Top-Down K-Best A* Parsing

Adam Pauls and Dan Klein
Computer Science Division
University of California at Berkeley
{adpauls,klein}@cs.berkeley.edu

Chris Quirk
Microsoft Research
Redmond, WA, 98052
chrisq@microsoft.com

Abstract

We propose a top-down algorithm for extracting \( k \)-best lists from a parser. Our algorithm, TKA*, is a variant of the \( k \)-best A* (KA*) algorithm of Pauls and Klein (2009). In contrast to KA*, which performs an inside and outside pass before performing \( k \)-best extraction bottom up, TKA* performs only the inside pass before extracting \( k \)-best lists top down. TKA* maintains the same optimality and efficiency guarantees of KA*, but is simpler both in implementation and specification.

1 Introduction

Many situations call for a parser to return a \( k \)-best list of parses instead of a single best hypothesis.\(^1\) Currently, there are two efficient approaches known in the literature. The \( k \)-best algorithm of Jiménez and Marzal (2000) and Huang and Chiang (2005), referred to hereafter as LAZY, operates by first performing an exhaustive Viterbi inside pass and then lazily extracting \( k \)-best lists in top-down manner. The \( k \)-best A* algorithm of Pauls and Klein (2009), hereafter KA*, computes Viterbi inside and outside scores before extracting \( k \)-best lists bottom up.

Because these additional passes are only partial, KA* can be significantly faster than LAZY, especially when a heuristic is used (Pauls and Klein, 2009). In this paper, we propose TKA*, a top-down variant of KA* that, like LAZY, performs only an inside pass before extracting \( k \)-best lists top-down, but maintains the same optimality and efficiency guarantees as KA*. This algorithm can be seen as a generalization of the lattice \( k \)-best algorithm of Soong and Huang (1991) to parsing. Because TKA* eliminates the outside pass from KA*, TKA* is simpler both in implementation and specification.

\(^1\)See Huang and Chiang (2005) for a review.

2 Review

Because our algorithm is very similar to KA*, which is in turn an extension of the (1-best) A* parsing algorithm of Klein and Manning (2003), we first introduce notation and review those two algorithms before presenting our new algorithm.

2.1 Notation

Assume we have a PCFG\(^2\) \( \mathcal{G} \) and an input sentence \( s_0 \ldots s_{n-1} \) of length \( n \). The grammar \( \mathcal{G} \) has a set of symbols denoted by capital letters, including a distinguished goal (root) symbol \( G \). Without loss of generality, we assume Chomsky normal form: each non-terminal (root) symbol \( G \) with form \( (A, i, j) \) are trees with root non-terminal \( A \), spanning \( s_i \ldots s_{j-1} \). The weight (negative log-probability) of the best (minimum) inside derivation for an edge \( e \) is called the Viterbi inside score \( \beta(e) \), and the weight of the best derivation of \( G \rightarrow s_0 \ldots s_{n-1} \) is called the Viterbi outside score \( \alpha(e) \). The goal of a \( k \)-best parsing algorithm is to compute the \( k \) best (minimum weight) inside derivations of the edge \( (G, 0, n) \).

We formulate the algorithms in this paper in terms of prioritized weighted deduction rules (Shieber et al., 1995; Nederhof, 2003). A prioritized weighted deduction rule has the form

\[
\phi_1 : w_1, \ldots, \phi_n : w_n \xrightarrow{p(w_1, \ldots, w_n)} \phi_0 : g(w_1, \ldots, w_n)
\]

where \( \phi_1, \ldots, \phi_n \) are the antecedent items of the deduction rule and \( \phi_0 \) is the conclusion item. A deduction rule states that, given the antecedents \( \phi_1, \ldots, \phi_n \) with weights \( w_1, \ldots, w_n \), the conclusion \( \phi_0 \) can be formed with weight \( g(w_1, \ldots, w_n) \) and priority \( p(w_1, \ldots, w_n) \).

\(^2\)While we present the algorithm specialized to parsing with a PCFG, this algorithm generalizes to a wide range of
These deduction rules are “executed” within a generic agenda-driven algorithm, which constructs items in a prioritized fashion. The algorithm maintains an agenda (a priority queue of items), as well as a chart of items already processed. The fundamental operation of the algorithm is to pop the highest priority item \( \phi \) from the agenda, put it into the chart with its current weight, and apply deduction rules to form any items which can be built by combining \( \phi \) with items already in the chart. When the resulting items are either new or have a weight smaller than an item’s best score so far, they are put on the agenda with priority given by \( p(\cdot) \). Because all antecedents must be constructed before a deduction rule is executed, we sometimes refer to particular conclusion item as “waiting” on another item before it can be built.

