Converting Dependency Structures to Phrase Structures

Fei Xia and Martha Palmer
University of Pennsylvania
Philadelphia, PA 19104, USA
{fxia,mpalmer}@linc.cis.upenn.edu

1. INTRODUCTION

Treebanks are of two types according to their annotation schemata: phrase-structure Treebanks such as the English Penn Treebank [8] and dependency Treebanks such as the Czech dependency Treebank [6]. Long before Treebanks were developed and widely used for natural language processing, there had been much discussion of comparison between dependency grammars and context-free phrase-structure grammars [5]. In this paper, we address the relationship between dependency structures and phrase structures from a practical perspective; namely, the exploration of different algorithms that convert dependency structures to phrase structures and the evaluation of their performance against an existing Treebank. This work not only provides ways to convert Treebanks from one type of representation to the other, but also clarifies the differences in representational coverage of the two approaches.

2. CONVERTING PHRASE STRUCTURES TO DEPENDENCY STRUCTURES

The notion of *head* is important in both phrase structures and dependency structures. In many linguistic theories such as X-bar theory and GB theory, each phrase structure has a head that determines the main properties of the phrase and a head has several levels of projections; whereas in a dependency structure the head is linked to its dependents. In practice, the head information is explicitly marked in a dependency Treebank, but not always so in a phrase-structure Treebank. A common way to find the head in a phrase structure is to use a head percolation table, as discussed in [7, 1] among others. For example, the entry (S right S/VP) in the head percolation table says that the head child[1] of an S node is the first child of the node from the right with the label S or VP.

Once the heads in phrase structures are found, the conversion from phrase structures to dependency structures is straightforward, as shown below:

(a) Mark the head child of each node in a phrase structure, using the head percolation table.

[1]The *head-child* of a node XP is the child of the node XP that is the ancestor of the head of the XP in the phrase structure.

.

(b) In the dependency structure, make the head of each non-head-child depend on the head of the head-child.

Figure 1 shows a phrase structure in the English Penn Treebank [8]. In addition to the syntactic labels (such as NP for a noun phrase), the Treebank also uses function tags (such as SBJ for the subject) for grammatical functions. In this phrase structure, the root node has two children: the NP and the VP. The algorithm would choose the VP as the head-child and the NP as a non-head-child, and make the head *Vinkin* of the NP depend on the head *join* of the VP in the dependency structure. The dependency structure of the sentence is shown in Figure 2. A more sophisticated version of the algorithm (as discussed in [10]) takes two additional tables (namely, the argument table and the tagset table) as input and produces dependency structures with the argument/adjunct distinction (i.e., each dependent is marked in the dependency structure as either an argument or an adjunct of the head).

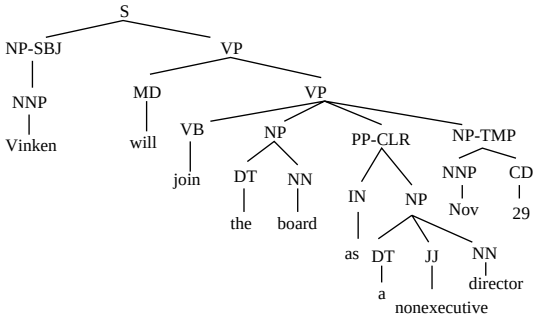


Figure 1: A phrase structure in the Penn Treebank

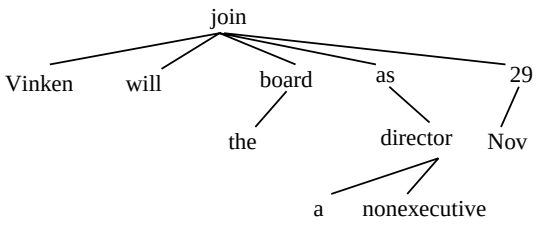


Figure 2: A dependency tree for the sentence in Figure 1. Heads are marked as parents of their dependents in an ordered tree.

