A* Parsing: Fast Exact Viterbi Parse Selection

Dan Klein and **Christopher D. Manning**
Computer Science Department
Stanford University
Stanford, CA 94305-9040
{klein, manning}@cs.stanford.edu

Abstract

A* PCFG parsing can dramatically reduce the time required to find the exact Viterbi parse by conservatively estimating outside Viterbi probabilities. We discuss various estimates and give efficient algorithms for computing them. On 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 even a simpler estimate which can be pre-computed in under a minute still reduces the work by a factor of 5. The algorithm extends the classic A* graph search procedure to a certain hypergraph associated with parsing. Unlike best-first and finite-beam methods for achieving this kind of speed-up, the A* parser is guaranteed to return the most likely parse, not just an approximation. The algorithm is also correct for a wide range of parser control strategies and maintains a worst-case cubic time bound.

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 methods of 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 amounts to a greedy strategy, and, like any greedy strategy, the actual Viterbi parse can be pruned from the beam because, though globally optimal, it is not locally optimal at every parse stage. Chitrao and Grishman (1990), Caraballo and Charniak (1998), and Charniak et al. (1998) 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. We discuss best-first parsing further in section 3.1. Note that it does not guarantee that the first parse returned is the actual Viterbi parse (nor does it preserve the worst-case cubic time bound).

Both of these speed-up techniques are based on greedy models of parser actions. The beam search makes greedy selections at each parse step, and a best-first FOM greedily compares parse items. However, if we view parsing as best-path finding in a certain hypergraph, then the natural way to avoid processing bad edges is to consider both the distance from the start point (the sentence) to an edge (the Viterbi inside score of that edge) and the distance remaining to the goal (the Viterbi outside score of that edge). We propose an A* algorithm (Russell and Norvig, 1995), which relies on combining a known best inside score of an edge with a conservative estimate of the outside score.

The A* formulation provides two 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. In section 3 we describe the estimates that we use, along with algorithms to efficiently calculate them, and illustrate their effectiveness in parsing sentences from the Penn Treebank. Second, it allows us to easily prove the correctness of our algorithm, over a broad range of control strategies, rule encodings, and input lattices.

2 An A* Algorithm

The parser is an agenda-based parser, and operates on *edges*, such as NP:[0,2], which represent a category over a span. It 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

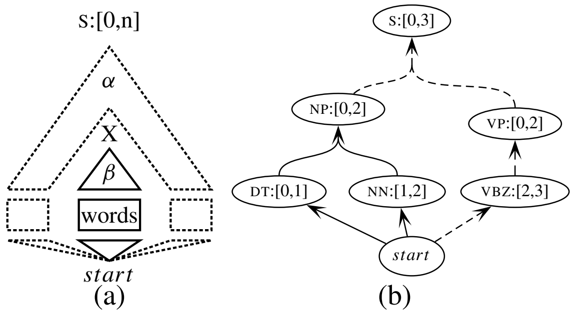


Figure 1: A* edge costs. The cost of an edge is a combination of the cost to build the edge (the Viterbi inside score) and the cost to incorporate it into a root parse (the outside Viterbi score). We will construct exact values for the inside scores and upper bounds on the outside scores.

core loop, as in any agenda-based tabular parser, involves removing an edge from the agenda and combining that edge with other edges to potentially create new edges. For example, NP:[0,2] might be removed from the agenda, and, if VP:[2,8] is already in the table, the edge S:[0,8] will be formed, and added to the agenda if it is not in the chart already. Pseudocode is shown in figure 10.

The way in which this algorithm differs from a classic chart parser is that, as in best-first parsing, agenda items are processed according to an edge priority. In best-first parsing, this priority is called a *figure-of-merit* (FOM), and is based on various approximations to the (conditional) inside probability $P(e|s)$, which is the fraction of parses of s which include e. Edges which are in few parses of s are left to wait on the agenda indefinitely. For parsing, this figure is heuristic; even if we know $P(e|s)$ exactly, we do not know whether e occurs in any *best* parse of s. Nonetheless, a good FOM leads quickly to good parses because of the strong correlation between summed inside scores and Viterbi inside scores. Best-first parsing aims to find a good parse quickly, but gives no guarantee that the first parse discovered is the Viterbi parse, nor does it allow one to verify a Viterbi parse without exhaustive parsing.

