Factored A* Search for Models over Sequences and Trees

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 investigate the calculation of A* bounds for sequence and tree models which are the explicit intersection of a set of simpler models or can be bounded by such an intersection. We provide a natural viewpoint which unifies various instances of factored A* models for trees and sequences, some previously known and others novel, including multiple sequence alignment, weighted finite-state transducer composition, and lexicalized statistical parsing. The specific case of parsing with a product of syntactic (PCFG) and semantic (lexical dependency) components is then considered in detail. We show that this factorization gives a modular lexicalized parser which is simpler than comparably accurate non-factored models, and which allows efficient exact inference with large treebank grammars.

1 Introduction

The primary challenge when using A* search is to find heuristic functions that simultaneously are admissible, close to actual completion costs, and efficient to calculate. In this paper, we describe a family of tree and sequence models in which path costs are either defined as or bounded by a combination of simpler component models, each of which scores some projection of the full structure. In such models, we can exploit the decoupled behavior over each projection to give sharp heuristics for the combined space. While we focus on models of trees and sequences within NLP applications, the approach can be applied more generally (and already has been, in the case of biological sequence models). All the concrete cases we consider here involve search over spaces which are equivalent to dynamic programming lattices, though this detail, too, is somewhat peripheral to the basic ideas.

2 Projection Models for Graphs

The core idea of factored A* search can apply to any graph search. Assume that $G = (N, A)$ is a very large graph, with a single source node s and a single goal node g, and that we wish to use A* search to efficiently find a best path from s to g. For concreteness, assume also that the score of a path is the sum of the scores of the arcs along the path, and that lower scores are better.[1] The particular assumption in this paper is that the arc scoring function has a special factored form. Specifically, there exists a set of projections $\{\pi_1, \ldots \pi_k\}$ of nodes (and therefore also arcs and graphs) such that for any arc (x, y), its score σ is given by:

$$\sigma_G(x, y) = \sum\nolimits_{i=1}^{k} \sigma_{\pi_i}(\pi_i(x), \pi_i(y))$$

Whenever the scoring function factors in this way, we have an immediate recipe for a *factored A* bound*, which we denote by h. Specifically, we can bound the shortest path in G from a node n to the goal g by the sum of the shortest paths inside each projection $\pi(G)$. Formally, if $\alpha_G(n, g)$ is the length of a shortest path from n to g in a graph G, then:

$$\alpha_G(n, g) \geq h(n, g) = \sum\nolimits_{i=1}^{k} \alpha_{\pi_i(G)}(\pi_i(n), \pi_i(g))$$

This follows immediately from the optimality of the projected paths and the structure of the scoring function. These projections need not be mutually compatible, and therefore the bound may not be tight. Broadly speaking, the greater the degree to which each projection prefers similar paths, the better the bound will be, and the more efficient our search will be.

3 Projection Models for Sequences

For intuition, we first consider applications to sequence models before extending to the more complex case of tree models.

3.1 Example: Multiple Sequence Alignment

A situation which fits this framework perfectly is the alignment of multiple genome sequences in bioinformatics, where such multiple sequence alignments (MSAs) are standardly evaluated by sum-of-pairs scoring [Durbin *et al.*, 1998]. MSA is a generalization of the longest-common-subsequence problem, in which one is given sequences like those in figure 1a, and asked to produce pointwise alignments. Alignments of d sequences $\{s_1, \ldots s_d\}$ consist of t vertical timeslices which specify, for each sequence, either a successive element of the sequence or a gap (–), and are such that if the gaps are removed, the rows contain the original sequences. The score of a timeslice is the sum of the scores of each of the pairs

[1] We will talk about minimum sums, but other semirings work as well, specifically maximum products.

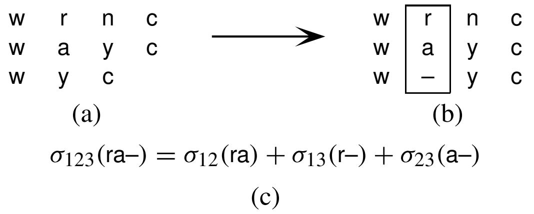


Figure 1: An example of a multiple sequence alignment. (a) The original sequences. (b) A multiple alignment with one timeslice distinguished. (c) The sum-of-pairs scoring function for that timeslice.

in that slice (where each pair of symbols is assigned some experimental goodness value).

