Fast Exact Inference with a Factored Model for Natural Language Parsing

Dan Klein
Department of Computer Science
Stanford University
Stanford, CA 94305-9040
klein@cs.stanford.edu

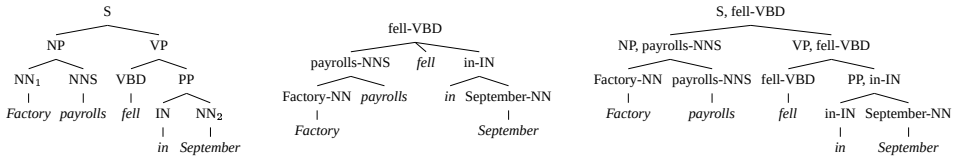
Christopher D. Manning
Department of Computer Science
Stanford University
Stanford, CA 94305-9040
manning@cs.stanford.edu

Abstract

We present a novel generative model for natural language tree structures in which semantic (lexical dependency) and syntactic structures are scored with separate models. This factorization provides conceptual simplicity, straightforward opportunities for separately improving the component models, and a level of performance already close to that of similar, non-factored models. Most importantly, unlike other modern parsing models, the factored model admits an extremely effective A* parsing algorithm, which makes efficient, exact inference feasible.

1 Introduction

Syntactic structure has standardly been described in terms of categories (phrasal and word class labels), with little mention of particular words. This is possible, since, with the exception of certain common function words, the acceptable syntactic configurations of a language are largely independent of the particular words that fill out a sentence. Conversely, for resolving the important attachment ambiguities of modifiers and arguments, lexical preferences are known to be very effective. But equally, methods based only on key lexical dependencies have been shown to be very effective in resolving ambiguities between valid syntactic forms [1]. Modern statistical parsers [2, 3] standardly use complex joint models of these two notions, where "everything is conditioned on everything" to the extent possible within the limits of data sparseness and finite computer memory. For example, the probability that a verb phrase will take a noun phrase object depends on the head word of the verb phrase. A VP headed by *acquired* will likely take an object, while a VP headed by *agreed* will likely not. There are certainly statistical interactions between syntactic and semantic structure, and, if deeper underlying variables of communication are not modeled, everything tends to be dependent on everything else in langauge [4]. However, the above considerations suggest that there might be considerable value in a factored model, which provides separate models of syntactic configurations and lexical depencies, and then combines them to determine optimal parses. For example, under this view, we may know that *acquired* takes right dependents headed by nouns such as *company* or *division*, while *agreed* takes no noun right dependents at all. If so, there is no need to explicitly model the phrasal selection on top of the lexical selection. We will show that such a model can indeed produce a high performance parser, though we focus particularly on how the factored model permits efficient exact inference, rather than the approximate heuristic inference normally used in large statistical parsers.



(a) PCFG Structure (b) Dependency Structure (c) Combined Structure

Figure 1: Three kinds of parse structures.

2 A Factored Model

Generative models for parsing typically model one of the kinds of structures shown in figure 1. Figure 1a is a plain phrase-structure tree T, which primarily models syntactic units, figure 1b is a dependency tree D, which primarily models word-to-word selectional affinities [5], and figure 1c is a lexicalized phrase-structure tree L, which carries both category and head word and its part-of-speech information at each node.

Lexicalized trees can be viewed as a pair $L = (T, D)$ of one phrase structure tree T and one dependency tree. In this view, generative models over lexicalized trees, of the sort standard in lexicalized PCFG parsing [2, 3] can then be regarded as assigning mass $P(T, D)$ to such pairs. To the extent that dependency and phrase structure need not be modeled jointly, we can factor our model as $P(T, D) = P(T)P(D)$: this approach is the basis of our proposed models, and its use is, to our knowledge, new. This factorization, of course, assigns mass to pairs which are incompatible, either because they do not generate the same terminal string or do not embody compatible bracketings. Therefore, the total mass assigned to valid structures will be less than one. We could imagine fixing this by renormalizing. For example, this situation fits into the product-of-experts framework [6], with one semantic expert and one syntactic expert that must agree on a single structure. However, since we are presently only interested in finding most-likely parses, no global renormalization constants need to be calculated.