In A* parsing, we wish to construct priorities which will speed up parsing, but still guarantee that the first parse returned is indeed a best parse. With a categorical CFG 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. With PCFG parsing, there is a subtlety involved. We want to score edges with best-parse score estimates as we go. If, during parsing, we find a better way to construct some edge e that has already been entered into the chart, we have also found a better way to construct all edges which were built from e. Best-first parsers can deal with this by having an upward propagation step 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.

In order to ensure correctness, it is sufficient that all edges f which are contained in a best parse of an edge e get processed before e itself. If we have a priority over edges which ensures this property, we can avoid upward propagation and be sure that each edge leaves the agenda with its correct Viterbi score. If the grammar is in CNF, then letting shorter spans have higher priority than longer ones works; this priority essentially gives the CKY algorithm. But one needs non-trivial modifications to handle grammars with unary, n-ary, and empty productions, and cannot incorporate non-bottom-up control.

If we use the current best known Viterbi score of an edge as that edge's priority, we show in Klein and Manning (2001a), that the algorithm is correct 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). We do not present the full details here, but the correspondence should be clear enough: parser edges are nodes in the hypergraph, hyperarcs take sets of edges to their result edge, and paths map to parses. Reachability from $start$ corresponds to parseability, and shortest paths to Viterbi parses.

The graph shown is a parse of the goal S:[0,3] which includes NP:[0,2]. This path, or parse, can be split into an inside portion (solid lines) and outside portion (dashed lines), as indicated in figure 1(a). The algorithm corresponds to a generalization of uniform cost search (Russell and Norvig, 1995) with the backward cost β (the cost of the solid path) being the inside Viterbi score of an edge. Using an analogous generalization of A* search, we can also use estimates a of the forward cost α (the cost of the dashed path) to focus exploration on regions of the graph which appear to have good total cost.

A* is correct as long as a is satisfies two conditions. First, it must be *admissible*, meaning that it must not overestimate the actual cost to the goal.

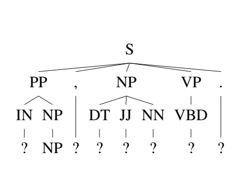
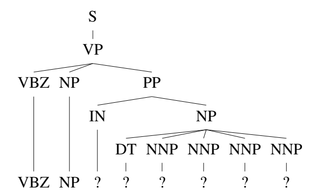
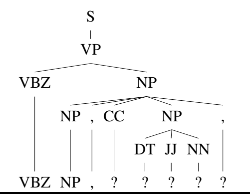
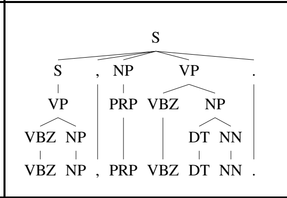
Estimate	SX	SXL	SXLR	TRUE
Summary	(1,6,NP)	(1,6,NP,VBZ)	(1,6,NP,VBZ,”,”)	(entire context)
Best Tree				
Score	-11.3	-13.9	-15.1	-18.1
	(a)	(b)	(c)	(d)

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) knowing the left and right tags, and (d – TRUE) knowing the entire outside context.

Second, it must be monotonic, meaning that as one explores a path to the goal, the combined cost $\beta + a$ does not decrease.

For parsing, this means we can add any admissible, monotonic estimate a of α to our current estimate b of β. We will still have a correct algorithm, and even rough heuristics can dramatically cut down the number of edges processed, and therefore the total work involved in parsing. We present several estimates, describe how to pre-compute them efficiently, and show the edge savings when parsing Penn treebank WSJ sentences. We also sketch proofs of correctness for A* parsing.