It is worth noting that quite often there is no consensus on what the *correct* dependency structure for a particular sentence should be. To build a dependency Treebank, the Treebank annotators must decide which word depends on which word; for example, they have to decide whether the subject *Vinken* in Figure 1 depends on the

modal verb *will* or the main verb *join*. In contrast, the annotators for phrase-structure Treebanks do not have to make such decisions. The users of phrase-structure Treebanks can modify the head percolation tables to get different dependency structures from the same phrase structure. In other words, phrase structures offer more flexibility than dependency structures with respect to the choices of heads.

The feasibility of using the head percolation table to identify the heads in phrase structures depends on the characteristics of the language, the Treebank schema, and the definition of the correct dependency structure. For instance, the head percolation table for a strictly head-final (or head-initial) language is very easy to build, and the conversion algorithm works very well. For the English Penn Treebank, which we used in this paper, the conversion algorithm works very well except for the noun phrases with the appositive construction. For example, the conversion algorithm would choose the appositive *the CEO of FNX* as the head child of the phrase *John Smith, the CEO of FNX*, whereas the correct head child should be *John Smith*.

3. CONVERTING DEPENDENCY STRUCTURES TO PHRASE STRUCTURES

The main information that is present in phrase structures but not in dependency structures is the type of syntactic category (e.g., NP, VP, and S); therefore, to recover syntactic categories, any algorithm that converts dependency structures to phrase structures needs to address the following questions:

Projections for each category: for a category X, what kind of projections can X have?

Projection levels for dependents: Given a category Y depends on a category X in a dependency structure, how far should Y project before it attaches to X's projection?

Attachment positions: Given a category Y depends on a category X in a dependency structure, to what position on X's projection chain should Y's projection attach?

In this section, we discuss three conversion algorithms, each of which gives different answers to these three questions. To make the comparison easy, we shall apply each algorithm to the dependency structure (*d-tree*) in Figure 2 and compare the output of the algorithm with the phrase structure for that sentence in the English Penn Treebank, as in Figure 1.

Evaluating these algorithms is tricky because just like dependency structures there is often no consensus on what the *correct* phrase structure for a sentence should be. In this paper, we measure the performance of the algorithms by comparing their output with an existing phrase-structure Treebank (namely, the English Penn Treebank) because of the following reasons: first, the Treebank is available to the public, and provides an objective although imperfect standard; second, one goal of the conversion algorithms is to make it possible to compare the performance of parsers that produce dependency structures with the ones that produce phrase structures. Since most state-of-the-art phrase-structure parsers are evaluated against an existing Treebank, we want to evaluate the conversion algorithms in the same way; third, a potential application of the conversion algorithms is to help construct a phrase-structure Treebank for one language, given parallel corpora and the phrase structures in the other language. One way to evaluate the quality of the resulting Treebank is to compare it with an existing Treebank.

3.1 Algorithm 1

According to X-bar theory, a category X projects to X', which further projects to XP. There are three types of rules, as shown in Figure 3(a). Algorithm 1, as adopted in [4, 3], strictly follows X-bar theory and uses the following heuristic rules to build phrase structures:

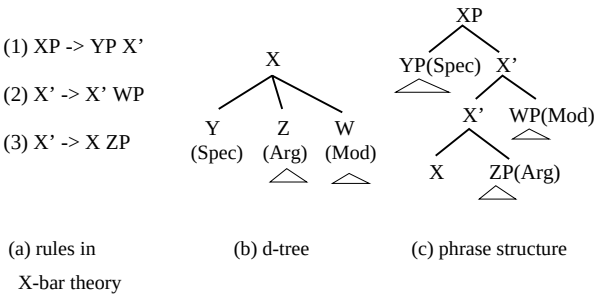


Figure 3: Rules in X-bar theory and Algorithm 1 (which is based on it)

Two levels of projections for any category: any category X has two levels of projection: X' and XP.

Maximal projections for dependents: a dependent Y always projects to Y' then YP, and the YP attaches to the head's projection.

