Span-based Statistical Dependency Parsing of Chinese

Tom B.Y. Lai,[*†] C.N. Huang,[†] Ming Zhou,[†] Jiangbo Miao[†] and Tony K.C. Siu[†]
[*]City University of Hong Kong and [†]Microsoft Research China
cttomlai@cityu.edu.hk, {cnhuang, mingzhou}@microsoft.com

Abstract

Using non-constituent spans, we have applied the CYK algorithm to statistical dependency parsing of Chinese, with an additional measure to avoid duplicated efforts. We used a rule-based Chinese parser to generate a training corpus of functionally labelled dependency annotations. In this paper, we describe our span-based CYK algorithm and report initial experimental results.

1 Introduction

Lexicalized (or bilexical) statistical parsing has been gaining popularity. Essentially, this involves the calculation of probabilities of syntactic structures on the basis of dependency relations between lexical items. Recently, an efficient $O(n^3)$ parsing algorithm based on a non-constituent construct called "span" has been suggested by Eisner (1996, 1997, 2000). We have made an initial attempt to use this algorithm in a statistical parser trained using a relatively large Chinese training set. To generate this training set, we used a rule-based Chinese parser (Zhou, 2000) to produce "block" dependency annotations.

1.1 Dependency Analysis

Dependency analysis of human languages has been studied by Tesnière (1959), Gaifman (1965), Hays (1964) and Robinson (1970).

Instead of focusing on constituent structures, dependency analysis produces an account of the dependencies between linguistic units. Dependencies are asymetrical governor-dependent relations carrying labels of grammatical functions (e.g. *subject*, *object*). In the Gaifman-Hays-Robinson tradition, dependency structures observe the following constraints:

- There are no crossing links.
- There are no cycles.
- There are no multiple governors (parents).

Dependency structures observing these constraints can be transformed into trees (as produced by context-free grammars), but conversion of trees into dependency structures requires the heads of "constituents" to be explictly marked.

Phrase structures dominate mainstream modern linguistics theory, but dependency analysis has also been employed by computational linguists. In particular, they have been used in probabilistic parsing because of the relatively low training cost made possible by handling binary relations between linguistic units directly (cf. more traditional stochastic phrase structure grammars).

1.2 Dependency-based Statistical Parsing

Early attempts in the use of dependency in natural language processing include Courtin (1973), Hellwig (1986) and, for the analysis of Chinese, Yuan et al. (1992).

Interest in dependency analysis received a boost when Collins (1996) suggested using bilexical dependency in the calculation of probabilities of phrase-structure trees.

Bilexical dependency probability models are easier to train than phrase-structure probabilistic models, but locating the headword in a constituent is a source of inefficiency. Eisner (1996, 1997, 2000) suggests using "non-constituent" spans in a CYK parsing algorithm. Doing so reduces complexity from $O(n^5)$ to $O(n^3)$.

Lee and Choi (2000) and Seo et al. (2000) have studied span-based dependency parsing for Korean. They have found faithful adherence to the algorithm somewhat expensive.

Bikel (2000) has applied bilexical statistical parsing to Chinese. However, he has not used spans.

We have made an initial attempt at span-based probabilistic Chinese dependency parsing. We have added a measure to avoid duplicated efforts in the parsing process. By using a rule-based parser to generate dependency structures, we have also been able to perform tests with a relatively large training set.

2 Probabilistic Model

2.1 Statistical Dependency Analysis

In our model, the input string is a sequence of <word, POS-tag> pairs (representing heads of chunks). BOP ('beginning of phrase') is added to the beginning.

$$S = \{BOP, <w_1, t_1>, <w_2, t_2>, \ldots, <w_n, t_n>\}$$

While Einser (1996) considers only "bare-bone" dependencies and Collins (1996) takes categories of headwords of constituents for functions, we incorporate grammatical functions (e.g. *subject*, *object*) in our model. Besides, the most recent sister (on the same side of the parent) is also included in the context (history) as a mild effort to capture the subcategorization requirements of the parent. Thus, for the j'th word (headword of the jth chunk in our experiment):

Dependency:
$D(j) = <h_j, R_j, s_j, R_{s_j}>$
or, when the intervening sister is not available:
$D'(j) = <h_j, R_j>$
or, with the head of the phrase:
$D''(j) = <BOP, R_j>$

where h_j = head, s_j = previous sister (same side), R_j and R_{s_j} are grammatical functions. BOP is a special entity introduced to represent the beginning of the "phrase".

