Deep, Consistent and also Useful: Extracting Vistas from Deep Corpora for Shallower Tasks

João Silva, António Branco, Sérgio Castro, Francisco Costa

University of Lisbon
Edifício C6, Departamento de Informática,
Faculdade de Ciências, Universidade de Lisboa,
Campo Grande, 1749-016 Lisboa, Portugal
{jsilva, antonio.branco, sergio.castro, fcosta}@di.fc.ul.pt

Abstract

Annotated corpora are fundamental for NLP, and the trend in their development is to move towards datasets with increasingly detailed linguistic annotation. To cope with the complexity of producing such resources, some approaches rely on a supporting deep processing grammar that provides annotation that is rich and consistent over its morphological, syntactic and semantic layers. However, for some purposes, the deep linguistic corpora thus produced are "too deep" and unwieldy. For instance, if one wishes to obtain a probabilistic constituency parser by learning a model over a treebank, the full extent of the annotation created by a deep grammar is not needed and can even be detrimental to training. In this paper, we report on procedures that, starting from a deep dataset produced by a deep processing grammar, extract a variety of *vistas*—that is, subsets of the information contained in the full dataset. This allows taking a single base dataset as a starting point and, from it, deliver a variety of corpora that are more streamlined and focused on particular tasks.

1. Introduction

Annotated corpora are key resources for NLP. Not only are they important materials for researchers investigating linguistic phenomena, they also allow one to automatically obtain data-driven models of language and evaluate the tools thus produced.

Annotating corpora with human-verified linguistic information is a time-consuming and often error-prone task. Early treebanks for NLP, like the well-known Penn Treebank corpus, were built with the help of automatic annotation tools that were used to provide a preliminary annotation which was then manually corrected (Marcus et al., 1993).

Performing such corrections by hand can introduce formatting errors since manual changes may easily be malformed (e.g. misspelled categories, forgetting to close a bracket, etc). As such, the manual correction step is often aided by a tool that ensures that at least the linguistic information is well formed

However, as the linguistic information one wishes to include in the corpus grows in complexity, this approach becomes increasingly hard to adopt since the human annotator, even with the help of supporting software, has to keep track of too much interconnected information.

To address this issue, approaches to corpus annotation have come to rely on an auxiliary deep processing grammar as a way of producing rich annotation that is consistent over its morphological, syntactic and semantic layers. Two examples of such an approach are (Dipper, 2000), using an LFG framework, and (Oepen et al., 2002), under HPSG.

Despite these advantages in terms of consistency and depth of the information encoded, the annotation produced by such grammars is often too theory-specific or too unwieldy for certain purposes. For instance, if one wishes to train a probabilistic constituency parser, the linguistic information on grammatical functions and semantic roles present in the output of a deep grammar is not needed and, if integrated into the model, might actually be detrimental to the performance of the parser due to data-sparseness issues.

The image in Figure 1 helps to illustrate the problem. It shows the fully-fledged grammatical representation, under the HPSG framework, for the rather simple sentence *Todos os computadores têm um disco* (Eng.: All computers have a disk).[1]

Thus, it is desirable to have a process that allows extracting *vistas*—that is, subsets of the information contained in the full dataset—such as text annotated with part-of-speech tags, a plain constituency tree or a grammatical dependency graph.

In this paper we present a set of procedures that allow extracting several such vistas from a deep linguistic dataset. In particular, we will use a dataset that has been produced by a computational grammar based on the HPSG framework (Pollard and Sag, 1994; Sag and Wasow, 1999; Copestake, 2002).

Section 2 provides an overview of the grammar-supported annotation procedure used to produce the full deep dataset, while Section 3 describes the deep dataset itself. This is followed by Section 4, where the vista extraction procedures are presented. Section 5 provides an extrinsic evaluation of the extracted vistas by inducing probabilistic parsers over them. Finally, Section 6 concludes with final remarks.

2. Grammar-Supported Treebanking

A grammar-supported approach to corpus annotation consists in using a computational grammar to produce all possible analyses for a given sentence. What is then asked of the human annotator is to select the correct parse among all those that were returned. In such a setup, the task of the

[1] The printout is in a 6pt font. The arm and hand holding a pen are there just to give a sense of the size of the grammatical representation.