The well-known dynamic program for calculating optimal multiple alignments involves a lattice of position nodes $n = [i_1, \ldots i_k]$ which specify an index along each sequence [Durbin *et al.*, 1998]. When each node is visited, each of its $2^k - 1$ successors p' (where each position is either incremented or not) are relaxed with the best score through p combined with the score of the timeslice that the change from p to p' represents. These nodes form a lattice of size $O(n^k)$, where n is the maximum length of the sequences. This becomes extremely inefficient as k grows.

The specific following idea has been used before ([Ikeda and Imai, 1994] is the earliest reference we could find) and it has been worked on recently ([Yoshizumi *et al.*, 2000] *inter alia*), though perhaps it has not received the attention it deserves in the bioinformatics literature. We present it here because (1) it is a good example that can be cast into our framework and (2) it gives a good starting intuition for the novel cases below. Since the score of an arc (timeslice) is a sum of pairwise goodness scores, we can define a set $\{\pi_{ab}\}$ of projections, one onto each pair of indices (i_a, i_b). Under π_{ab}, a node $[i_1, \ldots, i_k]$ will project to its a and b indices, $[i_a, i_b]$. It is easy to see that the optimal path in this projection is just the optimal 2-way alignment of the portions of sequences a and b which are to the right of the indices i_a and i_b, respectively. We can therefore bound the total completion cost of the k-way alignment from n onward with the sum of the pairwise completion costs of the 2-way alignments inside each projection.

Figure 2 shows some experimental speed-ups given by this method, compared to exhaustive search. We took several protein sequence groups from [McClure *et al.*, 1994], and, for each set, we aligned as large a subset of each group as was possible using uniform-cost search with 1GB of memory. The left four runs show the cost (in nodes visited) of aligning these subsets with both uniform-cost search and A* search. In the right four runs, we added another sequence to the subsets and solved the multiple alignment using only the A* search. The A* savings are substantial, usually providing several orders of magnitude over uniform-cost search, and many orders of magnitude over the exhaustive dynamic programming approach.[2] This verifies previous findings, and

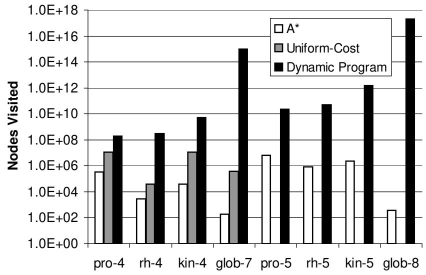


Figure 2: Effectiveness of the factored A* bound for the multiple alignment of several sequence sets. Numbers in the data set names give the number of sequences being aligned. Note the log scale: for example, on glob-7 the A* search is over a trillion times more efficient than exhaustive search.

shows that factored A* bounds can be highly effective.

One potential worry with A* search is that the cost of computing heuristics can be substantial. In this case, the $O(k^2)$ pairwise alignments could be calculated very efficiently; in our experiments this pre-search phase took up much less than 1% of the total time.

3.2 Example: Finite State Transducers

We briefly describe another potential application of factored A* search for sequence models: the intersection of weighted finite-state transducers (WFSTs).[3] WFSTs are probabilistic mappings between sequences in one alphabet to sequences in another, for example a transducer might map an input of written text to an output of that text's pronunciation as a phoneme sequence. Intersections of WFSTs have been applied to various tasks in speech and NLP [Mohri, 1997], such as text-to-speech, and, most famously in the NLP literature, modeling morphophonology [Kaplan and Kay, 1994; Albro, 2000]. In these cases, each transducer constrains some small portion of the overall output sequence. The case of finding the most likely intersected output of a set of WFSTs $\{M_i\}$ for an input sequence $\mathbf{w} = {}_0w_n$ involves the following:

