Generalized chart constraints for efficient PCFG and TAG parsing

Stefan Grünewald and **Sophie Henning** and **Alexander Koller**
Department of Language Science and Technology
Saarland University, Saarbrücken, Germany
{stefang|shenning|koller}@coli.uni-saarland.de

Abstract

Chart constraints, which specify at which string positions a constituent may begin or end, have been shown to speed up chart parsers for PCFGs. We generalize chart constraints to more expressive grammar formalisms and describe a neural tagger which predicts chart constraints at very high precision. Our constraints accelerate both PCFG and TAG parsing, and combine effectively with other pruning techniques (coarse-to-fine and supertagging) for an overall speedup of two orders of magnitude, while improving accuracy.

1 Introduction

Effective and high-precision pruning is essential for making statistical parsers fast and accurate. Existing pruning techniques differ in the source of parsing complexity they tackle. Beam search (Collins, 2003) bounds the number of entries in each cell of the parse chart; supertagging (Bangalore and Joshi, 1999; Clark and Curran, 2007; Lewis et al., 2016) bounds the number of lexicon entries for each input token; and coarse-to-fine parsing (Charniak et al., 2006) blocks chart cells that were not useful when parsing with a coarser-grained grammar.

One very direct method for limiting the chart cells the parser considers is through *chart constraints* (Roark et al., 2012): a tagger first identifies string positions at which constituents may begin or end, and the chart parser may then only fill cells which respect these constraints. Roark et al. found that begin and end chart constraints accelerated PCFG parsing by up to 8x. However, in their original form, chart constraints are limited to PCFGs and cannot be directly applied to more expressive formalisms, such as tree-adjoining grammar (TAG, Joshi and Schabes (1997)).

Chart constraints prune the ways in which smaller structures can be combined into bigger ones. Intuitively, they are complementary to supertagging, which constrains lexical ambiguity in lexicalized grammar formalisms such as TAG and CCG, and has been shown to drastically improve efficiency and accuracy for these (Bangalore et al., 2009; Lewis et al., 2016; Kasai et al., 2017). For CCG specifically, Zhang et al. (2010) showed that supertagging combines favorably with chart constraints. To our knowledge, similar results for other grammar formalisms are not available.

In this paper, we make two contributions. First, we generalize chart constraints to more expressive grammar formalisms by casting them in terms of *allowable parse items* that should be considered by the parser. The Roark chart constraints are the special case for PCFGs and CKY; our view applies to any grammar formalism for which a parser can be specified in terms of parsing schemata. Second, we present a neural tagger which predicts begin and end constraints with an accuracy around 98%. We show that these chart constraints speed up a PCFG parser by 18x and a TAG chart parser by 4x. Furthermore, chart constraints can be combined effectively with coarse-to-fine parsing for PCFGs (for an overall speedup of 70x) and supertagging for TAG (overall speedup of 124x), all while improving the accuracy over those of the baseline parsers. Our code is part of the Alto parser (Gontrum et al., 2017), available at http://bitbucket.org/tclup/alto.

2 Generalized chart constraints

Roark et al. define *begin* and *end* chart constraints. A begin constraint $\overline{B}$ for the string $\mathbf{w}$ is a set of positions in $\mathbf{w}$ at which no constituent of width two or more may start. Conversely, an end constraint $\overline{E}$ describes where constituents may not end.

Roark et al. focus on speeding up the standard

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 626–631
Melbourne, Australia, July 15 - 20, 2018.

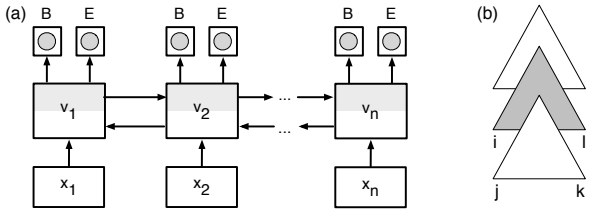

Figure 1: (a) Chart-constraint tagger; (b) TAG adjunction.

