train**	**development**
sentences	39279	1334
num tokens	997446	34702
unique tokens	39782	6085
unique lemmas	28258	4611
forms not in train (%)	—	2.32263
lemmas not in train (%)	—	1.70884

Table A.1: Overview of the corpus

The syntactic parsing algorithm costs depends on the sentence size. Most parsers have a cost of $O(n^3)$ or $O(n)$, being n the size of the parsed sentence. Thus we analyze in detail the sentence lengths.

statistic	train	development
Min. :	2.00	3.00
1st Qu.:	17.00	17.00
Median :	24.00	25.00
Mean :	25.39	26.01
3rd Qu.:	32.00	33.00
Max. :	144.00	119.0

Table A.2: Sentence length

Table A.2 shows that most sentences are around 25 words. Some outliers could pose a cost problem to $O(n^3)$ parsers, but these extremely long sentences could be discarded as they represent a small fraction of the corpus.

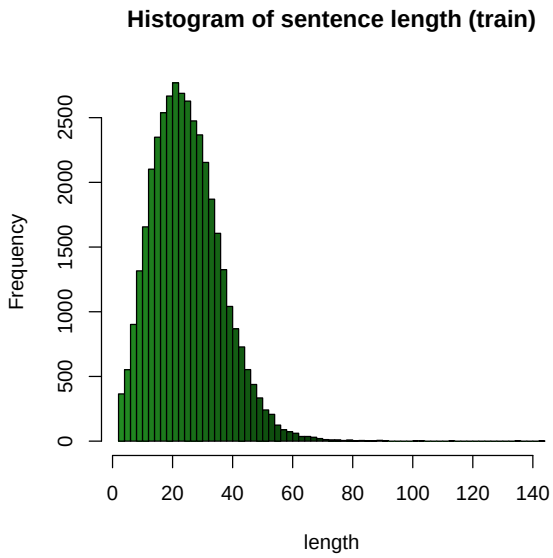

Figure A.1: Histogram of sentence length (train)

Figures A.1 and A.2 allows us to better understand the sentence size distribution and confirm that lengthy outliers can be safely discarded.

POS

The part of speech (POS) is the role that a word or phrase plays in a sentence. It represents an information intensively used as a feature for a wide variety of linguistic processing tasks. POS tags are predicted by state-of-the-art POS taggers (Surdeanu et al., 2008).

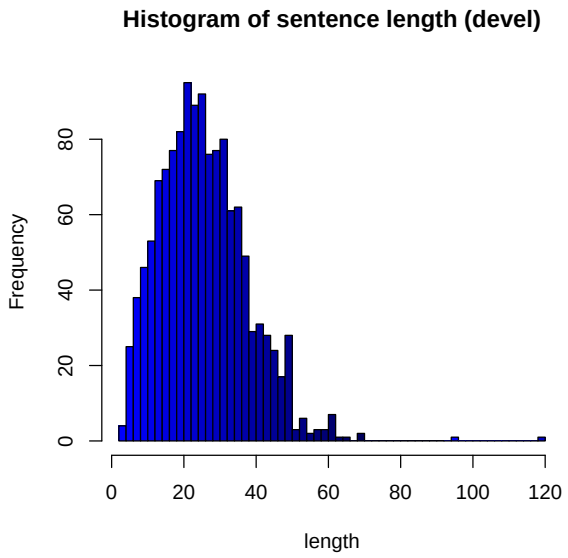

Figure A.2: Histogram of sentence length (devel)

POS tag	gold (counts)	predicted(counts)
NN	130595	143693
IN	96755	98988
NNP	89941	88044
DT	80449	82138
JJ	60084	52526
NNS	58718	62316
,	47931	47927
root	39279	39279
.	38918	38924
CD	36200	36974
RB	30499	28800
VBD	29438	29588
VB	25949	25514
CC	23515	23634
TO	21932	22022
VBZ	21360	21099
VBN	19677	22508
PRP	17181	17196
VBG	14554	15409
VBP	12270	12180
MD	9635	9651

POS	8487	8569
PRP$	8237	8263
$	7306	7306
“	6985	6985
’	6803	6804
:	4673	4734
WDT	4209	4321
JJR	3173	2248
NNPS	2636	404
RP	2609	2790
WP	2329	2456
WRB	2107	2217
JJS	1907	1421
RBR	1744	2874
)	1353	1352
(	1344	1344
EX	853	867
RBS	440	839
PDT	366	597
FW	232	0
WP$	159	159
#	142	142
UH	97	61
SYM	58	22
LS	36	17
PRF	0	233
HYPH	0	12011

Table A.3: POS of train corpus

Graphicaly the most frequent tags are plotted in figure A.3 and A.4.

Table A.3 shows a high number of predicted *HYPH* not appearing in the gold corpus. The reason is a finner tokenziation introduced in the data not present when gold tags were assigned. For example, *hand-crafted* is considered 1 token in the original annotation but 3 tokens ‘hand’,‘-’ and ‘crafted’ in the new annotation. Note that new hyphens are now tokens.

POS tag	**gold**	**predicted**
NN	4484	4984
IN	3560	3554
NNP	3170	3094
DT	2818	2875
NNS	2029	2183

JJ	2011	1843
,	1653	1653
.	1325	1325
CD	1039	1066
RB	1030	1047
VBD	1007	997
VB	1003	980
CC	863	861
TO	796	800
VBN	753	818
VBZ	695	682
PRP	535	538
VBG	521	518
VBP	418	422
MD	337	335
POS	298	303
PRP$	267	267
“	251	251
’	246	246
217	215	
$	210	210
NNPS	127	11
WDT	121	125
JJR	93	65
WP	89	94
RP	85	107
WRB	83	85
(	54	54
)	53	53
JJS	53	41
RBR	52	89
EX	37	37
PDT	22	24
RBS	20	36
SYM	10	0
FW	8	0
WP$	7	7
LS	5	5
UH	4	1
#	3	3
PRF	0	12
HYPH	0	452

Table A.4: POS of deve corpus

Table A.5 summarizes the accuracy of the POS taggers. Recall that gold POS is not available a test time.

	train (%)	**devel (%)**
All POS	93.6595	93.7122
Predicates	96.5237	96.6041
Noun preds	98.9675	99.2673
Verb preds	94.1331	94.0326

Table A.5: Accuracy of POS tagger

Table A.5 also accounts for the correct POS prediction restricted to noun and verb predicates. The POS tagging of this tokens is an important factor to correctly identify them as predicates. Not only nouns and verbs are adjectives. Figures A.5 and A.6 graphically show the distribution of the most frequent POS for predicates.

Verbs, nouns and adjectives can be predicates, the other POS tags of predicate tokens will probably be incorrect. This data points that we can directly discard some tokens by their POS in the predicate identification task.

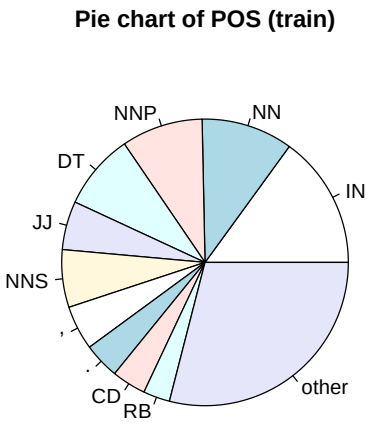

Figure A.3: Pie chart of POS distribution (train)

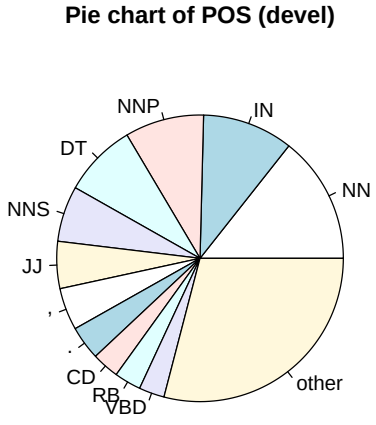

Figure A.4: Pie chart of POS distribution (train)

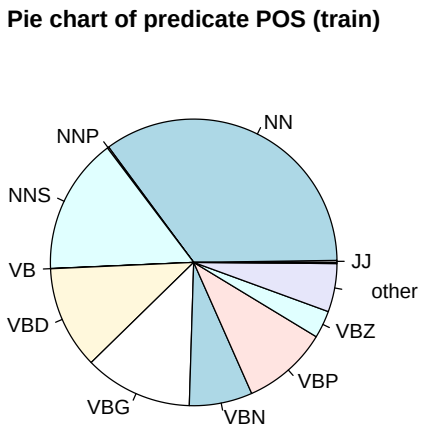

Figure A.5: Pie chart of POS distribution for predicates (train)

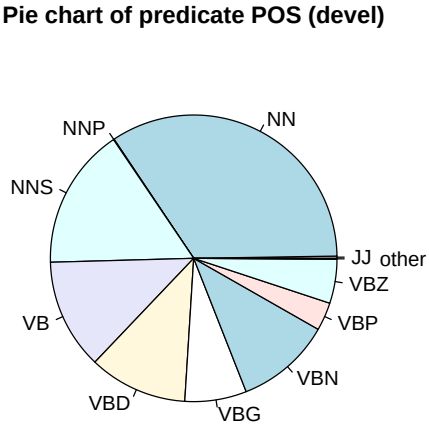

Figure A.6: Pie chart of POS distribution for predicates (devel)

