A* Parsing: Fast Exact Viterbi Parse Selection

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

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

Abstract

We present an extension of the classic A* search procedure to tabular PCFG parsing. The use of A* search can dramatically reduce the time required to find a best parse by conservatively estimating the probabilities of parse completions. We discuss various estimates and give efficient algorithms for computing them. On average-length Penn treebank sentences, our most detailed estimate reduces the total number of edges processed to less than 3% of that required by exhaustive parsing, and a simpler estimate, which requires less than a minute of precomputation, reduces the work to less than 5%. Unlike best-first and finite-beam methods for achieving this kind of speed-up, an A* method is guaranteed to find the most likely parse, not just an approximation. Our parser, which is simpler to implement than an upward-propagating best-first parser, is correct for a wide range of parser control strategies and maintains worst-case cubic time.

1 Introduction

PCFG parsing algorithms with worst-case cubic-time bounds are well-known. However, when dealing with wide-coverage grammars and long sentences, even cubic algorithms can be far too expensive in practice. Two primary types of methods for accelerating parse selection have been proposed. Roark (2001) and Ratnaparkhi (1999) use a beam-search strategy, in which only the best n parses are tracked at any moment. Parsing time is linear and can be made arbitrarily fast by reducing n. This is a greedy strategy, and the actual Viterbi (highest probability) parse can be pruned from the beam because, while it is globally optimal, it may not be locally optimal at every parse stage. Chitrao and Grishman (1990), Caraballo and Charniak (1998), Charniak et al. (1998), and Collins (1999) describe best-first parsing, which is intended for a tabular item-based framework. In best-first parsing, one builds a *figure-of-merit* (FOM) over parser items, and uses the FOM to decide the order in which agenda items should be processed. This approach also dramatically reduces the work done during parsing, though it, too, gives no guarantee that the first parse returned is the actual Viterbi parse (nor does it maintain a worst-case cubic time bound). We discuss best-first parsing further in section 3.3.

Both of these speed-up techniques are based on greedy models of parser actions. The beam search greedily prunes partial parses at each beam stage, and a best-first FOM greedily orders parse item exploration. If we wish to maintain optimality in a search procedure, the obvious thing to try is A* methods (see for example Russell and Norvig, 1995). We apply A* search to a tabular item-based parser, ordering the parse items based on a combination of their known internal cost of construction and a conservative estimate of their cost of completion (see figure 1). A* search has been proposed and used for speech applications (Goel and Byrne, 1999, Corazza et al., 1994); however, it has been little used, certainly in the recent statistical parsing literature, apparently because of difficulty in conceptualizing and computing effective admissible estimates. The contribution of this paper is to demonstrate effective ways of doing this, by precomputing grammar statistics which can be used as effective A* estimates.

The A* formulation provides three benefits. First, it substantially reduces the work required to parse a sentence, without sacrificing either the optimality of the answer or the worst-case cubic time bounds on the parser. Second, the resulting parser is structurally simpler than a FOM-driven best-first parser. Finally, it allows us to easily prove the correctness of our algorithm, over a broad range of control strategies and grammar encodings.

In this paper, we describe two methods of constructing A* bounds for PCFGs. One involves context summarization, which uses estimates of the sort proposed in Corazza et al. (1994), but considering richer summaries. The other involves grammar summarization, which, to our knowledge, is entirely novel. We present the estimates that we use, along with algorithms to efficiently calculate them, and illustrate their effectiveness in a tabular PCFG parsing algorithm, applied to Penn Treebank sentences.

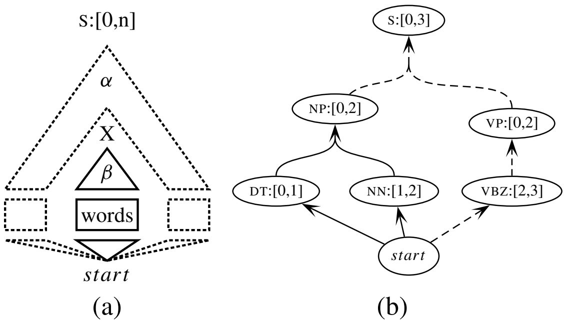


Figure 1: A* edge costs. (a) The cost of an edge X is a combination of the cost to build the edge (the Viterbi inside score β) and the cost to incorporate it into a root parse (the Viterbi outside score α). (b) In the corresponding hypergraph, we have exact values for the inside score from the explored hyperedges (solid lines), and use upper bounds on the outside score, which estimate the dashed hyperedges.

