K-Best A* Parsing

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

Abstract

A* parsing makes 1-best search efficient by suppressing unlikely 1-best items. Existing k-best extraction methods can efficiently search for top derivations, but only after an exhaustive 1-best pass. We present a unified algorithm for k-best A* parsing which preserves the efficiency of k-best extraction while giving the speed-ups of A* methods. Our algorithm produces optimal k-best parses under the same conditions required for optimality in a 1-best A* parser. Empirically, optimal k-best lists can be extracted significantly faster than with other approaches, over a range of grammar types.

1 Introduction

Many situations call for a parser to return the k-best parses rather than only the 1-best. Uses for k-best lists include minimum Bayes risk decoding (Goodman, 1998; Kumar and Byrne, 2004), discriminative reranking (Collins, 2000; Charniak and Johnson, 2005), and discriminative training (Och, 2003; McClosky et al., 2006). The most efficient known algorithm for k-best parsing (Jiménez and Marzal, 2000; Huang and Chiang, 2005) performs an initial bottom-up dynamic programming pass before extracting the k-best parses. In that algorithm, the initial pass is, by far, the bottleneck (Huang and Chiang, 2005).

In this paper, we propose an extension of A* parsing which integrates k-best search with an A*-based exploration of the 1-best chart. A* parsing can avoid significant amounts of computation by guiding 1-best search with heuristic estimates of parse completion costs, and has been applied successfully in several domains (Klein and Manning, 2002; Klein and Manning, 2003c; Haghighi et al., 2007). Our algorithm extends the speed-ups achieved in the 1-best case to the k-best case and is optimal under the same conditions as a standard A* algorithm. The amount of work done in the k-best phase is no more than the amount of work done by the algorithm of Huang and Chiang (2005). Our algorithm is also equivalent to standard A* parsing (up to ties) if it is terminated after the 1-best derivation is found. Finally, our algorithm can be written down in terms of deduction rules, and thus falls into the well-understood view of parsing as weighted deduction (Shieber et al., 1995; Goodman, 1998; Nederhof, 2003).

In addition to presenting the algorithm, we show experiments in which we extract k-best lists for three different kinds of grammars: the lexicalized grammars of Klein and Manning (2003b), the state-split grammars of Petrov et al. (2006), and the tree transducer grammars of Galley et al. (2006). We demonstrate that optimal k-best lists can be extracted significantly faster using our algorithm than with previous methods.

2 A k-Best A* Parsing Algorithm

We build up to our full algorithm in several stages, beginning with standard 1-best A* parsing and making incremental modifications.

2.1 Parsing as Weighted Deduction

Our algorithm can be formulated in terms of prioritized weighted deduction rules (Shieber et al., 1995; Nederhof, 2003; Felzenszwalb and McAllester, 2007). 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 ϕ_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 ϕ_0 can be formed with weight $g(w_1, \ldots, w_n)$ and priority $p(w_1, \ldots, w_n)$.

Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP, pages 958–966,
Suntec, Singapore, 2-7 August 2009. ©2009 ACL and AFNLP

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 unprocessed items), as well as a *chart* of items already processed. The fundamental operation of the algorithm is to pop the highest priority item ϕ from the agenda, put it into the chart with its current weight, and form using deduction rules any items which can be built by combining ϕ with items already in the chart. If new or improved, resulting items are put on the agenda with priority given by $p(\cdot)$.

2.2 A* Parsing

The A* parsing algorithm of Klein and Manning (2003c) can be formulated in terms of weighted deduction rules (Felzenszwalb and McAllester, 2007). We do so here both to introduce notation and to build to our final algorithm.

First, we must formalize some notation. Assume we have a PCFG[1] $\mathcal{G}$ and an input sentence $s_1 \dots s_n$ of length n. The grammar $\mathcal{G}$ has a set of symbols Σ, including a distinguished goal (root) symbol G. Without loss of generality, we assume Chomsky normal form, so each non-terminal rule r in $\mathcal{G}$ has the form $r = A \to B\ C$ with weight w_r (the negative log-probability of the rule). *Edges* are labeled spans $e = (A, i, j)$. *Inside derivations* of an edge (A, i, j) are trees rooted at A and spanning $s_{i+1} \dots s_j$. The total weight of the best (minimum) inside derivation for an edge e is called the Viterbi inside score $\beta(e)$. The goal of the 1-best A* parsing algorithm is to compute the Viterbi inside score of the edge $(G, 0, n)$; backpointers allow the reconstruction of a Viterbi parse in the standard way.

The basic A* algorithm operates on deduction items $I(A, i, j)$ which represent in a collapsed way the possible inside derivations of edges (A, i, j). We call these items *inside edge items* or simply *inside items* where clear; a graphical representation of an inside item can be seen in Figure 1(a). The space whose items are inside edges is called the *edge space*.