A.1 Syntactic dependencies

The next data to analyze are the syntactic labels. Table A.6 and shows the distribution of the rich set of syntactic labels.

syntactic label	train (counts)
NMOD	257549
P	107278
PMOD	92413
SBJ	71254
OBJ	53219
ROOT	39279
ADV	37890
NAME	32415
VC	28134
COORD	24897
DEP	23811
TMP	20769
CONJ	19684
LOC	14682
AMOD	14159
PRD	12991
APPO	12847
IM	12756
HYPH	11084
HMOD	10982
SUB	10237
OPRD	9313
SUFFIX	8417
TITLE	5267
DIR	4982
POSTHON	4042
MNR	3884
PRP	3167
PRT	2593
LGS	2471
EXT	1951
PRN	1215
LOC-PRD	625
EXTR	547
DTV	403
PUT	218
GAP-SBJ	93

GAP-OBJ	90
DEP-GAP	75
GAP-PRD	58
GAP-TMP	52
PRD-TMP	44
GAP-LGS	40
PRD-PRP	39
BNF	34
GAP-LOC	32
DIR-GAP	30
LOC-OPRD	25
VOC	19
GAP-PMOD	18
ADV-GAP	14
EXT-GAP	14
GAP-VC	14
GAP-NMOD	12
DTV-GAP	6
AMOD-GAP	5
GAP-LOC-PRD	5
GAP-PRP	4
DIR-PRD	3
GAP-MNR	3
EXTR-GAP	3
MNR-PRD	2
LOC-TMP	2
GAP-OPRD	1
DIR-OPRD	1
LOC-MNR	1
GAP-SUB	1
GAP-PUT	1
MNR-TMP	1
EXT	1951

Table A.6: Syntactic labels (train)

Next table A.7 contains the distribution for the development set.

syntactic label	**development (counts)**
NMOD	8922
P	3760
PMOD	3263
SBJ	2407

OBJ	1728
ROOT	1334
ADV	1256
NAME	1138
VC	953
COORD	915
DEP	772
TMP	755
CONJ	706
LOC	556
AMOD	536
IM	512
PRD	509
APPO	444
HMOD	423
HYPH	421
OPRD	373
SUB	371
SUFFIX	295
TITLE	173
POSTHON	125
DIR	119
MNR	109
PRP	105
PRT	97
LGS	93
EXT	52
PRN	52
LOC-PRD	26
EXTR	15
DTV	14
PUT	11
GAP-SBJ	5
LOC-OPRD	4
GAP-TMP	4
VOC	2
GAP-VC	2
GAP-OBJ	2
GAP-LOC	2
DEP-GAP	2
GAP-LOC-PRD	1
DIR-PRD	1
PRD-TMP	1

BNF	1
DIR-GAP	1

Table A.7: Syntactic labels (devel)

Again a graphical distribution of the most frequent syntactic labels is plotted in figures A.7 and A.8.

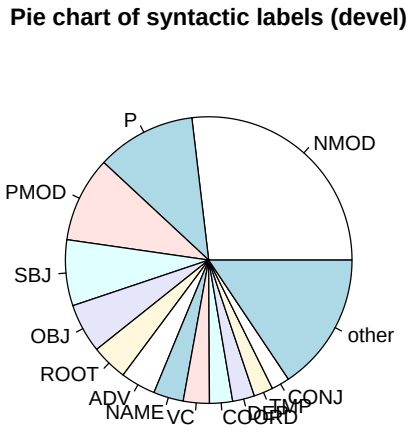

Figure A.7: Pie chart of syntactic labels (train)

Tables A.6 and A.7 pointed out that some really infrequent labels occurs in corpus. Therefore most classifiers will not be able to predict them due to the low number of associated examples. The removal of these labels should be considered

An important factor to predict syntactic labels is the distance, i.e., the number of tokens between the *head* and the *modifier* of the syntactic dependency (McDonald and Nivre, 2007).

Histograms in figure A.9 and A.10 shows very long, although rare, syntactic dependencies. Few syntactic dependencies are non-projective, the 0.4% of the training corpus dependencies. But these few dependencies are distributed among the 7.6% of the corpus sentences. See the definition of *projectivity* in section 3.1.

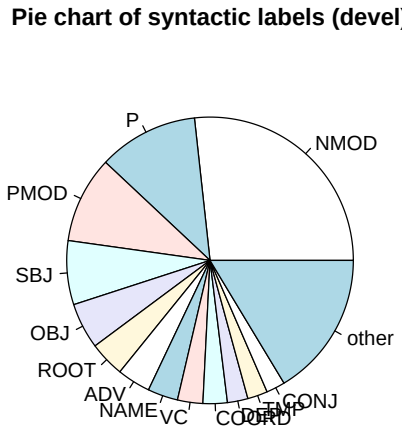

Figure A.8: Pie chart of syntactic labels (devel)

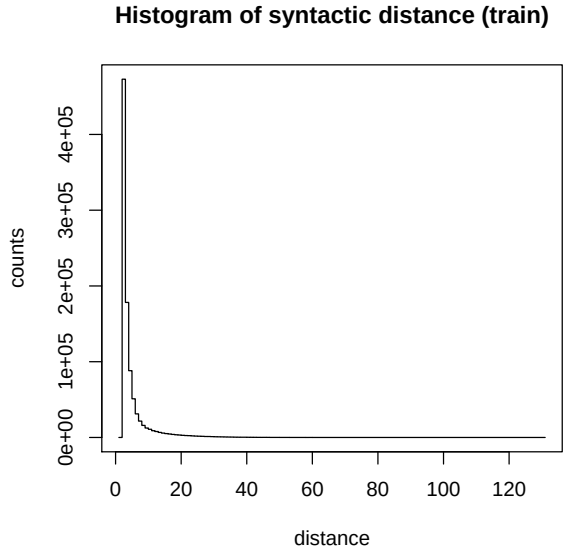

Figure A.9: Histogram of syntactic distance (train)

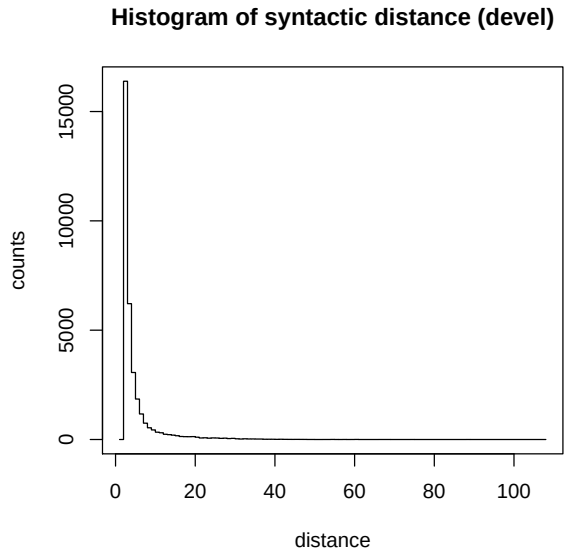

Figure A.10: Histogram of syntactic distance (devel)

A.2 Predicates

The identification of predicates is the second of the tree main task of the CoNLL-2008 shared task.

Table A.8 summarizes important statistics about the number of predicates per sentence.

statistic	train	devel
Min.	0.000	0.00
1st Qu.	3.000	3.00
Median	4.000	4.50
Mean	4.557	4.79
3rd Qu.	6.000	6.00
Max.	26.000	19.00

Table A.8: Predicates per sentence

In addition the distribution is plotted by histograms in figures A.11 and A.12.

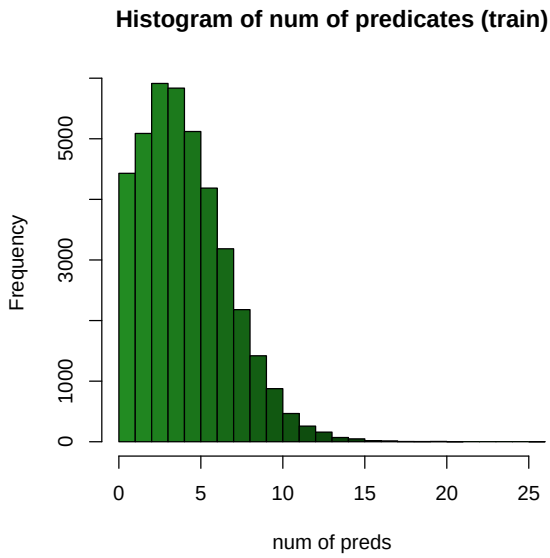

Figure A.11: Histogram of predicates per sentence (train)

A high number of predicates is found in this corpus, the reason is the tagging of nominal predicates for this shared task. Also the number of predicates is a factor that strongly determines the cost of the identification algorithms, see 5.8. We expect a relation between the number of predicates and sentence

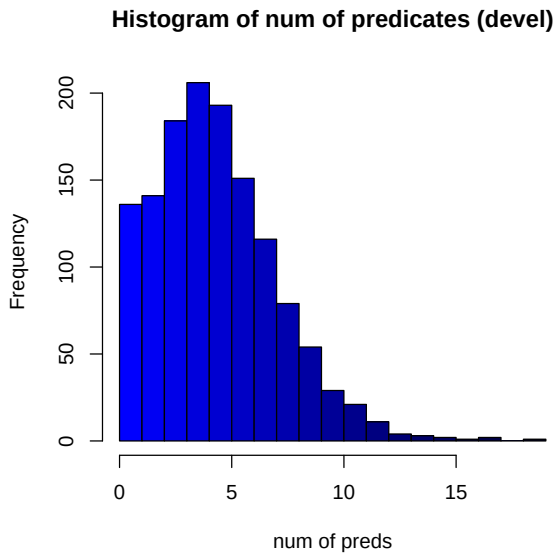

Figure A.12: Histogram of predicates per sentence (train)

length.