3 A* Estimates for Parsing

When parsing, each edge e spans some part $[i, j]$ of the sentence. Its yield is the terminals in the sentence it spans. Its *context* is the rest of the sentence. We called the spans $[0, i]$ and $[j, n]$ of the context the left and right span, respectively, or the split outside span as a pair. Their sum is the combined outside span. For concreteness, all scores will be log-probabilities, and lower cost is higher log-probability. To avoid confusion, ‘>’ or ‘better’ will mean higher probability.

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. If we give no information in the summary, the estimate will be 0. This is the trival estimate NULL, and corresponds to simply using inside scores alone as priorities. On the other hand, the ideal estimate for a is α, the true outside Viterbi score of that exact context, which we call TRUE. It would be impractical to precompute TRUE in practice, but we can still find its value for any given context by exhaustive parsing. It is worth noting 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.

We used various summaries, some illustrated in figure 2. S_1 consists of only the combined outside span, while S is the split outside span. SX also includes 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 uses the combined outside span. This was done solely to reduce memory usage; it is no quicker to calculate with combined outside span than split outside span, but more compact to store.

As the summaries become richer, the estimates become sharper. As an example, consider an NP in the context “VBZ NP , PRP VBZ DT NN .” shown in figure 2.[1] The summary SX tells us only that there is an NP with 6 words to the left and 1 to the right, and gives an estimate of -11.3. This score is backed by the concrete parse shown in figure 2(a). Now, 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 hope 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.

All of the summaries described above use somewhat local information, combined with spans. F is a less local estimate. It tracks, for each state, what

[1]Our examples, and our experiments, used delexicalized sentences from the Penn treebank.

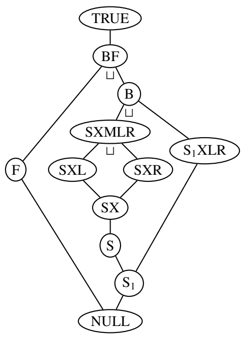


Figure 3: A* estimates form a lattice. Lines indicate subsumption, ⊔ indicates estimates which are the explicit join of lower estimates.

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

Figure 4: Two trie encodings of rules.

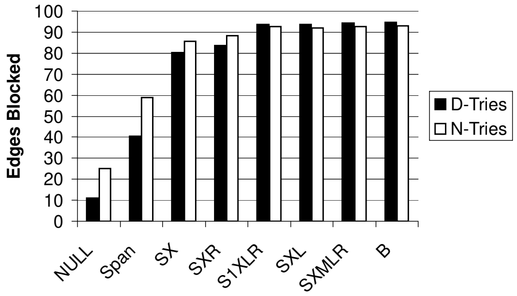


Figure 5: Fraction of edges saved by using various estimate methods, for two rule encodings. D-TRIE is a deterministic right-branching trie encoding with weights pushed left. N-TRIE is a non-deterministic left-branching trie with weights on rule entry as in (Charniak et al., 1998).

tags must occur to the left (and in what sequence) in order for that state to be reached. For example, a state labeled S → IN NP · cannot occur unless there is actually an IN somewhere to the left, and possibly the IN must be at least one terminal away (if there are no empties in the grammar). The estimate based on F returns $-\infty$ if the required tags do not exist, and 0 otherwise. Unlike the other estimates, it either rules an edge out entirely or says nothing about it. A standard chart parser with top-down control will render F redundant by never constructing an edge unless the requisite tags have already been found and added to the chart. It is only useful for these experiments because we use a true bottom-up parsing scheme to separate out the A* effects from gains gotten by selective parser control. Running the other estimates along with top-down control is quantitatively and qualitatively similar to combining them with F.

F can be computed very efficiently at parse-time for each left span and edge label, the rest are precomputed and require constant access time to retrieve. Section 3.2 discusses the precomutation time vs. benefit trade-offs involved.

These estimates form a join lattice, illustrated in figure 3; adding context information subdivides the partitions and 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 maximum of a set of admissible estimates is still an admissible estimate. We can use this to combine our basic estimates into composite estimates: SXMLR = ⊔ (SXL, SXR) will be vaild, and a better estimate than either SXL or SXR individually. Similarly, B is ⊔ (SXMLR, S_1XLR).