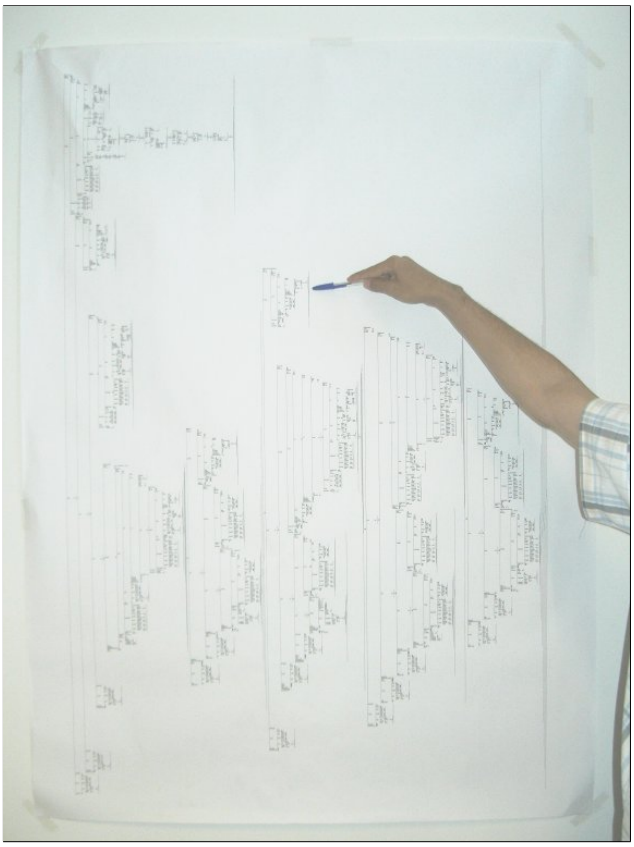


Figure 1: Full HPSG representation of the sentence

human annotator can be envisaged as being one of disambiguation.

Due to the inherent ambiguity of natural language, the parse forest that results from the grammar producing all possible analyses for a sentence may very well include hundreds of trees. Manually examining each individual tree in search for the correct one would prove unfeasible. Instead, the human disambiguator goes through a list of discriminants to reduce the number of parses. Discriminants are binary disambiguation decisions, many of which cut the parse forest in half.

For instance, PP-attachment is a common source of structural ambiguity, where a PP constituent may validly attach to more than one constituent in the parse tree. A discriminant would state whether the PP attaches to a given constituent. Choosing that discriminant as valid automatically prunes from the parse forest all parses where the attachment of that PP is different, while marking the discriminant as invalid discards all trees where the PP is attached to that same constituent.

For such an approach to work, it must be supported by a tool that provides the discriminants and handles the pruning of the parse forest in a manner that is unobtrusive for the human annotator. For datasets in the HPSG family, like the one used in this work, this can be done using the `[incr tsdb()]` tool (Oepen and Flickinger, 1998). Besides providing an interface for the disambiguation process described above, this tool integrates functionality for benchmarking, profiling and testing the grammar over test suites.

3. The Core Dataset

To create the core deep linguistic dataset from which the vistas will be extracted, we started with a corpus of Portuguese newspaper excerpts which had been previously annotated with manually verified shallow morpho-syntactic data, namely part-of-speech tags, lemmas, inflection features and information on named-entities.

This corpus was then treebanked according to the process outlined in Section 2. The supporting grammar that was used is LXGram, a deep computational grammar for Portuguese (Branco and Costa, 2008; Branco and Costa, 2010). It is worth of note that, for this dataset, annotation was done through a method of double-blind annotation followed by adjudication. In this setup, two human annotators work independently while pruning the parse forest returned by the grammar. If both annotators agree on the choice of an analysis, that analysis is added to the dataset. When the annotators disagree on what is the preferred analysis, a third human annotator, the adjudicator, is brought in to decide which analysis will be added to the dataset, if any (the adjudicator is free to choose a third analysis, rejecting the ones chosen by either annotator). This method of corpus annotation is resource-consuming, both in terms of human effort (three people are needed) and in terms of time (an adjudication round is required), but it allows a stricter quality control of the dataset being produced.