Given the factorization $P(T, D) = P(T)P(D)$, rather than engineering a complex joint model, we can instead build two simpler models. We show that the combination of even quite simple "off the shelf" implementations of the two half-models provides decent parsing performance. Further, the factorization makes it much easier to extend and optimize the individual components. We illustrate this by building somewhat improved versions of both models, but believe that there is considerable room for further optimization.

Concretely, we used the following two half-models. For the basic model of $P(T)$, we used the raw treebank grammar, taken directly from the training trees [7]. In this model, nodes rewrite atomically, in a top-down manner, in only the ways observed in the training data. Each tree node's label (excluding preterminals and terminals) was annotated with both its parent's label and its grandparent's label, since it is now well known that such parent annotation improves the accuracy of PCFG parsing by weakening the PCFG independence assumptions [8]. For example, the NP in figure 1a would actually have been labeled NPˆSˆROOT. Since the counts were not fragmented by head word or head tag, we were able to directly use the MLE parameters, without smoothing.[1] Formally, the probability of a rule r being generated from a node n is conditioned on the two dominating nodes: $P(r|n, n_{-1}, n_{-2})$. For the data used in section 4, the node NPˆSˆROOT occurs 40238 times, 1346 times with the expansion NN NNS, so that rewrite has probability 1346/40238.

The dependency model $P(D)$ deals with tagged words: pairs $\langle w, t \rangle$. First the head $\langle w_h, t_h \rangle$ of a constituent is generated, then successive right dependents $\langle w_d, t_d \rangle$ until a STOP token

[1] This is not to say that smoothing would not improve performance, but to underscore how the factored model encounters less sparsity problems than a joint model.

$\diamond$ is generated, then successive left dependents until $\diamond$ is generated. For example, in figure 1, first we choose *fell*-VBD as the head of the sentence. Then, we generate *in*-IN to the right, which then generates *September* to the right, which generates $\diamond$ on both sides in turn. At this point, we return up to *in*-IN and generate $\diamond$ to the right, and so on.

The dependency model requires smoothing, as the word-word dependency data is very sparse. In our basic model, we generate a dependent, conditional only on the head and direction, as follows. There is a top-level mixture of two dependent generation methods: head selection of specific words, and head selection of tags. For head selection of words, there is a prior distribution over dependents taken by a head's tag, for example, left dependents taken by a past tense verb VBD: $P(w_d, t_d|t_h, dir) = \text{count}(w_d, t_d, t_h, dir)/\text{count}(t_h, dir)$. Observations of bilexical pairs are taken against this prior, with some prior strength λ_1:

$$P(w_d, t_d|w_h, t_h, dir) = \frac{\text{count}(w_d, t_d, w_h, t_h, dir) + \lambda_1 P(w_d, t_d|t_h, dir)}{\text{count}(w_h, t_h, dir) + \lambda_1}$$

This model is meant to capture bilexical selection, such as the affinity between *payrolls* and *fell*. Alternately, the dependent can have only its tag selected, then the word generated independently: $P(w_d, t_d|w_h, t_h, dir) = P(w_d|t_d)P(t_d|w_h, t_h, dir)$. The estimates for $P(t_d|w_h, t_h, dir)$ are similar to the above, and the unsmoothed ML estimate for $P(w_d|t_d)$ is used. These two mixture components are then linearly interpolated, giving just two prior strengths and a mixing weight to be estimated on held-out-data. We discuss elaborations to both models in section 4.

At this point, one might wonder what has been gained. By factoring the semantic and syntactic models, we have simplified both (and fragmented the data less), but there are always simpler models, and researchers have adopted complex ones because of their parsing accuracy. In the remainder of the paper we demonstrate the three primary benefits of our model: it allows a fast, exact parsing algorithm, it empirically parses comparably to non-factored models, and its modularity makes it easily extensible.