3.1 Parsing Performance

After (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. We tried all estimates described above.

Rules were encoded as both deterministic (D) and non-deterministic (N) tries, shown in figure 4. Such an encoding binarizes the grammar, and compacts it. N-tries are as in (Charniak et al., 1998), where NP→ DT JJ NN becomes NP → DT $X_{JJ\ NN}$ and $X_{JJ\ NN}$ → JJ NN. D-tries, as in (Klein and Manning, 2001b), turn NP→ DT JJ NN into NP → DT $X_{NP\rightarrow\ DT\ \cdot}$ and $X_{NP\rightarrow\ DT\ \cdot}$ → JJ NN. Figure 5 shows the overall savings for several estimates for each type. The N-tries were superior for the coarser estimates, while D-tries were superior for the finer estimates. In addition, D-tries have a simpler (and more effective) version of F: $X_{NP\rightarrow\ DT\ \cdot}$ clearly must have a DT somewhere to the left, while $X_{JJ\ NN}$ might be compatible with any context, depending on the rest of the grammar. Additionally, with N-tries, only the top-level rule has probability less than 1, while for D-tries, one can back-weight probability as in (Mohri, 1997), also shown in figure 4, enabling subparts of rare rules to be penalized even before they are completed. For all future results, we discuss only the D-trie numbers.

Figure 8 lists the overall savings for each estimate, with and without the non-local F. Comparing the A* estimates, we see that the NULL estimate is

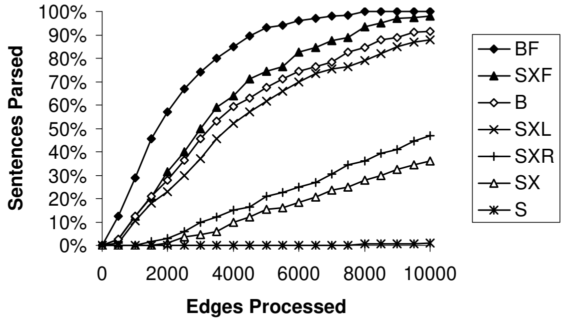


Figure 6: Number of sentences parsed as more edges are expanded. Sentences are Penn treebank sentences of length 18-26 parsed with the treebank grammar. A typical number of edges in an exhaustive parse is 150,000. Even relatively simple A* estimates allow substantial savings.

not very effective – alone it only blocks 13% of the edges. 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. Since in all cases, we stop when the combined score $\beta(e) + a(e) > \sigma$, NULL can still exploit $\beta(e)$. For even the simplest non-trivial outside estimate, S, already 40% of the edges have been blocked, 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 6 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 tuned, more complex FOM in (Charniak et al., 1998) is able to parse all of these sentences using around 2K edges, while BF requires 7K edges. In short, our estimates do not reduce the total edge count quite so far as the best FOMs can, but are in the same range. Crucially, however, our first parses are always optimal, while the FOM parses need not be and are not.[2] 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.

It is interesting that SXL is so much more effective than SXR; 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 left-filter and is partially redundant when added to SXL, but orthogonal to SXR.

3.2 Estimate Computation

The amount of work required to calculate A* 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. We do not claim that the estimates above are the best possible at the latter – one could easily imagine features of the context, such as number of verbs to the right or number of unsual tags in the context, which would partition the contexts more effectively than adjacent tags.

To calculate these A* estimates efficiently, we used a memoization approach rather than a dynamic programming approach. This resulted in code which was simpler to reason about, and, more importantly, allowed us to exploit sparseness when present. There are two kinds of sparseness. The first is request spareseness: not all context signatures will be asked about when parsing. For example with right-factored trie encodings, 76% of (state, left tag) combinations are simply impossible. The second is hierarchical sparsity. For example, 86% of the estimate values in the SXL tables were within a log-score of ± 0.1 of the values of the SX entries which subsumed them. This could have been exploited using a back-off method over the lattice, where the most specific entry which was present was used. However, the increase in space required by the hashtables negated the utility of this back-off for our particular estimate family. For space reasons, example code is only provided in figure 7 for the SX estimate, in the case where the grammar is in CNF, but the other cases are similar. 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 tables populate.

