Sentence Diagram Generation Using Dependency Parsing

Elijah Mayfield
Division of Science and Mathematics
University of Minnesota, Morris
mayf0016@morris.umn.edu

Abstract

Dependency parsers show syntactic relations between words using a directed graph, but comparing dependency parsers is difficult because of differences in theoretical models. We describe a system to convert dependency models to a structural grammar used in grammar education. Doing so highlights features that are potentially overlooked in the dependency graph, as well as exposing potential weaknesses and limitations in parsing models. Our system performs automated analysis of dependency relations and uses them to populate a data structure we designed to emulate sentence diagrams. This is done by mapping dependency relations between words to the relative positions of those words in a sentence diagram. Using an original metric for judging the accuracy of sentence diagrams, we achieve precision of 85%. Multiple causes for errors are presented as potential areas for improvement in dependency parsers.

1 Dependency parsing

Dependencies are generally considered a strong metric of accuracy in parse trees, as described in (Lin, 1995). In a dependency parse, words are connected to each other through relations, with a head word (the governor) being modified by a dependent word. By converting parse trees to dependency representations before judging accuracy, more detailed syntactic information can be discovered. Recently, however, a number of dependency parsers have been developed that have very different theories of a correct model of dependencies.

Dependency parsers define syntactic relations between words in a sentence. This can be done either through spanning tree search as in (McDonald et al., 2005), which is computationally expensive, or through analysis of another modeling system, such as a phrase structure parse tree, which can introduce errors from the long pipeline. To the best of our knowledge, the first use of dependency relations as an evaluation tool for parse trees was in (Lin, 1995), which described a process for determining heads in phrase structures and assigning modifiers to those heads appropriately. Because of different ways to describe relations between negations, conjunctions, and other grammatical structures, it was immediately clear that comparing different models would be difficult. Research into this area of evaluation produced several new dependency parsers, each using different theories of what constitutes a correct parse. In addition, attempts to model multiple parse trees in a single dependency relation system were often stymied by problems such as differences in tokenization systems. These problems are discussed by (Lin, 1998) in greater detail. An attempt to reconcile differences between parsers was described in (Marneffe et al., 2006). In this paper, a dependency parser (from herein referred to as the Stanford parser) was developed and compared to two other systems: MINIPAR, described in (Lin, 1998), and the Link parser of (Sleator and Temperley, 1993), which uses a radically different approach but produces a similar, if much more fine-grained, result.

Comparing dependency parsers is difficult. The main problem is that there is no clear way to compare models which mark dependencies differently. For instance, when clauses are linked by a conjunction, the Link parser considers the conjunction related to the subject of a clause, while the Stanford parser links the conjunction to the verb of a clause. In (Marneffe et al., 2006), a simple comparison was used to alleviate this problem, which was based only on the presence of dependencies, without semantic information. This solution loses information and is still subject to many problems in representational differences. Another problem with this approach is that they only used ten sentences for comparison, randomly selected from the Brown corpus. This sparse data set is not necessarily congruous with the overall accuracy of these parsers.

In this paper, we propose a novel solution to the difficulty of converting between dependency models. The options that have previously been presented for comparing dependency models are either too specific to be accurate (relying on annotation schemes that are not adequately parallel for comparison) or too coarse to be useful (such as merely checking for the existence of dependencies). By using a model of language which is not as fine-grained as the models used by dependency parsers, but still contains some semantic information beyond unlabelled relations, a compromise can be made. We show that using linear diagramming models can do this with acceptable error rates, and hope that future work can use this to compare multiple dependency models.

Section 2 describes structural grammar, its history, and its usefulness as a representation of syntax. Section 3 describes our algorithm for conversion from dependency graphs to a structural representation. Section 4 describes the process we used for developing and testing the accuracy of this algorithm, and Section 5 discusses our results and a variety of features, as well as limitations and weaknesses, that we have found in the dependency representation of (Marneffe et al., 2006) as a result of this conversion.

2 Introduction to structural grammar

Structural grammar is an approach to natural language based on the understanding that the majority of sentences in the English language can be matched to one of ten patterns. Each of these patterns has a set of slots. Two slots are universal among these patterns: the subject and the predicate. Three additional slots may also occur: the direct object, the subject complement, and the object complement. A head word fills each of these slots. In addition, any word in a sentence may be modified by an additional word. Finally, anywhere that a word could be used, a substitution may be made, allowing the position of a word to be filled by a multiple-word phrase or an entire subclause, with its own pattern and set of slots.