To demonstrate extension, we also evaluate a first attempt at extending the component models. The enhanced PCFG model adds marking infinitival VPs, spliting the preposition tag into sentential and non-sentential occurences, splitting the conjunction tag into contrastive and other occurrences, and adding spelling features (last two characters and capitalization) to the unknown word model (in the basic PCFG we simply use the MLE of $P(\text{tag}|\text{unknown})$ with ML estimates for the reserved mass per tag). In the enhanced dependency model, we condition not only on direction, but also on distance and valence. The decision of whether to generate $\diamond$ is conditioned on one of five values of distance between the head and the generation point: zero, one, 2–5, 6–10, and 11+. If we decide to generate a non-$\diamond$ dependent, the actual dependent depends only on whether the distance is zero or not. That is, we model only zero/non-zero valence. Note that this is (intentionally) very similar to the generative model of [2] in broad structure, but substantially less complex.

3 An A* Parser

In this section, we outline an efficient algorithm for finding the Viterbi, or most probable, parse for a given terminal sequence in our factored lexicalized model. The naive approach to lexicalized PCFG parsing is simply to act as if the lexicalized PCFG had many more symbols than its basic PCFG backbone. For example, while the basic PCFG might have a symbol NP, the lexicalized one has a symbol NP-x for every possible head x in the vocabulary. Further, rules like S $\rightarrow$ NP VP become a family of rules S-x $\rightarrow$ NP-y VP-x. Within a dynamic program, the core parse item in this case is the *edge*, shown in figure 2, which is specified by its start, end, root symbol, and head position.[2] Adjacent edges combine to

[2]The head position variable often, as in our case, also specifies the head's tag.

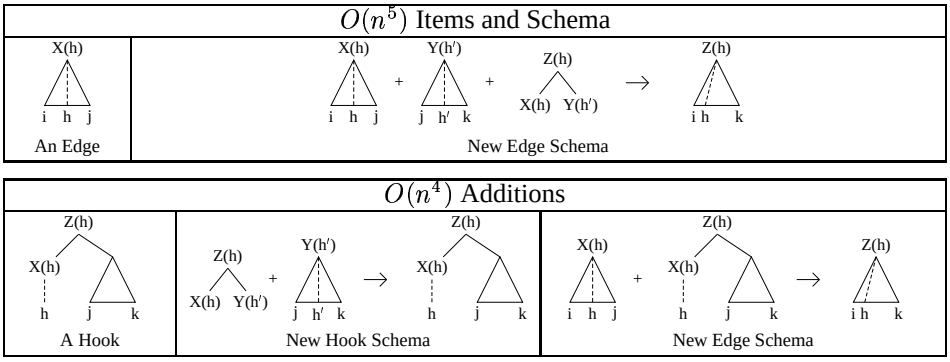


Figure 2: Items and selected schemata for a tabular parser: (top) an $O(n^5)$ naive implementation and (bottom) the $O(n^4)$ version from [9].

form larger edges, as in the top of figure 2.There are $O(n^3)$ edges, and two edges are potentially compatible whenever the left one ends where the right one starts. Therefore, there are $O(n^5)$ such combinations to check, giving an $O(n^5)$ algorithm.

Eisner and Satta [9] propose a clever $O(n^4)$ modification which separates this process into two steps by introducing an intermediate structure, as shown in the bottom part of figure 2. In addition to edges, which encapsulate a headed constituent, they add what we call *hooks*, which encapsulate a headed constituent, formed by a certain rule type, with the head edge unincorporated, and abstract over the non-head edge's label and head. There are again cubicly many of each item, but the win is that edges combine with hooks, not other edges. For an edge and a hook to be compatible, they must not only be adjacent, but the head of the edge must match the head of the hook, giving a total of $O(n^4)$ work. For details, see [9]. Nonethess, even the $O(n^4)$ formulation is impractical for exhaustive parsing with broad-coverage, lexicalized treebank grammars. There are several reasons for this: the constant factor due to the grammar is huge (often containing tens of thousands of rules once binarized), and larger sentences are more likely to contain structures which unlock increasingly large regions of the grammar.[3]