The dependency analysis (the parse) of an input is

$$D = \bigcup_{j=1}^{n} \{D(j)\} \text{ or } \{D'(j)\} \text{ or } \{D''(j)\}$$

(D'' can only occur once for the whole input; D' can only occur once on each side of parent. No crossing; no cycles; no multiple parents.)

The probability of a dependency analysis (given the input S) is:

$$P(D|S) = \prod_{j=1}^{n} P(D(j)|S) \text{ or } P(D'(j)|S) \text{ or } P(D''(j)|S)$$

2.2 Conditional Probabilities

For the conditional probabilities, Eisner (1996) considers $R, <w,t>$ to be generated by the context (history) of $<H, S, R_s>$.

We follow Collins (1996) in considering a relation R to be conditioned by the context (history) of $<<w,t>, <w_h, t_h>, <w_s, t_s>, R_s>$.

In the full context (when not head of "phrase" and when there are intervening sisters), the probability of a relation R is:

$$P(R \mid <w,t>, <w_h, t_h>, <w_s, t_s>, R_s)$$

Its estimation is:

$$\begin{aligned}&\hat{P}(R \mid <w,t>, <w_h, t_h>, <w_s, t_s>, R_s)\\ &= C(R, <w,t>, <w_h, t_h>, <w_s, t_s>, R_s) / \\ &\quad C(<w,t>, <w_h, t_h>, <w_s, t_s>, R_s).\end{aligned}$$

backing off to t, t_h and t_s when necessary. (Similar formulae for incomplete contexts with no intervening sisters or when the word is head of "phrase".)

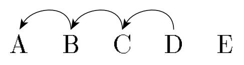
To simplify things, we do not supplement this model with a trigram Markov model to generate $<w,j>$ as in Eisner's (1996) Model A.

3 Span-based CYK Parsing

We use Eisner's (1996) "non-constituent spans" and $O(n^3)$ CYK parsing algorithm.

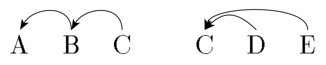
A span is a sequence of contiguous words in the input. We follow Eisner and require that except for the boundary vertices, vertices in the span should be "inert". That is to say, they are not allowed to be linked to any vertices outside the span.

For example, adding a link from E to internal vertices of the span $A \ldots D$ is not possible:

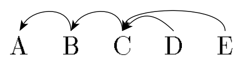


(Arrows point from dependents to governors.)

The internal vertices B and C are by definition "inert". They are not allowed to be linked to outside objects. We can however combine the two spans



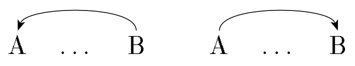
to form:



We always build a composite span from two constituent spans. It is thus possible to use a CYK-like algorithm to derive the best dependency structure of an input.

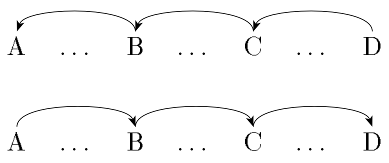
3.1 Nomenclature and Types of (Regular) Spans

1. *Covered span.* When a dependency link exists between the end vertices.



We call the link in a covered span a "covering link".

2. *Chain span.* A sequence of *covered spans*, with the same direction of dependency, and with over-lapping boundary vertices.



Covered spans are special cases of chain spans.

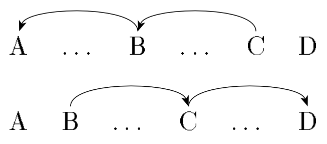
3. Covered spans and chain spans are left-headed or right-headed depending on whether the head of the span is the start or end vertex.