To understand these relationships better, a standardized system of sentence diagramming has been developed. With a relatively small number of rules, a great deal of information about the function of each word in a sentence can be represented in a compact form, using orientation and other spatial clues. This provides a simpler and intuitive means of visualizing relationships between words, especially when compared to the complexity of directed dependency graphs. For the purposes of this paper, we use the system of diagramming formalized in (Kolln and Funk, 2002).

2.1 History

First developed in the early 20th century, structural grammar was a response to the prescriptive grammar approach of the time. Structural grammar describes how language actually is used, rather than prescribing how grammar should be used. This approach allows an emphasis to be placed on the systematic and formulaic nature of language. A key change involved the shift to general role-based description of the usage of a word, whereas the focus before had been on declaring words to fall into strict categories (such as the eight parts of speech found in Latin).

Beginning with the work of Chomsky in the 1950s on transformational grammar, sentence diagrams, used in both structural and prescriptive approaches, slowly lost favor in educational techniques. This is due to the introduction of transformational grammar, based on generative theories and intrinsic rules of natural language structure. This generative approach is almost universally used in natural language processing, as generative rules are well-suited to computational representation. Nevertheless, both structural and transformational grammar are taught at secondary and undergraduate levels.

2.2 Applications of structural grammar

Structural grammar still has a number of advantages over generative transformational grammar. Because it is designed to emulate the natural usage of language, it is more intuitive for non-experts to understand. It also highlights certain features of sentences, such as dependency relationships between words and targets of actions. Many facets of natural language are difficult to describe using a parse tree or other generative data structure. Using structural techniques, many of these aspects are obvious upon basic analysis.

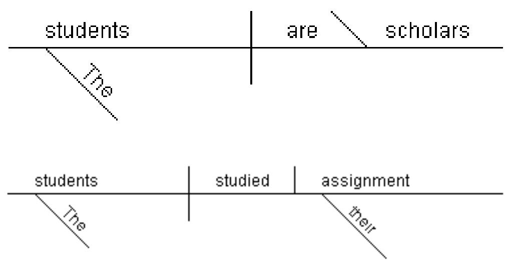


Figure 1: Diagram of "The students are scholars." and "The students studied their assignment."

By developing an algorithm to automatically analyze a sentence using structural grammar, we hope that the advantages of structural analysis can improve the performance of natural language parsers. By assigning roles to words in a sentence, patterns or structures in natural language that cannot be easily gleaned from a data structure are made obvious, highlighting the limitations of that structure. It is also important to note that while sentence diagrams are primarily used for English, they can be adapted to any language which uses subjects, verbs, and objects (word order is not important in sentence diagramming). This research can therefore be expanded into multilingual dependency parser systems in the future.

To test the effectiveness of these approaches, a system must be developed for structural analysis of sentences and subsequent conversion to a sentence diagram.

3 Sentence diagram generation algorithm

In order to generate a sentence diagram, we make use of typed dependency graphs from the Stanford dependency parser. To understand this process requires understanding both the underlying data structure representing a sentence diagram, and the conversion from a directed graph to this data structure.

3.1 Data structure

In order to algorithmically convert dependency parses to a structural grammar, we developed an original model to represent features of sentence diagrams. A sentence is composed of four slots (*Subject*, *Predicate*, *Object*, *Complement*). These slots are represented[1] in two sentences shown in Figure 1 by the words "students," "are," "assignment," and "scholars" respectively. Each slot contains three sets (*Heads*, *Expletives*, *Conjunctions*). With the exception of the *Heads* slot in *Subject* and *Predicate*, all sets may be empty. These sets are populated by words. A word is comprised of three parts: the string it represents, a set of modifying words, and information about its orientation in a diagram. Finally, anywhere that a word may fill a role, it can be replaced by a phrase or subclause. These phrases are represented identically to clauses, but all sets are allowed to be empty. Phrases and subclauses filling the role of a word are connected to the slot they are filling by a pedestal, as in Figure 2.