2 An A* Algorithm

An agenda-based PCFG parser operates on parse items called *edges*, such as NP:[0,2], which denote a grammar symbol over a span. The parser maintains two data structures: a chart or table, which records edges for which (best) parses have already been found, and an agenda of newly-formed edges waiting to be processed. The core loop involves removing an edge from the agenda and combining that edge with edges already in the chart to create new edges. For example, NP:[0,2] might be removed from the agenda, and, if there were a rule S $\rightarrow$ NP VP and VP:[2,8] was already entered into the chart, the edge S:[0,8] would be formed, and added to the agenda if it were not in the chart already.

The way an A* parser differs from a classic chart parser is that, like a best-first parser, agenda edges are processed according to a priority. In best-first parsing, this priority is called a *figure-of-merit* (FOM), and is based on various approximations to $P(e|s)$, the fraction of parses of a sentence s which include an edge e (though see Goodman (1997) for an alternative notion of FOM). Edges which seem promising are explored first; others can wait on the agenda indefinitely. Note that even if we did know $P(e|s)$ exactly, we still would not know whether e occurs in any *best* parse of s. Nonetheless, good FOMs empirically lead quickly to good parses. Best-first parsing aims to find a (hopefully good) parse quickly, but gives no guarantee that the first parse discovered is the Viterbi parse, nor does it allow one to recognize the Viterbi parse when it is found.

In A* parsing, we wish to construct priorities which will speed up parsing, yet still guarantee *optimality* (that the first parse returned is indeed a best parse). With a categorical CFG chart parser run to exhaustion, it does not matter in what order one removes edges from the agenda; all edges involved in full parses of the sentence will be constructed at some point. A cubic time bound follows straightforwardly by simply testing for edge existence, ensuring that we never process an edge twice. With PCFG parsing, there is a subtlety involved. In addition to knowing whether edges can be constructed, we also want to know the scores of edges' best parses. Therefore, we record estimates of best-parse scores, updating them as better parses are found. If, during parsing, we find a new, better way to construct some edge e that has previously been entered into the chart, we may also have found a better way to construct any edges which have already been built using e. Best-first parsers deal with this by allowing an upward propagation, which updates such edges' scores (Caraballo and Charniak, 1998). If run to exhaustion, all edges' Viterbi scores will be correct, but the propagation destroys the cubic time bound of the parser, since in effect each edge can be processed many times.

In order to ensure optimality, it is sufficient that, for any edge e, all edges f which are contained in a best parse of e get removed from the agenda before e itself does. If we have an edge priority which ensures this ordering, we can avoid upward propagation entirely (and omit the data structures involved in it) and still be sure that each edge leaves the agenda scored correctly. If the grammar happens to be in CNF, one way to do this is to give shorter spans higher priority than longer ones; this priority essentially gives the CKY algorithm.

Formally, assume we have a PCFG G and a sentence $s = {}_0w_n$ (we place indices as fenceposts between words). An *inside parse* of an edge $e = X$:$[i, j]$ is a derivation in G from X to ${}_iw_j$. Let $\beta_G(e, s)$ denote the log-probability of a best inside parse of e (its *Viterbi inside score*).[1] We will drop the G, s, and even e when context permits. Our parser, like a best-first parser, maintains estimates $b(e, s)$ of $\beta(e, s)$ which begin at $-\infty$, only increase over time, and always represent the score of the best parses of their edges e discovered so far. Optimality means that for any e, $b(e, s)$ will equal $\beta_G(e, s)$ when e is removed from the agenda.

If one uses $b(e, s)$ to prioritize edges, we show in Klein and Manning (2001a), that the parser is optimal over arbitrary PCFGs, and a wide range of control strategies. This is proved using an extension of Dijkstra's algorithm to a certain kind of hypergraph associated with parsing, shown in figure 1(b): parse items are nodes in the hypergraph, hyperarcs take sets of parse items to their result item, and hyperpaths map to parses. Reachability from *start* corresponds to parseability, and shortest paths to Viterbi parses.

[1] Our use of *inside score* and *outside score* evokes the same picture as talk about inside and outside probabilities, but note that in this paper inside and outside scores always refer to (a bound on) the maximum (Viterbi) probability parse inside or outside some edge, rather than to the sum for all such parses.

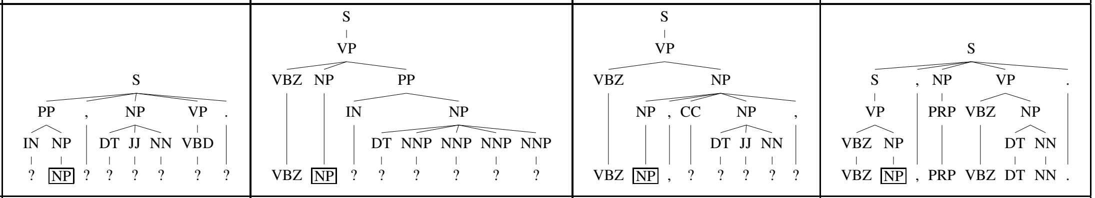
Estimate	SX	SXL	SXLR	TRUE
Summary	(1,6,NP)	(1,6,NP,VBZ)	(1,6,NP,VBZ,“,”)	(entire context)
Best Tree				
Score	-11.3	-13.9	-15.1	-18.1
	(a)	(b)	(c)	(d)

Figure 2: Best outside parses given richer summaries of edge context. (a – SX) Knowing only the edge state (NP) and the left and right outside spans, (b – SXL) also knowing the left tag, (c – SXLR) left and right tags, and (d – TRUE) the entire outside context.