The core of our parsing algorithm is a tabular agenda-based parser, using the $O(n^4)$ schemata above. The novelty is in the choice of agenda priority, where we exploit the rapid parsing algorithms available for the sub-models to speed up the otherwide impractical combined parser, but maintain that the first parse discovered will be the actual most-probable parse. Other parsers with comparable models accelerate parsing in ways that destroy this correctness guarantee. The top-level procedure is given in figure 3. First, we parse exhaustively with the two sub-models, not to find complete parses, but to find best *outside scores* for each edge e. An outside score is the score of the best parse structure which starts at the goal and includes e, the words before it, and the words after it, as depicted in figure 3. Outside scores are the Viterbi version of the standard outside probabilities given by the inside-outside algorithm [11]. The syntactic component, $P(T)$, is a standard PCFG, and well-known cubic inside parsing algorithms are easily adapted to also find outside scores. For the semantic component, $P(D)$, there are several presentations of cubic algorithms, including [9] and [12]. These can also be adapted to produce outside scores in cubic time, though since their basic data structures are not organized around edges, there is some subtlety. For space reasons, we omit the details of these phases.

[3] [10] describes how this can lead to empirical growth far faster than the predicted bounds as the sentence length leaks into constants hidden in the worst-case bounds, leading to implementations that seem fast for short sentences but bog down for longer ones.

1. Extract PCFG grammar and train PCFG parser
2. Use PCFG parser to find outside scores $\alpha_{\text{PCFG}}(e)$ for each edge
3. Extract dependency grammar train dependency parser
4. Use dependency parser to find outside scores $\alpha_{\text{DEP}}(e)$ for each edge
5. Combine PCFG and dependency grammars into lexicalized PCFG grammar
6. Build combined outside estimate $\alpha(e) = \alpha_{\text{PCFG}}(e) + \alpha_{\text{DEP}}(e)$
7. Use lexicalized A* parser, with $\alpha(e)$ as an A* estimate

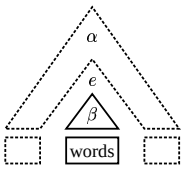


Figure 3: The top-level algorithm and a picture of an outside score α and an inside score β.

An agenda-based parser tracks all items (edges and hooks in our case) that have been constructed to date. When an item is first constructed, it is put on an agenda, which is a heap indexed by some score for that node. The agenda is a holding area for items which have been built in at least one way, but which have not yet been used to build other items. The core cycle of the parser is to remove the highest-scored item from the agenda, and act on it according to the relevant schemata, combining it with any appropriate previously removed items. This much is common to many parsers; such parsers primarily differ in their choice of agenda priority. If the priority is the best known inside score for an item, then the parser will be correct in that each item's score estimate is actually its true best score when it is removed form the agenda. In particular, the first goal parse found will be the exact most likely parse. The proof is a generalization of the proof of Dijsktra's algorithm, and is omitted for space reasons (but see [13] for a proof). However, removing edges by inside score is not feasible, because all small edges end up having better scores than any large edges; see section 4 for an empirical demonstration. Luckily, the correctness of the algorithm remains if, rather than removing items from the heap by their best scores, we add to those scores any optimistic (admissable) estimate of the cost to complete a parse using that item. The proof of this is a generalization of the proof of the correctness of A* search.

To our knowledge, no way of generating effective, admissible A* estimates for this task has been previously proposed. [4] However, because of the factored structure of our model, we can use the results of the individual models' parses to give us quite sharp A* estimates. Say we want to know the outside score of an edge e. That score will be the score of a certain structure (T, D) outside of e, where T and D are compatible. Now, from the initial phases, we know the best T' and the best D' which can occur outside of e. It may well be that T' and D' are not compatible. However, $score(T) \leq score(T')$ and $score(D) \leq score(D')$, and so $score(T, D) = score(T) + score(D) \leq score(T') + score(D')$. Therefore, we can use the individual models' outside scores to synthesize an outside score for the joint model which is not less that the true joint outside score. It is reasonable to assume that the two models will be broadly compatible, and will generally prefer similar structures, creating a sharp A* estimate, and greatly reducing the work needed to find the goal parse. We give empirical evidence of this in section 4.