4. A legal span may be:

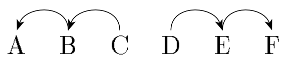
   (a) A chain span (possibly a covered span); or

   (b) Two adjacent unlinked vertices; or

   (c) A chain and a vertex; or



   (d) Two chains.



3.2 Signature of a Span

The "signature" of a span includes the following combinatorial features:

1. Start and end positions (explicit handling not required in CYK-like parsing)

2. Whether start vertex has a parent in span; end vertex has a parent in span; or neither (never both)

3. Whether the span is "minimal" ("simple"), see later section.

and the following information:

1. Right-most daughter of start vertex, with dependency name

2. Left-most daughter of end vertex, with dependency name

3.3 Implementation of Spans

Spans have lengths ≥ 2. All vertices in the interior of a span must have parents in the span. They are are not allowed to be linked to vertices outside the span. Only endwords (boundary vertices) are considered when two spans are combined ("covered-concatenated"). The structure of a span is:

$<$start, end, dependencies, combinatory-features, whether-minimal, probability, sister-info, head/non-const. ...$>$

"Spans" in $cell(1, n)$ is a headed structure. Otherwise, spans are not "constituents".

3.4 CYK-like Parsing

	1	2	3	4	5	6
6	(1, 6)					
5	(1, 5)	(2, 5)				
4	(1, 4)	(2, 4)	(3, 4)			
3	(1, 3)	(2, 3)	(3, 3)	(4, 3)		
2	(1, 2)	(2, 2)	(3, 2)	(4, 2)	(5, 2)	

1. The input sequence of blocks/chunks are indexed $i = 1, 2, \ldots, n$.

2. BOP (and EOP) is added as $i = 0$ (and $i = n + 1$).

3. The second axis k represents the number of vertices in the span, starting with $k = 2$, and ending with $k = n - i + 1$.

4. For $k = 2$, $\forall i < n$, fill in the cell with appropriate *initial spans* (see Section 3.7).

5. Loop begins: for $k = 3$ to $k = n - 1$.

6. For i from 1 to $n - k + 1$, $cell(i, k)$ contains spans formed by "covered-concatenating" (Eisner's usage) spans from $cell(i, \lambda)$ and spans from $cell(i + \lambda - 1, k - \lambda + 1)$, for λ from 2 to k. When spans are covered-concatenated (see Section 3.8), only their combinatorial features are considered.

7. When "covered-concatenating" two spans, the use of "minimal" spans (Eisner's usage) is restricted (see Section 3.5) to avoid generating the same span via multiple paths.

8. When generating spans in a cell, dependency constraints regarding link crossing, cycles and multiple heads apply. Inertness of internal vertices of spans is respected, and, except when in $cell(1, n)$, internal vertices of the resulting span must have a parent.

9. Spans with the same "signature" are identified. Their probabilities are compared, and the best one is retained for later use.

10. Loop ends.

11. When we come to $k = n$, a different kind of actions (internal heads are allowed) is used to generate the final dependency structure candidates. Probabilities of the links with BOP are taken into consideration when chosing the parse output.

3.5 Avoiding Multiple Generation Paths: Minimal (Simple) Spans

CYK by itself does not prevent *chains* from being generating in multiple ways. For example, a chain with 3 covered spans can be generated in 2 ways; a chain with 4 covered spans can be generated in 5 ways. To avoid this kind of over-generation, Eisner (1996) imposes restrictions on the combining power of "minimal spans".

A "minimal (simple) span" is:

1. A "covered span"; or

2. A span of length 2.

When two spans are concatenated, we require that the span on the right hand side should be minimal. In CYK-like parsing, this restriction is easy to realize efficiently. Minimal (simple) spans are marked as such when they are created or put in the CYK chart.

Eisner (1996) did not give a detailed description "minimal spans", and the significance of the requirement that the constituent span on one side should be minimal was pointed out clearly. These were done later (Eisner 2000).