CKY parser for PCFGs with chart constraints. They do this by declaring a cell $[i, k]$ of the CKY parse chart as *closed* if $i \in \overline{B}$ or $k \in \overline{E}$, and modifying the CKY algorithm such that no nonterminals may be entered into closed cells. They show this to be very effective for PCFG parsing; but in its reliance on CKY chart cells, their algorithm is not directly applicable to other parsing algorithms or grammar formalisms.

2.1 Allowable items

In this paper, we take a more general perspective on chart constraints, which we express in terms of *parsing schemata* (Shieber et al., 1995). A parsing schema consists of a set $\mathcal{I}$ of *items*, which are derived from initial items by applying inference rules. Once all derivable items have been calculated, we can calculate the best parse tree by following the derivations of the *goal items* backwards.

Many parsing algorithms can be expressed in terms of parsing schemata. For instance, the CKY algorithm for CFGs uses items of the form $[A, i, k]$ to express that the substring from i to k can be derived from the nonterminal A, and derives new items out of old ones using the inference rule

$$\frac{[B, i, j] \quad [C, j, k] \quad A \to B\ C}{[A, i, k]}$$

The purpose of a chart constraint is to describe a set of *allowable items* $\mathcal{A} \subseteq \mathcal{I}$. We restrict the parsing algorithm so that the consequent item of an inference rule may only be derived if it is allowable. If all items that are required for the best derivation are allowable, the parser remains complete, but may become faster because fewer items are derived.

For the specific case of the CKY algorithm for PCFGs, we can simulate the behavior of Roark et al.'s algorithm by defining an item $[A, i, k]$ as allowable if $i \notin \overline{B}$ and $k \notin \overline{E}$.

2.2 Chart constraints and binarization

One technical challenge regarding chart constraints arises in the context of binarization. Chart constraints are trained to identify constituent boundaries in the original treebank, where nodes may have more than two children. However, an efficient chart parser for PCFG can combine only two adjacent constituents in each step. Thus, if the original tree used the rule $A \to B\ C\ D$, the parser needs to first combine B with C, say into the substring $[i, k]$, and then the result with D (or vice versa). This intermediate parsing item for $[i, k]$ must be allowable, even if $k \in \overline{E}$, because it does not represent a real constituent; it is only a computation step on the way towards one.

We solve this problem by keeping track in the parse items whether they were an intermediate result caused by binarization, or a complete constituent. This generalizes Roark et al.'s cells that are "closed to complete constituents". For instance, when converting a PCFG grammar to Chomsky normal form, one can distinguish the "new" nonterminals generated by the CNF conversion from those that were already present in the original grammar. We can then let an item $[A, i, k]$ be allowable if $i \notin \overline{B}$ and either $k \notin \overline{E}$ or A is new.

2.3 Allowable items for TAG parsing

By interpreting chart constraints in terms of allowable parse items, we can apply them to a wide range of grammar formalisms beyond PCFGs. We illustrate this by defining allowable parse items for TAG. Parse items for TAG (Shieber et al., 1995; Kallmeyer, 2010) are of the form $[\mathcal{X}, i, j, k, l]$, where i, l are string positions, and j, k are either both string positions or both are NULL. $\mathcal{X}$ is a complex representation of a position in an elementary tree, which we do not go into here; see the literature for details. The item describes a derivation of the string from position i to l. If j and k are NULL, then the derivation starts with an initial tree and covers the entire substring. Otherwise, it starts with an auxiliary tree, and there is a gap in its string yield from j to k. Such an item will later be adjoined at a node which covers the substring from j to k using the following inference rule (see Fig. 1b):

$$\frac{[\mathcal{X}, i, j, k, l] \quad [\mathcal{Y}, j, r, s, k]}{[\mathcal{Y}', i, r, s, l]}$$

Assuming begin and end constraints as above, we define allowable TAG items as follows. First, an item $[\mathcal{X}, i, j, k, l]$ is not allowable if $i \in \overline{B}$ or $l \in \overline{E}$. Second, if j and k are not NULL, then the item is not allowable if $j \in \overline{B}$ or $k \in \overline{E}$ (else there will be

no constituent from j to k at which the item could be adjoined). Otherwise, the item is allowable.