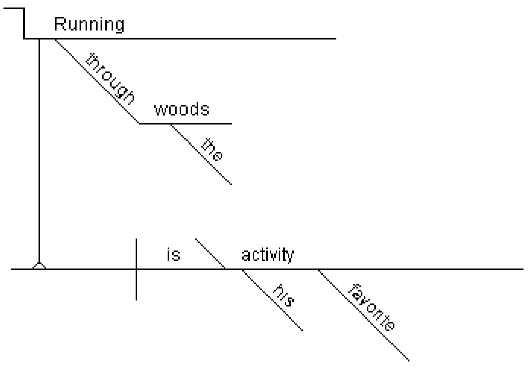


Figure 2: Diagram of "Running through the woods is his favorite activity."

3.2 Conversion from dependency graph

A typed dependency representation of a sentence contains a root – that is, a dependency relation in which neither the governor nor the dependent word in the relation is dependent in any other relation. We use this relation to determine the predicate of a sentence, which is almost always the governor of the root dependency. The dependent is added to the diagram data structure based on its relation to the governor.

Before analysis of dependency graphs begins, our algorithm takes in a set of dependency relations *S* and a set of actions (possible objects and methods to call) *A*. This paper describes an algorithm that takes in the 55 relations from (Marneffe et al., 2006) and the actions in Table 1. The algorithm then takes as input a directed graph *G* representing a sentence, composed of a node rep-

[1]All sentence diagram figures were generated by the algorithm described in this paper. Some diagrams have been edited for spacing and readability concerns. These changes do not affect their accuracy.

resenting each word in the sentence. These nodes are connected by edges in the form `reln(gov, dep)` representing a relation from S between a word `gov` and `dep`. Our algorithm performs the following steps:

1. **Determining root actions:** For each relation type $R \in S$, create an ordered list of actions $Root < R, A >$ from A to perform if that relation is the root relation in the graph.
2. **Determining regular actions:** For each relation type $R \in S$, create an ordered list of actions $Reln < R, A >$ from A to perform if R is found anywhere other than the root in G.
3. **Determining the root:** Using the root-finding process described in (Marneffe et al., 2006), find the root relation $\hat{R}(\hat{G}, \hat{D}) \in G$.
4. **Initialize a sentence diagram:** Find the set of actions$\hat{A}$ from $Root < \hat{R}, A >$and perform those actions.
5. **Finding children:** Create a set *Open* and add to it each relation $\in G$ in which $\hat{G}$ or $\hat{D}$ from step 3 is a governor.
6. **Processing children:** For each relation $\tilde{R}(\tilde{G}, \tilde{D})$ in *Open*,
   (a) **Populate the sentence diagram:** Find the set of actions $\tilde{A}$ from $Reln < \tilde{R}, A >$ and perform those actions.
   (b) **Finding children:** Add to *Open* each relation $R \in G$ in which $\tilde{G}$ or $\tilde{D}$ is a governor.

   This step continues until all relations have been found in a breadth-first order.

Our system of conversion makes the assumption that the governor of a typed dependency will already have been assigned a position in a diagram. This is due to the largely tree-like structure of dependency graphs generated by the dependency parser. Dependencies in most cases "flow" downwards to the root, and in exceptions, such as cycles, the governor will have been discovered by the time it is reached again. As we are searching for words breadth-first, we know that the dependent of any relation will have been discovered already so long as this tree-like structure holds. The number of cases where it does not is small compared to the overall error rate of the dependency parser, and does not have a large impact on the accuracy of the resulting diagram.

3.3 Single-relation analysis

A strength of this system for conversion is that information about the overall structure of a sentence is not necessary for determining the role of each individual word as it is added to the diagram. As each word is traversed, it is assigned a role relative to its parent only. This means that overall structure will be discovered naturally by tracing dependencies throughout a graph.