Fixed positions of attachment: Dependents are divided into three types: specifiers, modifiers, and arguments. Each type of dependent attaches to a fixed position, as shown in Figure 3(c).

The algorithm would convert the d-tree in Figure 3(b) to the phrase structure in Figure 3(c). If a head has multiple modifiers, the algorithm could use either a single X' or stacked X' [3]. Figure 4 shows the phrase structure for the d-tree in Figure 2, where the algorithm uses a single X' for multiple modifiers of the same head.[2]

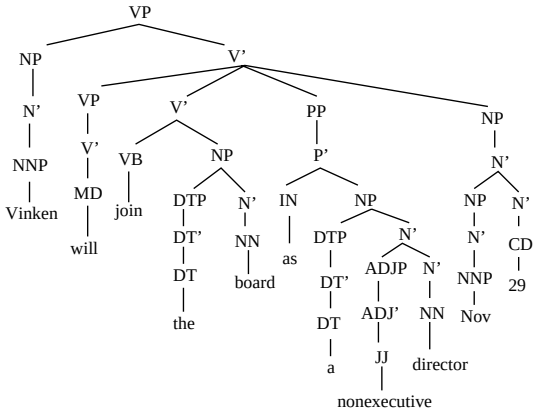


Figure 4: The phrase structure built by algorithm 1 for the d-tree in Figure 2

3.2 Algorithm 2

Algorithm 2, as adopted by Collins and his colleagues [2] when they converted the Czech dependency Treebank [6] into a phrase-structure Treebank, produces phrase structures that are as flat as possible. It uses the following heuristic rules to build phrase structures:

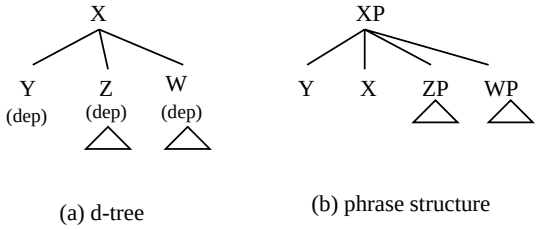
One level of projection for any category: X has only one level of projection: XP.

[2]To make the phrase structure more readable, we use N' and NP as the X' and XP for all kinds of POS tags for nouns (e.g., NNP, NN, and CD). Verbs and adjectives are treated similarly.

Minimal projections for dependents: A dependent Y does not project to YP unless it has its own dependents.

Fixed position of attachment: A dependent is a sister of its head in the phrase structure.[3]

The algorithm treats all kinds of dependents equally. It converts the pattern in Figure 5(a) to the phrase structure in Figure 5(b). Notice that in Figure 5(b), Y does not project to YP because it does not have its own dependents. The resulting phrase structure for the d-tree in Figure 2 is in Figure 6, which is much flatter than the one produced by Algorithm 1.



(a) d-tree (b) phrase structure

Figure 5: The scheme for Algorithm 2

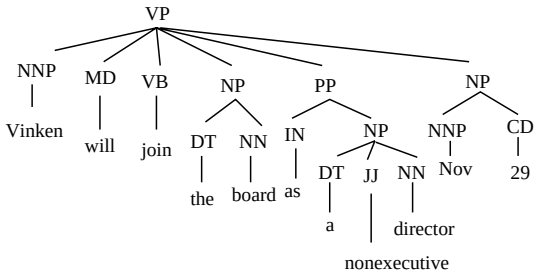


Figure 6: The phrase structure built by Algorithm 2 for the d-tree in Figure 2

3.3 Algorithm 3

The previous two algorithms are linguistically sound. They do not use any language-specific information, and as a result there are several major differences between the output of the algorithms and the phrase structures in an existing Treebank, such as the Penn English Treebank (PTB).