The optimal forward estimate, the actual distance to the closest goal, would mean that we never ex-

[2]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.

```
outsideSX(state, lspan, rspan)
  if (lspan+rspan == 0)
    if state is the top 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)
      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)
      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)
  return score;
```

Figure 7: Pseudocode for the SX estimate in the case where the grammar is in CNF. Other estimates and grammars are similar.

Estimate	Savings	w/ Filter	Storage	Precomp
S	40.5	77.8	5K	1 min
SX	80.3	95.3	10M	1 min
SXR	83.5	96.1	500M	30 min
S_1XLR	93.5	96.5	1G	480 min
SXL	93.8	96.9	500M	30 min
SXMLR	94.3	97.1	1G	60 min
B	94.6	97.3	2G	540 min

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

pand edges other than those in a best parse, but its computation would require parsing the rest of the sentence. On the other hand, no precomputation is needed for NULL. In between is a tradeoff 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.[3] 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 taken to simply read the Penn treebank in our system.

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

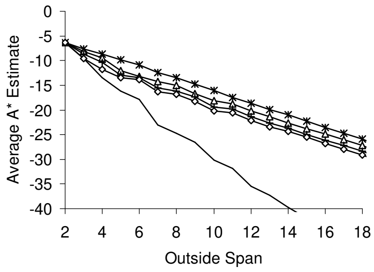
3.3 Estimate Effectiveness

A disadvantage of admissible 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 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 parse. The cost to "package up" the local restrictions creates their contant differences, and the shared ignorance about the wider context causes their same-slope linear drop-off.

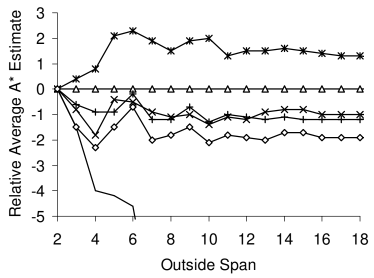
4 Correctness and Completeness

Though not in terms of A* search, we prove in Klein and Manning (2001a) the correctness of the null estimate for a range of grammars encodings, control strategies, and scan policies. All of those proofs go through with any monotonic admissible score function added to the edge priorities. For space reasons, we omit general proofs and assume bottom-up control and at-most-binary grammar rules.

First we need to know that our estimates are admissible and monotonic. Their admissibility should be clear from their construction: the max over all outside parses of all contexts compatible with a given summary includes the actual best outside parse of the actual context. For monotonicity, we need that if an edge f is contained in some best parse T_e of some edge e, then $\beta(e)+a(e) < \beta(f)+a(f)$. This is also true for all of our estimates, since



(a)



(b)

Figure 9: (a) The average estimate by outside span length for various methods. (b) The average difference from the SX method; for large outside spans, the estimates differ by relatively constant amounts.

the max over all outside parses involved in calculating $a(f)$ included all parses with the particular structure outside f in T_e.

For the following, we say an edge is *finished* when it is entered into the chart.

Claim: (Correctness) When the algorithm above terminates, every finished edge is correctly scored.

Proof: First, we need to know that the inside estimates always underestimate the true best score: $b(e) \leq \beta(e)$. Leaf edges are correctly scored at initialization. For all other edges, let traversal(e) be the traversal which either discovered e or last updated its score by relaxation. Then it is easy to see that we can take any edge at any time and use the recorded traversal values to (recursively) extract a concrete parse of e with score $b(e)$. That parse, and therefore e, can be scored no better than a best parse of e.