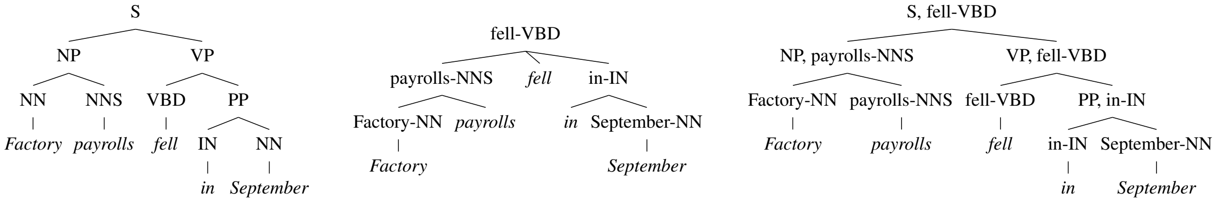
1. For each M_i, create the projection π_i of the full output space O onto M_i's output space (note that this can be the identity projection).[4]
2. For each index j along $\mathbf{w}$ and each output in $\pi_i(O)$, compute optimal completion costs $\alpha_i(j, \pi_i(O))$ for M_i.
3. Use $h(i, O) = \sum_i \alpha_i(j, \pi_i(O))$ as an A* heuristic.

While transduction intersection fits cleanly into the factored framework, the primary utility of transducers lies in their composition, not their intersection [Mohri, 1997]. In this case, transducers are chained together, with the output of one serving as the input to the next. In this case, it is worth switching from talk of summed distances to talk of multiplied probabilities. Say we have two transducers, M_{IX} which gives a distribution $P(X|I)$ from sequences I to sequences X, and M_{XO}, which gives $P(O|X)$ from X to O.

[2] Note that the DP was never run – it would not have fit in memory – but it is easy to calculate the size of the lattice. There are also subtleties in running the uniform-cost search since the score of a timeslice can be negative (we add in a worst possible negative score).

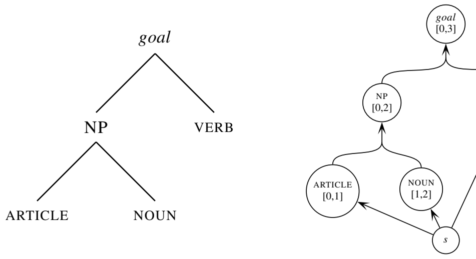
[3] WFSTs are equivalent to HMMs which have emission weights assigned to their transitions (not states) and which may have epsilon transitions.

[4] For simplicity, we assume all history relevant to any transducer is encoded into the state space O.



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

Figure 3: Three kinds of parse structures.



(a) (b)

Figure 4: Two representations of a parse: (a) a tree, (b) a path in the edge hypergraph.

We then wish to answer questions about their composed behavior. For example, we might want to find the output o which maximizes $P(o|i)$ according to this model. The common Viterbi approximation is to settle for the o from the pair (o, x) which maximizes $P(o, x|i)$. This problem would fit neatly into the factored framework if the (usually false) conditional independence $P(o, x|i) = P(o|i)P(x|i))$ where true – in fact it would then be WFST intersection. However, something close to this does trivially hold: $P(o, x|i) = P(o|x, i)P(x|i)$. Given this, we can define another model $R(o|i) = \max_x P(o|x, i)$. R is not a proper probabilistic model – it might well assign probability one to every transduction – but its intersection with $P(x|i)$ does upper-bound the actual composed model. Hence these two projections provide a factored bound for a non-factored model, with the practical utility of this bound depending on how tightly $R(o|i)$ typically bounds $P(o|x, i)$.

4 Projection Models for Trees

Search in the case of trees is not over standard directed graphs, but over a certain kind of directed hypergraph in which arcs can have multiple sources (but have a single target). This is because multiple sub-trees are needed to form a larger tree.[5] Figure 4 shows a fragment of such a hypergraph for a small parse tree (note that all the lines going to one arrowhead represent a *single* hyperarc). We don't give the full definitions of hypergraph search here (see [Gallo *et al.*, 1993] for details), but the basic idea is that one cannot traverse an arc until all its source nodes have been visited. In the parse case, for example, we cannot build a sentence node until we build both a noun phrase node and an (adjacent) verb phrase node. The nodes in this graph are identified by a grammar symbol, along with the region of the input it spans. The goal node is then a parse of the root symbol over the entire input. (Hyper)paths embody trees, and the score of a path is the combination of the scores of the arcs in the tree. One fine point is that, while a standard path from a source s to a goal g through a node n breaks up into two smaller paths (s to n and n to g), in the tree case there will be an inside path and an outside path, as shown in the right of figure 5. In general, then, the completion structures that represent paths to the goal (marked by α in the figure) are specified not only by a node n and goal g, but also by the original source s.