These inside items are combined using the single IN deduction schema shown in Table 1. This schema is instantiated for every grammar rule r in $\mathcal{G}$. For IN, the function $g(\cdot)$ simply sums the weights of the antecedent items and the grammar rule r, while the priority function $p(\cdot)$ adds a *heuristic* to this sum. The heuristic is a bound on the Viterbi *outside score* $\alpha(e)$ of an edge e; see Klein and Manning (2003c) for details. A good heuristic allows A* to reach the goal item $I(G, 0, n)$ while constructing few inside items.

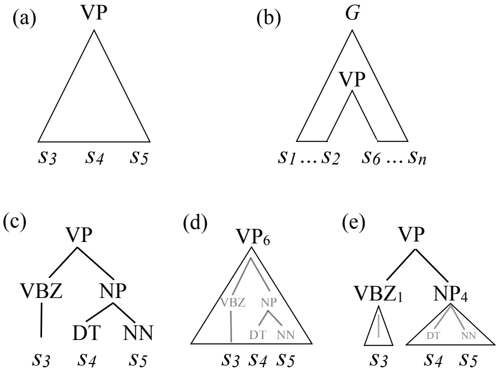

Figure 1: Representations of the different types of items used in parsing. (a) An inside edge item: $I(\text{VP}, 2, 5)$. (b) An outside edge item: $O(\text{VP}, 2, 5)$. (c) An inside derivation item: $D(T_{\text{VP}}, 2, 5)$ for a tree T_{VP}. (d) A ranked derivation item: $K(\text{VP}, 2, 5, 6)$. (e) A modified inside derivation item (with backpointers to ranked items): $D(\text{VP}, 2, 5, 3, \text{VP} \to \text{VBZ NP}, 1, 4)$.

If the heuristic is *consistent*, then A* guarantees that whenever an inside item comes off the agenda, its weight is its true Viterbi inside score (Klein and Manning, 2003c). In particular, this guarantee implies that the goal item $I(G, 0, n)$ will be popped with the score of the 1-best parse of the sentence. Consistency also implies that items are popped off the agenda in increasing order of bounded Viterbi scores:

$$\beta(e) + h(e)$$

We will refer to this monotonicity as the *ordering* property of A* (Felzenszwalb and McAllester, 2007). One final property implied by consistency is *admissibility*, which states that the heuristic never overestimates the true Viterbi outside score for an edge, i.e. $h(e) \leq \alpha(e)$. For the remainder of this paper, we will assume our heuristics are consistent.

2.3 A Naive k-Best A* Algorithm

Due to the optimal substructure of 1-best PCFG derivations, a 1-best parser searches over the space of edges; this is the essence of 1-best dynamic programming. Although most edges can be built

[1] While we present the algorithm specialized to parsing with a PCFG, it generalizes to a wide range of hypergraph search problems as shown in Klein and Manning (2001).

Inside Edge Deductions (Used in A* and KA*)

$$\textbf{IN:} \quad I(B,i,l) : w_1 \qquad I(C,l,j) : w_2 \quad \xrightarrow{w_1+w_2+w_r+h(A,i,j)} \quad I(A,i,j) \quad : \quad w_1 + w_2 + w_r$$

Table 1: The deduction schema (IN) for building inside edge items, using a supplied heuristic. This schema is sufficient on its own for 1-best A*, and it is used in KA*. Here, r is the rule $A \rightarrow B\ C$.

Inside Derivation Deductions (Used in NAIVE)

$$\textbf{DERIV:} \quad D(T_B,i,l) : w_1 \quad D(T_C,l,j) : w_2 \quad \xrightarrow{w_1+w_2+w_r+h(A,i,j)} \quad D\left(\begin{array}{c} \mathrm{A} \\ \diagup\diagdown \\ \mathrm{T}_B \quad \mathrm{T}_C \end{array}, i, j\right) \quad : \quad w_1 + w_2 + w_r$$

Table 2: The deduction schema for building derivations, using a supplied heuristic. T_B and T_C denote full tree structures rooted at symbols B and C. This schema is the same as the IN deduction schema, but operates on the space of fully specified inside derivations rather than dynamic programming edges. This schema forms the NAIVE k-best algorithm.

Outside Edge Deductions (Used in KA*)

$$\begin{array}{lllllll}
\textbf{OUT-B:} & I(G,0,n) : w_1 & & & \xrightarrow{w_1} & O(G,0,n) & : \ 0 \\
\textbf{OUT-L:} & O(A,i,j) : w_1 & I(B,i,l) : w_2 & I(C,l,j) : w_3 & \xrightarrow{w_1+w_3+w_r+w_2} & O(B,i,l) & : \ w_1 + w_3 + w_r \\
\textbf{OUT-R:} & O(A,i,j) : w_1 & I(B,i,l) : w_2 & I(C,l,j) : w_3 & \xrightarrow{w_1+w_2+w_r+w_3} & O(C,l,j) & : \ w_1 + w_2 + w_r
\end{array}$$

Table 3: The deduction schemata for building ouside edge items. The first schema is a base case that constructs an outside item for the goal $(G, 0, n)$ from the inside item $I(G, 0, n)$. The second two schemata build outside items in a top-down fashion. Note that for outside items, the completion cost is the weight of an inside item rather than a value computed by a heuristic.