3.6 Avoiding Another Kind of Multiple Generation

Another kind of multiple generation (not noted by Eisner) will occur when the overlapping boundary vertex is given a parent when two spans are "covered-concatenated".

$$\begin{tikzcd}
\text{span} & * \arrow[l, bend right] & \text{span}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{span} & * \arrow[r, bend left] & \text{span}
\end{tikzcd}$$

The participating span that provides the new parent will certainly have a counterpart with the new link already there.

We avoid this kind of multiple generation by not allowing the overlapping boundary vertex to have new links pointing to anywhere in the spans being concatenated.

3.7 Initial Spans and the Final Output

We kick off CYK parsing procedure by populating the (*, 2) cells with spans of length 2. There are three kinds of initial spans:

$$\begin{tikzcd}
A & B & & A \arrow[r, bend left] & B & & A & B \arrow[l, bend right]
\end{tikzcd}$$

All initial spans are marked "minimal (simple)".

The CYK procedure stops when a "span" is successfully placed in $cell(1, n)$. In this cell, a "span" has a different structure. There must be exactly one vertex without a parent. This is the head of the whole input expression, which is to be linked to BOP.

3.8 Combining Two Spans by "Covered Concatenation"

This procedure takes as input two spans $span(AB)$ and $span(BC)$ having a common boundary vertex B. It produces one or more spans (concatenations of the input spans) as OUTPUT. If no OUTPUT can be produced, it returns FAIL.

A ... B B ... C

When a span is OUTPUT, its probability, *combinatory features* and *sister-info* are updated (to ensure efficient computation).

3.8.1 Creating Concatenated Spans for "Regular" Cells ($2 < k < n$)

1. If $span(BC)$ is not "minimal (simple)" (i.e. the "simple" bit is not set), RETURN FAIL.

2. If NOT(B has parent in exactly one of $span(AB)$ and $span(BC)$), then RETURN FAIL.
   (The"parent" bit of B must be set in exactly one of the two spans. B will have Multiple parents if the bit is set in spans. Their will multiple generation if the bit is not set in both spans.)

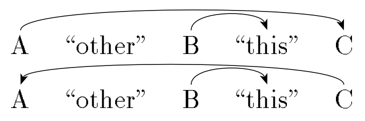
3. Now, B has a parent in exactly one of $span(AB)$ and $span(BC)$. Call the span containing the parent "this span". Call the other span "other span".

4. OUTPUT $span(AC)$ without a covering link between A and C.

5. If the other vertex of "other span" has a parent (the "parent" bit is set), then RETURN. (No further output.)

$$\begin{tikzcd}
A \arrow[r, bend left] & \text{“other”} & B \arrow[r, bend left] & \text{“this”} & C
\end{tikzcd}$$

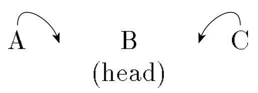
6. OUTPUT two spans with covering links between A and C ("minimal (simple)" bit set) and RETURN



3.8.2 Creating Concatenated “Spans” for Cell(1,n)

1. If B has a parent in both *span*(AB) and RETURN FAIL. (Multiple parents for B.)

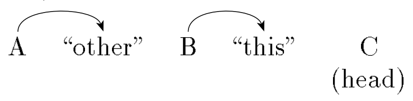
2. If both A and C have parents (i.e. ultimately depending on B), then OUTPUT *span*(AC) without a covering link and RETURN. (B is marked as head.)



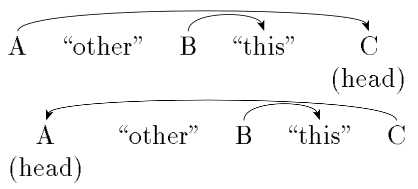
3. If B has no parents then RETURN FAIL. (To avoid multiple generation. Not being the head, B has to have a parent.)

4. Now, B has a parent in eactly one of *span*(AB) and *span*(BC). The head of the whole expression is yet to be determined.