With this modification, the recipe for the factored A* bound is now:

$$\alpha_G(s, n, g) \geq h(s, n, g) = \sum\nolimits_i \alpha_{\pi_i(G)}(\pi_i(s), \pi_i(n), \pi_i(g))$$

Next, we present a concrete projection model for scoring lexicalized trees, and construct an A* parser using the associated factored A* bound.

Generative models for parsing natural language typically model one of the kinds of structures shown in figure 3. While word-free syntactic configurations like those embodied by phrase structure trees (figure 3a) are good at capturing the broad linear syntax of a language [Charniak, 1996], word-to-word affinities like those embodied by lexical dependency trees (figure 3b) have been shown to be important in resolving difficult ambiguities [Hindle and Rooth, 1993]. Since both kinds of information are relevant to parsing, the trend has been to model lexicalized phrase structure trees like figure 3c.

In our current framework, it is natural to think of a lexicalized tree as a pair $L = (T, D)$ of a phrase structure tree T and a dependency tree D. In this view, generative models over lexicalized trees, of the sort standard in lexicalized PCFG parsing [Collins, 1999; Charniak, 2000], can be regarded as assigning mass $P(T, D)$ to such pairs. In the standard approach, one builds a joint model over $P(T, D)$, and, for a given word sequence ${}_0w_n$, one searches for the maximum posterior parse:

$$L^* = \max\nolimits_{L=(T,D)} P(T, D|w)$$

Since $P(w)$ is a constant, one operationally searches instead for the maximizer of $P(T, D, w)$.

The naive way to do this is an $O(n^5)$ dynamic program (often called a tabular parser or chart parser) that works as follows. The core declarative object is an *edge*, such as $e = [X, i, j, h]$ which encapsulates all parses of the span ${}_iw_j$ which are labeled with grammar symbol X and are headed by word w_h ($i \leq h < j$). Edges correspond to the nodes in the

[5]These directed B-hypergraphs model what has been explored as AND/OR trees in AI.

1. Extract the PCFG projection and set up the PCFG parser.
2. Use the PCFG parser to find projection scores $\alpha_{\text{PCFG}}(s, e, g)$ for each edge.
3. Extract the dependency projection and set up the dependency parser.
4. Use the dependency parser to find projection scores $\alpha_{\text{DEP}}(s, e, g)$ for each edge.
5. Combine PCFG and dependency projections into the full model.
6. Form the factored A* estimate $h(s, e, g) = \alpha_{\text{PCFG}}(s, e, g) + \alpha_{\text{DEP}}(s, e, g)$
7. Use the combined parser, with $h(s, e, g)$ as an A* estimate of $\alpha(s, e, g)$

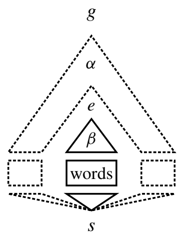


Figure 5: The top-level algorithm (left) and an illustration of how paths decompose in the parsing hypergraph (right).

parsing hypergraph. Two edges $[X, i, j, h]$ and $[Y, j, k, h']$ can be combined whenever they are contiguous (the right one starts where the left one ends) and the grammar permits the combination. For example, if there were a rewrite $Z[w_h] \rightarrow X[w_h]Y[w_{h'}]$, those two edges would combine to form $[Z, i, k, h]$, and that combination would be scored by some joint model over the word and symbol configuration: $P(XY, h'|Z, h)$.[6] These weighted combinations are the arcs in the hypergraph.

A natural projection of a lexicalized tree L is onto its components T and D (though, to our knowledge, this projection has not been exploited previously). In this case, the score for the combination above would be $P(XY, h'|Z, h) = P(XY|Z)P(h'|h)$.[7]

This kind of projected model offers two primary benefits. First, since we are building component models over much simpler projections, they can be designed, engineered, and tested modularly, and easily. To underscore this point, we built three PCFG models of $P(T)$ and two lexical dependency models of $P(T)$. In section 4.2, we discuss the accuracy of these models, both alone and in combination.