The hypergraph shown in figure 1(b) shows a parse of the goal S:[0,3] which includes NP:[0,2].[2] This parse can be split into an inside portion (solid lines) and an outside portion (dashed lines), as indicated in figure 1(a). The outside portion is an *outside parse*: formally, an outside parse of an edge X:$[i, j]$ in sentence $s = {}_0w_n$ is a derivation from G's root symbol to $w_{0i}Xw_{jn}$. We use $\alpha_G(e, s)$ to denote the score of a best outside parse of e.

Using $b(e, s)$ as the edge priority corresponds to a generalization of uniform cost search on graphs (Russell and Norvig, 1995). In the analogous generalization of A* search, we add to $b(e, s)$ an estimate $a(e, s)$ of the competion cost $\alpha_G(e, s)$ (the cost of the dashed outside parse) to focus exploration on regions of the graph which appear to have good *total* cost.

A* search is correct as long as the estimate a satisfies two conditions. First, it must be *admissible*, meaning that it must not underestimate the actual log-probability required to complete the parse. Second, it must be monotonic, meaning that as one builds up a tree, the combined log-probability $\beta + a$ never increases. The proof of this is very similar to the proof of the uniform-cost case in Klein and Manning (2001a), and so we omit it for space reasons (it can be found in Klein and Manning, 2002).

Concretely, we can use $b + a$ as the edge priority, provided a is an admissible, monotonic estimate of α. We will still have a correct algorithm, and even rough heuristics can dramatically cut down the number of edges processed (and therefore total work). We next discuss several estimates, describe how to compute them efficiently, and show the edge savings when parsing Penn treebank WSJ sentences.

3 A* Estimates for Parsing

When parsing with a PCFG G, each edge $e = X$:$[i, j]$ spans some interval $[i, j]$ of the sentence and is labeled by some grammar symbol (or *state*) X. Our presentation assumes that G is a binarized grammar, and so in general X may be either a *complete* state like NP that was in an original n-ary grammar, or an intermediate state, like an Earley dotted rule, that is the result of implicit or explicit grammar binarization. For the edge e, its yield in $s = {}_0w_n$ is the sequence of terminals that it spans (${}_iw_j$). Its *context* is its state X along with the rest of the terminals of sentence (${}_0w_i X {}_jw_n$). Scores are log-probabilities; lower cost is higher log-probability. So, ‘$>$’ or ‘better’ will mean higher log-probability.

3.1 Context Summary Estimates

One way to construct an admissible estimate is to summarize the context in some way, and to find the score of the best parse of *any* context that fits that summary. Let $c(e, s)$ be the context of e in s. Let σ be a summary function of contexts. We can then use the *context summary estimate*:

$$a_\sigma(e, s) = \max_{\substack{(e', s'):\sigma(c(e',s'))= \\ \sigma(c(e,s))}} \alpha_G(e', s') \geq \alpha_G(e, s)$$

That is, we return the exact Viterbi outside score for *some* context, generally not the actual context, whose summary matches the actual one's summary. If the number of summaries is reasonable, we can precompute and store the estimate for each summary once and for all, then retrieve them in constant time per edge at parse time.

If we give no information in the summary, the estimate will be constantly 0. This is the trivial estimate NULL, and corresponds to simply using inside estimates b alone as priorities. On the other extreme, if each context had a unique summary, then $a(e, s)$ would be $\alpha_G(e, s)$ itself. This is the ideal estimate, which we call TRUE. In practice, of course, precomputing TRUE would not be feasible.[3]

[2]The example here shows a bottom-up construction of a parse tree. However, the present algorithm and estimates work just as well for top-down chart parsing, given suitable active items as nodes; see (Klein and Manning, 2001a).

[3]Note that our ideal estimate is not $P(e|s)$ like the ideal FOM, rather it is $P(T_{g,e})/P(T_e)$ (where $T_{g,e}$ is a best parse of the goal g among those which contain e, and T_e is a best parse of e over the yield of e). That is, we are not estimating *parser choice* probabilities, but *parse tree probabilities*.