Delayed Inside Derivation Deductions (Used in KA*)

$$\textbf{DERIV:} \quad D(T_B,i,l) : w_1 \quad D(T_C,l,j) : w_2 \quad O(A,i,j) : w_3 \quad \xrightarrow{w_1+w_2+w_r+w_3} \quad D\left(\begin{array}{c} \mathrm{A} \\ \diagup\diagdown \\ \mathrm{T}_B \ \mathrm{T}_C \end{array}, i, j\right) \quad : \quad w_1 + w_2 + w_r$$

Table 4: The deduction schema for building derivations, using exact outside scores computed using OUT deductions. The dependency on the outside item $O(A, i, j)$ delays building derivation items until exact Viterbi outside scores have been computed. This is the final search space for the KA* algorithm.

Ranked Inside Derivation Deductions (Lazy Version of NAIVE)

$$\begin{array}{llllll}
\textbf{BUILD:} & K(B,i,l,u) : w_1 & K(C,l,j,v) : w_2 & \xrightarrow{w_1+w_2+w_r+h(A,i,j)} & D(A,i,j,l,r,u,v) & : \ w_1 + w_2 + w_r \\
\textbf{RANK:} & D_1(A,i,j,\cdot) : w_1 \ldots & D_k(A,i,j,\cdot) : w_k & \xrightarrow{\max_m w_m + h(A,i,j)} & K(A,i,j,k) & : \ \max_m w_m
\end{array}$$

Table 5: The schemata for simultaneously building and ranking derivations, using a supplied heuristic, for the lazier form of the NAIVE algorithm. BUILD builds larger derivations from smaller ones. RANK numbers derivations for each edge. Note that RANK requires distinct D_i, so a rank k RANK rule will first apply (optimally) as soon as the kth-best inside derivation item for a given edge is removed from the queue. However, it will also still formally apply (suboptimally) for all derivation items dequeued after the kth. In practice, the RANK schema need not be implemented explicitly – one can simply assign a rank to each inside derivation item when it is removed from the agenda, and directly add the appropriate ranked inside item to the chart.

Delayed Ranked Inside Derivation Deductions (Lazy Version of KA*)

$$\begin{array}{lllllll}
\textbf{BUILD:} & K(B,i,l,u) : w_1 & K(C,l,j,v) : w_2 & O(A,i,j) : w_3 & \xrightarrow{w_1+w_2+w_r+w_3} & D(A,i,j,l,r,u,v) & : \ w_1 + w_2 + w_r \\
\textbf{RANK:} & D_1(A,i,j,\cdot) : w_1 \ldots & D_k(A,i,j,\cdot) : w_k & O(A,i,j) : w_{k+1} & \xrightarrow{\max_m w_m + w_{k+1}} & K(A,i,j,k) & : \ \max_m w_m
\end{array}$$

Table 6: The deduction schemata for building and ranking derivations, using exact outside scores computed from OUT deductions, used for the lazier form of the KA* algorithm.

using many derivations, each inside edge item will be popped exactly once during parsing, with a score and backpointers representing its 1-best derivation.

However, k-best lists involve suboptimal derivations. One way to compute k-best derivations is therefore to abandon optimal substructure and dynamic programming entirely, and to search over the *derivation space*, the much larger space of fully specified trees. The items in this space are called *inside derivation items*, or *derivation items* where clear, and are of the form $D(T_A, i, j)$, specifying an entire tree T_A rooted at symbol A and spanning $s_{i+1} \dots s_j$ (see Figure 1(c)). Derivation items are combined using the DERIV schema of Table 2. The goals in this space, representing root parses, are any derivation items rooted at symbol G that span the entire input.

In this expanded search space, each distinct parse has its own derivation item, derivable only in one way. If we continue to search long enough, we will pop multiple goal items. The first k which come off the agenda will be the k-best derivations. We refer to this approach as NAIVE. It is very inefficient on its own, but it leads to the full algorithm.

The correctness of this k-best algorithm follows from the correctness of A* parsing. The derivation space of full trees is simply the edge space of a much larger grammar (see Section 2.5).

Note that the DERIV schema's priority includes a heuristic just like 1-best A*. Because of the context freedom of the grammar, any consistent heuristic for inside edge items usable in 1-best A* is also consistent for inside derivation items (and vice versa). In particular, the 1-best Viterbi outside score for an edge is a "perfect" heuristic for any derivation of that edge.