Second, our A* heuristic will be loose only to the degree that the two models prefer different structures. Therefore, the combined search only needs to figure out how to optimally reconcile these differences, not explore the entire space of legal structures. Figure 6 shows the amount of work done in the uniform-cost case versus the A* case. Clearly, the uniform-cost version of the parser is dramatically less efficient; by sentence length 15 it extracts over 800K edges, while even at length 40 the A* heuristics are so effective that only around 2K edges are extracted. At length 10, the average number is less than 80, and the fraction of edges not suppressed is better than 1/10K (and it improves as sentence length increases).[8] 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 initial, array-based PCFG phase (see figure 6b).[9]

4.1 Specific Projection Models for Parsing

To test our factored parser, we built several component models, which were intended to show the modularity of the approach. We merely sketch the individual models here; more details can be found in [Klein and Manning, 2003]. For $P(T)$, we built successively more accurate PCFGs. The simplest, PCFG-BASIC, used the raw treebank grammar, with nonterminals and rewrites taken directly from the training trees [Charniak, 1996]. In this model, nodes rewrite atomically, in a top-down manner, in only the ways observed in the training data. For improved models of $P(T)$, tree nodes' labels were annotated with various contextual markers. In PCFG-PA, each node was marked with its parent's label as in [Johnson, 1998]. It is now well known that such annotation improves the accuracy of PCFG parsing by weakening the PCFG independence assumptions. For example, the NP in figure 3a would actually have been labeled NPˆS. Since the counts were not fragmented by head word or head tag, we were able to directly use the MLE parameters, without smoothing.[10] The best PCFG model, PCFG-LING, involved selective parent splitting, order-2 rule markovization (similar to [Collins, 1999; Charniak, 2000]), and linguistically-derived feature splits.

[6] Most models, including ours, will also mention distance; we ignore this for now.

[7] As a probabilistic model, this formulation is mass deficient, assigning 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. In particular, this situation fits into the product-of-experts framework [Hinton, 2000], 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. In any case, the question of mass deficiency impacts only parameter estimation, not inference, which is our focus here.

[8] Note that the uniform-cost parser does enough work to exploit the shared structure of the dynamic program, and therefore edge counts appear to grow polynomially. However, the A* parser does so little work that there is minimal structure-sharing. Its edge counts therefore appear to grow *exponentially* over these sentence lengths, just like a non-dynamic-programming parser's would. With much longer sentences, or a less efficient estimate, the polynomial behavior would reappear.

[9] There are other ways of speeding up lexicalized parsing without sacrificing search optimality. Eisner and Satta [Eisner and Satta, 1999] propose a clever $O(n^4)$ modification which separates this process into two steps by introducing an intermediate object. However, even the $O(n^4)$ formulation is impractical for exhaustive parsing with broad-coverage, lexicalized treebank grammars. The essential reason is that the non-terminal set is just too large. We did implement a version of this parser using their $O(n^4)$ formulation, but, because of the effectiveness of the A* estimate, it was only marginally faster; as figure 6b shows, the combined search time is very small.

[10] 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.

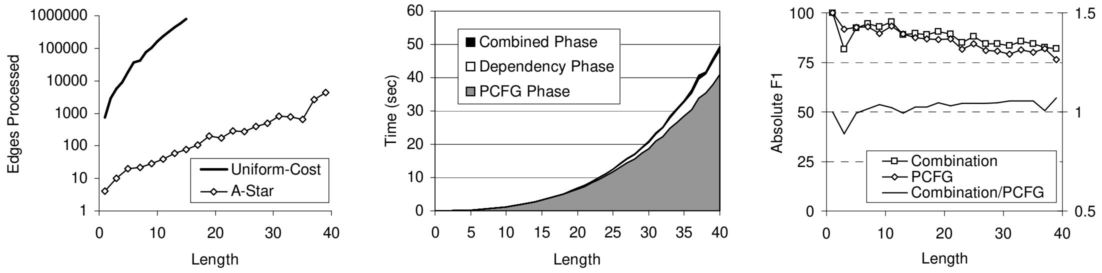


Figure 6: (a) A* effectiveness measured by edges expanded, (b) time spent on each phase, and (c) relative F_1, all as sentence length increases.

PCFG Model	Precision	Recall	F_1	Exact Match
PCFG-BASIC	75.3	70.2	72.7	11.0
PCFG-PA	78.4	76.9	77.7	18.5
PCFG-LING	83.7	82.1	82.9	25.7

(a) PCFG Models Alone

Dependency Model	Dependency Acc
DEP-BASIC	76.3
DEP-VAL	85.0

(b) Dependency Models Alone

Figure 7: Performance of the projection models alone.