Due to the way it was built, the core dataset only contains those sentences that the supporting grammar was able to parse. It is formed by 5,422 sentences, most of which (4,644, or 86%) from newspaper text. The remaining sentences (778, or 14%) were part of the LXGram distribution and consist of sentences used for regression testing of the grammar.

4. Extracting Vistas

In this work we cover three vistas: the TreeBank, the DependencyBank and the PropBank, A TreeBank vista is a constituency tree, the familiar structure that represents the various constituents of the sentence and their level of aggregation. A DependencyBank vista, instead of giving a tree structure describing syntactic constituency, is a graph that relates pairs of words by a syntactic function (i.e. subject, direct object, modifier, etc). The PropBank is a dataset similar to the one described in (Kingsbury and Palmer, 2003) in that it consists of a layer of semantic role annotation that is added to phrases in the syntactic structure of the sentence. The format of these extended nodes in the PropBank tree is C-GF-SR, where C is the constituency tag, GF corresponds to the grammatical function and SR to the semantic role.

What is important to note regarding these three vistas is that the information contained in a PropBank is a super-set of the information present in the other two vistas. Our approach is then to take the PropBank as the main vista since the other two vistas, viz. the TreeBank and the DependencyBank, can in turn be obtained directly from it instead of having to extract each of them independently from the deep dataset.

For this we began by creating a PropBank extraction tool that runs over the deep representations resulting from the grammar-supported treebanking process. This tool makes

use of the Tregex library created by the Stanford NLP Group (Levy and Andrew, 2006),[2] which provides a language for pattern matching over tree structures and regular expression matches over tree nodes.

4.1. PropBank Vista

The procedure that creates the PropBank consists of several steps, each having to deal with non-trivial issues. These steps are described in this Section.

4.1.1. Retrieving the Exported Tree

The deep representation of a sentence that is exported by `[incr tsdb()]` at the end of the manual disambiguation process includes the derivation tree, which encodes the rules that were used by the grammar during analysis of that sentence and the order in which they were applied. The exported deep representation also includes a second tree which has the same structure as the derivation tree, but where the rule names have been mapped into syntactic categories. This second tree, which we will call the *exported tree*, is taken by the tool as the starting point of the vista extraction procedure.

4.1.2. Tokenization

Due to the inner workings of `[incr tsdb()]`, the leafs in the exported tree are all converted to lowercase and truncated to the first 30 characters. Moreover, given the grammar used, the original newspaper corpus that was treebanked contains information not present in the deep dataset that has been created by the grammar (e.g. information on named entities). Thus, in order not to lose this data, we want it to be possible to incorporate it into the vistas. The most straightforward way of fixing each leaf is to replace it by the corresponding token from the original sentence.

For either of these procedures to work, leafs and tokens must be aligned. However, there is not a one-to-one correspondence between the leafs in the exported tree and the tokens in the sentence due to the original corpus and the grammar having different criteria for tokenization.

This is readily apparent in punctuation symbols, which are still attached to words in the exported tree, while they are found tokenized (i.e. detached from words) in the sentence. Given that the purpose of the tokenization stage is only to obtain a one-to-one correspondence between the leafs in the exported tree and the tokens in the sentence, punctuation symbols are simply detached from words and temporarily placed in a newly created sister node. The process of moving the punctuation symbols to their correct position in the final tree merits a slightly more detailed explanation and is addressed further ahead

4.1.3. Feature Bundles

Following the tokenization step, the leafs in the exported tree will be aligned one-to-one with the tokens from the original corpus, which allows the tool to easily copy the morpho-syntactic information from the corpus over to the tree as feature bundles that are appended to the leafs.