Figure A.13 and A.14 show a relation between the sentence length and the number of predicates. We can observe that as the sentence grows a minimum number of predicates is identified.

The next task is the predicate disambiguation, i.e., to assign a sense tag to each identified predicate.

sense	**train**			**devel**		
	global	nouns	verbs	global	nouns	verbs
1	86.58	90.45	82.69	85.92	88.96	82.92
2	8.12	6.26	10	9.36	7.73	10.96
3	3.27	1.99	4.55	2.68	1.8	3.54
4	0.85	0.66	1.05	0.97	0.69	1.24
5	0.4	0.31	0.48	0.27	0.19	0.34
6	0.25	0.21	0.28	0.39	0.44	0.34
7	0.09	0.04	0.14	0.06	0.06	0.06
8	0.08	0.02	0.13	0.11	0.03	0.19
9	0.05	0.02	0.08	0.02	0	0.03
10	0.06	0.01	0.12	0.02	0	0.03
11	0.09	0.03	0.14	0.11	0.06	0.16
12	0.06	0	0.12	0	0	0
13	0.03	0	0.05	0.03	0.03	0.03
14	0.03	0	0.05	0.02	0	0.03

15	0.03	0	0.05	0.02	0	0.03
16	0.02	0	0.03	0.02	0	0.03
17	0.01	0	0.01	0	0	0
18	0	0	0	0	0	0
19	0	0	0.01	0	0	0
20	0	0	0	0	0	0
21	0	0	0	0.03	0	0.06

Table A.9: Syntactic labels (devel)

Table A.9 shows that the most frequent sense is the '.01' sense. But the rule to assign the tag '.01' to the most frequency sense of each predicate is not always followed.

Another important feature to identify and disambiguate the predicate is the lemma. Recall that predicted lemmas are provided in the corpus.

Even for a simple strategy as the disambiguation using the most frequent sense, lemmas are important features. In this case, an incorrect lemma will cause the labeling of the most frequent sense of *another* predicate. For, the predicate *fall* is always incorrectly lemmatized as *fell*.

	train (%)	**devel (%)**
accuracy	95.9288	95.9311
no roleset	3.64608	5.05477

Table A.10: Lemmatizer accuracy on predicates

Table A.10 shows the high lemmatizer accuracy only for predicates. The lemmatizer extracts the lemma from the most common sense of token in WordNet. The *no roleset* entry measures the number of predicate lemmas without associate roleset in frames files due to an incorrect tagging.

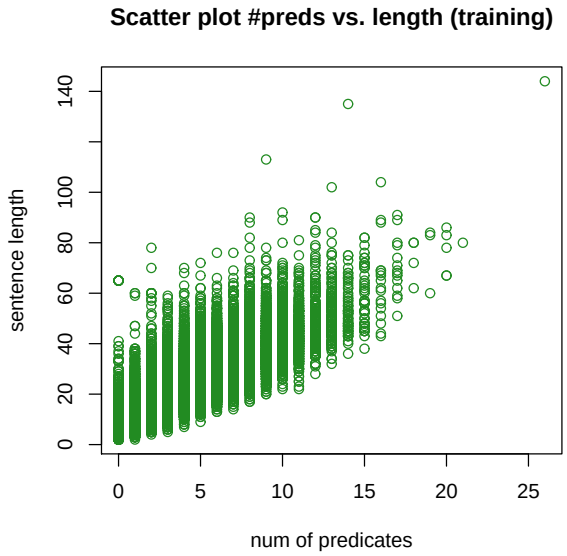

Figure A.13: Scatter plot of number of predicates vs. sentence length (train)

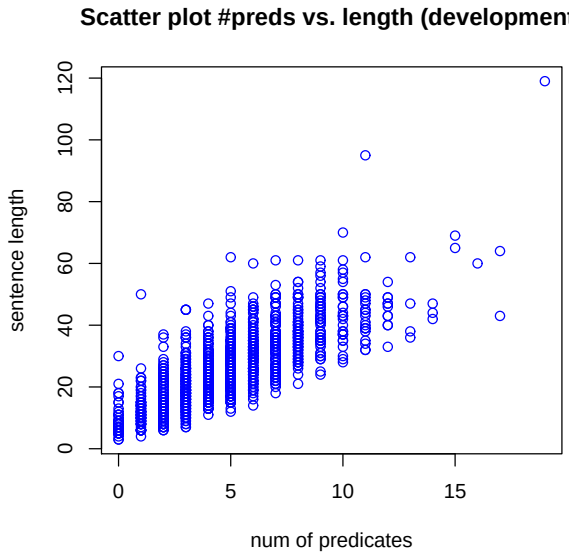

Figure A.14: Scatter plot of number of predicates vs. sentence length (devel)

A.3 Semantic dependencies

Finally we explore the data concerning semantic arguments.

measure	train (%)	devel (%)
verb predicates/noun predicates	49.8894	50.3912
verbs args / total args	60.7101	60.6996
noun args / total args	39.2899	39.3004

Table A.11: Semantic labels (train)

Table A.11 shows an even distribution of noun and verb predicates and a reasonably balanced argument distribution for nous and verbs.

We considered separate prediction tasks the classification of arguments for noun and verbs (i.e., an A0 for noun predicates and an A0 for verb predicates are classified by different classifiers), see section 5.4 for further discussion. We appended 'v' or 'n' to represent the arguments corresponding to noun and verb predicates.

semantic label	train (counts)	train (%)
A1v	83569	26.920
A1n	62981	20.288
A0n	38969	12.553
A2n	27148	8.745
A2v	19595	6.312
AM-TMPv	16089	5.183
AM-ADVv	8072	2.600
AM-MNRv	6243	2.011
A3n	6073	1.956
AM-LOCv	5833	1.879
AM-MNRn	5596	1.803
AM-DISv	4825	1.554
R-A0v	4039	1.301
A3v	3360	1.082
AM-NEGv	3173	1.022
C-A1v	2755	0.887
A4v	2688	0.866
R-A1v	2306	0.743
AM-PNCv	2232	0.719
AM-CAUv	1179	0.380
AM-DIRv	1125	0.362
R-AM-TMPv	705	0.227
AM-EXTv	622	0.200

A4n	378	0.122
R-A2v	287	0.092
A5v	68	0.022
AM-PRDv	67	0.022
C-A0v	65	0.021
C-A2v	62	0.020
AM-CAUn	43	0.014
R-AM-CAUv	42	0.014
C-A3v	36	0.012
R-A3v	29	0.009
C-AM-ADVv	21	0.007
C-AM-MNRv	21	0.007
A5n	16	0.005
AAv	14	0.005
R-AM-PNCv	13	0.004
C-AM-EXTv	12	0.004
C-A4v	12	0.004
C-AM-TMPv	11	0.004
C-AM-LOCv	10	0.003
R-A4v	8	0.003
AMv	8	0.003
C-AM-CAUv	7	0.002
R-AM-EXTv	5	0.002
R-AAv	3	0.001
AM-PRTv	3	0.001
AM-TMv	3	0.001
AM-MODn	3	0.001
C-R-AM-TMPv	2	0.001
C-AM-NEGv	2	0.001
R-AM-DIRv	2	0.001

Table A.12: Semantic labels (train)

Table A.13 regards to semantic labels on the development set.

semantic label	**devel (counts)**	**devel (%)**
A1v	2972	21.366
A1n	2252	16.190
A0v	2061	14.817
A0n	1387	9.971
A2n	928	6.671
A2v	669	4.809
AM-TMPv	595	4.277

AM-MODv	316	2.272
AM-TMPn	285	2.049
AM-ADVv	278	1.999
AM-MNRv	238	1.711
AM-DISv	203	1.459
AM-MNRn	193	1.387
AM-LOCv	192	1.380
A3n	191	1.373
AM-LOCn	151	1.086
R-A0v	145	1.042
C-A1v	140	1.006
A3v	113	0.812
AM-NEGv	105	0.755
R-A1v	83	0.597
AM-PNCv	81	0.582
A4v	66	0.474
AM-CAUv	46	0.331
AM-DIRv	35	0.252
R-AM-TMPv	32	0.230
AM-EXTv	29	0.208
A4n	20	0.144
AM-EXTn	20	0.144
AM-NEGn	20	0.144
AM-ADVn	11	0.079
R-AM-LOCv	10	0.072
R-AM-MNRv	7	0.050
R-A2v	6	0.043
R-AM-CAUv	4	0.029
C-A2v	4	0.029
AM-PRDv	4	0.029
C-AM-MNRv	3	0.022
A5v	3	0.022
AM-CAUn	2	0.014
C-AM-DIRv	2	0.014
A5n	2	0.014
C-AM-CAUv	2	0.014
AAv	2	0.014
R-AM-EXTv	2	0.014

Table A.13: Semantic labels (devel)

Tables A.12 and A.13 clearly points the core arguments (A0-A5) as the most frequent argument excluding A3-A5. A large number of argument labels

are really infrequent thus hard to classify by most machine learning methods. Some of these rare labels are in fact continuations, references or combination of other labels. A research direction to explore is to exploit this information.