2.4 Allowable states for IRTG parsing

Allowable items have a particularly direct interpretation when parsing with Interpreted Regular Tree Grammars (IRTGs, Koller and Kuhlmann (2011)), a grammar formalism which generalizes PCFG, TAG, and many others. Chart parsers for IRTG describe substructures of the input object as *states* of a finite tree automaton D. When we encode a PCFG as an IRTG, these states are of the form $[i, k]$; when we encode a TAG grammar, they are of the form $[i, j, k, l]$. Thus chart constraints describe *allowable states* of this automaton, and we can prune the chart simply by restricting D to rules that use only allowable states.

In the experiments below, we use the Alto IRTG parser (Gontrum et al., 2017), modified to implement chart constraints as allowable states. We convert the PCFG and TAG grammars into IRTG grammars and use the parsing algorithms of Groschwitz et al. (2016): "condensed intersection" for PCFG parsing and the "sibling-finder" algorithm for TAG. Both of these implement the CKY algorithm and compute charts which correspond to the parsing schemata sketched above.

3 Neural chart-constraint tagging

Roark et al. predict the begin and end constraints for a string $\mathbf{w}$ using a log-linear model with manually designed features. We replace this with a neural tagger (Fig. 1a), which reads the input sentence token by token and jointly predicts for each string position whether it is in $\overline{B}$ and/or $\overline{E}$.

Technically, our tagger is a two-layer bidirectional LSTM (Kiperwasser and Goldberg, 2016; Lewis et al., 2016; Kummerfeld and Klein, 2017). In each time step, it reads as input a pair $x_i = (w_i, p_i)$ of one-hot encodings of a word w_i and a POS tag p_i, and embeds them into dense vectors (using pretrained GloVe word embeddings (Pennington et al., 2014) for w_i and learned POS tag embeddings for p_i). It then computes the probability that a constituent begins (ends) at position i from the concatenation $v_i = v_i^{F2} \circ v_i^{B2}$ of the hidden states v^{F2} and v^{B2} of the second forward and backward LSTM at position i:

$$\begin{aligned} P(B \mid \mathbf{w}, i) &= \mathrm{softmax}(W_B \cdot v_i + b_B) \\ P(E \mid \mathbf{w}, i) &= \mathrm{softmax}(W_E \cdot v_i + b_E) \end{aligned}$$

	$\overline{B}$			$\overline{E}$		
θ	acc	prec	recall	acc	prec	recall
0.5	97.6	97.4	97.8	98.1	98.7	98.7
0.9	96.7	98.8	95.2	97.2	99.4	96.7
0.99	93.7	99.6	87.9	93.0	99.7	90.5

Figure 2: Chart-constraint tagging accuracy.

We let $\overline{B} = \{i \mid P(B|\mathbf{w}, i) < 1 - \theta\}$; that is, the network predicts a begin constraint if the probability of $\overline{B}$ exceeds a threshold θ (analogously for $\overline{E}$). The threshold allows us to trade off precision against recall; this is important because false positives can prevent the parser from discovering the best tree.

4 Evaluation

We evaluated the efficacy of chart-constraint pruning for PCFG and TAG parsing. All runtimes are on an AMD Opteron 6380 CPU at 2.5 GHz, using Oracle Java version 8. See the Supplementary Materials for details on the setup.

4.1 PCFG parsing

We trained the chart-constraint tagger on WSJ Sections 02–21. The tagging accuracy on WSJ Section 23 is shown in Fig. 2. As expected, an increasing threshold θ increases precision and decreases recall. Precision and recall are comparable to Roark et al.'s log-linear model for $\overline{E}$. Our tagger achieves 94% recall for $\overline{B}$ at a precision of 99%, compared to Roark et al.'s recall of just over 80% – without the feature engineering effort required by their system.[1]

We extracted a PCFG grammar from a right-binarized version of WSJ Sections 02–21 using maximum likelihood estimation, applying a horizontal markovization of 2 and using POS tags as terminal symbols to avoid sparse data issues. We parsed Section 23 using a baseline parser which does not prune the chart, obtaining a low f-score of 71, which is typical for such a simple PCFG. We also parsed Section 23 with parsers which utilize the chart constraints predicted by the tagger (on the original sentences and gold POS tags) and the gold chart constraints from Section 23. The results are shown in Fig. 3; "time" is the mean time to compute the chart for each sentence, in milliseconds.