4 Empirical Performance

In this section, we demonstrate that (i) the factored model's parsing performance is comparable to non-factored models which use similar features, (ii) there is an advantage to exact

[4] The basic idea of changing edge priorities to more effectively guide parser work is standardly used, and other authors have made very effective use of inadmissable estimates. [2] uses extensive probabilistic pruning – this amounts to giving pruned edges infinitely low priority. Absolute pruning can, and does, prevent the true maximum a posteriori parse from being returned at all. [14] removes edges in order of estimates of their correcteness. This may result in the first parse found not being the true maximum parse, and has another more subtle drawback: we may hold back an edge e for a very long time, finally remove it from the agenda, and then discover a better way of constructing another edge f using e. If f has already been used to construct larger edges, we must propagate its new score upwards, which may trigger yet further propagation.

PCFG Model	Precision	Recall	F_1	Exact Match
Basic	81.2	80.5	80.8	19.0
Enhanced	82.7	82.2	82.4	20.8

(a) The PCFG Model

Dependency Model	Dependency Acc
Basic	78.9
Enhanced	84.0

(b) The Dependency Model

Figure 4: Performance of the sub-models alone.

PCFG Model	Dependency Model	Precision	Recall	F_1	Exact Match	Dependency Acc
Basic	Basic	83.5	84.1	83.8	23.4	88.6
Basic	Enhanced	85.0	85.8	85.4	24.5	89.9
Enhanced	Basic	84.3	85.1	84.7	25.5	89.0
Enhanced	Enhanced	85.6	86.6	86.0	26.4	90.2

PCFG Model	Dependency Model	Thresholded?	F_1	Exact Match	Dependency Acc
Basic	Basic	No	83.5	23.4	88.6
		Yes	83.8	23.0	88.3
Enhanced	Enhanced	No	86.0	26.4	90.2
		Yes	85.6	25.7	89.9

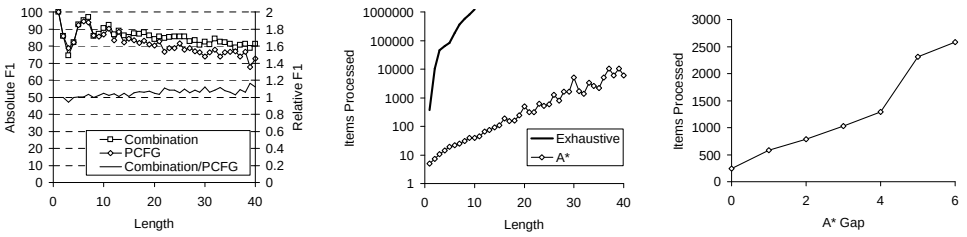
Figure 5: The combined model, with various sub-models, and with/without thresholding.

inference, and (iii) the A* savings are substantial. First, we give parsing figures on the standard Penn treebank parsing task. We trained the two sub-models, separately, on sections 02–21 of the WSJ section of the treebank. The numbers reported here are the result of then testing on section 23 (length ≤ 40). The treebank only supplies node labels (like NP), and does not contain head information. Heads were calculated for each node according to the deterministic rules given in [2]. These rules are broadly correct, but not perfect.