Now we review the semantic dependency distance between arguments and their predicates.

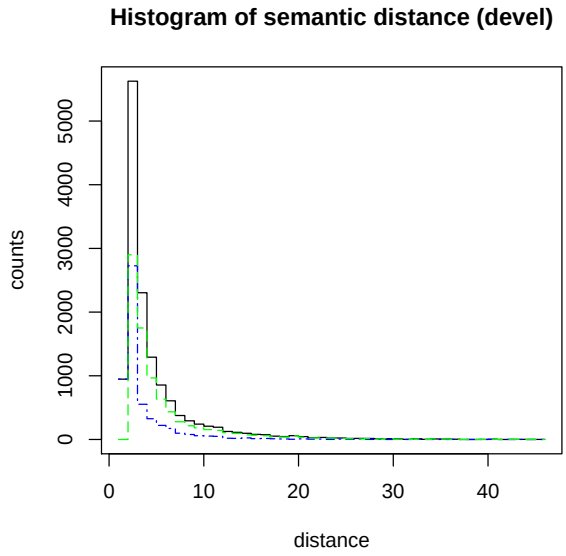

Figure A.15: Histogram of semantic distance (devel)

In figure A.15 the solid black line represents the histogram of the distance between the semantic arguments and their predicates. The histogram for noun arguments is plotted in dashed-doted lines. Verb argument distance in dashed lines. We can appreciate that the noun arguments can receive semantic dependencies to themselves, see the counts in figure for a 0 distance.

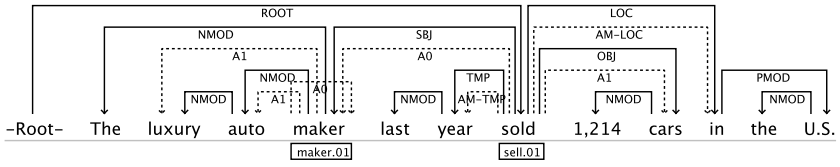

Figure A.16: A self semantic dependency link

Example A.3.0.1 Self reference

Figure A.16 contains a semantic dependency between *maker* and itself. Note that this situation only occurs for nominal predicates.

Syntactic and semantic overlap

The relations between syntax and semantics are of special interest for us as our joint system should exploit them. A high degree of overlap, i.e., syntactic and semantic dependencies relating the same tokens, will facilitate a joint processing of this two kinds of dependencies.

pred is	**train**			**devel**		
	global (%)	verb (%)	noun (%)	global (%)	verb (%)	noun (%)
head	63.629	66.211	59.640	62.610	64.365	59.900
self	6.659	0.00209	16.945	6.830	0	17.379
ancestor	8.656	11.029	4.990	9.231	11.680	5.450
descendant	1.120	0.7556	1.683	1.190	0.9624	1.541
direct son	5.333	7.734	1.623	5.769	0.7604	1.890
grandfather	0.9248	0.6351	1.372	0.95203	0	1.247
other	19.934	22.002	16.739	20.137	22.9919	15.727

Table A.14: Syntactic-semantic overlapping

One may think that the predicate of an argument is always the syntactic head. Table A.14 shows the wide variety of syntactic relations in semantic links. Note that some semantic dependencies relates a token with its direct son, thus this semantic dependencies will overlap with syntax but will appear as *reversed*.

In a bottom-up parsing processing of the sentence, only relations inside the current substree are accessible. The relations considered in the table as *other* and *ancestor*, excluding *head* are not visible during a bottom up parsing.

As we previously commented, verb arguments never have semantic self-dependencies, but table A.14 accounts for a small degree (0.002) of self dependencies in verb arguments. The table was computed in a realistic manner from the predicted POS. This implies that if a nominal predicate is incorrectly tagged with a verbal POS is considered as a verbal predicate.

	train			**devel**		
	global (%)	verb (%)	noun (%)	global (%)	verb (%)	noun (%)
is core	77.5349	70.9976	87.6361	76.8265	69.8432	87.6124
core no head	29.5401	23.2768	39.218	30.3498	24.6554	39.1448

Table A.15: Syntactic-semantic overlapping

Table A.15 shows the core arguments vs. the core arguments that occupies an non-overlapping relation with syntax *core no head*. We can see that

most(63.6%), but not all, core arguments occurs in overlapping positions. We cannot restrict the labelling of core arguments to overlapping dependencies.

In previous sections we have seen histograms of the *surface* distance about syntactic and semantic dependencies. We now review the semantic distance but measured in the syntactic graph.

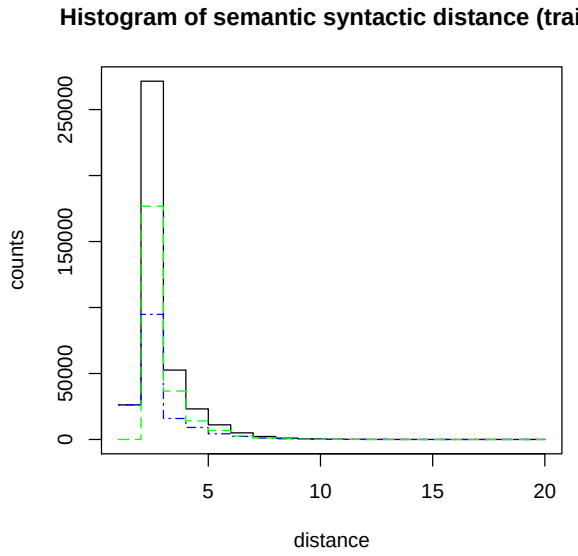

Figure A.17: Histogram of semantic distance (devel)

Figures A.17 and A.18 proves that the simultaneous prediction of syntactic and semantic dependencies between two tokens is not possible for a significant amount of dependencies. This point will significantly increase the difficulty in the design and development of a joint model. With these statistics we conclude the initial data exploration.

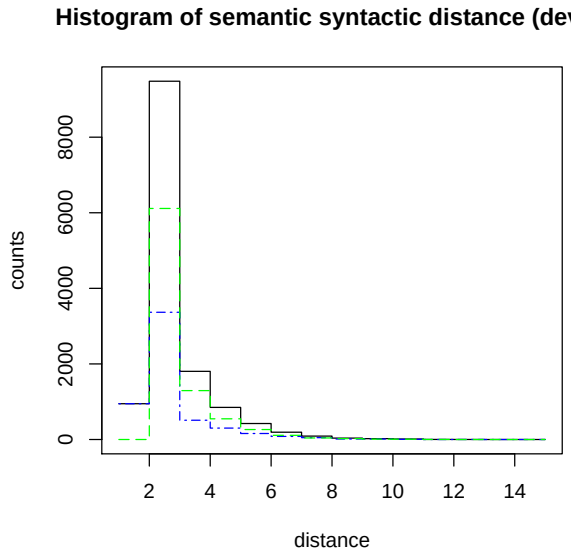

Figure A.18: Histogram of semantic distance (devel)

Appendix B

Data structures and files

In this appendix we explore the data managed by our implemented model. It complements the previous efficiency section 5.11.

The appropriate structures for data are selected given the performance requirements. For example, filters were implemented as vectors. The syntactic tree structure originally implemented as a vector was implemented in a adjacency list. This new implementation allowed a reduction from $O(n)$ to a near constant time for operations related to feature extraction.

The table B.1 contains a description of the datafiles used.

File Name	description	type	format	size
inputCorpus	The training corpus	text	CoNLL	52MB
inputTest	The development corpus	text	CoNLL	1.8MB
goldTest	The development corpus with gold arguments and predicates	text	CoNLL	1.8MB
outputSyntax	The development corpus with predicted syntax	text	CoNLL	1.4MB
outputPreds	The development corpus with syntax and predicates identified	text	CoNLL	1.4MB
output	The final output	text	CoNLL	1.4MB
secondBestFileSemantic	The best final output and the second best	text	CoNLL extended	3MB
preprocessedCorpus.gz	The extracted features for syntactic parsing from the Training corpus	binary	preproc	1.2BG
preprocessedCorpus-Predicates.gz	The extracted features for predicate identification from the corpus	binary	preproc	1GB

preprocessedCorpus-Semantics	The extracted features for semantic parsing from the Training corpus	binary	preproc	71.3GB
preprocessedCorpus-Semantics.gz1-10	Same previous file splitted in 10 files	binary	preproc	10×0.9GB
preprocessedTest.gz	The extracted features for syntactic parsing from the test corpus	binary	preproc	44.3MB
preprocessedTest-Predicates.gz	The extracted features for predicate identification from the test corpus	binary	preproc	7.3MB
preprocessedTest-Semantics.gz	The extracted features for semantic parsing from the test corpus	binary	preproc	342MB
synLabels	The set of syntactic labels with their assigned id	binary	dictionary string-int	<1KB
semLabels	The set of syntactic labels with their assigned id	binary	dictionary string-int	1.1KB
semLabelsNouns	The set of syntactic labels with their assigned id for nominal predicates	binary	dictionary string-int	3.8KB
semLabelsVerbs	The set of syntactic labels with their assigned id for verbal predicates	binary	dictionary string-int	3.8KB
synTagSelected	The set of syntactic labels filtered by frequency	binary	dictionary string-bool	<1KB
semTagSelected	The set of semantic labels filtered by frequency	binary	dictionary string-bool	<1KB
synFilter	The set of syntactic labels for each POS-POS combination	binary	filter by POS-POS	160KB
semFilter	The set of semantic labels for each POS-POS combination	binary	filter by POS-POS	180KB
semFilterByLemma-Splitted	The set of syntactic labels with their assigned id extracted from semLabels-Nouns/verbs	binary	filter by lem-maid	10MB
posIntMap	The dictionary to transform POS into integers	binary	dictionary string-int	1KB