There is one exception to this rule: when comparing relationships of type *cop* (copula, a linking verb, usually a variant of "to be"), three words are involved: the linking verb, the subject, and the subject complement. However, instead of a transitive relationship from one word to the next, the parser assigns the subject and subject complement as dependent words of the linking verb. An example is the sentence "The students are scholars" as in Figure 1. This sentence contains three relations:

```
det(students, The)
nsubj(scholars, students)
cop(scholars, are)
```

A special case exists in our algorithm to check the governor of a *cop* relation for another relation (usually *nsubj*). This was a necessary exception to make given the frequency of linking verbs in the English language. Dependency graphs from (Marneffe et al., 2006) are defined as a singly rooted directed acyclic graph with no re-entrancies; however, they sometimes share nodes in the tree, with one word being a dependent of multiple relations. An example of this exists in the sentence "I saw the man who loves you." The word "who" in this sentence is dependent in two relations:

```
ref(man, who)
rel(loves, who)
```

We here refer to this phenomenon as breaking the tree structure. This is notable because it causes a significant problem for our approach. While the correct relation is identified and assigned in most cases, a duplicated copy of the dependent word will appear in the resulting diagram. This is because the dependent word in each relation is added to the diagram, even if it has already been added. Modifiers of these words are then assigned to each copy, which can result in large areas of duplication. We decided this duplication was acceptable

Term	Definition	Example	
		Input	Output
`GOV, DEP, RELN`	Elements of a relation	`det(``woods", ``the").GOV`	` ``woods"`
`SBJ, PRD, OBJ, CMP`	Slots in a clause	`CLAUSE.PRD`	`HEADS(``is"), EXPL(), CONJ()`
`HEADS, EXPL, CONJ`	Sets of words in a slot	`CLAUSE.PRD.HEADS()`	` ``is"`
`MODS`	Set of modifiers of a word	` ``activity".MODS`	`(``his", ``favorite")`
`SEGMENT, CLAUSE`	Set or clause of word	` ``is".SEGMENT()`	`CLAUSE.PRD`
`NEW[WORD, Slot]`	New clause constructor	`NEW(``is", PRD)`	`CLAUSE(SBJ(), PRD(``is"), OBJ(), CMP())`
`ADD(WORD[,ORIENT])`	Word added to modifiers	` ``activity".ADD(``his")`	
`APP(WORD[,RIGHT?])`	Word appended to phrasal head	` ``down".APP(``shut", false)`	
`SET(ORIENT)`	Word orientation set	` ``his".SET(DIAGONAL)`	

Periods represent ownership, parentheses represent parameters passed to a method, separated by commas, and brackets represent optional parameters.

Orientations include HORIZONTAL, DIAGONAL, VERTICAL, GERUND, BENT, DASHED, and CLAUSE as defined in (Kolln and Funk, 2002) .

Table 1: Terms and methods defined in our algorithm.

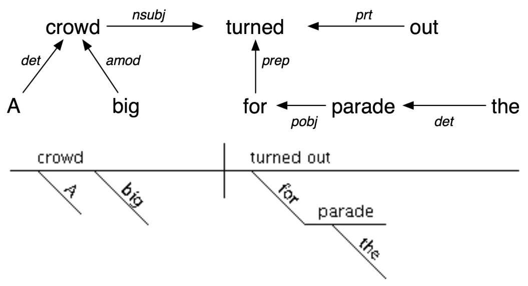


Figure 3: The sentence "A big crowd turned out for the parade." shown as a dependency graph (top) and a sentence diagram.

to maintain the simplicity of single-relation conversion rules, though remedying this problem is an avenue for further research. For testing purposes, if duplicate copies of a word exist, the correct one is given preference over the incorrect copy, and the diagram is scored as correct if either copy is correctly located.

3.4 An example diagram conversion

To illustrate the conversion process, consider the sentence "A big crowd turned out for the parade." The dependency graph for this, as generated by the Stanford dependency parser, is shown in Figure 3. The following relations are found, with the actions taken by the conversion algorithm described:

Root: **nsubj(turned, crowd)**

```
NEW(GOV, PRD);
GOV.CLAUSE.SBJ.ADD(DEP);
```

Finding Children: `det(crowd, A)`, `amod(crowd, big)`, `prt(turned, out)`, `prep(turned, for)` added to *Open*.