5. If the other vertex of “other span” has a parent, then OUTPUT *span*(AC) and RETURN. (Mark the other vertex of “this span” as head.)



6. OUTPUT two spans with covering links between A and C and RETURN. (If the covering link is from A to C, then mark C as head. Otherwise, mark A as head.)



4 Training and Statistical Parser Implementation

4.1 Training

We use Zhou’s (2000) parser to generate a large training set of dependency structures. This is a robust rule- and heuristics-based parser that breaks up Chinese sentences into “chunks”, which are represented by their *headwords*. Grammatical functions are also indicated in its output. 17 POS tags and 19 grammatical relations are recognized by this parser.

Two gigabytes of text corpus from the 1982-1998 issues of the *People’s Daily*, Beijing, has been parsed using Zhou’s parser. Parse outputs for the year 1996 have been checked automatically for well-formedness (e.g. non-cyclicity). The final training corpus consists of dependency parse results of about 40M of text from January 1996 to April 1996, which contains about 60,000 word types (counting homonyms with different parts of speech as different word types).

4.2 Statistical Parser Implementation

In our implementation, parsing is carried out in two steps. First, the span-based CYK statistical parsing algorithm is applied to the input string of words-POS pairs (headwords of chunks). Non-constituent spans and the final dependency structure are ensured to be well-formed by carefully designing the span-combining algorithm. Care is also taken to avoid generating redundant intermediate spans. At this stage, there is no fall-back to POS tags. Zero counts are handled by adding 0.001 to numerators and 0.1 to denominators. There is no smoothing and no sister statistics (which are very sparse) are used. The dominance structure is determined entirely in this step.

Then, if there are “unknown” dependency types in the dependency structure obtained in the first stage, an attempt will be made to resolve them locally. Probabilities are used, but spans and CYK parsing are not. The whole n-gram context (or history), including the intervening sister, is taken into account in this stage, and backing off to POS tags is done progressively.

In our statistical parser, punctuation marks in the input are treated as “words”. At the same time, they also play the role of BOP (‘beginning of phrase’). Different punctuation marks function as different kinds of BOP.

Progressively relaxed pruning in the manner of Collins (1996) is not applied.

5 Tests and Results

5.1 Evaluation Tests

Fifty files not included in the training set were selected to perform an open test. Eighty files from the training data set were used in a closed test. Results are shown in Figure 1.

In these tests, an analysis by the statistical parser is deemed to be correct if it agrees with that produced by Zhou’s (2000) parser. We use the ratio (number of correctly identified items : total no. of items) as our precision measure. Input units with only 2 blocks (a “chunk” and a punctuation mark) are not counted because correctness is trivial in such cases. Sentences containing more than 15 “words” are also excluded

```
-------------------------------------------------------------------
                            Open Test            Closed Test
-------------------------------------------------------------------
 Total no. of sentences:     2013                 2224
   Correctly parsed:         1224   60.80%        1644   73.92%
-------------------------------------------------------------------
 Total no. of relations:     7473                 7980
   Dominance-correct:        5343   71.50%        6812   85.36%
   Relations also correct:   4473   59.86%        6538   81.93%
-------------------------------------------------------------------
 Processing time:            904.622 seconds      661.850 seconds
-------------------------------------------------------------------
```

Figure 1: Evaluation Test Results

because they do not have much chance of being correctly parsed by the trainer parser in the first place.