The rest of the proof is by contradiction. Let e be the first edge which is finished with $b(e) \neq \beta(e)$. By our first fact, it must be that $b(e) < \beta(e)$, that it is scored too low. But consider a best parse T of e. The leaves of T have all been finished, because e's discovery means that some parse T' of e has had all of its non-root nodes finished, and all parses of e share the same leaves. Therefore, there must be at least one node m in T (possibly e) which has the property that both of its children f_1 and f_2 have been finished, but m itself has not been finished. But whenever the latter of f_1 and f_2 was finished, m was discovered (if it had not been earlier) and relaxed with the traversal $m \rightarrow f_1\ f_2$. Since f_1 and f_2 were finished before e, they were correctly scored at their finishing. By first lemma, they still are. So it must be that m is correctly scored, since its score in T, a best parse, must be its true best score. If m is unfinished but correctly scored when e is removed from the agenda, it must be that $b(e) + a(e) > b(m) + a(m)$. We know that $b(m) = \beta(m)$ at that time, so $b(m) + a(m) = \beta(m) + a(m)$. By the monotonicity assumption on a, we also have $\beta(m) + a(m) > \beta(e) + a(e)$. Since $b(e) < \beta(e)$, we know $\beta(e) + a(e) > b(e) + a(e)$, but that is a contradiction.

Claim (Completeness): When the algorithm above terminates, the goal has been finished if it is parsable. Moreover, all edges whose A* score $\beta + a$ is less than the score of a best parse of the goal have been finished, and no edges whose A* scores are more than the best parse score have been finished.

Proof: If the goal edge g is parsable, then there is some concrete tree T rooted at g. All of the leaves are discovered during initialization. At termination, the agenda is empty. If g is not finished at that time, it must be that there is some lowest node m in T which is unfinished, but whose children are finished. But the second child finished would have triggered m's discovery, and m must have left the agenda for the agenda to be empty at termination, a contradiction. Moreover, edges enter the agenda only once, and there are finitely many edges, so the algorithm will terminate.

For the rest of the claim, agenda priorities at extraction are of the form $\beta(e) + a(e)$. Since a is admissable, the priority of g at extraction must have been $\beta(g)$. Therefore, it suffices to show that the priorities of the sequence of items extracted from the agenda is non-increasing. But this follows easily from the monotonicity of a and arguments similar to those above.

Claim (Time and Space): If a lookup is constant time, the algorithm can be made to run in amortized time $O(SCn^3)$ and in space $O(Sn^2)$, where S is the number of non-terminals (states) in the grammar, and C is maximum number of rules with any fixed right symbol on the RHS.

We do not prove this; it is identical to the proof in Klein and Manning (2001a). These bounds will

```
parse(sentence, goal, estimate)
  initialize(sentence, goal)
  while the agenda is non-empty
    e = nextEdge(agenda, estimate)
    finishEdge(e)

nextEdge(agenda, estimate)
  priority of e is score(e)+estimate(e)
  return the e in agenda with best priority

initialize(sentence, goal)
  create a new chart and new agenda
  for each word w:[start,end] in the sentence
    add w:[start,end] to the agenda

exploreTraversal(Traversal t)
  e = result(t)
  if notYetDiscovered(e)
    add e to the agenda
  relaxEdge(e, t)

relaxEdge(Edge e, Traversal t)
  newScore = combineScores(t)
  if (newScore is better than score(e))
    score(e) = score(t)
    bestTraversal(e) = t

finishEdge(Edge e)
  add e to the chart
  for all adjacent edges f in the chart
    for all labels x
      let t = combine(e,f)
      if t is valid
        exploreTraversal(t)
```

Figure 10: Pseudocode for the A* parser.

not be true of a best-first parser (because of the upward percolation), and, if one wants to ensure a best parse is in a beam parser's results, one must have an exponentially wide beam (though of course that would defeat the point of the beam parser).

5 Conclusions

We have demonstrated that A* estimates can be used to accelerate parsing work by orders of magnitude, without sacrificing either the guarantee that the first returned parse is the Viterbi parse or the worst-case asymptotic complexity of the parser. On average-size Penn treebank sentences, the parser is capable of finding the Viterbi parse in a very few seconds by processing less than 3% of the edges which would be constructed by an exhaustive parser. To do this, we sketched a view of parsing as hypergraph analysis which naturally leads to this A* formulation, rather than to a model which prunes at the parser-action level (such as finite-beam or best-first parsing), and proved the correctness of our algorithm.

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.

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*, pages 330–337.

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

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.