### 2.2 A*

A*-parsing (Klein and Manning, 2003) is an algorithm for computing the 1-best parse of a sentence. A* operates on items called inside edge items \( I(A, i, j) \), which represent the many possible inside derivations of an edge \((A, i, j)\). Inside edge items are constructed according to the IN deduction rule of Table 1. This deduction rule constructs inside edge items in a bottom-up fashion, combining items representing smaller edges \( I(B, i, k) \) and \( I(C, k, j) \) with a grammar rule \( r = A \rightarrow B C \) to form a larger item \( I(A, i, j) \). The weight of a newly constructed item is given by the sum of the weights of the antecedent items and the grammar rule \( r \), and its priority is given by

\[
\text{weight of a newly constructed item} = \text{sum of the weights of the antecedent items and the grammar rule } r,
\]

and its priority is given by

\[
\text{priority of a newly constructed item} = \alpha(e) + \text{true Viterbi outside score}.
\]

Hypergraph search problems as shown in Klein and Manning (2001).
down fashion, all we would need to compute opti-
up; if we constructed partial derivations in a top-
because we construct partial derivations bottom-
tion costs. Outside costs are thus only necessary
items for a particular edge wait until the exact out-
side score of that edge has been computed. The al-
terminates when $k$ derivation items rooted at $(G, 0, n)$ have been popped from the agenda.

3 TK$A^*$

KA$^*$ efficiently explores the space of inside
derivation items because it waits for the exact
Viterbi outside cost before building each deriv-
ation item. However, these outside costs and asso-
related deduction items are only auxiliary quanti-
ties used to guide the exploration of inside deriv-
tions: they allow KA$^*$ to prioritize currently con-
structed inside derivation items (i.e., constructed
derivations of the goal) by their optimal comple-
tion costs. Outside costs are thus only necessary
case we construct partial derivations bottom-
if we constructed partial derivations in a top-
down fashion, all we would need to compute opti-
mal completion costs are Viterbi inside scores, and
we could forget the outside pass.

TK$A^*$ does exactly that. Inside edge items are
constructed in the same way as KA$^*$, but once the
inside edge item $I(G, 0, n)$ has been discovered,
TK$A^*$ begins building partial derivations from the
goal outwards. We replace the inside derivation
items of KA$^*$ with outside derivation items, which
represent trees rooted at the goal and expanding
downwards. These items bottom out in a list of
edges called the *frontier* edges. See Figure 1(d)
for a graphical representation. When a frontier
edge represents a single word in the input, i.e. is
of the form $(s_i, i, i + 1)$, we say that edge is *com-
plete*. An outside derivation can be expanded by
applying a rule to one of its incomplete frontier
dges; see Figure 2. In the same way that inside
derivation items wait on exact outside scores be-
fore being built, outside derivation items wait on
the inside edge items of all frontier edges before
they can be constructed.

Although building derivations top-down obvi-
nates the need for a 1-best outside pass, it raises a
new issue. When building derivations bottom-up,
the only way to expand a particular partial inside
derivation is to combine it with another partial in-
side derivation to build a bigger tree. In contrast,
an outside derivation item can be expanded any-
where along its frontier. Naïvely building deriv-
tions top-down would lead to a prohibitively large
number of expansion choices.

We solve this issue by always expanding the
left-most incomplete frontier edge of an outside
derivation item. We show the deduction rule
OUT-D which performs this deduction in Fig-
ure 1(d). We denote an outside derivation item as
$Q(T_A i, j, F)$, where $T_A$ is a tree rooted at the
goal with left-most incomplete edge $(A, i, j)$, and
$F$ is the list of incomplete frontier edges exclud-
ing $(A, i, j)$, ordered from left to right. Whenever
the application of this rule “completes” the left-
most edge, the next edge is removed from \( \mathcal{F} \) and is used as the new point of expansion. Once all frontier edges are complete, the item represents a correctly scored derivation of the goal, explored in a pre-order traversal.

### 3.1 Correctness

It should be clear that expanding the left-most incomplete frontier edge first eventually explores the same set of derivations as expanding all frontier edges simultaneously. The only worry in fixing this canonical order is that we will somehow explore the \( Q \) items in an incorrect order, possibly building some complete derivation \( Q'_{C} \) before a more optimal complete derivation \( Q_{C} \). However, note that all items \( Q \) along the left-most construction of \( Q_C \) have priority equal to or better than any less optimal complete derivation \( Q'_{C} \). Therefore, when \( Q'_{C} \) is enqueued, it will have lower priority than all \( Q \); \( Q'_{C} \) will therefore not be dequeued until all \( Q \) – and hence \( Q_{C} \) – have been built.

Furthermore, it can be shown that the top-down expansion strategy maintains the same efficiency and optimality guarantees as KA\(^{\ast}\) for all item types: for consistent heuristics \( h \), the first \( k \) entirely complete outside derivation items are the true \( k \)-best derivations (modulo ties), and that only derivation items which participate in those \( k \)-best derivations will be removed from the queue (up to ties).