Figure 2 shows the breakdown of a feature bundle into its parts. Having the POS tag as a feature might at first seem

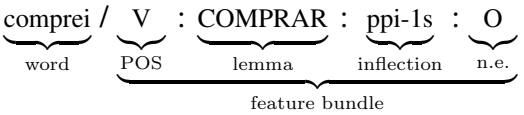


Figure 2: Leaf with added feature bundle

unnecessary, since that information is given by the preterminal node of the tree, but the POS tagset of the original corpus is different from the one used by the grammar and, in this way, no information is lost. Named-entity information is encoded using a tagset like the one from the CoNLL shared task (Tjong Kim Sang, 2002): B is used for the first word in an entity, and I for any subsequent words in the same entity. A string representing the semantic type of the entity (e.g. PER for person, LOC for location, etc.) is appended. The letter O marks a word not belonging to any named entities.

Note that, in the following examples, the feature bundles appended to the leafs are not shown for the sake of readability.

4.1.4. Moving Punctuation

Punctuation symbols were detached from words during tokenization and placed in a temporary position. The current step is concerned with deciding where in the final tree to place the node with the detached punctuation symbol since its final position will depend on the syntactic construction the symbol is a part of.

Coordination is represented as a recursive tree structure where several constituents of the same type are combined together. Usually, a comma is used to separate each constituent, except for the last one which is delimited by an explicit conjunction, such as *e* (Eng.: and), *ou* (Eng.: or), etc.

As the example in Figure 3 shows, the comma is initially attached to the final word in a constituent of the enumeration. After being detached from the word, it is placed under a new node (PNT) which is in an adjunction position to the node to the right.

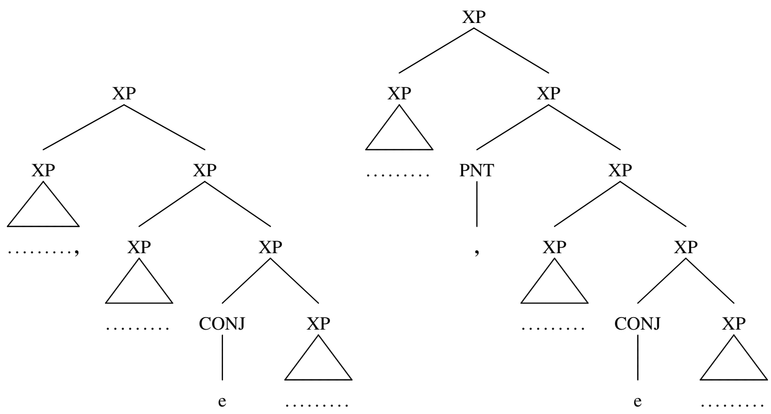
Appositions inside NPs are delimited by commas. This is made explicit in the tree representation by placing the apposition in a sub-tree that itself is delimited on either side by a pair of matching punctuation nodes, as shown in Figure 4. Parenthetical structures and quoted expressions are represented in a similar way. These are also the only situations where ternary nodes are used.

In all other cases, such as sentence-ending punctuation and topicalization, punctuation is adjoined far up as possible without crossing constituent boundaries. Figure 6 shows an example.

4.1.5. Collapsing Unary Chains

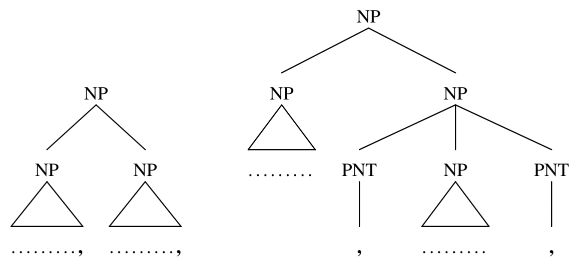
The syntactic representation of the exported tree contains unary chains of nodes with the same label. As mentioned above, this happens because the structure of the exported syntactic tree mirrors that of the derivation tree, which represents the rules applied by the grammar. Each node in these chains corresponds to the application of a unary morphological rule by the grammar (cf. (Copestake, 2002, Section 5.2) for more on such rules).

[2]Tregex website at `http://nlp.stanford.edu/software/tregex.shtml`.



(a) exported tree (b) final tree

Figure 3: Coordination



(a) exported tree (b) final tree

Figure 4: Apposition

For instance, the unary chain of three N nodes that dominates the word *computadores* (Eng.: computers) in Figure 5 corresponds, from the bottom up, to the application of the following morphological rules: COMPUTADOR (the rule for the lexical entry of the word), MASC-NOMINAL (flags a feature that marks the word as having gender inflection) and PL-NOMINAL (flags a feature that marks the word as having number inflection).