Projections for each category: Both algorithms assume that the numbers of projections for all the categories are the same, whereas in the PTB the number of projections varies from head to head. For example, in the PTB, determiners do not project, adverbs project only one level to adverbial phrases, whereas verbs project to VP, then to S, then to SBAR.[4]

Projection levels for dependents: Algorithm 1 assumes the maximal projections for all the dependents, while Algorithm 2 assumes minimal projections; but in the PTB, the level of projection of a dependent may depend on several factors such as the categories of the dependent and the head, the position of the dependent with respect to the head, and the dependency type. For example, when a noun modifies a verb (or VP) such as *yesterday* in *he came yesterday*, the noun always projects to NP, but when a noun N_1 modifiers another noun N_2, N_1 projects to NP if N_1 is to the right of N_2 (e.g., in an appositive construction) and it does not project to NP if N_1 is to the left of N_2.

Attachment positions: Both algorithms assume that all the dependents of the same dependency type attach at the same level (e.g., in Algorithm 1, modifiers are sisters of X', while in Algorithm 2, modifiers are sisters of X); but in the PTB, that is not always true. For example, an ADVP, which depends on a verb, may attach to either an S or a VP in the phrase structure according to the position of the ADVP with respect to the verb and the subject of the verb. Also, in noun phrases, left modifiers (e.g., JJ) are sisters of the head noun, while the right modifiers (e.g., PP) are sisters of NP.

For some applications, these differences between the Treebank and the output of the conversion algorithms may not matter much, and by no means are we implying that an existing Treebank provides the gold standard for what the phrase structures should be. Nevertheless, because the goal of this work is to provide an algorithm that has the flexibility to produce phrase structures that are as close to the ones in an existing Treebank as possible, we propose a new algorithm with such flexibility. The algorithm distinguishes two types of dependents: arguments and modifiers. The algorithm also makes use of language-specific information in the form of three tables: the projection table, the argument table, and the modification table. The projection table specifies the projections for each category. The argument table (the modification table, resp.) lists the types of arguments (modifiers, resp) that a head can take and their positions with respect to the head. For example, the entry $V \rightarrow VP \rightarrow S$ in the projection table says that a verb can project to a verb phrase, which in turn projects to a sentence; the entry (P 0 1 NP/S) in the argument table indicates that a preposition can take an argument that is either an NP or an S, and the argument is to the right of the preposition; the entry (NP DT/JJ PP/S) in the modification table says that an NP can be modified by a determiner and/or an adjective from the left, and by a preposition phrase or a sentence from the right.

Given these tables, we use the following heuristic rules to build phrase structures:[5]

One projection chain per category: Each category has a unique projection chain, as specified in the projection table.

Minimal projection for dependents: A category projects to a higher level only when necessary.

Lowest attachment position: The projection of a dependent attaches to a projection of its head as lowly as possible.

The last two rules require further explanation, as illustrated in Figure 7. In the figure, the node X has three dependents: Y and Z are arguments, and W is a modifier of X. Let's assume that the algorithm has built the phrase structure for each dependent. To form the phrase structure for the whole d-tree, the algorithm needs to attach the phrase structures for dependents to the projection chain $X^0, X^1, ...X^k$ of the head X. For an argument such as Z, suppose its projection chain is $Z^0, Z^1, ...Z^u$ and the root of the phrase structure headed by Z is Z^s. The algorithm would find the lowest position X^h on the head projection chain, such that Z has a projection Z^t that can be an argument of X^{h-1} according to the argument table and Z^t is no lower than Z^s on the projection chain for Z. The algorithm then makes Z^t a child of X^h in the phrase structure. Notice that based on the second heuristic rule (i.e., minimal projection for dependents), Z^t does not further project to Z^u in

[3] If a dependent Y has its own dependents, it projects to YP and YP is a sister of the head X; otherwise, Y is a sister of the head X.

[4] S is similar to IP (IP is the maximal projection of INFL) in GB theory, so is SBAR to CP (CP is the maximal projection of Comp); therefore, it could be argued that only VP is a projection of verbs in the PTB. Nevertheless, because PTB does not mark INFL and Comp, we treat S and SBAR as projections of verbs.