We effectively have three parsers: the PCFG parser alone, which produces nonlexical phrase structures like figure 1a, the dependency parser, which produces dependency structures like figure 1b, and the combination parser with produces lexicalized phrase structure, like figure 1c. The output of the combination parser can also be projected down to either nonlexical phrase structure or dependency structure. We score the output of our parsers in two ways. First, the phrase structure of the PCFG and combination parsers can be compared to the treebank parses. The parsing measures standardly used for this task are labeled precision and recall.[5] We also report F_1, the harmonic mean of these two quantities. Second, for the depenendency and combination parsers, we can score the dependency structures. A dependency structure D is viewed as a set of head-dependent pairs $\langle h, d \rangle$, with an extra dependency $\langle root, x \rangle$ where $root$ is a special symbol and x is the head of the sentence. Although the dependency model generates part-of-speech tags as well, these are ignored for dependency accuracy. Punctuation is not scored. Since all dependency structures over n non-punctuation terminals contain n dependencies ($n - 1$ plus the root dependency), we report only accuracy, which is identical to both precision and recall. It should be stressed that the "correct" dependency structures, though generally correct, are generated from the PCFG structures by linguistically motivated, but automatic and only heuristic rules.

Figure 4 shows the respective scores for the PCFG and dependency half-parsers. For each, we give figures for both the basic model and the enhanced model, with the enhanced models improving the component models' F_1 performance by 1.6% and 4.1%, respectively. The combination parser's performance is given in figure 5. As each individual model is improved, the combination is improved on all measures, with a top F_1 of 86.0%. For that setting, figure 6 shows the relative F_1 of the combination parser to the PCFG component alone, showing the unsurpising trend that the addition of the dependency model helps more

[5] A tree T is viewed as a set of constituents $c(T)$. Constituents in the correct and the proposed tree must have the same start, end, and label to be considered identical. For this measure, the lexical heads of nodes are irrelevant. The actual measures used are detailed in [15], and involve minor normalizations like the removal of punctuation in the comparison.



(a) (b) (c)

Figure 6: (a) Relative F_1 by length, (b) A* effectivness shown by items expanded, and (c) Work increasing as the true score $-$ A* estimate gap width increases.

for longer sentences, which, on average, contain more attachment ambiguity. An F_1 of 86.0 is greater than that of the lexicalized parsers presented in [15, 16], but less than that of the newer, more complex, parsers presented in [3, 2], which reach as high as 90.1% F_1. However, it is worth pointing out that these higher-accuracy parsers incorporate many finely wrought enhancements which could presumably be extracted and applied to benefit our individual models.[6]

The primary goal of this paper is not to present a maximally tuned parser, but to demonstrate a method for fast, exact inference usable in parsing. Given the impracticality of exact inference for standard parsers, a common strategy is to take the PCFG backbone, extract a set of top parses, either the top k or all parses within a score threshold of the top parse, and rerank them [3, 17]. This pruning is done for efficiency; the question is whether it is hurting accuracy. That is, would exact inference be preferable? Figure 5 shows the result of parsing with our combined model where the A* estimates were altered to block parses whose PCFG score was further than a threshold $\delta = 2$ in log-probability from the best PCFG parse, for both the top- and bottom-performing versions of our parser. With the basic model, F_1 performance actually increases with pruning. But, in both cases, dependency accuracy and exact match scores drop.

While pruning typically buys speed at the expense of some accuracy, the observation that pruning can actually improve F_1 has been made before: Charniak et al. [14] find that pruning based on estimates for $P(e|s)$, that is, the fraction of parses of a sentence s which contain a certain edge e, raises accuracy slightly, for a non-lexicalized PCFG. As they note, this increase seems to be due to the fact that this pruning metric mimics Goodman's maximum-constituents parsing [18], which maximizes the expected number of correct nodes rather than the likelihood of the entire parse. The trade-off between F_1, exact match, dependency accuracy, and parsing speed is certainly a value judgment. However, it is worth pointing out that if, as is common, parsing is meant to be a prelude to semantic processing, the accuracy of the dependency structure is likely to be of primary concern, and the present evidence suggests that this accuracy is hurt by naive pruning. In any case, we see it as valuable to have an exact parser with which these types of questions can be investigated at all.