These various nodes in these unary chains are collapsed into a single node in the final tree.

4.1.6. Adding Phonetically Null Items

Nodes marking null subjects (*NULL*), null heads (*ELLIPSIS*), traces of constituents (*GAP*) and tough objects (*TOUGH*) are explicitly added to the final tree. There are several details concerning this step that are worth pointing out.

Pattern matching over the exported syntactic tree is not enough to always detect where one should add the nodes for phonetically null items. Instead, to do that, one must look at the derivation tree, since the relevant information can be found in the name of the derivation rule.

However, at this stage of processing, the syntactic tree and the derivation tree, which began by being isomorphic, do not have matching structures anymore, since the syntactic tree has been altered (viz. when moving punctuation and when collapsing unary chains). This issue was overcome by decorating the nodes in the syntactic tree with information taken from the derivation tree while both structures are still isomorphic.

Having decorated the syntactic tree, adding tree branches representing null subjects and null heads is quite straightforward.

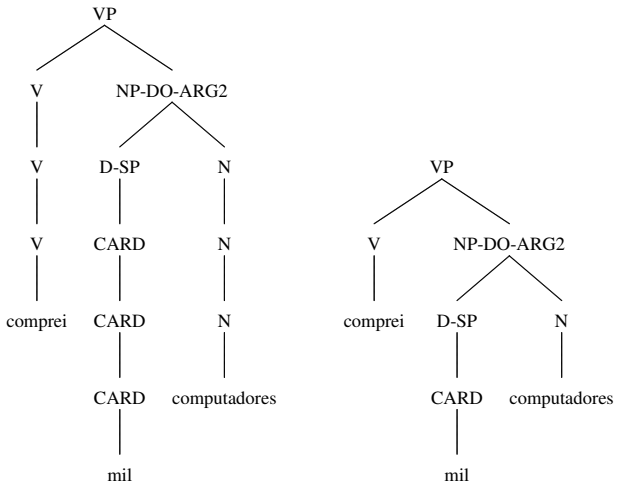
Null subjects are found by looking for SNS nodes in the exported syntactic tree, which are the way the grammar categorizes a sentence with a null subject. However, to properly assign a semantic role, the tool needs to look at the rule name from the corresponding node in the derivation tree, since the rule name indicates whether the missing NP-SJ node is an expletive (no semantic role), a passive construction (ARG2), a causative alternation (ARGA) or falls under the default case (ARG1).

Null heads are found by searching the derivation tree for certain rule names. The rule name not only indicates the category of the missing head (nominal or verbal) but also whether the head is the left or right child of the node.

Figure 6 shows an example of a parse tree with a null subject and a null nominal head.

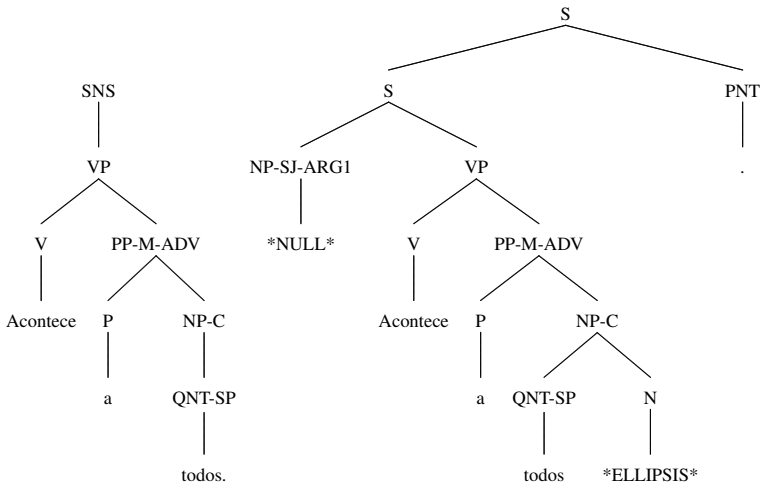
Nodes with a trace constituent are decorated by searching the derivation tree for a rule that indicates the extraction of a constituent and marking the corresponding node in the syntactic tree. The rule name also indicates whether the extracted constituent is on the left or on the right side of the node. The category of the extracted constituent is given by the usual HPSG slash notation, where a node labeled with X/Y indicates a constituent of type X that is missing a constituent of type Y.