lemmaIntMap	The dictionary to transform predicate lemmas into integers	binary	dictionary string-int	120KB
similarLemma	The list of lemmas that can be confused or erroneously lemmatized	binary	dictionary string-list of int	120KB
predFreq	The frequency for each predicate sense	binary	list of senses	157KB
ambiguousLemmas	The lemmas present in corpus that could be predicates with a certain degree of certainty	binary	list of strings	73KB
notAmbiguousLemmas	The lemmas present in corpus that could be predicates with a certain degree of certainty	binary	list of strings	4.5KB
featureMap	The list of selected features for syntactic parsing	binary	dictionary string-int	3.5MB
dynamicFeatureMap	The list of selected dynamic features for syntactic parsing	binary	dictionary string-int	8.2KB
featureMapSemantic	The list of selected features for semantic parsing	binary	dictionary string-int	1.1MB
modelSyntactic	The averaged perceptron weight vector for syntactic parsing	binary	int vector	95 MB
modelPredicates	The averaged perceptron weight vector for predicate identification	binary	int vector	55 MB
modelSemantic	The averaged perceptron weight vector for joint parsing	binary	int vector	232 MB
backup	The averaged weight vector and the weight vector file for resume the training if it is stopped	binary	backup	460 MB
tmpKeywords	List of keywords related to *AM-TMP* arguments	binary	list of string	10KB
bow	List of the most frequent words extracted from corpus (only nouns, verbs and adjectives)	list of string		5KB

errorFile	The trace of the errors made at each training sentence	Text	list of doubles	200MB
predList	The list of predicates extracted from the XML frame files	binary	list of int	74KB

Table B.1: List of data saved in files

File formats:

CoNLL Previous CoNLL shared task formats inspired the present year format but are not compatible.

CoNLL extended This file contains extra columns with the second best output and scores for each argument.

Dictionary Pairs of strings and integers sequentially recorded.

The preprocessed files are the largest files employed. They cointain all features for all combinations of head and modifier tokens. Note that a lot of features for a given *head-modifier* dependency will be shared with other dependencies with the same *head* or *modifier*. The redundancy is even greater in the semantic corpus file. As this file contains features for each *head-modifier-predicate*. We used a compressor library to deal with this files. Unfortunately the library was unable to deal with the huge files we had and we splitted the files into a set of chunks.

Although different files can share the same format it does not means that the data structure read is the same. In addition the same information contained in a given data file can be used differently depending on the actual processing phase. E.g., the list of syntactic labels is loaded as a dictionary(hash-table) when we need to index the labels and it is loaded as a vector we we need to generate an output.

Appendix C

Detailed Scores of CoNLL-2008 participants

This are the results of the systems for the closed challenge[1]. For a detailed explanation of each scoring metric see 2.2.

C.1 Labeled Macro F_1 score

Group	Name	WSJ + Brown	WSJ	Brown
Lund (*)	Johansson (*)	85.49	86.61	76.34
Yahoo! (*)	Ciaramita (*)	82.69	83.83	73.51
HIT-IR	Che	82.66	83.78	73.57
Hong Kong (*)	Zhao (*)	82.24	83.41	72.70
Geneva (*)	Henderson (*)	80.48	81.53	71.93
Koc	Yuret	79.84	80.97	70.55
GSLT	ML2 Samuelsson	79.79	80.92	70.49
DFKI 2	Zhang	79.32	80.41	70.48
NAIST	Watanabe	79.1	80.3	69.29
Antwerp	Morante	78.43	79.52	69.55
HIT-ICR	Li	78.35	79.38	70.01
UPC (*)	Lluís (*)	78.11	79.16	69.84
UT Austin	Baldridge	77.49	78.57	68.53
Koc	Yatbaz	77.45	78.43	69.61
USTC	Chen	77	77.95	69.23
Korea	Lee	76.9	77.96	68.34
Peking	Sun	76.28	77.1	69.58
Colorado	Choi	71.23	72.22	63.44

[1]This data was retrieved from the CoNLL-2008 Shared Task web site, see `http://www.yr-bcn.es/conll2008`

UAIC	Trandabat	63.45	64.21	57.41
DFKI 1	Neumann	19.93	20.13	18.14

Table C.1: Labeled Macro F_1 score

C.2 Exact match for syntax and semantics

Group	Name	WSJ + Brown	WSJ	Brown
Lund (*)	Johansson (*)	13.10	13.12	13.15
HIT-IR	Che	10.37	10.21	11.5
Hong Kong (*)	Zhao (*)	9.98	9.92	10.56
Yahoo! (*)	Ciaramita (*)	9.88	9.62	11.50
Geneva (*)	Henderson (*)	9.52	9.38	10.56
NAIST	Watanabe	7.79	7.54	9.39
Koc	Yuret	7.65	7.33	9.62
DFKI 2	Zhang	7.4	7.46	7.28
HIT-ICR	Li	7.12	6.71	9.62
GSLT ML2	Samuelsson	6.94	6.62	8.92
USTC	Chen	6.83	6.46	9.15
Korea	Lee	6.69	6.29	9.15
Antwerp	Morante	6.44	6.04	8.92
UPC (*)	Lluís (*)	6.23	5.79	8.92
Koc	Yatbaz	5.91	5.29	9.62
Peking	Sun	5.38	4.96	7.98
UT Austin	Baldridge	5.24	4.92	7.28
Colorado	Choi	3.33	3.5	2.58
UAIC	Trandabat	3.26	3.08	4.46
DFKI 1	Neumann	0.11	0.12	0.23

Table C.2: Exact match for syntax and semantics

C.3 Labeled attachment score of syntactic depedencies

Group	Name	WSJ + Brown	WSJ	Brown
Lund (*)	Johansson (*)	89.32	90.13	82.84
Hong Kong (*)	Zhao (*)	87.68	88.51	80.99
Geneva (*)	Henderson (*)	87.64	88.46	81.05
Yahoo! (*)	Ciaramita (*)	87.37	88.21	80.60
DFKI 2	Zhang	87.32	88.14	80.8

NAIST	Watanabe	87.18	88.06	80.17
HIT-IR	Che	86.75	87.51	80.73
HIT-ICR	Li	86.69	87.42	80.8
UT Austin	Baldridge	86.67	87.42	80.64
GSLT ML2	Samuelsson	86.63	87.36	80.77
Koc	Yatbaz	86.62	87.39	80.46
Koc	Yuret	86.62	87.39	80.46
Antwerp	Morante	86.07	86.88	79.58
UPC (*)	Lluís (*)	85.84	86.54	80.24
Peking	Sun	85.75	86.37	80.75
UAIC	Trandabat	85.21	85.96	79.24
Korea	Lee	84.82	85.69	77.83
USTC	Chen	84.47	85.2	78.58
Colorado	Choi	77.56	78.58	69.46
DFKI 1	Neumann	16.25	16.22	16.47

Table C.3: Labeled attachment score of syntactic depedencies

C.4 Labeled F_1 semantic score

Group	**Name**	**WSJ + Brown**	**WSJ**	**Brown**
Lund (*)	Johansson (*)	81.65	83.09	69.85
HIT-IR	Che	78.52	80	66.37
Yahoo! (*)	Ciaramita (*)	78	79.43	66.41
Hong Kong (*)	Zhao (*)	76.75	78.25	64.35
Geneva (*)	Henderson (*)	73.09	74.36	62.56
Koc	Yuret	73.06	74.54	60.62
GSLT ML2	Samuelsson	72.94	74.47	60.18
DFKI 2	Zhang	71.31	72.67	60.16
NAIST	Watanabe	70.84	72.37	58.21
Antwerp	Morante	70.51	71.88	59.23
UPC (*)	Lluís (*)	70.35	71.74	59.42
HIT-ICR	Li	69.95	71.27	59.17
USTC	Chen	69.45	70.62	59.81
Korea	Lee	68.71	69.95	58.63
Koc	Yatbaz	68.26	69.44	58.76
UT Austin	Baldridge	67.92	69.35	55.95
Peking	Sun	66.61	67.62	58.26
Colorado	Choi	64.78	65.72	57.4
UAIC	Trandabat	40.63	41.36	34.75
DFKI 1	Neumann	22.36	22.86	17.94