We used various intermediate summaries, some illustrated in figure 2. S_1 specifies only the total number of words outside e, while S specifies separately the number to the left and right. SX also specifies e's label. SXL and SXR add the tags adjacent to e on the left and right respectively. S_1XLR includes both the left and right tags, but merges the number of words to the left and right.[4]

As the summaries become richer, the estimates become sharper. As an example, consider an NP in the context "VBZ $\boxed{\text{NP}}$, PRP VBZ DT NN ." shown in figure 2.[5] The summary SX reveals only that there is an NP with 1 word to the left and 6 the right, and gives an estimate of -11.3. This score is backed by the concrete parse shown in figure 2(a). This is a best parse of a context compatible with what little we specified, but very optimistic. It assumes very common tags in very common patterns. SXL adds that the tag to the left is VBZ, and the hypothesis that the NP is part of a sentence-initial PP must be abandoned; the best score drops to -13.9, backed by the parse in figure 2(b). Specifying the right tag to be "," drops the score further to -15.1, given by figure 2(c). The actual best parse is figure 2(d), with a score of -18.1.

These estimates are similar to quantities calculated in Corazza et al. (1994); in that work, they are interested in the related problem of finding best completions for strings which contain gaps. For the SX estimate, for example, the string would be the edge's label and two (fixed-length) gaps. They introduce quantities essentially the same as our SX estimate to fill gaps, and their one-word update algorithms are similarly related to those we use here. The primary difference here is in the application of these quantities, not their calculation.

3.2 Grammar Projection Estimates

The context summary estimates described above use local information, combined with span sizes. This gives the effect that, for larger contexts, the best parses which back the estimates will have less and less to do with the actual contexts (and hence will become increasingly optimistic). Context summary estimates do not pin down the exact context, but do use the original grammar G. For *grammar projection estimates*, we use the exact context, but project the grammar to some G' which is so much simpler that it is feasible to first exhaustively parse with G' and then use the result to guide the search in the full grammar G.

Formally, we have a projection π which maps grammar states of G (that is, the dotted rules of an Earley-style parser) to some reduced set. This projection of states induces a projection of rules. If a set $R = \{r\}$ of rules in G collide as the rule r' in G', we give r' the probability $P(r') = \max_{r \in R} P(r)$. Note that the resulting grammar G' will *not* generally be a proper PCFG; it may assign more than probability 1 to the set of trees it generates. In fact, it will usually assign infinite mass. However, all that matters for our purposes is that every tree in G projects under π to a tree in G' with the same or higher probability, which is true because every rule in G does. Therefore, we know that $\alpha_G(e, s) \leq \alpha_{G'}(e, s)$. If G' is much more compact than G, for each new sentence s, we can first rapidly calculate $a_\pi = \alpha_{G'}$ for all edges, then parse with G.

[4] Merging the left and right outside span sizes in S_1XLR was done solely to reduce memory usage.

[5] Our examples, and our experiments, use delexicalized sentences from the Penn treebank.

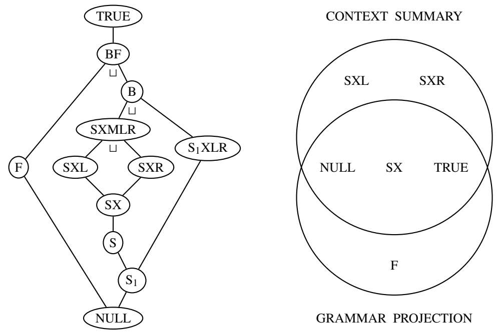
	Grammar State		
Projection	NP	CC	NP → · CC NP CC NP
NULL	X	X	X
SX	NP	X	NP → · X NP X NP
XBAR	NP	CC	NP′
F	X	CC	X → · CC X CC X
TRUE	NP	CC	NP → · CC NP CC NP

Figure 3: Examples of grammar state images under several grammar projections.

The identity projection ι returns G and therefore a_ι is TRUE. On the other extreme, a constant projection gives NULL (if any rewrite has probability 1). In between, we tried three other grammar projection estimates (examples in figure 3). First, consider mapping all terminal states to a single terminal token, but not altering the grammar in any other way. If we do this projection, then we get the SX estimate from the last section (collapsing the terminals together effectively hides which terminals are in the context, but not their number). However, the resulting grammar is nearly as large as G, and therefore it is much more efficient to use the precomputed context summary formulation. Second, for the projection XBAR, we tried collapsing all the incomplete states of each complete state to a single state (so NP→ · CC NP and NP→ · PP would both become NP′). This turned out to be ineffective, since most productions then had merged probability 1.