When adding the trace, it suffices searching for the decorated node and add the *GAP* node as its left of right child, depending on the marking. In addition, the trace is



(a) exported tree (b) final tree

Figure 5: Unary chains
(Eng.: I bought a thousand computers)



(a) exported tree (b) final tree

Figure 6: Null subjects and null heads
(Eng.: Happens to everyone)

co-indexed with the displaced node by affixing the same index number to the trace and to the corresponding displaced constituent, as shown in Figure 7.

The displaced node is found by following the path of slashed constituents from the trace up to the topmost slash, which is the sister node of the displaced node.

When the sister of the topmost slash is not of the expected category it indicates a "tough" construction, and the trace node is marked with *TOUGH*, as shown in Figure 8.

4.1.7. Extending Semantic Role Annotation

The semantic role tags present in some of the nodes are at a different abstraction level than the constituency information conveyed by the phrase labels and tree structure. In particular, some role annotations show cross-tree dependency, where they need to refer to more than one constituent although the exported trees do not make this explicit.

This is the case with complex predicates, such as modals, auxiliaries and raising verbs. In such cases, the semantic role tag is suffixed with "cp" (for complex predicate). Anticausative constructions are handled in a similar way, but using "ac" as a suffix to the role tag.

For instance, the tree snippet shown in Figure 9 indicates that, though the NP is the subject of the VP, it is not the ARG1 of the head verb of the VP, but instead it is the ARG1 of some verb that is located down in the complex predicate topped by the VP.

Arguments of control verbs are handled in a similar manner, but one needs to look at the lexical type of the verb to determine whether it is a subject, direct object or indirect object control verb. To achieve this, the grammar lexicon is used to map the derivation rule for the lexical entry of a word (i.e. the pre-terminal node in the derivation tree) into the corresponding lexical type.

For instance, the ARG11 tag in Figure 10 indicates that the NP is both the subject of the control verb, *querem* (Eng.:

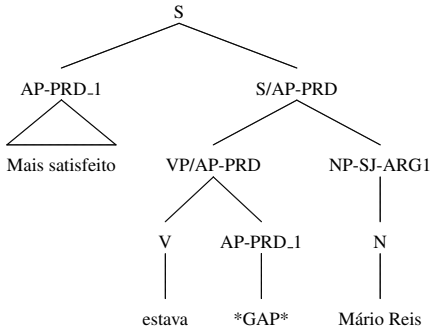


Figure 7: Traces and co-indexation
(Eng.: More pleased was Mário Reis)

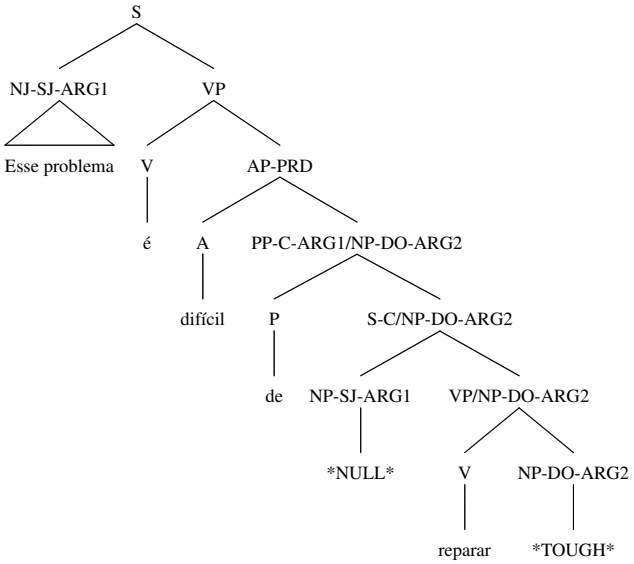


Figure 8: "Tough" constructions
(Eng.: That problem is tough to repair)