We conclude with data on the effectiveness of the A* method. Figure 6b shows the average number of items extracted from the agenda as sentence length increases. Numbers both with and without the A* estimates are shown. Clearly, the best-first version of the parser is dramatically less efficient; by sentence length 10 it processes 1.2M items, while even at length 40 the A* heuristics are so effective that only around 50K items are processed. At length 10, the fraction of items supressed is 99.997%, and further approaches 100% as

[6] For example, the dependency distance function of [2] registers punctuation and verb counts, and both variously add further node annotation, such as a distinction between finite and non-finite S nodes, and Markovized rules, which could improve the PCFG grammar. In addition, the easy intelligibility of our factored models suggests to us a number of other things we might try.

length increases. To explain this effectiveness, we suggest that the combined parsing phase is really only figuring out how to reconcile the two models. To support this claim, figure 6c shows the average number of items processed versus the binned difference between a parse's joint best score and separate best score. This amount is the width of the area explored by the A* search, and empirically work increased with that difference. Average width varied by model, but generally averaged less than 3. As a result, the A* estimates were so effective that even with our object-heavy Java implementation of the combined parser, total parse time was dominated by the array-based initial PCFG phase.[7]

5 Conclusion

The factored model framework over lexicalized trees has several advantages. It is conceptually simple, and modularizes the model design and estimation problems. The concrete model presented performs comparably to other, more complex, non-exact models proposed, and can be easily extended in the same ways that other parser models have been. Most importantly, it admits a novel A* parsing approach which allows fast, exact inference of the most probable parse.

References

[1] M. J. Collins and J. Brooks. Prepositional phrase attachment through a backed-off model. *WVLC 3*, pp. 27–38, 1995.

[2] M. Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[3] E. Charniak. A maximum-entropy-inspired parser. *NAACL 1*, pp. 132–139, 2000.

[4] R. Bod. What is the minimal set of fragments that achieves maximal parse accuracy? *ACL 39*, pp. 66–73, 2001.

[5] I. A. Mel′čuk. *Dependency Syntax: theory and practice*. State University of New York Press, Albany, NY, 1988.

[6] G. E. Hinton. Training products of experts by minimizing contrastive divergence. Technical Report GCNU TR 2000-004, GCNU, University College London, 2000.

[7] E. Charniak. Tree-bank grammars. *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI '96)*, pp. 1031–1036, 1996.

[8] M. Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632, 1998.

[9] J. Eisner and G. Satta. Efficient parsing for bilexical context-free grammars and head-automaton grammars. *ACL 37*, pp. 457–464, 1999.

[10] D. Klein and C. D. Manning. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn treebank. *ACL 39/EACL 10*, pp. 330–337, 2001.

[11] J. K. Baker. Trainable grammars for speech recognition. *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pp. 547–550, 1979.

[12] J. Lafferty, D. Sleator, and D. Temperley. Grammatical trigrams: A probabilistic model of link grammar. *1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, 1992.

[13] D. Klein and C. D. Manning. Parsing and hypergraphs. *Proceedings of the 7th International Workshop on Parsing Technologies (IWPT-2001)*, 2001.

[14] E. Charniak, S. Goldwater, and M. Johnson. Edge-based best-first chart parsing. *Proceedings of the Sixth Workshop on Very Large Corpora*, pp. 127–133, 1998.

[15] D. M. Magerman. Statistical decision-tree models for parsing. *ACL 33*, pp. 276–283, 1995.

[16] M. J. Collins. A new statistical parser based on bigram lexical dependencies. *ACL 34*, pp. 184–191, 1996.

[17] M. Collins. Discriminative reranking for natural language parsing. *ICML 17*, pp. 175–182, 2000.

[18] J. Goodman. Parsing algorithms and metrics. *ACL 34*, pp. 177–183, 1996.

[7] The average time to parse a sentence on a 750Mhz Pentium III with 2GB RAM was: for 20 words, PCFG 8 sec, dependencies 0.5 sec, combination 0.5 sec; 40 words, PCFG 70 sec, dependencies 14 sec, combination 7 sec.