Table C.4: Labeled F_1 semantic score

C.5 Semantic perfect propositions

Group	Name	WSJ + Brown	WSJ	Brown
Lund (*)	Johansson (*)	55.15	57.34	37.11
HIT-IR	Che	48.05	50.15	30.9
Yahoo! (*)	Ciaramita (*)	47.36	49.25	31.93
Hong Kong (*)	Zhao (*)	44.05	45.98	27.88
Geneva (*)	Henderson (*)	41.47	42.97	28.87
NAIST	Watanabe	36.44	38.09	22.72
GSLT ML2	Samuelsson	35.2	36.96	20.22
DFKI 2	Zhang	34.96	36.25	24.22
Koc	Yuret	34.61	36.13	21.78
UPC (*)	Lluís (*)	32.90	34.32	21.75
HIT-ICR	Li	32.08	33.45	20.62
Korea	Lee	31.4	32.52	22.18
USTC	Chen	31.02	32.08	22.14
Peking	Sun	30.43	31.51	21.4
Antwerp	Morante	30.41	31.97	17.49
Koc	Yatbaz	26.14	27.01	18.81
UT Austin	Baldridge	25.35	26.57	15.26
Colorado	Choi	24.77	25.71	17.37
UAIC	Trandabat	6.59	6.81	4.76
DFKI 1	Neumann	0.3	0.31	0.2

Table C.5: Semantic perfect propositions

Appendix D

Detailed System Results

Contents

D.1 Development

Evaluation metric	detail	result
SYNTACTIC SCORES:		
Labeled attachment score:	28203 / 33368	84.52 %
Unlabeled attachment score:	29371 / 33368	88.02 %
Label accuracy score:	30112 / 33368	90.24 %
Exact syntactic match:	223 / 1335	16.70 %
SEMANTIC SCORES:		
Labeled precision:	(8585 + 5099) / (11994 + 6414)	74.34 %
Labeled recall:	(8585 + 5099) / (13865 + 6390)	67.56 %
Labeled F1:		70.79
Unlabeled precision:	(9922 + 5726) / (11994 + 6414)	85.01 %
Unlabeled recall:	(9922 + 5726) / (13865 + 6390)	77.25 %
Unlabeled F1:		80.95
Proposition precision:	2132 / 6414	33.24 %

Proposition recall:	2132 / 6390	33.36 %
Proposition F1:		33.30
Exact semantic match:	77 / 1335	5.77 %
OVERALL MACRO SCORES (Wsem = 0.50):		
Labeled macro precision:		79.43 %
Labeled macro recall:		76.04 %
Labeled macro F1:		77.70 %
Unlabeled macro precision:		86.51 %
Unlabeled macro recall:		82.64 %
Unlabeled macro F1:		84.53 %
Exact overall match:	40 / 1335	3.00 %
OVERALL MICRO SCORES:		
Labeled micro precision:	(28203 + 8585 + 5099) / (33368 + 11994 + 6414)	80.90 %
Labeled micro recall:	(28203 + 8585 + 5099) / (33368 + 13865 + 6390)	78.11 %
Labeled micro F1:		79.48
Unlabeled micro precision:	(29371 + 9922 + 5726) / (33368 + 11994 + 6414)	86.95 %
Unlabeled micro recall:	(29371 + 9922 + 5726) / (33368 + 13865 + 6390)	83.95 %
Unlabeled micro F1:		85.43

Table D.1: Global development score

D.1.1 Syntactic depedencies

Syn label	gold	correct	system	recall (%)	precision (%)
ADV	1256	879	1338	69.98	65.70
AMOD	536	334	452	62.31	73.89
APPO	444	299	439	67.34	68.11
BNF	1	0	1	0.00	0.00
CONJ	706	612	730	86.69	83.84
COORD	915	597	842	65.25	70.90
DEP	772	591	724	76.55	81.63
DEP-GAP	2	0	0	0.00	NaN
DIR	119	68	97	57.14	70.10
DIR-GAP	1	0	0	0.00	NaN
DIR-PRD	1	0	0	0.00	NaN
DTV	14	12	13	85.71	92.31

EXT	52	40	47	76.92	85.11
EXTR	15	9	15	60.00	60.00
GAP-LOC	2	0	0	0.00	NaN
GAP-LOC-PRD	1	0	0	0.00	NaN
GAP-OBJ	2	0	1	0.00	0.00
GAP-SBJ	5	0	1	0.00	0.00
GAP-TMP	4	0	1	0.00	0.00
GAP-VC	2	0	2	0.00	0.00
HMOD	423	403	433	95.27	93.07
HYPH	421	421	437	100.00	96.34
IM	512	507	510	99.02	99.41
LGS	93	80	95	86.02	84.21
LOC	556	327	549	58.81	59.56
LOC-OPRD	4	0	0	0.00	NaN
LOC-PRD	26	15	30	57.69	50.00
MNR	109	54	110	49.54	49.09
NAME	1138	837	1024	73.55	81.74
NMOD	8922	8077	9048	90.53	89.27
OBJ	1728	1550	1774	89.70	87.37
OPRD	373	307	359	82.31	85.52
P	3760	2892	3771	76.91	76.69
PMOD	3263	3016	3334	92.43	90.46
POSTHON	125	109	127	87.20	85.83
PRD	509	416	487	81.73	85.42
PRD-TMP	1	0	1	0.00	0.00
PRN	52	9	42	17.31	21.43
PRP	105	65	108	61.90	60.19
PRT	97	80	99	82.47	80.81
PUT	11	8	10	72.73	80.00
ROOT	1334	1244	1334	93.25	93.25
SBJ	2407	2207	2471	91.69	89.32
SUB	371	328	376	88.41	87.23
SUFFIX	295	289	301	97.97	96.01
TITLE	173	138	140	79.77	98.57
TMP	755	460	707	60.93	65.06
VC	953	923	988	96.85	93.42
VOC	2	0	0	0.00	NaN

Table D.2: Precision and recall for syntactic depedencies

D.1.2 Predicate identification

PPOSS	**gold**	**correct**	**system**	**recall (%)**	**precision (%)**	**F_1**

CD	1	0	0	0.00	NaN	NaN
JJ	16	0	0	0.00	NaN	NaN
NN	2181	1721	2024	78.91	85.03	81.86
NNP	5	0	0	0.00	NaN	NaN
NNS	1021	848	1038	83.06	81.70	82.37
PRF	1	0	0	0.00	NaN	NaN
RB	1	0	0	0.00	NaN	NaN
RP	3	1	1	33.33	100.00	50.00
VB	796	793	804	99.62	98.63	99.12
VBD	710	710	738	100.00	96.21	98.07
VBG	445	445	492	100.00	90.45	94.99
VBN	689	688	718	99.85	95.82	97.79
VBP	203	203	245	100.00	82.86	90.63
VBZ	317	317	354	100.00	89.55	94.49
WP	1	0	0	0.00	NaN	NaN

Table D.3: Precision and recall for predicate identification

D.1.3 Predicate classification

PPOSS	**gold**	**correct**	**system**	**recall (%)**	**precision (%)**	**F_1**
CD	1	0	0	0.00	NaN	NaN
JJ	16	0	0	0.00	NaN	NaN
NN	2181	1549	2024	71.02	76.53	73.67
NNP	5	0	0	0.00	NaN	NaN
NNS	1021	783	1038	76.69	75.43	76.05
PRF	1	0	0	0.00	NaN	NaN
RB	1	0	0	0.00	NaN	NaN
RP	3	1	1	33.33	100.00	50.00
VB	796	675	804	84.80	83.96	84.38
VBD	710	629	738	88.59	85.23	86.88
VBG	445	382	492	85.84	77.64	81.53
VBN	689	611	718	88.68	85.10	86.85
VBP	203	182	245	89.66	74.29	81.25
VBZ	317	287	354	90.54	81.07	85.54
WP	1	0	0	0.00	NaN	NaN