[5] In theory, the last two heuristic rules may conflict each other in some cases. In those cases, we prefer the third rule over the second. In practice, such conflicting cases are very rare, if exist.

this case although Z^u is a valid projection of Z. The attachment for modifiers is similar except that the algorithm uses the modification table instead of the argument table.[6]

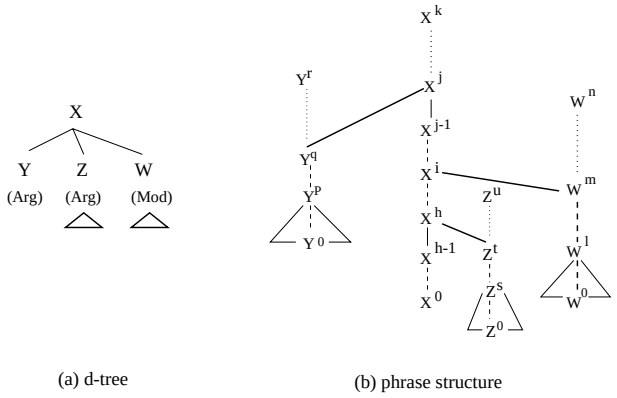


Figure 7: The scheme for Algorithm 3

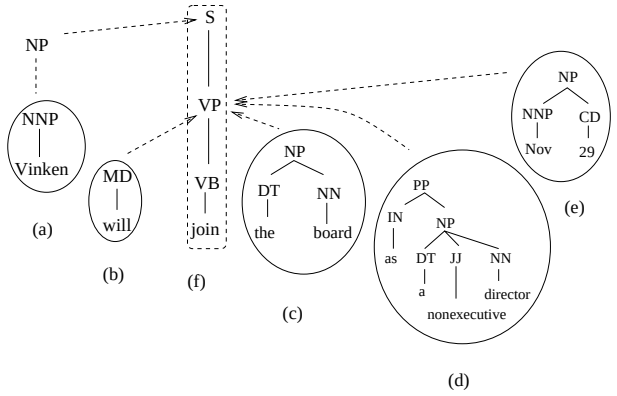


Figure 8: The phrase structure produced by Algorithm 3

The phrase structure produced by Algorithm 3 for the d-tree in Figure 2 is in Figure 8. In Figure 8, (a)-(e) are the phrase structures for five dependents of the head *join*; (f) is the projection chain for the head. The arrows indicate the positions of the attachment. Notice that to attach (a) to (f), the NNP *Vinken* needs to further project to NP because according to the argument table, a VP can take an NP, but not an NNP, as its argument.

In the PTB, a modifier either sister-adjoins or Chomsky-adjoins to the modifiee. For example, in Figure 1, the MD *will* Chomsky-adjoins whereas the NP *Nov. 29* sister-adjoins to the VP node. To account for that, we distinguish these two types of modifiers in the modification table and Algorithm 3 is extended so that it would attach Chomsky-adjoining modifiers higher by inserting extra nodes. To convert the d-tree in Figure 2, the algorithm inserts an extra VP node in the phrase structure in Figure 8 and attaches the MD *will* to the new VP node; the final phrase structure produced by the algorithm is identical to the one in Figure 1.

3.4 Algorithm 1 and 2 as special cases of Algorithm 3

[6]Note that once Z^t becomes a child of X^h, other dependents of X (such as W) that are on the same side as Z but are further away from X can attach only to X^h or higher on the projection chain of X.

Although the three algorithms adopt different heuristic rules to build phrase structures, the first two algorithms are special cases of the last algorithm; that is, we can design a distinct set of projection/argument/modification tables for each of the first two algorithms so that running Algorithm 3 with the associated set of tables for Algorithm 1 (Algorithm 2, respectively) would produce the same results as running Algorithm 1 (Algorithm 2, respectively).

For example, to produce the results of Algorithm 2 with the code for Algorithm 3, the three tables should be created as follows:

(a) In the projection table, each head X has only one projection XP;

(b) In the argument table, if a category Y can be an argument of a category X in a d-tree, then include both Y and YP as arguments of X;

(c) In the modification table, if a category Y can be a modifier of a category X in a d-tree, then include both Y and YP as sister-modifiers of XP.