5.2 When the Test Data Are "Better"

It is understood that the training data produced using Zhou's (2000) parser is liable to contain errors. In order to monitor the quality of the input to our statistical parser, we have carried out a small-scale evaluation of Zhou's (2000) parsing output. 4 files containing 511 input "sentences" were chosen from the parsed newspaper corpus (January 1996) for human evaluation. The results are:

```
4 files:
  Total number of sentences: 511
    Containing dependencies: 440
  No. of relations: 1487
  No. of correct relations: 1081
  Percent correct: 72.70
1st file:
  Total number of sentences: 173
  Percent correct: 78.76
2nd file:
  Total number of sentences: 61
  Percent correct: 74.17
3rd file:
  Total number of sentences: 148
  Percent correct: 73.12
4th file:
  Total number of sentences: 129
  Percent correct: 66.15
  (Main problem with this file:
   an unknown person name.)
```

Errors in the training data (as produced by Zhou's parser) may have had effects on our test results, In order to get an idea on how things would be like if the training data had been error-free, we have studied and compared our statistical parser's performance with that of Zhou's parser for sentences that are at least correctly broken up into "chunks" (by Zhou's parser). Results of open and closed tests are shown in Figure 2.

6 Discussion

As shown in Figure 1, parsing speed is acceptable. For inputs not exceeding 15 "words" in length, the average processing time is 904.6/2013 = 0.45 sec. in the open test and 661.9/2224 = 0.30 sec. in the closed test. This shows that if the span-based CYK algorithm is carefully designed to avoid redundancy, parsing speed is acceptable even for an effort involving about 60,000 word types.

As for correctness, 60.8% of the sentences (*whole* sentences) are correct (i.e. agree with Zhou's parser) in the open test in terms of both dominance and grammatical functions (Figure 1). This is significantly lower than in the closed test (73.9%). We believe the main reason for this is that the training corpus is still not large enough, though we have already used a training corpus that is much bigger than is usually possible.

If we follow the practice of counting correctly identified dominance relations (without requiring the relation name to be correct), then the precision rate will be over 71.5% in the open test and 85.4% in the closed test. In the closed test, even if grammatical relations are also taken into account, the percentage is still 81.9%.

The closed test results suggest that it should be possible for us to improve on the open test results by using larger training sets. While we have used only 40 megabytes of training data, we have applied Zhou's parser to 2 gigabytes of data. Applying the parser to more data should not be difficult.

In the above discussion, we have compared the performance of the statistical parser against that of Zhou's parser. An analysis of the statistical parser is considered to be "correct" if it agrees with that of the Zhou's parser.

We have studied how things will be like if the test sentences are "better" in the sense that they

```
------------------------------------------------------------------------
                                         Open Test      Closed Test
------------------------------------------------------------------------
 Total no. of sentences in test:         400            400
   Correctly chunked sentences:          236            248
   Incorrectly chunked sentences:        135            126
   Incorrectly segmented input sent.:     29             26
   Chunking precision:                 63.61%         66.31%
------------------------------------------------------------------------
 Total number of dependencies in test:   778            746
------------------------------------------------------------------------
 Trainer parser:
   Correct parents:                       755 (97.04%)   728 (97.59%)
   Dominance-correct sentences:           227 (96.19%)   238 (95.97%)
   Correct relation names:                742 (95.37%)   716 (95.98%)
   Correct (dom. & rel. name) sentences:  224 (94.92%)   237 (95.57%)
------------------------------------------------------------------------
 Statistical parser:
   Correct parents:                       670 (86.12%)   695 (93.16%)
   Dominance-correct sentences:           184 (77.97%)   224 (90.32%)
   Correct relation names:                598 (76.86%)   680 (91.15%)
   Correct (dom. & rel. name) sentences:  175 (74.15%)   235 (94.76%)
------------------------------------------------------------------------
 Comparision:
   Same parents:                          664            702 (94.10%)
                                     (664/778=85.35%)
       Correct:                           657            692 (98.58%)
                                     (657/664=98.95%)
   Sentences with same dominance trees:   179            222 (89.52%)
                                     (179/236=75.85%)
                              Correct:    177            221 (99.55%)
                                     (177/179=98.88%)
   Same relation names:                   597            692 (92.76%)
                                     (597/778=76.74%)
           Correct:                       548            656 (94.80%)
                                     (548/597=91.79%)
   Sentences with same parses:            157            226 (91.13%)
                                     (157/236=66.525%)
   Sentences with same parses (correct):  147            215 (95.13%)
                                     (147/157=93.631%)
------------------------------------------------------------------------
```