Table D.4: Precision and recall for predicate classification

D.1.4 Precision and recall for semantic dependencies

PPOSS + ARG	gold	correct	system	recall (%)	precision (%)	F_1
CD* + A1	1	0	0	0.00	NaN	NaN
JJ* + A0	5	0	0	0.00	NaN	NaN
JJ* + A1	15	0	0	0.00	NaN	NaN
JJ* + A2	1	0	0	0.00	NaN	NaN
JJ* + AM-EXT	1	0	0	0.00	NaN	NaN
JJ* + AM-MNR	1	0	0	0.00	NaN	NaN
JJ* + AM-MOD	1	0	0	0.00	NaN	NaN
JJ* + AM-TMP	2	0	0	0.00	NaN	NaN
NN* + A0	1419	542	785	38.20	69.04	49.19
NN* + A1	2278	1282	1961	56.28	65.37	60.49
NN* + A2	934	347	537	37.15	64.62	47.18
NN* + A3	192	87	135	45.31	64.44	53.21
NN* + A4	19	3	6	15.79	50.00	24.00
NN* + A5	1	0	0	0.00	NaN	NaN
NN* + AM-ADV	17	0	5	0.00	0.00	NaN
NN* + AM-CAU	3	0	0	0.00	NaN	NaN
NN* + AM-DIR	3	0	0	0.00	NaN	NaN
NN* + AM-DIS	5	0	0	0.00	NaN	NaN
NN* + AM-EXT	18	6	13	33.33	46.15	38.71
NN* + AM-LOC	154	64	139	41.56	46.04	43.69
NN* + AM-MNR	198	75	146	37.88	51.37	43.61
NN* + AM-MOD	4	0	0	0.00	NaN	NaN
NN* + AM-NEG	19	6	6	31.58	100.00	48.00
NN* + AM-TMP	290	162	228	55.86	71.05	62.55
NN* + R-A0	2	0	0	0.00	NaN	NaN
NN* + R-A1	2	0	0	0.00	NaN	NaN
NN* + R-AM-TMP	2	0	0	0.00	NaN	NaN
PR* + A0	1	0	0	0.00	NaN	NaN
PR* + A1	1	0	0	0.00	NaN	NaN
RB* + A0	1	0	0	0.00	NaN	NaN
RB* + A1	1	0	0	0.00	NaN	NaN
RB* + AM-DIS	1	0	0	0.00	NaN	NaN
RP* + A1	2	1	1	50.00	100.00	66.67
RP* + AM-TMP	1	0	0	0.00	NaN	NaN
VB* + A0	2019	1498	1941	74.20	77.18	75.66
VB* + A1	2924	2459	3043	84.10	80.81	82.42
VB* + A2	660	397	607	60.15	65.40	62.67
VB* + A3	110	54	95	49.09	56.84	52.68
VB* + A4	65	45	60	69.23	75.00	72.00
VB* + A5	2	0	0	0.00	NaN	NaN
VB* + AA	1	0	0	0.00	NaN	NaN
VB* + AM-ADV	270	125	256	46.30	48.83	47.53
VB* + AM-CAU	43	30	36	69.77	83.33	75.95
VB* + AM-DIR	31	10	22	32.26	45.45	37.74
VB* + AM-DIS	196	118	171	60.20	69.01	64.30
VB* + AM-EXT	28	11	20	39.29	55.00	45.84
VB* + AM-LOC	187	119	225	63.64	52.89	57.77
VB* + AM-MNR	230	100	195	43.48	51.28	47.06
VB* + AM-MOD	309	271	283	87.70	95.76	91.55
VB* + AM-NEG	104	89	125	85.58	71.20	77.73
VB* + AM-PNC	80	36	59	45.00	61.02	51.80
VB* + AM-PRD	3	0	1	0.00	0.00	NaN
VB* + AM-TMP	585	383	541	65.47	70.79	68.03
VB* + C-A1	139	87	128	62.59	67.97	65.17
VB* + C-A2	3	0	0	0.00	NaN	NaN
VB* + C-AM-CAU	1	0	0	0.00	NaN	NaN

VB* + C-AM-DIR	1	0	0	0.00	NaN	NaN
VB* + C-AM-DIS	0	0	2	NaN	0.00	NaN
VB* + C-AM-EXT	0	0	1	NaN	0.00	NaN
VB* + C-AM-MNR	2	0	1	0.00	0.00	NaN
VB* + R-A0	142	118	133	83.10	88.72	85.82
VB* + R-A1	79	47	62	59.49	75.81	66.67
VB* + R-A2	5	2	2	40.00	100.00	57.14
VB* + R-AM-CAU	3	0	1	0.00	0.00	NaN
VB* + R-AM-EXT	1	0	1	0.00	0.00	NaN
VB* + R-AM-LOC	9	1	2	11.11	50.00	18.18
VB* + R-AM-MNR	6	0	1	0.00	0.00	NaN
VB* + R-AM-TMP	29	10	18	34.48	55.56	42.55
WP* + A0	1	0	0	0.00	NaN	NaN
WP* + AM-MOD	1	0	0	0.00	NaN	NaN
WP* + R-A1	1	0	0	0.00	NaN	NaN

Table D.5: Precision and recall for semantic dependencies, excluding predicate classification

D.2 Test

This section contains the scorer output for the *test* dataset. Note that this are the official scoring results using for the *test* dataset. To improve, evaluate and analize the errors of our system the *development* dataset must be used. See the 6 and the previous section for detailed results using the *development* corpus.

Evaluation metric	**detail**	**result**
SYNTACTIC SCORES:		
Labeled attachment score:	49911 / 57676	86.54 %
Unlabeled attachment score:	51458 / 57676	89.22 %
Label accuracy score:	52931 / 57676	91.77 %
Exact syntactic match:	528 / 2400	22.00 %
SEMANTIC SCORES:		
Labeled precision:	(14960 + 8501) / (20673 + 10948)	74.19 %
Labeled recall:	(14960 + 8501) / (23286 + 10498)	69.44 %
Labeled F1:		71.74
Unlabeled precision:	(16933 + 9486) / (20673 + 10948)	83.55 %
Unlabeled recall:	(16933 + 9486) / (23286 + 10498)	78.20 %
Unlabeled F1:		80.79
Proposition precision:	3680 / 10948	33.61 %
Proposition recall:	3680 / 10498	35.05 %
Proposition F1:		34.32
Exact semantic match:	217 / 2400	9.04 %
OVERALL MACRO SCORES (Wsem = 0.50):		

Labeled macro precision:		80.37 %
Labeled macro recall:		77.99 %
Labeled macro F1:		79.16 %
Unlabeled macro precision:		86.38 %
Unlabeled macro recall:		83.71 %
Unlabeled macro F1:		85.03 %
Exact overall match:	139 / 2400	5.79 %
OVERALL MICRO SCORES:		
Labeled micro precision:	(49911 + 14960 + 8501) / (57676 + 20673 + 10948)	82.17 %
Labeled micro recall:	(49911 + 14960 + 8501) / (57676 + 23286 + 10498)	80.22 %
Labeled micro F1:		81.18
Unlabeled micro precision:	(51458 + 16933 + 9486) / (57676 + 20673 + 10948)	87.21 %
Unlabeled micro recall:	(51458 + 16933 + 9486) / (57676 + 23286 + 10498)	85.15 %
Unlabeled micro F1:		86.17

Table D.6: Test development score

D.2.1 Syntactic depedencies

Deprel	gold	correct	system	recall (%)	precision (%)
ADV	2091	1541	2323	73.70	66.34
ADV-GAP	0	0	1	NaN	0.00
AMOD	913	598	761	65.50	78.58
APPO	727	541	796	74.42	67.96
BNF	1	0	0	0.00	NaN
CONJ	1103	954	1149	86.49	83.03
COORD	1374	980	1330	71.32	73.68
DEP	1302	1045	1266	80.26	82.54
DEP-GAP	3	0	2	0.00	0.00
DIR	292	139	183	47.60	75.96
DIR-GAP	0	0	1	NaN	0.00
DTV	19	13	17	68.42	76.47
EXT	105	79	93	75.24	84.95
EXT-GAP	1	0	0	0.00	NaN
EXTR	29	15	23	51.72	65.22

GAP-LGS	3	2	3	66.67	66.67
GAP-LOC	4	1	3	25.00	33.33
GAP-OBJ	2	0	1	0.00	0.00
GAP-PMOD	1	0	2	0.00	0.00
GAP-PRD	4	1	3	25.00	33.33
GAP-SBJ	4	0	3	0.00	0.00
GAP-TMP	6	0	1	0.00	0.00
HMOD	702	676	723	96.30	93.50
HYPH	707	705	723	99.72	97.51
IM	782	770	777	98.47	99.10
LGS	159	154	171	96.86	90.06
LOC	955	633	949	66.28	66.70
LOC-OPRD	1	0	0	0.00	NaN
LOC-PRD	71	30	35	42.25	85.71
MNR	179	105	180	58.66	58.33
NAME	2002	1547	1823	77.27	84.86
NMOD	15286	13908	15439	90.99	90.08
OBJ	3073	2819	3193	91.73	88.29
OPRD	568	482	554	84.86	87.00
P	6831	5514	6851	80.72	80.48
PMOD	5469	5143	5566	94.04	92.40
POSTHON	327	311	336	95.11	92.56
PRD	835	702	862	84.07	81.44
PRD-PRP	3	1	2	33.33	50.00
PRD-TMP	1	0	0	0.00	NaN
PRN	67	12	36	17.91	33.33
PRP	205	82	144	40.00	56.94
PRT	157	122	156	77.71	78.21
PUT	9	6	10	66.67	60.00
ROOT	2399	2288	2399	95.37	95.37
SBJ	4332	4016	4277	92.71	93.90
SUB	655	601	684	91.76	87.87
SUFFIX	533	527	538	98.87	97.96
TITLE	238	187	191	78.57	97.91
TMP	1341	901	1273	67.19	70.78
VC	1804	1760	1823	97.56	96.54
VOC	1	0	0	0.00	NaN

Table D.7: Precision and recall for syntactic depedencies (test)

D.2.2 Predicate classification

PPOSS	**gold**	**correct**	**system**	**recall (%)**	**precision (%)**	**F_1**

CC	3	0	0	0.00	NaN	NaN
CD	1	0	0	0.00	NaN	NaN
IN	3	1	1	33.33	100.00	50.00
JJ	16	0	0	0.00	NaN	NaN
NN	3635	2710	3464	74.55	78.23	76.35
NNP	10	0	0	0.00	NaN	NaN
NNS	1648	1274	1737	77.31	73.34	75.27
PDT	2	0	0	0.00	NaN	NaN
RP	4	2	2	50.00	100.00	66.67
VB	1278	1098	1299	85.92	84.53	85.22
VBD	1320	1197	1371	90.68	87.31	88.96
VBG	742	623	842	83.96	73.99	78.66
VBN	985	831	1195	84.37	69.54	76.24
VBP	343	306	452	89.21	67.70	76.98
VBZ	504	459	585	91.07	78.46	84.30
WP	2	0	0	0.00	NaN	NaN
WRB	2	0	0	0.00	NaN	NaN