While correct, NAIVE is massively inefficient. In comparison with A* parsing over $\mathcal{G}$, where there are $O(n^2)$ inside items, the size of the derivation space is exponential in the sentence length. By the ordering property, we know that NAIVE will process all derivation items d with

$$\delta(d) + h(d) \leq \delta(g_k)$$

where g_k is the kth-best root parse and $\delta(\cdot)$ is the inside score of a derivation item (analogous to β for edges).[2] Even for reasonable heuristics, this number can be very large; see Section 3 for empirical results.

This naive algorithm is, of course, not novel, either in general approach or specific computation. Early k-best parsers functioned by abandoning dynamic programming and performing beam search on derivations (Ratnaparkhi, 1999; Collins, 2000). Huang (2005) proposes an extension of Knuth's algorithm (Knuth, 1977) to produce k-best lists by searching in the space of derivations, which is essentially this algorithm. While Huang (2005) makes no explicit mention of a heuristic, it would be easy to incorporate one into their formulation.

2.4 A New k-Best A* Parser

While NAIVE suffers severe performance degradation for loose heuristics, it is in fact very efficient if $h(\cdot)$ is "perfect," i.e. $h(e) = \alpha(e)\ \forall e$. In this case, the ordering property of A* guarantees that only inside derivation items d satisfying

$$\delta(d) + \alpha(d) \leq \delta(g_k)$$

will be placed in the chart. The set of derivation items d satisfying this inequality is exactly the set which appear in the k-best derivations of $(G, 0, n)$ (as always, modulo ties). We could therefore use NAIVE quite efficiently if we could obtain exact Viterbi outside scores.

One option is to compute outside scores with exhaustive dynamic programming over the original grammar. In a certain sense, described in greater detail below, this precomputation of exact heuristics is equivalent to the k-best extraction algorithm of Huang and Chiang (2005). However, this exhaustive 1-best work is precisely what we want to use A* to avoid.

Our algorithm solves this problem by integrating three searches into a single agenda-driven process. First, an A* search in the space of inside edge items with an (imperfect) external heuristic $h(\cdot)$ finds exact inside scores. Second, exact outside scores are computed from inside and outside items. Finally, these exact outside scores guide the search over derivations. It can be useful to imagine these three operations as operating in phases, but they are all interleaved, progressing in order of their various priorities.

In order to calculate outside scores, we introduce *outside items* $O(A, i, j)$, which represent best derivations of $G \rightarrow s_1 \dots s_i\ A\ s_{j+1} \dots s_n$; see Figure 1(b). Where the weights of inside items

[2] The new symbol emphasizes that δ scores a specific derivation rather than a minimum over a set of derivations.

compute Viterbi inside scores, the weights of outside items compute Viterbi outside scores.

Table 3 shows deduction schemata for building outside items. These schemata are adapted from the schemata used in the general hierarchical A* algorithm of Felzenszwalb and McAllester (2007). In that work, it is shown that such schemata maintain the property that the weight of an outside item is the true Viterbi outside score when it is removed from the agenda. They also show that outside items o follow an ordering property, namely that they are processed in increasing order of

$$\beta(o) + \alpha(o)$$

This quantity is the score of the best root derivation which includes the edge corresponding to o. Felzenszwalb and McAllester (2007) also show that both inside and outside items can be processed on the same queue and the ordering property holds jointly for both types of items.

If we delay the construction of a derivation item until its corresponding outside item has been popped, then we can gain the benefits of using an exact heuristic $h(\cdot)$ in the naive algorithm. We realize this delay by modifying the DERIV deduction schema as shown in Table 4 to trigger on and prioritize with the appropriate outside scores.

We now have our final algorithm, which we call KA*. It is the union of the IN, OUT, and new "delayed" DERIV deduction schemata. In words, our algorithm functions as follows: we initialize the agenda with $I(s_i, i-1, i)$ and $D(s_i, i-1, i)$ for $i = 1 \ldots n$. We compute inside scores in standard A* fashion using the IN deduction rule, using any heuristic we might provide to 1-best A*. Once the inside item $I(G, 0, n)$ is found, we automatically begin to compute outside scores via the OUT deduction rules. Once $O(s_i, i-1, i)$ is found, we can begin to also search in the space of derivation items, using the perfect heuristics given by the just-computed outside scores. Note, however, that all computation is done with a single agenda, so the processing of all three types of items is interleaved, with the k-best search possibly terminating without a full inside computation. As with NAIVE, the algorithm terminates when a k-th goal derivation is dequeued.

2.5 Correctness

We prove the correctness of this algorithm by a reduction to the hierarchical A* (HA*) algorithm of Felzenszwalb and McAllester (2007). The input to HA* is a target grammar $\mathcal{G}_m$ and a list of grammars $\mathcal{G}_0 \ldots \mathcal{G}_{m-1}$ in which $\mathcal{G}_{t-1}$ is a relaxed projection of $\mathcal{G}_t$ for all $t = 1 \ldots m$. A grammar $\mathcal{G}_{t-1}$ is a *projection* of $\mathcal{G}_t$ if there exists some onto function $\pi_t : \Sigma_t \mapsto \Sigma_{t-1}$ defined for all symbols in $\mathcal{G}_t$. We use A_{t-1} to represent $\pi_t(A_t)$. A projection is *relaxed* if, for every rule $r = A_t \rightarrow B_t C_t$ with weight w_r there is a rule $r' = A_{t-1} \rightarrow B_{t-1} C_{t-1}$ in $\mathcal{G}_{t-1}$ with weight $w_{r'} \leq w_r$.

We assume that our external heuristic function $h(\cdot)$ is constructed by parsing our input sentence with a relaxed projection of our target grammar. This assumption, though often true anyway, is to allow proof by reduction to Felzenszwalb and McAllester (2007).[3]

We construct an instance of HA* as follows: Let $\mathcal{G}_0$ be the relaxed projection which computes the heuristic. Let $\mathcal{G}_1$ be the input grammar $\mathcal{G}$, and let $\mathcal{G}_2$, the target grammar of our HA* instance, be the grammar of derivations in $\mathcal{G}$ formed by expanding each symbol A in $\mathcal{G}$ to all possible inside derivations T_A rooted at A. The rules in $\mathcal{G}_2$ have the form $T_A \rightarrow T_B\ T_C$ with weight given by the weight of the rule $A \rightarrow B\ C$. By construction, $\mathcal{G}_1$ is a relaxed projection of $\mathcal{G}_2$; by assumption $\mathcal{G}_0$ is a relaxed projection of $\mathcal{G}_1$. The deduction rules that describe KA* build the same items as HA* with same weights and priorities, and so the guarantees from HA* carry over to KA*.

We can characterize the amount of work done using the ordering property. Let g_k be the kth-best derivation item for the goal edge g. Our algorithm processes all derivation items d, outside items o, and inside items i satisfying

$$\begin{aligned} \delta(d) + \alpha(d) &\leq \delta(g_k) \\ \beta(o) + \alpha(o) &\leq \delta(g_k) \\ \beta(i) + h(i) &\leq \delta(g_k) \end{aligned}$$

We have already argued that the set of derivation items satisfying the first inequality is the set of subtrees that appear in the optimal k-best parses, modulo ties. Similarly, it can be shown that the second inequality is satisfied only for edges that appear in the optimal k-best parses. The last inequality characterizes the amount of work done in the bottom-up pass. We compare this to 1-best A*, which pops all inside items i satisfying

$$\beta(i) + h(i) \leq \beta(g) = \delta(g_1)$$

[3] KA* is correct for any consistent heuristic but a non-reductive proof is not possible in the present space.

Thus, the "extra" inside items popped in the bottom-up pass during k-best parsing as compared to 1-best parsing are those items i satisfying

$$\delta(g_1) \leq \beta(i) + h(i) \leq \delta(g_k)$$

The question of how many items satisfy these inequalities is empirical; we show in our experiments that it is small for reasonable heuristics. At worst, the bottom-up phase pops all inside items and reduces to exhaustive dynamic programming.

Additionally, it is worth noting that our algorithm is naturally online in that it can be stopped at any k without advance specification.

2.6 Lazy Successor Functions

The global ordering property guarantees that we will only dequeue derivation fragments of top parses. However, we will *enqueue* all combinations of such items, which is wasteful. By exploiting a local ordering amongst derivations, we can be more conservative about combination and gain the advantages of a lazy successor function (Huang and Chiang, 2005).

To do so, we represent inside derivations not by explicitly specifying entire trees, but rather by using *ranked backpointers*. In this representation, inside derivations are represented in two ways, shown in Figure 1(d) and (e). The first way (d) simply adds a rank u to an edge, giving a tuple (A, i, j, u). The corresponding item is the *ranked derivation item* $K(A, i, j, u)$, which represents the uth-best derivation of A over (i, j). The second representation (e) is a backpointer of the form (A, i, j, l, r, u, v), specifying the derivation formed by combining the uth-best derivation of (B, i, l) and the vth-best derivation of (C, l, j) using rule $r = A \rightarrow B\ C$. The corresponding items $D(A, i, j, l, r, u, v)$ are the new form of our inside derivation items.

The modified deduction schemata for the NAIVE algorithm over these representations are shown in Table 5. The BUILD schema produces new inside derivation items from ranked derivation items, while the RANK schema assigns each derivation item a rank; together they function like DERIV. We can find the k-best list by searching until $K(G, 0, n, k)$ is removed from the agenda. The k-best derivations can then be extracted by following the backpointers for $K(G, 0, n, 1)\ \ldots\ K(G, 0, n, k)$. The KA* algorithm can be modified in the same way, shown in Table 6.

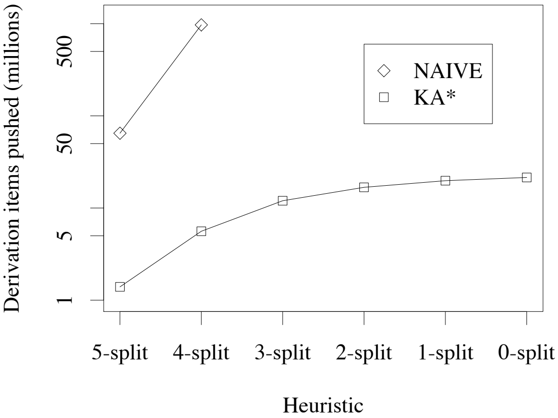

Figure 2: Number of derivation items enqueued as a function of heuristic. Heuristics are shown in decreasing order of tightness. The y-axis is on a log-scale.

The actual laziness is provided by additionally delaying the combination of ranked items. When an item $K(B, i, l, u)$ is popped off the queue, a naive implementation would loop over items $K(C, l, j, v)$ for all v, C, and j (and similarly for left combinations). Fortunately, little looping is actually necessary: there is a partial ordering of derivation items, namely, that $D(A, i, j, l, r, u, v)$ will have a lower computed priority than $D(A, i, j, l, r, u-1, v)$ and $D(A, i, j, l, r, u, v-1)$ (Jiménez and Marzal, 2000). So, we can wait until one of the latter two is built before "triggering" the construction of the former. This triggering is similar to the "lazy frontier" used by Huang and Chiang (2005). All of our experiments use this lazy representation.

3 Experiments

3.1 State-Split Grammars

We performed our first experiments with the grammars of Petrov et al. (2006). The training procedure for these grammars produces a hierarchy of increasingly refined grammars through state-splitting. We followed Pauls and Klein (2009) in computing heuristics for the most refined grammar from outside scores for less-split grammars.

We used the Berkeley Parser[4] to learn such grammars from Sections 2-21 of the Penn Treebank (Marcus et al., 1993). We trained with 6 split-merge cycles, producing 7 grammars. We tested these grammars on 100 sentences of length at most 30 of Section 23 of the Treebank. Our "target grammar" was in all cases the most split grammar.

[4] http://berkeleyparser.googlecode.com

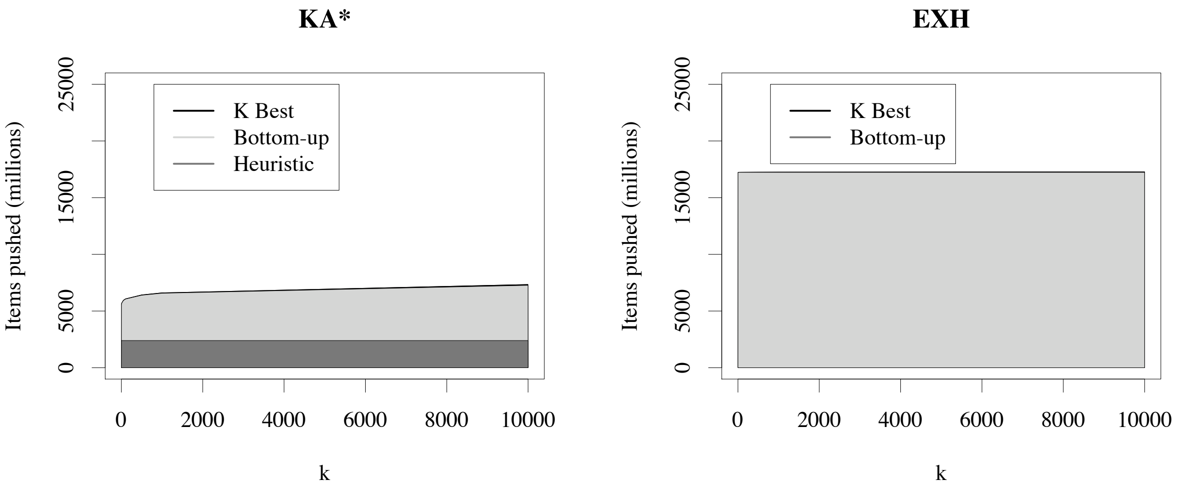

Figure 3: The cost of k-best extraction as a function of k for state-split grammars, for both KA* and EXH. The amount of time spent in the k-best phase is negligible compared to the cost of the bottom-up phase in both cases.

Heuristics computed from projections to successively smaller grammars in the hierarchy form successively looser bounds on the outside scores. This allows us to examine the performance as a function of the tightness of the heuristic. We first compared our algorithm KA* against the NAIVE algorithm. We extracted 1000-best lists using each algorithm, with heuristics computed using each of the 6 smaller grammars.

In Figure 2, we evaluate only the k-best extraction phase by plotting the number of derivation items and outside items added to the agenda as a function of the heuristic used, for increasingly loose heuristics. We follow earlier work (Pauls and Klein, 2009) in using number of edges pushed as the primary, hardware-invariant metric for evaluating performance of our algorithms.[5] While KA* scales roughly linearly with the looseness of the heuristic, NAIVE degrades very quickly as the heuristics get worse. For heuristics given by grammars weaker than the 4-split grammar, NAIVE ran out of memory.

Since the bottom-up pass of k-best parsing is the bottleneck, we also examine the time spent in the 1-best phase of k-best parsing. As a baseline, we compared KA* to the approach of Huang and Chiang (2005), which we will call EXH (see below for more explanation) since it requires exhaustive parsing in the bottom-up pass. We performed the exhaustive parsing needed for EXH in our agenda-based parser to facilitate comparison. For KA*, we included the cost of computing the heuristic, which was done by running our agenda-based parser exhaustively on a smaller grammar to compute outside items; we chose the 3-split grammar for the heuristic since it gives the best overall tradeoff of heuristic and bottom-up parsing time. We separated the items enqueued into items enqueued while computing the heuristic (not strictly part of the algorithm), inside items ("bottom-up"), and derivation and outside items (together "k-best"). The results are shown in Figure 3. The cost of k-best extraction is clearly dwarfed by the the 1-best computation in both cases. However, KA* is significantly faster over the bottom-up computations, even when the cost of computing the heuristic is included.

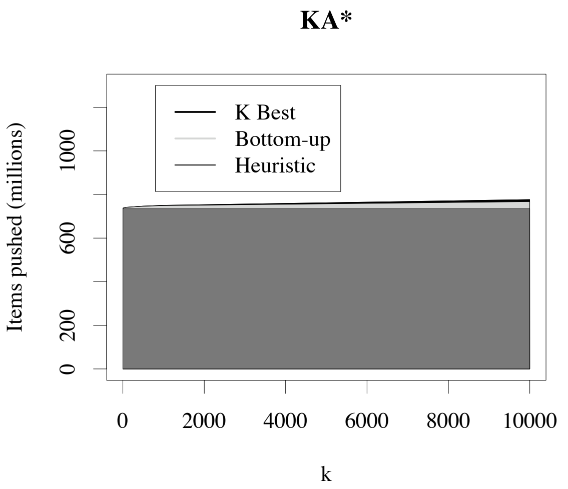

Figure 4: The performance of KA* for lexicalized grammars. The performance is dominated by the computation of the heuristic, so that both the bottom-up phase and the k-best phase are barely visible.

3.2 Lexicalized Parsing

We also experimented with the lexicalized parsing model described in Klein and Manning (2003b). This model is constructed as the product of a dependency model and the unlexicalized PCFG model in Klein and Manning (2003a). We

[5] We found that edges pushed was generally well correlated with parsing time.

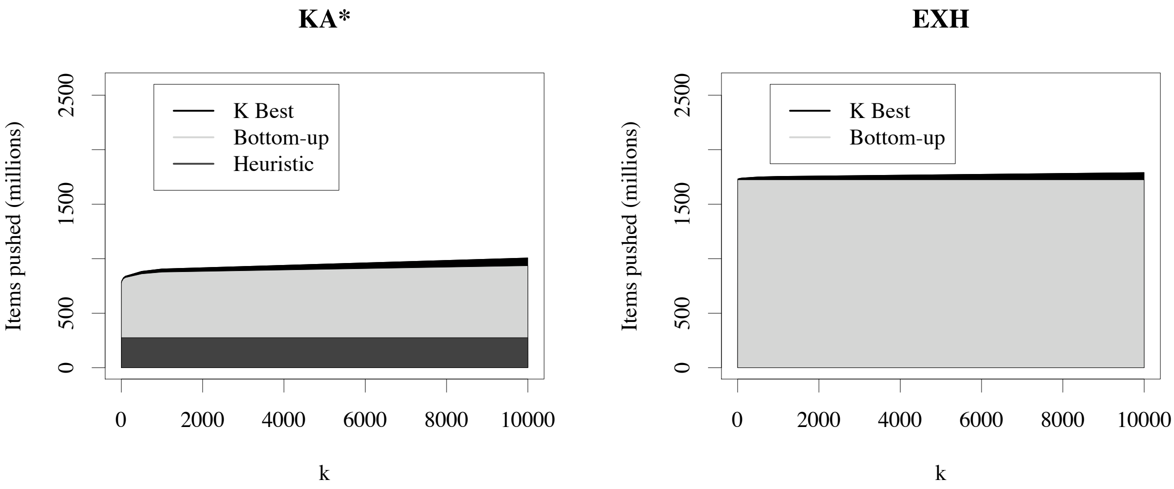

Figure 5: k-best extraction as a function of k for tree transducer grammars, for both KA* and EXH.

constructed these grammars using the Stanford Parser.[6] The model was trained on Sections 2-20 of the Penn Treebank and tested on 100 sentences of Section 21 of length at most 30 words.

For this grammar, Klein and Manning (2003b) showed that a very accurate heuristic can be constructed by taking the sum of outside scores computed with the dependency model and the PCFG model individually. We report performance as a function of k for KA* in Figure 4. Both NAIVE and EXH are impractical on these grammars due to memory limitations. For KA*, computing the heuristic is the bottleneck, after which bottom-up parsing and k-best extraction are very fast.

3.3 Tree Transducer Grammars

Syntactic machine translation (Galley et al., 2004) uses tree transducer grammars to translate sentences. Transducer rules are synchronous context-free productions that have both a source and a target side. We examine the cost of k-best parsing in the source side of such grammars with KA*, which can be a first step in translation.

We extracted a grammar from 220 million words of Arabic-English bitext using the approach of Galley et al. (2006), extracting rules with at most 3 non-terminals. These rules are highly lexicalized. About 300K rules are applicable for a typical 30-word sentence; we filter the rest. We tested on 100 sentences of length at most 40 from the NIST05 Arabic-English test set.

We used a simple but effective heuristic for these grammars, similar to the FILTER heuristic suggested in Klein and Manning (2003c). We projected the source projection to a smaller grammar by collapsing all non-terminal symbols to X, and also collapsing pre-terminals into related clusters. For example, we collapsed the tags NN, NNS, NNP, and NNPS to N. This projection reduced the number of grammar symbols from 149 to 36. Using it as a heuristic for the full grammar suppressed $\sim 60\%$ of the total items (Figure 5).

4 Related Work

While formulated very differently, one limiting case of our algorithm relates closely to the EXH algorithm of Huang and Chiang (2005). In particular, if all inside items are processed before any derivation items, the subsequent number of derivation items and outside items popped by KA* is nearly identical to the number popped by EXH in our experiments (both algorithms have the same ordering bounds on which derivation items are popped). The only real difference between the algorithms in this limited case is that EXH places k-best items on local priority queues per edge, while KA* makes use of one global queue. Thus, in addition to providing a method for speeding up k-best extraction with A*, our algorithm also provides an alternate form of Huang and Chiang (2005)'s k-best extraction that can be phrased in a weighted deduction system.

5 Conclusions

We have presented KA*, an extension of A* parsing that allows extraction of optimal k-best parses without the need for an exhaustive 1-best pass. We have shown in several domains that, with an appropriate heuristic, our algorithm can extract k-best lists in a fraction of the time required by current approaches to k-best extraction, giving the best of both A* parsing and efficient k-best extraction, in a unified procedure.

[6] http://nlp.stanford.edu/software/

References

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*.

P. Felzenszwalb and D. McAllester. 2007. The generalized A* architecture. *Journal of Artificial Intelligence Research*.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT-ACL)*.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *The Annual Conference of the Association for Computational Linguistics (ACL)*.

Joshua Goodman. 1998. *Parsing Inside-Out*. Ph.D. thesis, Harvard University.

Aria Haghighi, John DeNero, and Dan Klein. 2007. Approximate factoring for A* search. In *Proceedings of HLT-NAACL*.

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

Liang Huang. 2005. Unpublished manuscript. `http://www.cis.upenn.edu/~lhuang3/knuth.pdf`.

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 *IWPT*, pages 123–134.

Dan Klein and Chris Manning. 2002. Fast exact inference with a factored model for natural language processing,. In *Proceedings of NIPS*.

Dan Klein and Chris Manning. 2003a. Accurate unlexicalized parsing. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Dan Klein and Chris Manning. 2003b. Factored A* search for models over sequences and trees. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

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

Donald Knuth. 1977. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5.

Shankar Kumar and William Byrne. 2004. Minimum bayes-risk decoding for statistical machine translation. In *Proceedings of The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. In *Computational Linguistics*.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 152–159.

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

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics (ACL)*, pages 160–167, Morristown, NJ, USA. Association for Computational Linguistics.

Adam Pauls and Dan Klein. 2009. Hierarchical search for parsing. In *Proceedings of The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of COLING-ACL 2006*.

Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. In *Machine Learning*, volume 34, pages 151–5175.

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