Chart constraints by themselves speed the parser up by factor of 18x at $\theta = 0.5$; higher values of θ did not increase the parsing accuracy further, but

[1] Note that the numbers are not directly comparable because Roark et al. evaluate their tagger on Section 24.

Parser	f-score	time	speedup	% gold
Unpruned	71.0	2599	1.0x	4.4
CC ($\theta = 0.5$)	75.0	143	18.2x	91.8
CC (gold)	77.6	143	18.2x	100.0
CTF	67.6	194	13.4x	20.1
CTF + CC (θ=0.5)	72.4	37	**70.1x**	94.3
CTF + CC (gold)	75.3	38	68.4x	100.0

Figure 3: Results for PCFG parsing.

yielded smaller speedups. This compares to an 8x speedup in Roark et al.; the difference may be due to the higher $\overline{B}$ recall of our neural tagger. Furthermore, when we combine chart constraints with the coarse-to-fine parser of Teichmann et al. (2017), using their threshold of 10^{-5} for CTF pruning, the two pruning methods amplify each other, yielding an overall speedup of up to 70x.[2]

4.2 TAG parsing

For the TAG experiments, we converted WSJ Sections 02–21 into a TAG corpus using the method of Chen and Vijay-Shanker (2004). This method sometimes adjoins multiple auxiliary trees to the same node. We removed all but the last adjunction at each node to make the derivations compatible with standard TAG, shortening the sentences by about 40% on average. To combat sparse data, we replaced all numbers by NUMBER and all words that do not have a GloVe embedding by UNK.

The neural chart-constraint tagger, trained on the shortened corpus, achieves a recall of 93% for $\overline{B}$ and 98% for $\overline{E}$ at 99% precision on the (shortened) Section 00. We chose a value of $\theta = 0.95$ for the experiments, since in the case of TAG parsing, false positive chart constraints frequently prevent the parser from finding any parse at all, and thus lower values of θ strongly degrade the f-scores.

We read a PTAG grammar (Resnik, 1992) with 4731 unlexicalized elementary trees off of the training corpus, binarized it, and used it to parse Section 00. This grammar struggles with unseen words, and thus achieves a rather low f-score (see Fig. 4). Chart constraints by themselves speed the TAG parser up by 3.8x, almost matching the performance of gold chart constraints. This improvement is remarkable in that Teichmann et al. (2017) found that coarse-to-fine parsing, which also prunes the substrings a finer-grained parser considers, did not improve TAG parsing performance.

[2]Our CTF numbers differ slightly from Teichmann et al.'s because they only parse sentences with up to 40 words and use a different binarization method.

	Parser	f-score	time	speedup	% gold
binarized	Unpruned	51.4	9483	1.0x	5.3
	CC ($\theta = 0.95$)	53.6	2489	3.8x	76.7
	CC (gold)	53.9	2281	4.2x	100.0
	supertag ($k = 3$)	77.5	137	69.4x	29.7
unbinarized	supertag ($k = 3$)	78.5	132	72.0x	30.2
	… + CC (0.95)	78.4	76	**124.3x**	91.6
	… + CC (0.99)	79.2	80	119.2x	86.1
	… + CC (gold)	78.3	74	127.9x	100.0
	… + B/E (0.95)	79.2	87	108.9x	74.5
	… + B/E (0.8)	78.4	84	113.3x	76.9
	supertag ($k = 10$)	79.4	1768	5.4x	1.5
	… + CC (0.95)	80.6	265	35.8x	71.3
	… + CC (0.99)	81.0	288	33.0x	60.3
	… + CC (gold)	81.9	252	37.6x	100.0
	… + B/E (0.95)	81.1	397	23.9x	35.6
	… + B/E (0.8)	80.7	386	24.6x	38.6

Figure 4: Results for TAG parsing.

Supertagging. We then investigated the combination of chart constraints with a neural supertagger along the lines of Lewis et al. (2016). We modified the output layer of Fig. 1a such that it predicts the supertag (= unlexicalized elementary tree) for each token. Each input token is represented by a 200D GloVe embedding.