Figure 2: Tests with Correctly Chunked Sentences

are at least correctly "chunked" by Zhou's parser. Figure 2 shows that the performance of our statistical parser on closed test data is comparable to Zhou's parser. Its scores of 90.3% to 94.8% are close to Zhou's parser's scores of 95.6% to 97.6%. Its open test scores of 74.2% to 86.1% are farther away from the trainer parser's scores of 94.9% to 97.0%, but should still be acceptable. We should note that the precision rate for correctly identified dependencies (dominance only) are 86.1% (trainer parser 97.0%) for the open test and 93.2% (trainer parser 97.6%) for the closed set. Even for the more stringent indicator of getting the whole sentence correct in terms of both dominance and grammatical functions, the statistical parser still have scores of 74.2% (trainer parser 94.9%) and 94.7% (trainer parser 95.6%).

The results show that the two parsers agree (according to a range of indicators) from 66.5% to 98.9% in the open test and from 89.5% to 99.6% in the closed test. This suggests that the statistical parser should be able to perform very well if the annotation of the training corpus is of good quality.

7 Conclusion

We have studied statistical dependency parsing using a CYK algorithm based on non-constituent spans. We have added a measure to avoid duplicated efforts in the parsing process and have shown that acceptable parsing speed can be obtained for an effort involving about 60,000 word types (Chinese). We have used an existing parser to produce syntactically annotated training data.

We have added grammatical functions and the most recent (elder) sister to our statistical model.

Our test results suggest that with the use of large training data, the performance of the statistical parser can come close to that of the parser that provides the training data.

References

Daniel M. Bikel and David Chiang. Two statical parsing models applied to the Chinese treebank. In *Proceedings of the ACL-2000 Workshop on Chinese Language Processing (CLP-2000)*, Hong Kong, October 2000.

Michael John Collins. A new statistical parser based on bigram lexical dependencies. In *34th Annual Meeting of ACL (ACL'96)*, Santa Cruz, California, 1996.

Jacques Courtin. Un analyseur syntaxique interactif pour la communication homme-machine. In *Proceedings of International Conference of Computational Linguistics*, volume 1, Pise, Italy, August 1973.

Jason M. Eisner. Three new probabilistic models for dependency parsing: an exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhagen, August 1996.

Jason M. Eisner. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the International Workshop on Parsing Technologies (IWPT'97)*, pages 54–65, MIT, September 1997.

Jason M. Eisner. Bilexical grammars and their cubic-time parsing algorithms. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer, 2000.

Haim Gaifman. Dependency Systems and Phrase-Structure Systems. *Information and Control*, 8:304–337, 1965.

David G. Hays. Dependency Theory: A Formalism and Some Observations. *Language*, 40:511–525, 1964.

Peter Hellwig. Dependency Unification Grammar. In *Proceedings of 11th International Conference on Computational Linguistics (COLING'86)*, pages 195–199, 1986.

Seungmi Lee and Key-Sun Choi. Learning probabilistic dependency grammars for korean. *Computer Processing of Oriental Languages*, 12(3):251–268, Mar 2000.

Jane J. Robinson. Dependency Structures and Transformation Rules. *Language*, 46:259–285, 1970.

Kwang-Jun Seo, Ki-Chun Nam, and Key-Sun Choi. A probalistic model of the dependency parse of the variable-word-order languages by using ascending dependency. *Computer Processing of Oriental Languages*, 12(3):309–322, Mar 2000.

Lucien Tesnière. *Elements de syntaxe structurale.* Klincksieck, Paris, 1959.

Chunfa Yuan and Changning Huang. Knowledge Acquisition and Chinese Parsing Based on Corpus. In *Proceedings of COLING'92*, pages 13000–13004, Nantes, 1992.

Ming Zhou. A block-based robust dependency parser for unrestricted Chinese text. In *Proceedings of ACL-2000 Workshop on Chinese Language Processing (CLP2000)*, Hong Kong, October 2000.