For our current grammar, the best estimate of this type was one we called F, for *filter*, which collapsed all complete (passive) symbols together, but did not collapse any terminal symbols. So, for example, a state like NP→ · CC NP CC NP would become X→ · CC X CC X (see section 3.3 for a description of our grammar encodings). This estimate has an interesting behavior which is complementary to the context summary estimates. It does not indicate well when an edge would be moderately expensive to integrate into a sentence, but it is able to completely eliminate certain edges which are impossible to integrate into a full parse (for example in this case maybe the two CC tags required to complete the NP are not present in the future context).

A close approximation to the F estimate can also be computed online especially quickly during parsing. Since



(a) (b)

Figure 4: (a) The A* estimates form a lattice. Lines indicate subsumption, $\sqcup$ indicates estimates which are the explicit join of lower estimates. (b) Context summary vs. grammar projection estimates: some estimates can be cast either way.

we are parsing with the Penn treebank covering grammar, almost any (phrasal) non-terminal can be built over almost any span. As discussed in Klein and Manning (2001b), the only source of constraint on what edges can be built where is the tags in the rules. Therefore, an edge with a label like NP$\rightarrow$ · CC NP CC NP can essentially be built whenever (and only whenever) two CC tags are in the edge's right context, one of them being immediately to the right. To the extent that this is true, F can be approximated by simply scanning for the tag configuration required by a state's local rule, and returning 0 if it is present and $-\infty$ otherwise. This is the method we used to implement F; exactly parsing with the projected grammar was much slower and did not result in substantial improvement.

It is worth explicitly discussing how the F estimate differs from top-down grammar-driven filtering standardly used by top-down chart parsers; in the treebank grammar, there is virtually no top-down filtering to be exploited (again, see Klein and Manning (2001b)). In a left-to-right parse, top-down filtering is a prefix licensing condition; F is more of a sophisticated lookahead condition on suffixes.

The relationships between all of these estimates are shown in figure 4. The estimates form a join lattice (figure 4(a)): adding context information to a merged context estimate can only sharpen the individual outside estimates. In this sense, for example S $\prec$ SX. The lattice top is TRUE and the bottom is NULL. In addition, the minimum ($\sqcup$) of a set of admissible estimates is still an admissible estimate. We can use this to combine our basic estimates into composite estimates: SXMLR = $\sqcup$ (SXL, SXR) will be valid, and a better estimate than either SXL or SXR individually. Similarly, B is $\sqcup$ (SXMLR, S_1XLR).

There are other useful grammar projections, which are beyond the scope of this paper. First, much recent statistical parsing work has gotten value from splitting grammar

Original Rules		Outside-Trie Rules		Inside-Trie Rules	
NP $\rightarrow$ DT JJ NN	0.3	NP $\rightarrow$ $X_{NP\rightarrow\cdot NN}$ NN	0.4	NP $\rightarrow$ $X_{DT\ JJ}$ NN	0.3
NP $\rightarrow$ DT NN NN	0.1	$X_{NP\rightarrow\cdot NN}$ $\rightarrow$ DT JJ	0.75	NP $\rightarrow$ $X_{DT\ NN}$ NN	0.1
		$X_{NP\rightarrow\cdot NN}$ $\rightarrow$ DT NN	0.25	$X_{DT\ JJ}$ $\rightarrow$ DT JJ	1.0
				$X_{DT\ NN}$ $\rightarrow$ DT NN	1.0

Figure 5: Two trie encodings of rules.

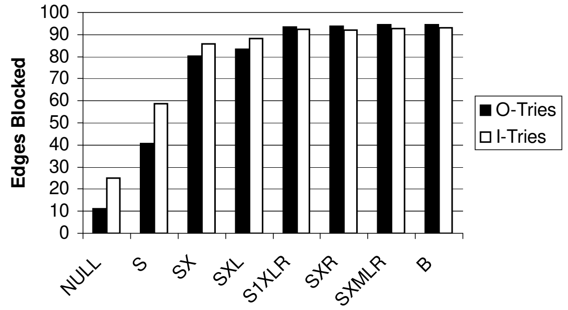


Figure 6: Fraction of edges saved by using various estimate methods, for two rule encodings. O-TRIE is a deterministic right-branching trie encoding (Leermakers, 1992) with weights pushed left (Mohri, 1997). I-TRIE is a non-deterministic left-branching trie with weights on rule entry as in Charniak et al. (1998).

states, such as by annotating nodes with their parent and even grandparent categories (Johnson, 1998). This annotation multiplies out the state space, giving a much larger grammar, and projecting back to the unannotated state set can be used as an outside estimate. Second, and perhaps more importantly, this technique can be applied to lexical parsing, where the state projections are onto the delexicalized PCFG symbols and/or onto the word-word dependency structures. This is particularly effective when the tree model takes a certain factored form; see Klein and Manning (2003) for details.