PCFG Model	Dependency Model	Prec	Rec	F_1	Exact	DepAcc
PCFG-BASIC	DEP-BASIC	80.1	78.2	79.1	16.7	87.2
PCFG-BASIC	DEP-VAL	82.5	81.5	82.0	17.7	89.2
PCFG-PA	DEP-BASIC	82.1	82.2	82.1	23.7	88.0
PCFG-PA	DEP-VAL	84.0	85.0	84.5	24.8	89.7
PCFG-LING	DEP-BASIC	85.4	84.8	85.1	30.4	90.3
PCFG-LING	DEP-VAL	86.6	86.8	86.7	32.1	91.0

Figure 8: The combined model, with various projection models.

Models of $P(D)$ were lexical dependency models, which deal with part-of-speech 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 $\diamond$ is generated, then successive left dependents until $\diamond$ is generated again. For example, in figure 3, first we choose *fell*-VBD as the head of the sentence. Then, we generate *in*-IN to the right, which then generates *September*-NN to the right, which generates $\diamond$ on both sides. We then return to *in*-IN, generate $\diamond$ to the right, and so on. The dependency models required smoothing, as the word-word dependency data is very sparse. In our basic model, DEP-BASIC, we generate a dependent conditioned on the head and direction, requiring a model of $P(w_d, t_d|w_h, t_h, dir)$. This was estimated using a back-off model which interpolated the sparse bilexical counts with the denser but less specific counts given by ignoring the head word or by first generating the dependent tag and then generating the dependent word given only the dependent tag. The interpolation parameters were estimated on held-out data. The resulting model can thus capture classical bilexical selection, such as the affinity between *payrolls* and *fell*, as well as monolexical preferences, such as the tendency for *of* to modify nouns. In the enhanced dependency model, DEP-VAL, we condition not only on direction, but also on distance and valence. Note that this is (intentionally) very similar to the generative model of [Collins, 1999] in broad structure, but substantially less complex.

4.2 Parsing Performance

In this section, we describe our various projection models and test their empirical performance. There are two ways to measure the accuracy of the parses produced by our system. First, the phrase structure of the PCFG and the phrase structure projection of the combination parsers can be compared to the treebank parses. The parsing measures standardly used for this task are labeled precision and recall.[11] We also report F_1, the harmonic mean of these two quantities. Second, for the dependency 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 7 shows the relevant scores for the various PCFG and dependency parsers alone. The valence model increases the dependency model's accuracy from 76.3% to 85.0%, and each successive enhancement improves the F_1 of the PCFG models, from 72.7% to 77.7% to 82.9%. The combination parser's performance is given in figure 8. As each individual model is improved, the combination F_1 is also improved, from 79.1% with the pair of basic models to 86.7% with the pair of top models. The dependency accuracy also goes up: from 87.2% to 91.0%. Note, however, that even the pair of basic models has a combined dependency accuracy higher than the enhanced dependency model alone, and the top three have combined F_1 better than the best PCFG model alone. For the top pair, figure 6c illustrates the relative F_1 of the combination parser to the PCFG component alone, showing the unsurprising trend that the addition of the dependency model helps

[11] 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 [Magerman, 1995], and involve minor normalizations like the removal of punctuation in the comparison.

more for longer sentences, which, on average, contain more attachment ambiguity. The top F_1 of 86.7% is greater than that of the lexicalized parsers presented in [Magerman, 1995; Collins, 1996], but less than that of the newer, more complex, parsers presented in [Charniak, 2000; Collins, 1999], 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 applied to benefit our individual models.[12]