To parse a sentence $\mathbf{w}$ of length n, we ran the trained supertagger on $\mathbf{w}$ and extracted the top k supertags for each token w_i of $\mathbf{w}$. We then ran the Alto PTAG parser on an artificial string "*1 2 … n*" and a sentence-specific TAG grammar which contains, for each i, the top k elementary trees for w_i, lexicalized with the "word" i and weighted with the probability of its supertag. This allowed us to use the unmodified Alto parser, while avoiding the possible mixing of supertags for multiple occurrences of the same word. We then obtained the best parse trees for the original sentence $\mathbf{w}$ by replacing each artificial token i in the parse tree by the original token w_i.

The sentence-specific grammars are so small that we can parse the test corpus without binarizing them. As Fig. 4 indicates, supertagging speeds up the parser by 5x ($k = 10$) to 70x ($k = 3$); the use of word embeddings boosts the coverage to almost 100% and the f-score to around 80. Adding chart constraints on top of supertagging further improves the parser, yielding the best speed (at $k = 3$) and accuracy (at $k = 10$). We achieve an overall speedup of two orders of magnitude with a drastic increase in accuracy.

Allowable items for TAG. Instead of requiring that a TAG chart item is only allowable if neither the string $[i, l]$ nor its gap $[j, k]$ violate a chart constraint (as in Section 2.3), one could instead adopt

a simpler definition by which a TAG chart item is allowable if i and l satisfy the chart constraints, regardless of the gap.[3]

We evaluated the original definition from Section 2.3 ("CC") against this baseline definition ("B/E"). As the results in Fig. 4 indicate, the B/E strategy achieves higher accuracy and lower parsing speeds than the CC strategy at equal values of θ. This is to be expected, because CC has more opportunities to prune chart items early, but false positive chart constraints can cause it to overprune. When θ is scaled so both strategies achieve the same accuracy – i.e., B/E $\theta = 0.8$ for CC $\theta = 0.95$, or CC $\theta = 0.99$ for B/E $\theta = 0.95$ –, CC is faster than B/E. This suggests that imposing chart constraints on the gap is beneficial and illustrates the flexibility and power of the "admissible items" approach we introduce here.

4.3 Discussion

The effect of using chart constraints is that the parser considers fewer substructures of the input object – potentially to the point that the asymptotic parsing complexity is reduced below that of the underlying grammar formalism (Roark et al., 2012). In practice, we observe that the percentage of chart items whose begin positions and end positions are consistent with the gold standard tree ("% gold" in the figures) is increased by CTF and supertagging, indicating that these suppress the computation of many spans that are not needed for the best tree. However, chart constraints prune useless spans out much more directly and completely, leading to a further boost in parsing speed.

Because we remove multiple adjunctions in the TAG experiment, most sentences in the corpus are shorter than in the original. This might skew the parsing results in favor of pruning techniques that work best on short sentences. We checked this by plotting sentence lengths against mean parsing times for a number of pruning methods in Fig. 5 (supertagging with $k = 10$, chart constraints with $\theta = 0.95$). As the sentence length increases, parsing times of supertagging together with chart constraints grows much more slowly than the other methods. Thus we can expect the relative speedup to increase for corpora of longer sentences.

[3] We thank an anonymous reviewer for suggesting this comparison.

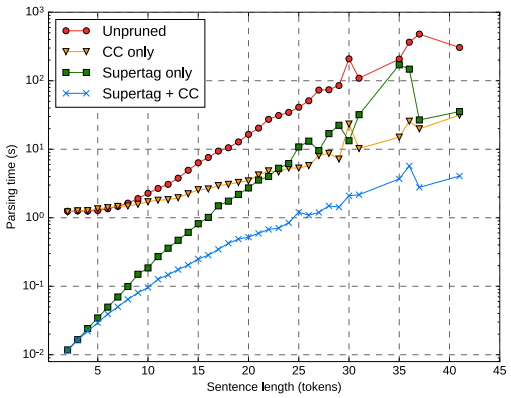

Figure 5: TAG parsing speed as a function of sentence length.