4. EXPERIMENTS

So far, we have described two existing algorithms and proposed a new algorithm for converting d-trees into phrase structures. As explained at the beginning of Section 3, we evaluated the performance of the algorithms by comparing their output with an existing Treebank. Because there are no English dependency Treebanks available, we first ran the algorithm in Section 2 to produce d-trees from the PTB, then applied these three algorithms to the d-trees and compared the output with the original phrase structures in the PTB.[7] The process is shown in Figure 9.

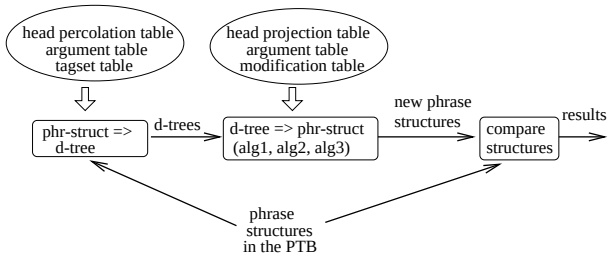


Figure 9: The flow chart of the experiment

The results are shown in Table 1, which use Section 0 of the PTB. The precision and recall rates are for unlabelled brackets. The last column shows the ratio of the number of brackets produced by the algorithms and the number of brackets in the original Treebank. From the table (especially the last column), it is clear that Algorithm 1 produces many more brackets than the original Treebank, resulting in a high recall rate but low precision rate. Algorithm 2 produces very flat structures, resulting in a low recall rate and high precision rate. Algorithm 3 produces roughly the same number of brackets as the Treebank and has the best recall rate, and its precision rate is almost as good as that of Algorithm 2.

The differences between the output of the algorithms and the phrase structures in the PTB come from four sources:

(S1) Annotation errors in the PTB

(S2) Errors in the Treebank-specific tables used by the algorithms in Sections 2 and 3 (e.g., the head percolation table, the projection table, the argument table, and the modification table)

[7]Punctuation marks are not part of the d-trees produced by LexTract. We wrote a simple program to attach them as high as possible to the phrase structures produced by the conversion algorithms.

	recall (%)	prec (%)	no-cross (%)	ave cross	test/ gold
Alg1	81.34	32.81	50.81	0.90	2.48
Alg2	54.24	91.50	94.90	0.10	0.59
Alg3	86.24	88.72	84.33	0.27	0.98

Table 1: Performance of three conversion algorithms on the Section 0 of the PTB

(S3) The imperfection of the conversion algorithm in Section 2 (which converts phrase structures to d-trees)

(S4) Mismatches between the heuristic rules used by the algorithms in Section 3 and the annotation schemata adopted by the PTB

To estimate the contribution of (S1)–(S4) to the differences between the output of Algorithm 3 and the phrase structures in the PTB, we manually examined the first twenty sentences in Section 0. Out of thirty-one differences in bracketing, seven are due to (S1), three are due to (S2), seven are due to (S3), and the remaining fourteen mismatches are due to (S4).

While correcting annotation errors to eliminate (S1) requires more human effort, it is quite straightforward to correct the errors in the Treebank-specific tables and therefore eliminate the mismatches caused by (S2). For (S3), we mentioned in Section 2 that the algorithm chose the wrong heads for the noun phrases with the appositive construction. As for (S4), we found several exceptions (as shown in Table 2) to the one-projection-chain-per-category assumption (i.e., for each POS tag, there is a unique projection chain), an assumption which was used by all three algorithms in Section 3. The performance of the conversion algorithms in Section 2 and 3 could be improved by using additional heuristic rules or statistical information. For instance, Algorithm 3 in Section 3 could use a heuristic rule that says that an adjective (JJ) projects to an NP if the JJ follows the determiner *the* and the JJ is not followed by a noun as in *the rich are getting richer*, and it projects to an ADJP in other cases. Notice that such heuristic rules are Treebank-dependent.