Table D.8: Precision and recall for predicate classification (test)

D.2.3 Precision and recall for semantic dependencies

PPOSS(pred) + ARG	gold	correct	system	recall (%)	precision (%)	F_1
CC* + A1	3	0	0	0.00	NaN	NaN
CC* + A2	2	0	0	0.00	NaN	NaN
CD* + A1	1	0	0	0.00	NaN	NaN
IN* + A1	2	1	1	50.00	100.00	66.67
IN* + A2	1	0	0	0.00	NaN	NaN
IN* + AM-LOC	2	0	0	0.00	NaN	NaN
IN* + AM-MNR	1	0	1	0.00	0.00	NaN
JJ* + A0	5	0	0	0.00	NaN	NaN
JJ* + A1	15	0	0	0.00	NaN	NaN
JJ* + A2	7	0	0	0.00	NaN	NaN
JJ* + AM-ADV	2	0	0	0.00	NaN	NaN
JJ* + AM-DIS	1	0	0	0.00	NaN	NaN
JJ* + AM-LOC	1	0	0	0.00	NaN	NaN
JJ* + AM-MNR	2	0	0	0.00	NaN	NaN
JJ* + AM-MOD	1	0	0	0.00	NaN	NaN
JJ* + AM-PNC	1	0	0	0.00	NaN	NaN
JJ* + AM-TMP	3	0	0	0.00	NaN	NaN
JJ* + C-A1	1	0	0	0.00	NaN	NaN
NN* + A0	2339	956	1356	40.87	70.50	51.74
NN* + A1	3757	2232	3221	59.41	69.30	63.98
NN* + A2	1537	607	1000	39.49	60.70	47.85
NN* + A3	349	180	249	51.58	72.29	60.20
NN* + A4	18	2	8	11.11	25.00	15.38
NN* + A5	1	0	0	0.00	NaN	NaN

NN* + AM-ADV	32	2	7	6.25	28.57	10.26
NN* + AM-CAU	2	0	0	0.00	NaN	NaN
NN* + AM-DIR	4	0	0	0.00	NaN	NaN
NN* + AM-DIS	4	0	0	0.00	NaN	NaN
NN* + AM-EXT	33	10	23	30.30	43.48	35.71
NN* + AM-LOC	232	94	215	40.52	43.72	42.06
NN* + AM-MNR	344	160	275	46.51	58.18	51.69
NN* + AM-MOD	7	0	0	0.00	NaN	NaN
NN* + AM-NEG	35	6	8	17.14	75.00	27.90
NN* + AM-PNC	1	0	0	0.00	NaN	NaN
NN* + AM-TMP	492	274	363	55.69	75.48	64.09
NN* + C-A1	2	0	0	0.00	NaN	NaN
NN* + R-A0	2	0	0	0.00	NaN	NaN
PD* + A1	2	0	0	0.00	NaN	NaN
PD* + A2	1	0	0	0.00	NaN	NaN
PD* + AM-LOC	1	0	0	0.00	NaN	NaN
RP* + A0	1	0	0	0.00	NaN	NaN
RP* + A1	2	1	1	50.00	100.00	66.67
RP* + AM-LOC	2	1	2	50.00	50.00	50.00
RP* + AM-TMP	2	1	1	50.00	100.00	66.67
VB* + A0	3509	2756	3448	78.54	79.93	79.23
VB* + A1	4844	4053	5163	83.67	78.50	81.00
VB* + A2	1085	644	1042	59.35	61.80	60.55
VB* + A3	169	78	123	46.15	63.41	53.42
VB* + A4	99	71	101	71.72	70.30	71.00
VB* + A5	5	1	3	20.00	33.33	25.00
VB* + AM-ADV	488	247	504	50.61	49.01	49.80
VB* + AM-CAU	70	45	66	64.29	68.18	66.18
VB* + AM-DIR	81	23	44	28.40	52.27	36.80
VB* + AM-DIS	315	195	281	61.90	69.40	65.44
VB* + AM-EXT	32	17	32	53.12	53.12	53.12
VB* + AM-LOC	355	212	356	59.72	59.55	59.63
VB* + AM-MNR	335	173	338	51.64	51.18	51.41
VB* + AM-MOD	539	493	505	91.47	97.62	94.44
VB* + AM-NEG	227	217	300	95.59	72.33	82.35
VB* + AM-PNC	113	31	70	27.43	44.29	33.88
VB* + AM-PRD	5	0	0	0.00	NaN	NaN
VB* + AM-REC	2	0	0	0.00	NaN	NaN
VB* + AM-TMP	1068	756	1001	70.79	75.52	73.08
VB* + C-A0	5	0	0	0.00	NaN	NaN
VB* + C-A1	192	132	185	68.75	71.35	70.03
VB* + C-A2	0	0	1	NaN	0.00	NaN
VB* + C-A3	2	0	0	0.00	NaN	NaN
VB* + C-AM-DIR	1	0	1	0.00	0.00	NaN
VB* + C-AM-MNR	1	0	0	0.00	NaN	NaN
VB* + C-AM-TMP	1	0	0	0.00	NaN	NaN
VB* + R-A0	222	179	216	80.63	82.87	81.73
VB* + R-A1	155	94	132	60.65	71.21	65.51
VB* + R-A2	16	4	4	25.00	100.00	40.00
VB* + R-A3	1	0	0	0.00	NaN	NaN
VB* + R-A4	1	0	0	0.00	NaN	NaN
VB* + R-AM-ADV	2	0	0	0.00	NaN	NaN
VB* + R-AM-CAU	4	0	0	0.00	NaN	NaN
VB* + R-AM-EXT	1	0	0	0.00	NaN	NaN
VB* + R-AM-LOC	21	2	4	9.52	50.00	15.99
VB* + R-AM-MNR	6	0	1	0.00	0.00	NaN
VB* + R-AM-TMP	52	10	21	19.23	47.62	27.40
WP* + A0	1	0	0	0.00	NaN	NaN
WP* + A1	1	0	0	0.00	NaN	NaN

WP* + A2	1	0	0	0.00	NaN	NaN
WP* + AM-LOC	1	0	0	0.00	NaN	NaN
WP* + AM-TMP	1	0	0	0.00	NaN	NaN
WR* + A0	1	0	0	0.00	NaN	NaN
WR* + A1	2	0	0	0.00	NaN	NaN
WR* + AM-EXT	1	0	0	0.00	NaN	NaN

Table D.9: Precision and recall for semantic dependencies, excluding predicate classification (test)

Appendix E

Bookmarks

This section collects only the most rellevant bookmarks for the subject of this thesis. Note that this collection does not comprise the vast amount of sources that are available on dependency parsing, natural language processing toolkits and machine learning.

CoNLL

The CoNLL-2008 Shared task web site and results `http://www.yr-bcn.es/conll2008`

The CoNLL-2008 Proceedings (including description papers) `http://www.cnts.ua.ac.be/conll2008/proceedings.html`

Dependency parsing

MaltParser `http://w3.msi.vxu.se/~nivre/research/MaltParser.html`

MST McDonald's framework `http://ryanmcd.googlepages.com/MSTParser.html`

Nivre and McDonald dependency parsing course slides `http://dp.esslli07.googlepages.com/`

Glossary

arc-factored model a model that defines the score or probability of a graph assuming that all arcs are conditionally independent, 23

Brown corpus a compiled corpus of current American English extracted from a wide variety of sources [1] , 9

chunk parser a parser that segments a sentence in contiguous units , no tree structure or constituent structure is generated., 47

CoNLL is a yearly meeting of the SIGNLL, the ACL's Special Interest Group on Natural Language Learning., 5

corpus a collection of documents. , 9

dependency parsing is the task of deriving a graph representing the syntactic structure of the sentence., 1

dependency structure a directed graph expressing syntactic relations between tokens with labeled arcs, 18

dynamic features features that are extracted on-line during parsing from the partially constructed tree., 67

frameNet a collection of semantic frames. , 9

joint parsing a combined processing of the syntactic and semantic structure, 2

Named Entity atomic element in text that can comprise more than one word and designates specified things e.g., names of persons, organizations, locations)., 70

[1] http://khnt.aksis.uib.no/icame/manuals/brown/

Natural Language Understanding (NLU) is the set of tasks that deals with the exploitation of semantic knowledge present in natural language text, 1

Penn treebank a Treebank for the English language. , 9

POS the linguistic category of the words. , 9

projective link a link (i, j) such that $i \rightarrow j$, then $\forall i'$ such that $i \leftarrow i'$, $i < i' < j$ or $j < i' < i$., 19

projective sentence a sentence with one or more projective link., 19

propBank a corpus annotated with verbal propositions and their arguments. , 9

semantic frame a predicate and its receiving or admissible arguments., 14

semantic shallow parsing is the annotation semantic arguments for each sentence predicate., 2

span a dependency structure for a substring with the properties of no outgoing nor incoming edges from the rest of the input sentence., 23

split lemma/form the lemma or form of a token but considering a finer tokenization. For instance the split forms of *Atlanta-based* are *Atlanta*, "-" and *based*. , 65

treebank a text corpus with annotated syntactic structure. , 9

Index