4.3 Factored Bounds for Non-Projection Models

Arbitrary tree models will not be factored projection models. For example, while our parsing model was expressly designed so that $P(T, D) = P(T)P(D)$, to our knowledge no other model over lexicalized trees with this decomposition has been proposed. Nonetheless, non-factored models can still have factored bounds. Given any model $P(A, B)$, we can imagine bounds $R(A)$ and $R(B)$ that obey:

$$\forall A, B : P(A, B) < R(A)R(B)$$

Trivially, $R(A) = R(B) = 1$ will do. To get a non-trivial bound, consider a joint (local) model $P(XY, h'|Z, h)$ of lexicalized tree rewrites. Early lexicalized parsing work [Charniak, 1997] used models of exactly this form. We can use the chain rule to write $P(XY, h'|Z, h) = P(XY|Z, h)P(h'|XY, Z, h)$. Then, we can form $R(XY|Z) = \max_h P(XY|Z, h)$ and $R(h'|h) = \max_{XY,Z} P(h'|XY, Z, h)$. This technique allows one to use factored A* search for non-factored models, though one might reasonably expect such bounds to be much less sharp for non-factored models than for factored models. A particular application of this method for future work would be the exact parsing of the models in [Charniak, 1996; Collins, 1999; Charniak, 2000], as the details of their estimation suggest that their word dependency and phrase structure aspects would be approximately factorizable.

5 Conclusion

Not all models will factor, nor will all models which factor necessarily have tight factored bounds (for example MSA with many sequences or parsing if the component models do not prefer similar structures). However, when we can design factored models or find good factored bounds, the method of factored A* search has proven very effective. For the MSA problem, A* methods allow exact alignment of up to 9 protein sequences (though 5–6 is more typical) of length 100–300, when even three-way exhaustive alignment can easily exhaust memory. For the parsing problem, we have presented here the first optimal lexicalized parser which can exactly parse sentences of reasonable length using large real-world Penn Treebank grammars. The projected models can be designed and improved modularly, with improvements to each model raising the combined accuracy. Finally, we hope that this framework can be profitably used on the other sequence models we outlined, and on any large space which can naturally be viewed as a composition of (possibly overlapping) projections.

[12] For example, the dependency distance function of [Collins, 1999] registers punctuation and verb counts, and both smooth the PCFG production probabilities.

Acknowledgements

We would like to thank Lillian Lee, Fernando Pereira, and Dan Melamed for advice and discussion about this work.

References

[Albro, 2000] Daniel M. Albro. Taking primitive optimality theory beyond the finite state. In *Proceedings of the Special Interest Group in Computational Phonology*, 2000.

[Charniak, 1996] Eugene Charniak. Tree-bank grammars. In *AAAI 13*, pages 1031–1036, 1996.

[Charniak, 1997] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *AAAI 14*, pages 598–603, 1997.

[Charniak, 2000] Eugene Charniak. A maximum-entropy-inspired parser. In *NAACL 1*, pages 132–139, 2000.

[Collins, 1996] Michael John Collins. A new statistical parser based on bigram lexical dependencies. In *ACL 34*, pages 184–191, 1996.

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

[Durbin *et al.*, 1998] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[Eisner and Satta, 1999] Jason Eisner and Giorgio Satta. Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *ACL 37*, pages 457–464, 1999.

[Gallo *et al.*, 1993] G. Gallo, G. Longo, S. Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42:177–201, 1993.

[Hindle and Rooth, 1993] Donald Hindle and Mats Rooth. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120, 1993.

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

[Ikeda and Imai, 1994] T. Ikeda and T. Imai. Fast A* algorithms for multiple sequence alignment. In *Genome Informatics Workshop V*, pages 90–99, 1994.

[Johnson, 1998] Mark Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24:613–632, 1998.

[Kaplan and Kay, 1994] Ron Kaplan and Martin Kay. Regular model of phonological rule systems. *Computational Linguistics*, 20:331–378, 1994.

[Klein and Manning, 2003] Dan Klein and Christopher D. Manning. Fast exact inference with a factored model for natural language parsing. In *NIPS*, volume 15. MIT Press, 2003.

[Magerman, 1995] David M. Magerman. Statistical decision-tree models for parsing. In *ACL 33*, pages 276–283, 1995.

[McClure *et al.*, 1994] M.A McClure, T.K. Vasi, and W.M. Fitch. Comparative analysis of multiple protein-sequence alignment methods. *Molecular Biology and Evolution*, 11:571–592, 1994.

[Mohri, 1997] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(4):269–311, 1997.

[Yoshizumi *et al.*, 2000] Takayuki Yoshizumi, Teruhisa Miura, and Toru Ishida. A* with partial expansion for large branching factor problems. In *AAAI/IAAI*, pages 923–929, 2000.