3.3 Parsing Performance

Following (Charniak et al., 1998), we parsed unseen sentences of length 18–26 from the Penn Treebank, using the grammar induced from the remainder of the treebank.[6] We tried all estimates described above.

Rules were encoded as both inside (I) and outside (O) tries, shown in figure 5. Such an encoding binarizes the grammar, and compacts it. I-tries are as in Charniak et al. (1998), where NP$\rightarrow$ DT JJ NN becomes NP $\rightarrow$ $X_{DT\ JJ}$ NN and $X_{DT\ JJ}$ $\rightarrow$ DT JJ, and correspond to dropping the portion of an Earley dotted rule after the dot.[7] O-tries, as in Leermakers (1992), turn NP$\rightarrow$ DT JJ NN into NP $\rightarrow$ $X_{NP\rightarrow\cdot NN}$ NN and $X_{NP\rightarrow\cdot NN}$ $\rightarrow$ DT JJ, and correspond to

[6] We chose the data set used by Charniak and coauthors, so as to facilitate comparison with previous work. We do however acknowledge that many of our current local estimates are less effective on longer spans, and so would work less well on 40–50 word sentences. This is an area of future research.

[7] In Charniak et al. (1998), the binarization is in the reverse direction; we binarize into a left chain because it is the standard direction implicit in chart parsers' dotted rules, and the direction makes little difference in edge counts.

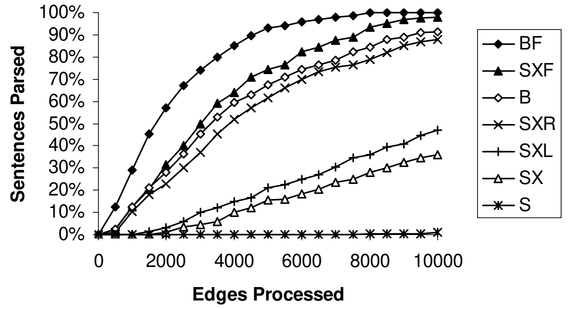


Figure 7: Number of sentences parsed as more edges are expanded. Sentences are Penn treebank sentences of length 18–26 parsed with the treebank grammar. A typical number of edges in an exhaustive parse is 150,000. Even relatively simple A* estimates allow substantial savings.

dropping the portion which precedes the dot. Figure 6 shows the overall savings for several estimates of each type. The I-tries were superior for the coarser estimates, while O-tries were superior for the finer estimates. In addition, only O-tries permit the accelerated version of F, since they explicitly declare their right requirements. Additionally, with I-tries, only the top-level intermediate rules have probability less than 1, while for O-tries, one can back-weight probability as in (Mohri, 1997), also shown in figure 5, enabling sub-parts of rare rules to be penalized even before they are completed.[8] For all subsequent results, we discuss only the O-trie numbers.

Figure 8 lists the overall savings for each context summary estimate, with and without F joined in. We see that the NULL estimate (i.e., uniform cost search) is not very effective – alone it only blocks 11% of the edges. But it is still better than exhaustive parsing: with it, one stops parsing when the best parse is found, while in exhaustive parsing one continues until no edges remain. Even the simplest non-trivial estimate, S, blocks 40% of the edges, and the best estimate BF blocks over 97% of the edges, a speed-up of over 35 times, without sacrificing optimality or algorithmic complexity.

For comparison to previous FOM work, figure 7 shows, for an edge count and an estimate, the proportion of sentences for which a first parse was found using at most that many edges. To situate our results, the FOMs used by (Caraballo and Charniak, 1998) require 10K edges to parse 96% of these sentences, while BF requires only 6K edges. On the other hand, the more complex, tuned FOM in (Charniak et al., 1998) is able to parse all of these sentences using around 2K edges, while BF requires 7K edges. Our estimates do not reduce the total edge count quite as much as the best FOMs can, but they are in the same range. This is as much as one could possibly expect, since, crucially, our first parses are always optimal, while the FOM parses need not be (and indeed sometimes are not).[9] Also, our parser never needs to propagate score changes upwards, and so may be expected to do less work overall per edge, all else being equal. This savings is substantial, even if no propagation is done, because no data structure needs to be created to track the edges which are supported by each given edge (for us, this represents a factor of approximately 2 in memory savings). Moreover, the context summary estimates require only a single table lookup per edge, while the accelerated version of F requires only a rapid quadratic scan of the input per sentence (less than 1% of parse time per sentence), followed by a table lookup per edge. The complex FOMs in (Charniak et al., 1998) require somewhat more online computation to assemble.