Relation: **det(crowd, A)**

```
GOV.ADD(DEP,DIAGONAL);
```

Relation: **amod(crowd, big)**

```
GOV.ADD(DEP,DIAGONAL);
```

Relation: **prt(turned, out)**

```
GOV.APP(DEP,TRUE);
```

Relation: **prep(turned, for)**

Finding Children: `pobj(for, parade)` added to *Open*.

```
GOV.ADD(DEP,DIAGONAL);
```

Relation: **pobj(for, parade)**

Finding Children: `det(parade, the)` added to *Open*.

```
GOV.ADD(DEP,HORIZONTAL);
```

Relation: **det(parade, the)**

```
GOV.ADD(DEP,DIAGONAL);
```

4 Experimental setup

In order to test our conversion algorithm, a large number of sentence diagrams were needed in order to ensure a wide range of structures. We decided to use an undergraduate-level English grammar textbook that uses diagramming as a teaching tool for two reasons. The first is a pragmatic matter: the sentences have already been diagrammed accurately for comparison to algorithm output. Second, the breadth of examples necessary to allow students a thorough understanding of the process is beneficial in assuring the completeness of the conversion system. Cases that are especially difficult for students are also likely to be stressed with multiple examples, giving more opportunities to determine the problem if parsers have similar difficulty.

Therefore, (Kolln and Funk, 2002) was selected to be used as the source of this testing data. This textbook contained 292 sentences, 152 from examples and 140 from solutions to problem sets. 50% of the example sentences (76 in total, chosen by selecting every other example) were set aside to use for development. The remaining 216 sentences were used to gauge the accuracy of the conversion algorithm.

Our implementation of this algorithm was developed as an extension of the Stanford dependency parser. We developed two metrics of precision to evaluate the accuracy of a diagram. The first approach, known as the *inheritance* metric, scored the results of the algorithm based on the parent of each word in the output sentence diagram. Head words were judged on their placement in the correct slot, while modifiers were judged on whether they modified the correct parent word. The second approach, known as the *orientation* metric, judged each word based solely on its orientation. This distinction judges whether a word was correctly identified as a primary or modifying element of a sentence.

These scoring systems have various advantages. By only scoring a word based on its immediate parent, a single mistake in the diagram does not severely impact the result of the score, even if it is at a high level in the diagram. Certain mistakes are affected by one scoring system but not the other; for instance, incorrect prepositional phrase attachment will not have an effect on the orientation score, but will reduce the value of the inheritance score. Alternatively, a mistake such as failing to label a modifying word as a participial modifier will reduce the orientation score, but will not reduce the value of the inheritance score. Generally, orientation scoring is more forgiving than inheritance scoring.

5 Results and discussion

The results of testing these accuracy metrics are given in Figure 4 and Table 2. Overall inheritance precision was 85% and overall orientation precision was 92%. Due to the multiple levels of analysis (parsing from tree to phrase structure to dependency graph to diagram), it is sometimes difficult to assign fault to a specific step of the algorithm.

There is clearly some loss of information when converting from a dependency graph to a sentence diagram. For example, fifteen dependency relations are represented as diagonal modifiers in a sentence diagram and have identical conversion rules. Interestingly, these relations are not necessarily grouped together in the hierarchy given in (Marneffe et al., 2006). This suggests that the syntactic information represented by these words may not be as critical as previously thought, given enough semantic information about the words. In total, six sets of multiple dependency relations mapping to the same conversion rule were found, as shown in Table 3.

The vast majority of mistakes that were made came from one of two sources: an incorrect conversion from a correct dependency parse, or a failure of the dependency parser to correctly identify a relation between words in a sentence. Both are examined below.

5.1 Incorrect conversion rules

On occasion, a flaw in a diagram was the result of an incorrect conversion from a correct interpretation in a dependency parse. In some cases, these were because of simple changes due to inaccuracies not exposed from development data. In some cases, this was a result of an overly general relationship, in which one relation correctly describes two or more possible structural patterns in sentences. This can be improved upon by specializing dependency relation descriptions in future versions of the dependency parser.