### 3.2 Implementation Details

Building derivations bottom-up is convenient from an indexing point of view: since larger derivations are built from smaller ones, it is not necessary to construct the larger derivation from scratch. Instead, one can simply construct a new tree whose children point to the old trees, saving both memory and CPU time.

In order keep the same efficiency when building trees top-down, a slightly different data structure is necessary. We represent top-down derivations as a lazy list of expansions. The top node \( T_{G}^{A} \) is an empty list, and whenever we expand an outside derivation item \( Q(T_{A}^{G}, i, j, \mathcal{F}) \) with a rule \( r = A \rightarrow B \ C \) and split point \( l \), the resulting derivation \( T_{B}^{G} \) is a new list item with \((r, l)\) as the head data, and \( T_{B}^{G} \) as its tail. The tree can be reconstructed later by recursively reconstructing the parent, and adding the edges \((B, i, l)\) and \((C, l, j)\) as children of \((A, i, j)\).

### 3.3 Advantages

Although our algorithm eliminates the 1-best outside pass of KA\(^{\ast}\), in practice, even for \( k = 10^{4} \), the 1-best inside pass remains the overwhelming bottleneck (Pauls and Klein, 2009), and our modifications leave that pass unchanged.

However, we argue that our implementation is simpler to specify and implement. In terms of deduction rules, our algorithm eliminates the 2 outside deduction rules and replaces the IN-D rule with the OUT-D rule, bringing the total number of rules from four to two.

The ease of specification translates directly into ease of implementation. In particular, if high-quality heuristics are not available, it is often more efficient to implement the 1-best inside pass as an exhaustive dynamic program, as in Huang and Chiang (2005). In this case, one would only need to implement a single, agenda-based \( k \)-best extraction phase, instead of the 2 needed for KA\(^{\ast}\).

### 3.4 Performance

The contribution of this paper is theoretical, not empirical. We have argued that TKA\(^{\ast}\) is simpler than KA\(^{\ast}\), but we do not expect it to do any more or less work than KA\(^{\ast}\), modulo grammar specific optimizations. Therefore, we simply verify, like KA\(^{\ast}\), that the additional work of extracting \( k \)-best lists with TKA\(^{\ast}\) is negligible compared to the time spent building 1-best inside edges.

We examined the time spent building 100-best lists for the same experimental setup as Pauls and Klein (2009).\(^{4}\) On 100 sentences, our implementation of TKA\(^{\ast}\) constructed 3.46 billion items, of which about 2% were outside derivation items. Our implementation of KA\(^{\ast}\) constructed 3.41 billion edges, of which about 0.1% were outside edge items or inside derivation items. In other words, the cost of \( k \)-best extraction is dwarfed by the the 1-best inside edge computation in both cases. The reason for the slight performance advantage of KA\(^{\ast}\) is that our implementation of KA\(^{\ast}\) uses lazy optimizations discussed in Pauls and Klein (2009), and while such optimizations could easily be incorporated in TKA\(^{\ast}\), we have not yet done so in our implementation.

---

\(^{4}\)This setup used 3- and 6-round state-split grammars from Petrov et al. (2006), the former used to compute a heuristic for the latter, tested on sentences of length up to 25.
4 Conclusion

We have presented TKA*, a simplification to the KA* algorithm. Our algorithm collapses the 1-best outside and bottom-up derivation passes of KA* into a single, top-down pass without sacrificing efficiency or optimality. This reduces the number of non base-case deduction rules, making TKA* easier both to specify and implement.

Acknowledgements

This project is funded in part by the NSF under grant 0643742 and an NSERC Postgraduate Fellowship.

References

Liang Huang and David Chiang. 2005. Better k-best parsing. In Proceedings of the International Workshop on Parsing Technologies (IWPT), pages 53–64.

Víctor M. Jiménez and Andrés Marzal. 2000. Computation of the n best parse trees for weighted and stochastic context-free grammars. In Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition, pages 183–192, London, UK. Springer-Verlag.

Dan Klein and Christopher D. Manning. 2001. Parsing and hypergraphs. In Proceedings of the International Workshop on Parsing Technologies (IWPT), pages 123–134.

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact Viterbi parse selection. In Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL), pages 119–126.

Mark-Jan Nederhof. 2003. Weighted deductive parsing and Knuth’s algorithm. Computational Linguistics, 29(1):135–143.

Adam Pauls and Dan Klein. 2009. K-best A* parsing. In Proceedings of the Association for Computational Linguistics (ACL).

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In Proceedings of the Association for Computational Linguistics (ACL).

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. Journal of Logic Programming, 24:3–36.

Frank K. Soong and Eng-Fong Huang. 1991. A trellis based fast search for finding the n best sentence hypotheses in continuous speech recognition. In Proceedings of the Workshop on Speech and Natural Language.