most likely projection	other projection(s)
JJ $\rightarrow$ ADJP	JJ $\rightarrow$ NP
CD $\rightarrow$ NP	CD $\rightarrow$ QP $\rightarrow$ NP
VBN $\rightarrow$ VP $\rightarrow$ S	VBN $\rightarrow$ VP $\rightarrow$ RRC
NN $\rightarrow$ NP	NN $\rightarrow$ NX $\rightarrow$ NP
VBG $\rightarrow$ VP $\rightarrow$ S	VBG $\rightarrow$ PP

Table 2: Some examples of heads with more than one projection chain

Empty categories are often explicitly marked in phrase-structures, but they are not always included in dependency structures. We believe that including empty categories in dependency structures has many benefits. First, empty categories are useful for NLP applications such as machine translation. To translate a sentence from one language to another, many machine translation systems first create the dependency structure for the sentence in the source language, then produce the dependency structure for the target language, and finally generate a sentence in the target language. If the source language (e.g., Chinese and Korean) allows argument deletion and the target language (e.g., English) does not, it is crucial that the dropped argument (which is a type of empty category) is explicitly marked in the source dependency structure, so that the machine translation systems are aware of the existence of the dropped argument and can handle the situation accordingly. The second benefit of including empty categories in dependency structures is that it can improve the performance of the conversion algorithms in Section 3, because the phrase structures produced by the algorithms would then have empty categories as well, just like the phrase structures in the PTB. Third, if a sentence includes a non-projective construction such as wh-movement in English, and if the dependency tree did not include an empty category to show the movement, traversing the dependency tree would yield the wrong word order.[8]

5. CONCLUSION

We have proposed a new algorithm for converting dependency structures to phrase structures and compared it with two existing ones. We have shown that our algorithm subsumes the two existing ones. By using simple heuristic rules and taking as input certain kinds of Treebank-specific information such as the types of arguments and modifiers that a head can take, our algorithm produces phrase structures that are very close to the ones in an annotated phrase-structure Treebank; moreover, the quality of the phrase structures produced by our algorithm can be further improved when more Treebank-specific information is used. We also argue for including empty categories in the dependency structures.

6. ACKNOWLEDGMENTS

This research reported here was made possible by NSF under Grant NSF-89-20230-15 and by DARPA as part of the Translingual Information Detection, Extraction and Summarization (TIDES) program under Grant N66001-00-1-8915.

7. REFERENCES

[1] M. Collins. Three Generative, Lexicalised Models for Statistical Parsing. In *Proc. of the 35th ACL*, 1997.

[2] M. Collins, J. Hajič, L. Ramshaw, and C. Tillmann. A Statistical Parser for Czech. In *Proc. of ACL-1999*, pages 505–512, 1999.

[3] M. Covington. An Empirically Motivated Reinterpretation of Dependency Grammar, 1994. Research Report AI-1994-01.

[4] M. Covington. GB Theory as Dependency Grammar, 1994. Research Report AI-1992-03.

[5] H. Gaifman. Dependency Systems and Phrase-Structure Systems. *Information and Control*, pages 304–337, 1965.

[6] J. Hajič. Building a Syntactically Annotated Corpus: The Prague Dependency Treebank, 1998. Issues of Valency and Meaning (Festschrift for Jarmila Panevová.

[7] D. M. Magerman. Statistical Decision-Tree Models for Parsing. In *Proc. of the 33rd ACL*, 1995.

[8] M. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a Large Annotated Corpus of English: the Penn Treebank. *Computational Lingustics*, 1993.

[9] O. Rambow and A. K. Joshi. A formal look at dependency grammars and phrase structure grammars with special considertaion of word-order phenomena. In L. Wenner, editor, *Recent Trends in Meaning-Text Theory*. John Benjamin, Amsterdam, Philadelphia, 1997.

[10] F. Xia, M. Palmer, and A. Joshi. A Uniform Method of Grammar Extraction and its Applications. In *Proc. of Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC)*, 2000.

[8] For more discussion of non-projective constructions, see [9].