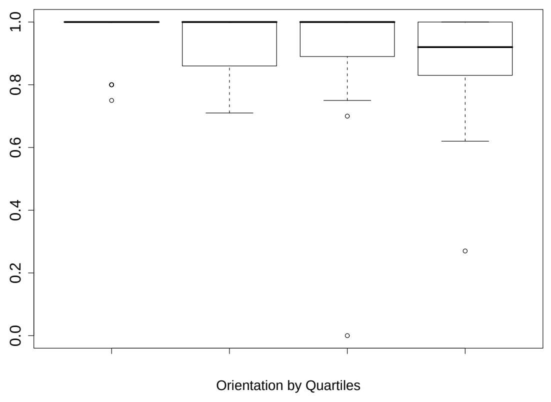
One frequent failure of the conversion rules is due to the overly generalized handling of the root of sentences. It is assumed that the governing word in the root relation of a dependency graph is the main verb of a sentence. Our algorithm has very general rules for root handling. Exceptions to these general cases are possible, especially in

Sentence Length	Ori Mean	Ori Std.Dev.	Inh Mean	Inh Std.Dev.	Count
3-6	96.61	7.42	90.34	15.20	56
7-8	92.37	15.77	86.00	19.34	57
9-10	92.80	8.18	82.73	17.15	45
11-20	89.97	12.54	82.52	15.51	58
3-20	92.91	11.84	85.51	17.05	216

Table 2: Precision of diagramming algorithm on testing data.

Relations	Rule
abbrev, *advmod*, *amod*, *dep*, *det*, *measure*, *neg*, *nn*, *num*, *number*, *poss*, *predet*, *prep*, *quantmod*, *ref*	`GOV.ADD(DEP,DIAGONAL)`
iobj, *parataxis*, *pobj*	`GOV.ADD(DEP,HORIZONTAL)`
appos, *possessive*, *prt*	`GOV.APP(DEP,TRUE)`
aux, *tmod*	`GOV.APP(DEP,FALSE)`
advcl, *csubj*, *pcomp*, *rcmod*	`GOV.ADD(NEW(DEP,PRD))`
complm, *expl*, *mark*	`GOV.SEGMENT.EXPL.ADD(DEP)`

Table 3: Sets of multiple dependency relations which are converted identically.



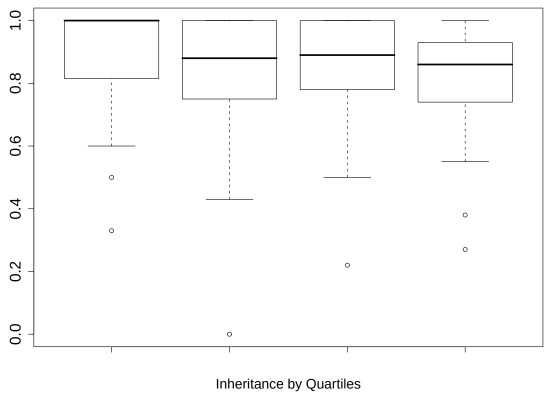


Figure 4: Inheritance (top) and Orientation precision results of diagramming algorithm on testing data. Results are separated by sentence length into quartiles.

interrogative sentences, e.g. the root relation of the sentence "What have you been reading?" is `dobj(reading, What)`. This should be handled by treating "What" as the object of the clause. This problem can be remedied in the future by creating specialized conversion rules for any given relation as a root of a dependency graph.

A final issue is the effect of a non-tree structure on the conversion algorithm. Because relationships are evaluated individually, multiple inheritance for words can sometimes create duplicate copies of a word which are then modified in parallel. An example of this is shown in Figure 5, which is caused due to the dependency graph for this sentence containing the following relations:

```
nsubj(is-4, hope-3)
xsubj(beg-6, hope-3)
xcomp(is-4, beg-6)
```

Because the tree structure is broken, a word (hope) is dependent on two different governing words. While the xsubj relation places the phrase "to beg for mercy" correctly in the diagram, a second copy is created because of the xcomp dependency. A more thorough analysis approach that checks for breaking of the tree structure may be useful in avoiding this problem in the future.

5.2 Exposed weaknesses of dependency parsers

A number of consistent patterns are poorly diagrammed by this system. This is usually due to

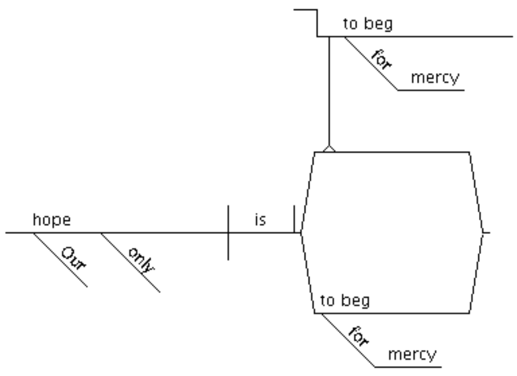


Figure 5: Duplication in the sentence diagram for "Our only hope is to beg for mercy."