Estimate	Savings	w/ Filter	Storage	Precomp
NULL	11.2	58.3	0K	none
S	40.5	77.8	2.5K	1 min
SX	80.3	95.3	5M	1 min
SXL	83.5	96.1	250M	30 min
S_1XLR	93.5	96.5	500M	480 min
SXR	93.8	96.9	250M	30 min
SXMLR	94.3	97.1	500M	60 min
B	94.6	97.3	1G	540 min

Figure 8: The trade-off between online savings and precomputation time.

It is interesting that SXR is so much more effective than SXL; this is primarily because of the way that the rules have been encoded. If we factor the rules in the other direction, we get the opposite effect. Also, when combined with F, the difference in their performance drops from 10.3% to 0.8%; F is a right-filter and is partially redundant when added to SXR, but is orthogonal to SXL.

3.4 Estimate Sharpness

A disadvantage of admissibility for the context summary estimates is that, necessarily, they are overly optimistic as to the contents of the outside context. The larger the outside context, the farther the gap between the true cost and the estimate. Figure 9 shows average outside estimates for Viterbi edges as span size increases. For small outside spans, all estimates are fairly good approximations of TRUE. As the span increases, the approximations fall behind. Beyond the smallest outside spans, all of the curves are approximately linear, but the actual value's slope is roughly twice that of the estimates. The gap between our empirical methods and the true cost grows fairly steadily, but the differences between the empirical methods themselves stay relatively constant. This reflects

[8]However, context summary estimates which include the state compensate for this automatically.

[9]In fact, the bias from the FOM commonly raises the bracket accuracy slightly over the Viterbi parses, but that difference nevertheless demonstrates that the first parses are not always the Viterbi ones. In our experiments, non-optimal pruning sometimes bought slight per-node accuracy gains at the cost of a slight drop in exact match.

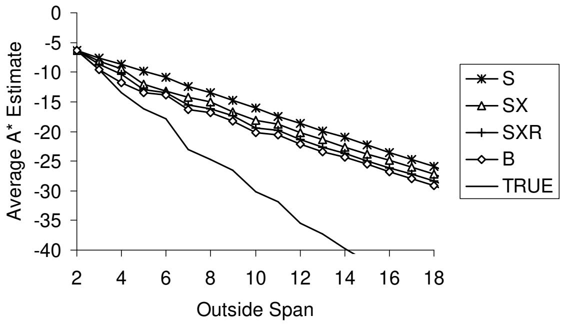


Figure 9: The average estimate by outside span length for various methods. For large outside spans, the estimates differ by relatively constant amounts.

the nature of these estimates: they have differing local information in their summaries, but all are equally ignorant about the more distant context elements. The various local environments can be more or less costly to integrate into a parse, but, within a few words, the local restrictions have been incorporated one way or another, and the estimates are all free to be equally optimistic about the remainder of the context. The cost to "package up" the local restrictions creates their constant differences, and the shared ignorance about the wider context causes their same-slope linear drop-off. This suggests that it would be interesting to explore other, more global, notions of context. We do not claim that our context estimates are the best possible – one could hope to find features of the context, such as number of verbs to the right or number of unusual tags in the context, which would partition the contexts more effectively than adjacent tags, especially as the outside context grows in size.

3.5 Estimate Computation

The amount of work required to (pre)calculate context summary estimates depends on how easy it is to efficiently take the max over all parses compatible with each context summary. The benefit provided by an estimate will depend on how well the restrictions in that summary nail down the important features of the full context.

Figure 10 shows recursive pseudocode for the SX estimate; the others are similar. To precalculate our A* estimates efficiently, we used a memoization approach rather than a dynamic programming approach. This resulted in code comparable in efficiency, but which was simpler to reason about, and, more importantly, allowed us to exploit sparseness when present. For example with left-factored trie encodings, 76% of (state, right tag) combinations are simply impossible. Tables which mapped arguments to returned results were used to memoize each procedure. In our experiments, we forced these tables to be filled in a precomputation step, but depending on the situation it might be advantageous to allow them to fill as needed, with early parses proceeding slowly while the

```
outsideSX(state, lspan, rspan)
  if (lspan+rspan == 0)
    if state is the root then 0 else −∞
  score = −∞
  % could have a left sibling
  for sibsize in [0,lspan-1]
    for (x→y state) in grammar
      cost = insideSX(y,sibsize)+
             outsideSX(x,lspan-sibsize,rspan)+
             log P(x→y state)
      score = max(score,cost)
  % could have a right sibling
  for sibsize in [0,rspan-1]
    for (x→state y) in grammar
      cost = insideSX(y,sibsize)+
             outsideSX(x,lspan,rspan-sibsize)+
             log P(x→state y)
      score = max(score,cost)
  return score;

insideSX(state, span)
  if (span == 0)
    if state is a terminal then 0 else −∞
  score = −∞
  % choose a split point
  for split in [1,span-1]
    for (state→x y) in grammar
      cost = insideSX(x,split)+
             insideSX(y,span-split)+
             log P(state→x y)
      score = max(score,cost)
  return score;
```

Figure 10: Pseudocode for the SX estimate in the case where the grammar is in CNF. Other estimates and more general grammars are similar.

tables populate.