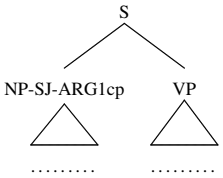


Figure 9: Complex predicate

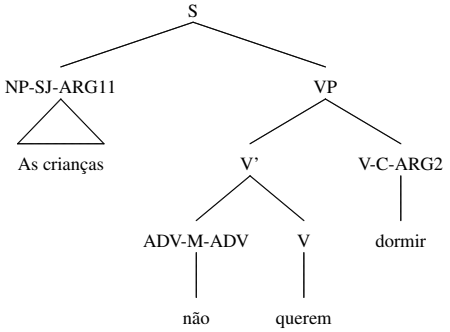


Figure 10: Control verbs
(Eng.: The children don't want to sleep)

want), and subject in the clause occurring as direct object of that verb.

4.2. TreeBank Vista

Having extracted the PropBank vista, the TreeBank vista is straightforward to obtain by simply discarding all information on grammatical function and semantic roles, leaving only the lexical and phrasal constituency information in the nodes of the tree.

4.3. DependencyBank Vista

To obtain the DependencyBank vista, one would like to make use of the extracted PropBank vista as an intermediate representation since it has already gone through an extensive normalization process. Fortunately, this is possible given that the trees that form the PropBank also include information on grammatical function in tags that are attached to the labels of some constituency nodes (e.g. SJ for subject, DO for direct object, M for modifier, etc). This gives us a straightforward way to automatically extract a dependency dataset from the PropBank.

Given that the PropBank adheres to an X-bar representation, phrasal nodes will have two children, one of which will be marked with a grammatical function. The (head of the) child that is marked is dependent on the (head of the) other child under the given grammatical function. The head of the phrasal node is the head of the unmarked child.

For instance, the tree fragment shown in Figure 11 yields a dependency where the head of ZP depends on the head of YP under relation F. The head of XP is the head of YP.

In the DependencyBank, displaced constituents are not represented by a *GAP* node. Instead, the head of the displaced node is dominated by the governor of its co-indexed node. For instance, in Figure 7 the head of the AP-PRD constituent is dependent on the verb *estava* (Eng.: was).

For complex predicates and anticausative constructions, the grammatical function tag is suffixed with the corresponding tag (i.e. either "cp" or "ac"). For instance, in Figure 9, the head of the NP is dependent on the head of the VP under the SJcp relation.

This procedure is carried out by a second tool that takes the PropBank as input and outputs the DependencyBank in the format of dependency triples and also in the widely-used CoNLL format (Nivre et al., 2007).

5. Evaluation

The extraction tool and the resulting vistas were evaluated extrinsically by measuring the performance of constituency and dependency probabilistic parsers trained over the corresponding vistas. The rationale for this approach being that a high quality dataset, with a consistent representation, should allow training probabilistic parsers that perform with high accuracy.

Note that, for the purpose of linguistic studies, both the TreeBank and the DependencyBank contain nodes that correspond to phonetically null items. These items, however, do not correspond to actual tokens that will appear in the input to the parser. Accordingly, they are removed from the TreeBank and DependencyBank vistas when training the parsers. In the TreeBank, the branches formed by a phonetically null item and the pre-terminal node immediately above it are pruned from the tree. Other information associated with these items, such as co-indexes and the slash notation used for traces, is also removed from the tree. In the PropBank, any dependencies involving the null items are discarded.

Since the focus is not on the development and tuning of the parsers, we opted for simply taking freely available third-party tools and running them out-of-the-box.

For constituency, we ran the Stanford parser (Klein and Manning, 2003), using the default parameters, over the 5,422 sentences in the TreeBank. This parser induces separate models, one for phrase-structure and one for lexical dependencies, which are then factored together during annotation. Following a standard 10-fold cross-validation approach, we obtained an 88% score under the Parseval metric. This is on par with the performance scores obtained by the same parser for English when training over the much larger Wall Street Journal dataset.

For evaluating the DependencyBank, we used MSTParser (McDonald et al., 2006), again using the default parameters. This parser works in two stages, the first assigning unlabeled dependency edges which are then labeled in the second stage using a sequence classifier. Under a 10-fold cross-validation evaluation methodology, we obtained a 87% labeled accuracy score, which is also a state-of-the-art score for this task.

6. Conclusion and Final Remarks

In this paper we presented and assessed a procedure for extracting vistas from a core deep dataset.

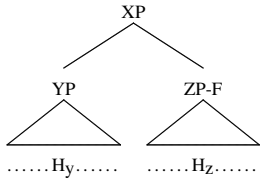
Deep processing grammars provide rich, accurate and consistent grammatical analyses for sentences, as well as much-needed support for the effective treebanking of corpora being annotated with rich linguistic information.

However, the output of such grammars may be too complex and unwieldy for what is required by certain tasks, motivating the need for creating procedures that extract streamlined and focused vistas. Such procedures allow taking a single, deep dataset as a starting point, with all the linguistic richness and annotation consistency that it offers, and extract subsets of the information contained in it.

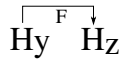
For the work described in this paper, this core dataset is composed of 5,422 sentences of mostly newspaper text. It was created with the help of an HPSG deep processing grammar by manual double-blind disambiguation of the analyses produced by the grammar.

A tool was described that extracts a PropBank vista, a syntactic structure where phrases are enriched with a layer of semantic role annotation. The extracted PropBank was then used as a super-vista from which TreeBank and DependencyBank vistas were in turn also extracted.

These latter two vistas were evaluated by training probabilistic parsers over them, namely a constituency parser for the TreeBank and a dependency parser for the DependencyBank. In both cases, under 10-fold cross-validation, the parsers achieved state-of-the-art scores.



(a) PropBank snippet



H_Z depends on H_Y under relation F
H_Y becomes the head of XP

(b) dependency relation

Figure 11: Extracting dependencies

7. References

António Branco and Francisco Costa. 2008. A computational grammar for deep linguistic processing of Portuguese: LX-Gram, version A.4.1. Technical Report DI-FCUL-TR-08-17, University of Lisbon.

António Branco and Francisco Costa. 2010. A deep linguistic processing grammar for Portuguese. In *Proceedings of the 9th Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR)*, LNAI, pages 86–89. Springer.

Ann Copestake. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Publications.

Stefanie Dipper. 2000. Grammar-based corpus annotation. In *Proceedings of the Workshop on Linguistically Interpreted Corpora*, pages 56–64.

Paul Kingsbury and Martha Palmer. 2003. Propbank: The next level of treebank. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 105–116.

Dan Klein and Christopher Manning. 2003. Fast exact inference with a factored model for NLP. *Advances in Neural Language Processing Systems*, 15:3–10.

Roger Levy and Galen Andrew. 2006. Tregex and Tsurgeon: Tools for querying and manipulating tree data structures. In *Proceedings of the 5th Language Resources and Evaluation Conference (LREC)*.

Mitchell Marcus, Mary Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the 10th Conference on Natural Language Learning (CoNLL)*, pages 216–220.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the 11th Conference on Natural Language Learning (CoNLL)*, pages 915–932.

Stephan Oepen and Daniel Flickinger. 1998. Towards systematic grammar profiling. Test suite technology ten years after. *Journal of Computer Speech and Language*, 12(4):411–436.

Stephan Oepen, Kristina Toutanova, Stuart Shieber, Christopher Manning, Dan Flickinger, and Thorsten Brants. 2002. The LinGO Redwoods treebank: Motivation and preliminary applications. In *Proceedings of the 19th Conference on Computational Linguistics (COLING)*.

Carl Pollard and Ivan Sag. 1994. *Head-Driven Phrase-Structure Grammar*. The University of Chicago Press.

Ivan Sag and Thomas Wasow. 1999. *Syntactic Theory: A Formal Introduction*. CSLI Publications.

Erik Tjong Kim Sang. 2002. Introduction to the CoNLL 2002 shared task: Language-independent named entity recognition. In *Proceedings of the 6th Conference on Natural Language Learning (CoNLL)*, pages 155–158.