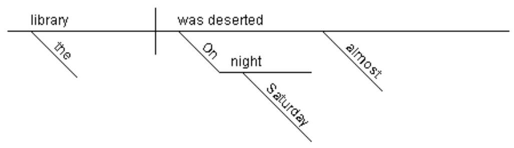


Figure 6: Diagram of "On Saturday night the library was almost deserted."

limitations in the theoretical model of the dependency parser. These differences between the actual structure of the sentence and the structure the parser assigns can lead to a significant difference in semantic value of phrases. Improving the accuracy of this model to account for these situations (either through more fine-grained separation of relationships or a change in the model) may improve the quality of meaning extraction from sentences.

One major shortcoming of the dependency parser is how it handles prepositional phrases. As described in (Atterer and Schutze, 2007), this problem has traditionally been framed as involving four words (v, n1, p, n2) where v is the head of a verb phrase, n1 is the head of a noun phrase dominated by v, p is the head of a prepositional phrase, and n2 the head of a noun phrase dominated by p. Two options have generally been given for attachment, either to the verb v or the noun n1. This parser struggles to accurately determine which of these two possibilities should be used. However, in the structural model of grammar, there is a third option, treating the prepositional phrase as an object complement of n1. This possibility occurs frequently in English, such as in the sentence "We elected him as our secretary." or with idiomatic expressions such as "out of tune." The current dependency parser cannot represent this at all.

5.3 Ambiguity

A final case is when multiple correct structural analyses exist for a single sentences. In some cases, this causes the parser to produce a gramatically and semantically correct parse which, due to ambiguity, does not match the diagram for comparison. An example of this can be seen in Figure 6, in which the dependency parser assigns the predicate role to "was deserted" when in fact deserted is acting as a subject complement. However, the phrase "was deserted" can accurately act as a predicate in that sentence, and produces a semantically valid interpretation of the phrase.

6 Conclusion

We have demonstrated a promising method for conversion from a dependency graph to a sentence diagram. However, this approach still has the opportunity for a great deal of improvement. There are two main courses of action for future work to reap the benefits of this approach: analyzing current results, and extending this approach to other parsers for comparison. First, a more detailed analysis of current errors should be undertaken to determine areas for improvement. There are two broadly defined categories of error (errors made before a dependency graph is given to the algorithm for conversion, and errors made during conversion to a diagram). However, we do not know what percent of mistakes falls into those two categories. We also do not know what exact grammatical idiosyncracy caused each of those errors. With further examination of current data, this information can be determined.

Second, it must be determined what level of conversion error is acceptable to begin making quantitative comparisons of dependency parsers. Once the level of noise introduced by the conversion process is lowered to the point that the majority of diagram errors are due to mistakes or shortfalls in the dependency graph itself, this tool will be much more useful for evaluation. Finally, this system should be extended to other dependency parsers so that a comparison can be made between multiple systems.

References

Michaela Atterer and Hinrich Schutze. 2007. Prepositional Phrase Attachment without Oracles. In *Com-*

putational Linguistics.

John Carroll, Guido Minnen, and Ted Briscoe. 1999. Corpus annotation for parser evaluation. In *Proceedings of the EACL workshop on Linguistically Interpreted Corpora.*

Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics.*

Martha Kolln and Robert Funk. 2002. Understanding English Grammar, Sixth Edition. *Longman Publishers*.

Dekang Lin. 1995. A Dependency-based Method for Evaluating Broad-Coverage Parsers. In *Natural Language Engineering*.

Dekang Lin. 1998. Dependency-based evaluation of MINIPAR. In *Workshop on the Evaluation of Parsing Systems*

Marie-Catherine de Marneffe, Bill MacCartney and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In *International Conference on Language Resources and Evaluation.*

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing.*

Daniel D. Sleator and Davy Temperley. 1993. Parsing English with a link grammar. In *Third International Conference on Parsing Technologies.*