5 Conclusion

Chart constraints, computed by a neural tagger, robustly accelerate parsers both for PCFGs and for more expressive formalisms such as TAG. Even highly effective pruning techniques such as CTF and supertagging can be further improved through chart constraints, indicating that they target different sources of complexity.

By interpreting chart constraints in terms of allowable chart items, we can apply them to arbitrary chart parsers, including ones for grammar formalisms that describe objects other than strings, e.g. graphs (Chiang et al., 2013; Groschwitz et al., 2015). The primary challenge here is to develop a high-precision tagger that identifies allowable subgraphs, which requires moving beyond LSTMs.

An intriguing question is to what extent chart constraints can speed up parsing algorithms that do not use charts. It is known that chart constraints can speed up context-free shift-reduce parsers (Chen et al., 2017). It would be interesting to see how a neural parser, such as (Dyer et al., 2016), would benefit from chart constraints calculated by a neural tagger.

Acknowledgments. We are grateful to Jonas Groschwitz, Christoph Teichmann, Stefan Thater, and the anonymous reviewers for discussions and comments. This work was supported through the DFG grant KO 2916/2-1.

References

Srinivas Bangalore, Pierre Boulllier, Alexis Nasr, Owen Rambow, and Benoît Sagot. 2009. MICA: A

probabilistic dependency parser based on tree insertion grammars application note. In *Proceedings of NAACL-HLT (Short Papers)*.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computataional Linguistics* 25(2):237–265.

Eugene Charniak, Mark Johnson, Micha Elsner, Joseph Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R. Shrivaths, Jeremy Moore, Michael Pozar, and Theresa Vu. 2006. Multilevel coarse-to-fine PCFG parsing. In *Proceedings of NAACL-HLT*.

John Chen and K. Vijay-Shanker. 2004. Automatic extraction of TAGs from the Penn Treebank. In *New developments in parsing technology*, Springer, pages 73–89.

Wenliang Chen, Muhua Zhu, Min Zhang, Yue Zhang, and Jingbo Zhu. 2017. Improving shift-reduce phrase-structure parsing with constituent boundary information. *Computational Intelligence* 33(3):428–447.

David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. 2013. Parsing graphs with hyperedge replacement grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics* 33(4):493–552.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics* 29(4):589–637.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. pages 199–209.

Johannes Gontrum, Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2017. Alto: Rapid prototyping for parsing and translation. In *Proceedings of the EACL Demo Session*. Valencia.

Jonas Groschwitz, Alexander Koller, and Mark Johnson. 2016. Efficient techniques for parsing with tree automata. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2015. Graph parsing with s-graph grammars. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*.

Aravind K. Joshi and Yves Schabes. 1997. Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, Springer-Verlag, volume 3.

Laura Kallmeyer. 2010. *Parsing Beyond Context-Free Grammars*. Springer.

Jungo Kasai, Bob Frank, Tom McCoy, Owen Rambow, and Alexis Nasr. 2017. TAG parsing with neural networks and vector representations of supertags. In *Proceedings of EMNLP*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics* 4:313–327.

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies*.

Jonathan K. Kummerfeld and Dan Klein. 2017. Parsing with traces: An $O(n^4)$ algorithm and a structural representation. *Transactions of the ACL* 5:441–454.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG parsing. In *Proceedings of NAACL-HLT 2016*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of EMNLP*.

Phil Resnik. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the 14th COLING*.

Brian Roark, Kristy Hollingshead, and Nathan Bodenstab. 2012. Finite-state chart constraints for reduced complexity context-free parsing pipelines. *Computational Linguistics* 38(4):719–753.

Stuart Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming* 24(1–2):3–36.

Christoph Teichmann, Alexander Koller, and Jonas Groschwitz. 2017. Coarse-to-fine parsing for expressive grammar formalisms. In *Proceedings of the 15th International Conference on Parsing Technologies (IWPT)*. Pisa.

Yue Zhang, Byung-Gyu Ahn, Stephen Clark, Curt Van Wyk, James R. Curran, and Laura Rimell. 2010. Chart pruning for fast lexicalised-grammar parsing. In *Proceedings of COLING*.