With the optimal forward estimate TRUE, the actual distance to the closest goal, we would never expand edges other than those in best parses, but computing TRUE is as hard as parsing the sentence in the first place. On the other hand, no precomputation is needed for NULL. In between is a trade off of space/time requirements for precomputation and the online savings during the parsing of new sentences. Figure 8 shows the average savings versus the precomputation time.[10] Where on this curve one chooses to be depends on many factors; 9 hours may be too much to spend computing B, but an hour for SXMLR gives nearly the same performance, and the one minute required for SX is comparable to the I/O time to read the Penn treebank in our system.

The grammar projection estimate F had to be recomputed for each sentence parsed, but took less than 1% of the total parse time. Although this method alone was less effective than SX (only 58.3% edge savings), it was extremely effective in combination with the context summary methods. In practice, the combination of F and SX is easy to implement, fast to initialize, and very effective:

[10] All times are for a Java implementation running on a 2GB 700MHz Intel machine.

one cuts out 95% of the work in parsing at the cost of one minute of precomputation and 5 Mb of storage for outside estimates for our grammar.

4 Extension to Other Models

While the A* estimates given here can be used to accelerate PCFG parsing, most high-performance parsing has utilized models over lexicalized trees. These A* methods can be adapted to the lexicalized case. In Klein and Manning (2003), we apply a pair of grammar projection estimates to a lexicalized parsing model of a certain factored form. In that model, the score of a lexicalized tree is the product of the scores of two projections of that tree, one onto unlexicalized phrase structure, and one onto phrasal-category-free word-to-word dependency structure. Since this model has a projection-based form, grammar projection methods are easy to apply and especially effective, giving over three orders of magnitude in edge savings. The total cost per sentence includes the time required for two exhaustive PCFG parses, after which the A* search takes only seconds, even for very long sentences. Even when a lexicalized model is not in this factored form, it still admits factored grammar projection bounds; we are currently investigating this case.

5 Conclusions

An A* parser is simpler to build than a best-first parser, does less work per edge, and provides both an optimality guarantee and a worst-case cubic time bound. We have described two general ways of constructing admissible A* estimates for PCFG parsing and given several specific estimates. Using these estimates, our parser is capable of finding the Viterbi parse of an average-length Penn treebank sentence in a few seconds, processing less than 3% of the edges which would be constructed by an exhaustive parser.

Acknowledgements.

We would like to Joshua Goodman and Dan Melamed for advice and discussion about this work. This paper is based on work supported by the National Science Foundation (NSF) under Grant No. IIS-0085896, by the Advanced Research and Development Activity (ARDA)'s Advanced Question Answering for Intelligence (AQUAINT) Program, by an NSF Graduate Fellowship to the first author, and by an IBM Faculty Partnership Award to the second author.

References

Sharon A. Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24:275–298.

Eugene Charniak, Sharon Goldwater, and Mark Johnson. 1998. Edge-based best-first chart parsing. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pages 127–133. Morgan Kaufmann.

Mahesh V. Chitrao and Ralph Grishman. 1990. Statistical parsing of messages. In *Proceedings of the DARPA Speech and Natural Language Workshop, Hidden Valley, PA*, pages 263–266. Morgan Kaufmann.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Anna Corazza, Renato De Mori, Roberto Gretter, and Giorgio Satta. 1994. Optimal probabilistic evaluation functions for search controlled by stochastic context-free grammars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(10):1018–1027.

Vaibhava Goel and William J. Byrne. 1999. Task dependent loss functions in speech recognition: A-star search over recognition lattices. In *Eurospeech-99*, pages 1243–1246.

Joshua Goodman. 1997. Global thresholding and multiple-pass parsing. In *EMNLP 2*, pages 11–25.

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

Dan Klein and Christopher D. Manning. 2001a. Parsing and hypergraphs. In *Proceedings of the 7th International Workshop on Parsing Technologies (IWPT-2001)*.

Dan Klein and Christopher D. Manning. 2001b. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn treebank. In *ACL 39/EACL 10*, pages 330–337.

Dan Klein and Christopher D. Manning. 2002. A* parsing: Fast exact Viterbi parse selection. Technical Report dbpubs/2002-16, Stanford University, Stanford, CA.

Dan Klein and Christopher D. Manning. 2003. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press.

René Leermakers. 1992. A recursive ascent Earley parser. *Information Processing Letters*, 41:87–91.

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

Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27:249–276.

Stuart J. Russell and Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ.