Efficient Parsing for Transducer Grammars

John DeNero, Mohit Bansal, Adam Pauls, and **Dan Klein**
Computer Science Division
University of California, Berkeley
{denero, mbansal, adpauls, klein}@cs.berkeley.edu

Abstract

The tree-transducer grammars that arise in current syntactic machine translation systems are large, flat, and highly lexicalized. We address the problem of parsing efficiently with such grammars in three ways. First, we present a pair of grammar transformations that admit an efficient cubic-time CKY-style parsing algorithm despite leaving most of the grammar in n-ary form. Second, we show how the number of intermediate symbols generated by this transformation can be substantially reduced through binarization choices. Finally, we describe a two-pass coarse-to-fine parsing approach that prunes the search space using predictions from a subset of the original grammar. In all, parsing time reduces by 81%. We also describe a coarse-to-fine pruning scheme for forest-based language model reranking that allows a 100-fold increase in beam size while reducing decoding time. The resulting translations improve by 1.3 BLEU.

1 Introduction

Current approaches to syntactic machine translation typically include two statistical models: a syntactic transfer model and an n-gram language model. Recent innovations have greatly improved the efficiency of language model integration through multi-pass techniques, such as forest reranking (Huang and Chiang, 2007), local search (Venugopal et al., 2007), and coarse-to-fine pruning (Petrov et al., 2008; Zhang and Gildea, 2008). Meanwhile, translation grammars have grown in complexity from simple inversion transduction grammars (Wu, 1997) to general tree-to-string transducers (Galley et al., 2004) and have increased in size by including more synchronous tree fragments (Galley et al., 2006; Marcu et al., 2006; DeNeefe et al., 2007). As a result of these trends, the syntactic component of machine translation decoding can now account for a substantial portion of total decoding time. In this paper, we focus on efficient methods for parsing with very large tree-to-string grammars, which have flat n-ary rules with many adjacent non-terminals, as in Figure 1. These grammars are sufficiently complex that the purely syntactic pass of our multi-pass decoder is the compute-time bottleneck under some conditions.

Given that parsing is well-studied in the monolingual case, it is worth asking why MT grammars are not simply like those used for syntactic analysis. There are several good reasons. The most important is that MT grammars must do both analysis and generation. To generate, it is natural to memorize larger lexical chunks, and so rules are highly lexicalized. Second, syntax diverges between languages, and each divergence expands the minimal domain of translation rules, so rules are large and flat. Finally, we see most rules very few times, so it is challenging to subcategorize non-terminals to the degree done in analytic parsing. This paper develops encodings, algorithms, and pruning strategies for such grammars.

We first investigate the qualitative properties of MT grammars, then present a sequence of parsing methods adapted to their broad characteristics. We give normal forms which are more appropriate than Chomsky normal form, leaving the rules mostly flat. We then describe a CKY-like algorithm which applies such rules efficiently, working directly over the n-ary forms in cubic time. We show how thoughtful

Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the ACL, pages 227–235,
Boulder, Colorado, June 2009. ©2009 Association for Computational Linguistics

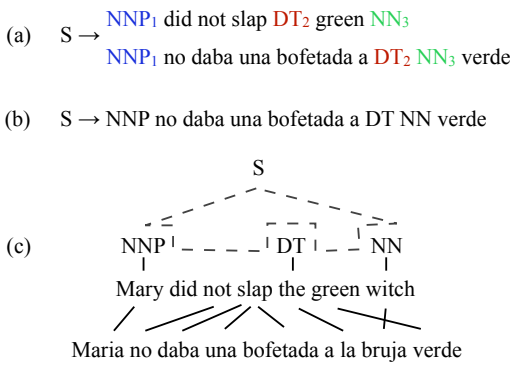

Figure 1: (a) A synchronous transducer rule has co-indexed non-terminals on the source and target side. Internal grammatical structure of the target side has been omitted. (b) The source-side projection of the rule is a monolingual source-language rule with target-side grammar symbols. (c) A training sentence pair is annotated with a target-side parse tree and a word alignment, which license this rule to be extracted.

binarization can further increase parsing speed, and we present a new coarse-to-fine scheme that uses rule subsets rather than symbol clustering to build a coarse grammar projection. These techniques reduce parsing time by 81% in aggregate. Finally, we demonstrate that we can accelerate forest-based reranking with a language model by pruning with information from the parsing pass. This approach enables a 100-fold increase in maximum beam size, improving translation quality by 1.3 BLEU while decreasing total decoding time.

2 Tree Transducer Grammars

Tree-to-string transducer grammars consist of weighted rules like the one depicted in Figure 1. Each n-ary rule consists of a root symbol, a sequence of lexical items and non-terminals on the source-side, and a fragment of a syntax tree on the target side. Each non-terminal on the source side corresponds to a unique one on the target side. Aligned non-terminals share a grammar symbol derived from a target-side monolingual grammar.

These grammars are learned from word-aligned sentence pairs annotated with target-side phrase structure trees. Extraction proceeds by using word alignments to find correspondences between target-side constituents and source-side word spans, then discovering transducer rules that match these constituent alignments (Galley et al., 2004). Given this correspondence, an array of extraction procedures yields rules that are well-suited to machine translation (Galley et al., 2006; DeNeefe et al., 2007; Marcu et al., 2006). Rule weights are estimated by discriminatively combining relative frequency counts and other rule features.

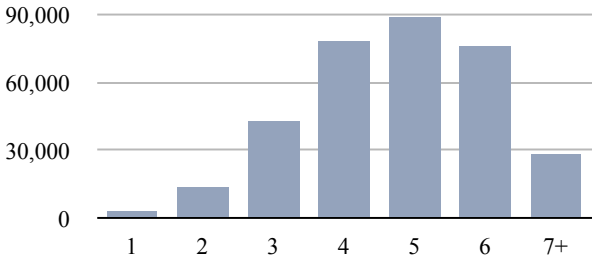

Figure 2: Transducer grammars are composed of very flat rules. Above, the histogram shows rule counts for each rule size among the 332,000 rules that apply to an individual 30-word sentence. The size of a rule is the total number of non-terminals and lexical items in its source-side yield.

A transducer grammar G can be projected onto its source language, inducing a monolingual grammar. If we weight each rule by the maximum weight of its projecting synchronous rules, then parsing with this projected grammar maximizes the translation model score for a source sentence. We need not even consider the target side of transducer rules until integrating an n-gram language model or other non-local features of the target language.

We conduct experiments with a grammar extracted from 220 million words of Arabic-English bitext, extracting rules with up to 6 non-terminals. A histogram of the size of rules applicable to a typical 30-word sentence appears in Figure 2. The grammar includes 149 grammatical symbols, an augmentation of the Penn Treebank symbol set. To evaluate, we decoded 300 sentences of up to 40 words in length from the NIST05 Arabic-English test set.

3 Efficient Grammar Encodings

Monolingual parsing with a source-projected transducer grammar is a natural first pass in multi-pass decoding. These grammars are qualitatively different from syntactic analysis grammars, such as the lexicalized grammars of Charniak (1997) or the heavily state-split grammars of Petrov et al. (2006).

In this section, we develop an appropriate grammar encoding that enables efficient parsing.

It is problematic to convert these grammars into Chomsky normal form, which CKY requires. Because transducer rules are very flat and contain specific lexical items, binarization introduces a large number of intermediate grammar symbols. Rule size and lexicalization affect parsing complexity whether the grammar is binarized explicitly (Zhang et al., 2006) or implicitly binarized using Early-style intermediate symbols (Zollmann et al., 2006). Moreover, the resulting binary rules cannot be Markovized to merge symbols, as in Klein and Manning (2003), because each rule is associated with a target-side tree that cannot be abstracted.

We also do not restrict the form of rules in the grammar, a common technique in syntactic machine translation. For instance, Zollmann et al. (2006) follow Chiang (2005) in disallowing adjacent non-terminals. Watanabe et al. (2006) limit grammars to Griebach-Normal form. However, general tree transducer grammars provide excellent translation performance (Galley et al., 2006), and so we focus on parsing with all available rules.

3.1 Lexical Normal Form

Sequences of consecutive non-terminals complicate parsing because they require a search over non-terminal boundaries when applied to a sentence span. We transform the grammar to ensure that all rules containing lexical items (lexical rules) do not contain sequences of non-terminals. We allow both unary and binary non-lexical rules.

Let L be the set of lexical items and V the set of non-terminal symbols in the original grammar. Then, *lexical normal form* (LNF) limits productions to two forms:

$$\textbf{Non-lexical: } X \rightarrow X_1(X_2)$$
$$\textbf{Lexical: } X \rightarrow (X_1)\alpha(X_2)$$
$$\alpha = w^+(X_i w^+)^*$$

Above, all $X_i \in V$ and $w^+ \in L^+$. Symbols in parentheses are optional. The nucleus α of lexical rules is a mixed sequence that has lexical items on each end and no adjacent non-terminals.

Converting a grammar into LNF requires two steps. In the *sequence elimination* step, for every lexical rule we replace each sequence of consecutive non-terminals $X_1 \ldots X_n$ with the intermediate symbol $X_1{+}\ldots{+}X_n$ (abbreviated $X_{1:n}$) and introduce a non-lexical rule $X_1{+}\ldots{+}X_n \rightarrow X_1 \ldots X_n$. In the *binarization* step, we introduce further intermediate symbols and rules to binarize all non-lexical rules in the grammar, including those added by sequence elimination.

Original grammar rules are flat and lexical

S → NNP no daba una bofetada a DT NN verde

NP → DT NN NNS

LNF replaces non-terminal sequences in lexical rules

S → NNP no daba una bofetada a DT+NN verde

DT+NN → DT NN

Non-lexical rules are binarized using few symbols

Non-lexical rules before binarization:

NP → DT NN NNS DT+NN → DT NN

Equivalent binary rules, minimizing symbol count:

NP → DT+NN NNS DT+NN → DT NN

Anchored LNF rules are bounded by lexical items

S\NNP → no daba una bofetada a DT+NN verde

NP → DT+NN NNS DT+NN → DT NN

S → NNP S\NNP

Figure 3: We transform the original grammar by first eliminating non-terminal sequences in lexical rules. Next, we binarize, adding a minimal number of intermediate grammar symbols and binary non-lexical rules. Finally, anchored LNF further transforms lexical rules to begin and end with lexical items by introducing additional symbols.

3.2 Non-terminal Binarization

Exactly how we binarize non-lexical rules affects the total number of intermediate symbols introduced by the LNF transformation.

Binarization involves selecting a set of symbols that will allow us to assemble the right-hand side $X_1 \ldots X_n$ of every non-lexical rule using binary productions. This symbol set must at least include the left-hand side of every rule in the grammar (lexical and non-lexical), including the intermediate

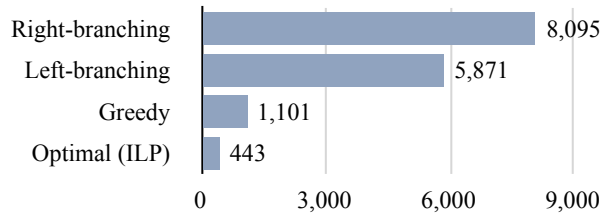

Figure 4: The number of non-terminal symbols introduced to the grammar through LNF binarization depends upon the policy for binarizing type sequences. This experiment shows results from transforming a grammar that has already been filtered for a particular short sentence. Both the greedy and optimal binarizations use far fewer symbols than naive binarizations.

symbols $X_{1:n}$ introduced by sequence elimination.

To ensure that a symbol sequence $X_1 \dots X_n$ can be constructed, we select a split point k and add intermediate types $X_{1:k}$ and $X_{k+1:n}$ to the grammar. We must also ensure that the sequences $X_1 \dots X_k$ and $X_{k+1} \dots X_n$ can be constructed. As baselines, we used *left-branching* (where $k = 1$ always) and *right-branching* (where $k = n - 1$) binarizations.

We also tested a *greedy* binarization approach, choosing k to minimize the number of grammar symbols introduced. We first try to select k such that both $X_{1:k}$ and $X_{k+1:n}$ are already in the grammar. If no such k exists, we select k such that one of the intermediate types generated is already used. If no such k exists again, we choose $k = \lfloor \frac{1}{2} n \rfloor$. This policy only creates new intermediate types when necessary. Song et al. (2008) propose a similar greedy approach to binarization that uses corpus statistics to select common types rather than explicitly reusing types that have already been introduced.

Finally, we computed an *optimal* binarization that explicitly minimizes the number of symbols in the resulting grammar. We cast the minimization as an integer linear program (ILP). Let V be the set of all base non-terminal symbols in the grammar. We introduce an indicator variable T_Y for each symbol $Y \in V^+$ to indicate that Y is used in the grammar. Y can be either a base non-terminal symbol X_i or an intermediate symbol $X_{1:n}$. We also introduce indicators $A_{Y,Z}$ for each pairs of symbols, indicating that both Y and Z are used in the grammar. Let $\mathcal{L} \subseteq V^+$ be the set of left-hand side symbols for all lexical and non-lexical rules already in the grammar. Let $\mathcal{R}$ be the set of symbol sequences on the right-hand side of all non-lexical rules. Then, the ILP takes the form:

$$\min \sum_{Y \in V^+} T_Y \quad (1)$$

$$\text{s.t. } T_Y = 1 \qquad \forall\, Y \in \mathcal{L} \quad (2)$$

$$1 \leq \sum_k A_{X_{1:k}, X_{k+1:n}} \quad \forall\, X_1 \dots X_n \in \mathcal{R} \quad (3)$$

$$T_{X_{1:n}} \leq \sum_k A_{X_{1:k}, X_{k+1:n}} \qquad \forall\, X_{1:n} \quad (4)$$

$$A_{Y,Z} \leq T_Y \;,\; A_{Y,Z} \leq T_Z \qquad \forall\, Y, Z \quad (5)$$

The solution to this ILP indicates which symbols appear in a minimal binarization. Equation 1 explicitly minimizes the number of symbols. Equation 2 ensures that all symbols already in the grammar remain in the grammar.

Equation 3 does not require that a symbol represent the entire right-hand side of each non-lexical rule, but does ensure that each right-hand side sequence can be built from two subsequence symbols. Equation 4 ensures that any included intermediate type can also be built from two subsequence types. Finally, Equation 5 ensures that if a pair is used, each member of the pair is included. This program can be optimized with an off-the-shelf ILP solver.[1]

Figure 4 shows the number of intermediate grammar symbols needed for the four binarization policies described above for a short sentence. Our ILP solver could only find optimal solutions for very short sentences (which have small grammars after relativization). Because *greedy* requires very little time to compute and generates symbol counts that are close to *optimal* when both can be computed, we use it for our remaining experiments.

3.3 Anchored Lexical Normal Form

We also consider a further grammar transformation, *anchored lexical normal form* (ALNF), in which the yield of lexical rules must begin and end with a lexical item. As shown in the following section, ALNF improves parsing performance over LNF by shifting work from lexical rule applications to non-lexical

[1] We used lp_solve: *http://sourceforge.net/projects/lpsolve*.

rule applications. ALNF consists of rules with the following two forms:

$$\textbf{Non-lexical: } X \rightarrow X_1(X_2)$$
$$\textbf{Lexical: } X \rightarrow w^+(X_i w^+)^*$$

To convert a grammar into ALNF, we first transform it into LNF, then introduce additional binary rules that split off non-terminal symbols from the ends of lexical rules, as shown in Figure 3.

4 Efficient CKY Parsing

We now describe a CKY-style parsing algorithm for grammars in LNF. The dynamic program is organized into spans S_{ij} and computes the Viterbi score $w(i, j, X)$ for each edge $S_{ij}[X]$, the weight of the maximum parse over words $i+1$ to j, rooted at symbol X. For each S_{ij}, computation proceeds in three phases: binary, lexical, and unary.

4.1 Applying Non-lexical Binary Rules

For a span S_{ij}, we first apply the binary non-lexical rules just as in standard CKY, computing an intermediate Viterbi score $w_b(i, j, X)$. Let ω_r be the weight of rule r. Then, $w_b(i, j, X) =$

$$\max_{r=X\rightarrow X_1X_2} \omega_r \max_{k=i+1}^{j-1} w(i, k, X_1) \cdot w(k, j, X_2).$$

The quantities $w(i, k, X_1)$ and $w(k, j, X_2)$ will have already been computed by the dynamic program. The work in this phase is cubic in sentence length.

4.2 Applying Lexical Rules

On the other hand, lexical rules in LNF can be applied without binarization, because they only apply to particular spans that contain the appropriate lexical items. For a given S_{ij}, we first compute all the legal mappings of each rule onto the span. A *mapping* consists of a correspondence between non-terminals in the rule and subspans of S_{ij}. In practice, there is typically only one way that a lexical rule in LNF can map onto a span, because most lexical items will appear only once in the span.

Let m be a legal mapping and r its corresponding rule. Let $S^{(i)}_{k\ell}[X]$ be the edge mapped to the ith non-terminal of r under m, and ω_r the weight of r. Then,

$$w_l(i, j, X) = \max_m \omega_r \prod_{S^{(i)}_{k\ell}[X]} w(k, \ell, X).$$

Again, $w(k, \ell, X)$ will have been computed by the dynamic program. Assuming only a constant number of mappings per rule per span, the work in this phase is quadratic. We can then merge w_l and w_b:

$$w(i, j, X) = \max(w_l(i, j, X), w_b(i, j, X)).$$

To efficiently compute mappings, we store lexical rules in a trie (or suffix array) – a searchable graph that indexes rules according to their sequence of lexical items and non-terminals. This data structure has been used similarly to index whole training sentences for efficient retrieval (Lopez, 2007). To find all rules that map onto a span, we traverse the trie using depth-first search.

4.3 Applying Unary Rules

Unary non-lexical rules are applied after lexical rules and non-lexical binary rules.

$$w(i, j, X) = \max_{r:r=X\rightarrow X_1} \omega_r w(i, j, X_1).$$

While this definition is recursive, we allow only one unary rule application per symbol X at each span to prevent infinite derivations. This choice does not limit the generality of our algorithm: chains of unaries can always be collapsed via a unary closure.

4.4 Bounding Split Points for Binary Rules

Non-lexical binary rules can in principle apply to any span S_{ij} where $j - i \geq 2$, using any split point k such that $i < k < j$. In practice, however, many rules cannot apply to many (i, k, j) triples because the symbols for their children have not been constructed successfully over the subspans S_{ik} and S_{kj}. Therefore, the precise looping order over rules and split points can influence computation time.

We found the following nested looping order for the binary phase of processing an edge $S_{ij}[X]$ gave the fastest parsing times for these grammars:

1. Loop over symbols X_1 for the left child
2. Loop over all rules $X \rightarrow X_1X_2$ containing X_1
3. Loop over split points $k : i < k < j$
4. Update $w_b(i, j, X)$ as necessary

This looping order allows for early stopping via additional bookkeeping in the algorithm. We track the following statistics as we parse:

Grammar	Bound checks	Parsing time
LNF	no	264
LNF	yes	181
ALNF	yes	104

Table 1: Adding bound checks to CKY and transforming the grammar from LNF to anchored LNF reduce parsing time by 61% for 300 sentences of length 40 or less. No approximations have been applied, so all three scenarios produce no search errors. Parsing time is in minutes.

$\min_{\text{END}}(i, X)$, $\max_{\text{END}}(i, X)$: The minimum and maximum position k for which symbol X was successfully built over S_{ik}.

$\min_{\text{START}}(j, X)$, $\max_{\text{START}}(j, X)$: The minimum and maximum position k for which symbol X was successfully built over S_{kj}.

We then bound k by $\min_k$ and $\max_k$ in the inner loop using these statistics. If ever $\min_k > \max_k$, then the loop is terminated early.

1. set $\min_k = i + 1, \max_k = j - 1$
2. loop over symbols X_1 for the left child
   $\min_k = \max(\min_k, \min_{\text{END}}(i, X_1))$
   $\max_k = \min(\max_k, \max_{\text{END}}(i, X_1))$
3. loop over rules $X \rightarrow X_1 X_2$
   $\min_k = \max(\min_k, \min_{\text{START}}(j, X_2))$
   $\max_k = \min(\max_k, \max_{\text{START}}(j, X_2))$
4. loop over split points $k : \min_k \leq k \leq \max_k$
5. update $w_b(i, j, X)$ as necessary

In this way, we eliminate unnecessary work by avoiding split points that we know beforehand cannot contribute to $w_b(i, j, X)$.

4.5 Parsing Time Results

Table 1 shows the decrease in parsing time from including these bound checks, as well as switching from lexical normal form to anchored LNF.

Using ALNF rather than LNF increases the number of grammar symbols and non-lexical binary rules, but makes parsing more efficient in three ways. First, it decreases the number of spans for which a lexical rule has a legal mapping. In this way, ALNF effectively shifts work from the lexical phase to the binary phase. Second, ALNF reduces the time spent searching the trie for mappings, because the first transition into the trie must use an edge with a lexical item. Finally, ALNF improves the frequency that, when a lexical rule matches a span, we have successfully built every edge $S_{k\ell}[X]$ in the mapping for that rule. This frequency increases from 45% to 96% with ALNF.

5 Coarse-to-Fine Search

We now consider two coarse-to-fine approximate search procedures for parsing with these grammars. Our first approach clusters grammar symbols together during the coarse parsing pass, following work in analytic parsing (Charniak and Caraballo, 1998; Petrov and Klein, 2007). We collapse all intermediate non-terminal grammar symbols (e.g., NP) to a single coarse symbol X, while pre-terminal symbols (e.g., NN) are hand-clustered into 7 classes (nouns, verbals, adjectives, punctuation, etc.). We then project the rules of the original grammar into this simplified symbol set, weighting each rule of the coarse grammar by the maximum weight of any rule that mapped onto it.

In our second and more successful approach, we select a subset of grammar symbols. We then include only and all rules that can be built using those symbols. Because the grammar includes many rules that are compositions of smaller rules, parsing with a subset of the grammar still provides meaningful scores that can be used to prune base grammar symbols while parsing under the full grammar.

5.1 Symbol Selection

To compress the grammar, we select a small subset of symbols that allow us to retain as much of the original grammar as possible. We use a voting scheme to select the symbol subset. After conversion to LNF (or ALNF), each lexical rule in the original grammar votes for the symbols that are required to build it. A rule votes as many times as it was observed in the training data to promote frequent rules. We then select the top n_l symbols by vote count and include them in the coarse grammar C.

We would also like to retain as many non-lexical rules from the original grammar as possible, but the right-hand side of each rule can be binarized in many ways. We again use voting, but this time each non-

Pruning	Minutes	Model score	BLEU
No pruning	104	60,179	44.84
Clustering	79	60,179	44.84
Subsets	50	60,163	44.82

Table 2: Coarse-to-fine pruning speeds up parsing time with minimal effect on either model score or translation quality. The coarse grammar built using symbol *subsets* outperforms *clustering* grammar symbols, reducing parsing time by 52%. These experiments do not include a language model.

lexical rule votes for its yield, a sequence of symbols. We select the top n_u symbol sequences as the set $\mathcal{R}$ of right-hand sides.

Finally, we augment the symbol set of C with intermediate symbols that can construct all sequences in $\mathcal{R}$, using only binary rules. This step again requires choosing a binarization for each sequence, such that a minimal number of additional symbols is introduced. We use the greedy approach from Section 3.2. We then include in C all rules from the original grammar that can be built from the symbols we have chosen. Surprisingly, we are able to retain 76% of the grammar rules while excluding 92% of the grammar symbols[2], which speeds up parsing substantially.

5.2 Max Marginal Thresholding

We parse first with the coarse grammar to find the Viterbi derivation score for each edge $S_{ij}[X]$. We then perform a Viterbi outside pass over the chart, like a standard outside pass but replacing $\sum$ with $\max$ (Goodman, 1999). The product of an edge's Viterbi score and its Viterbi outside score gives a *max marginal*, the score of the maximal parse that uses the edge.

We then prune away regions of the chart that deviate in their coarse max marginal from the global Viterbi score by a fixed margin tuned on a development set. Table 2 shows that both methods of constructing a coarse grammar are effective in pruning, but selecting symbol subsets outperformed the more typical clustering approach, reducing parsing time by an additional factor of 2.

[2]We used n_l of 500 and n_u of 4000 for experiments. These parameters were tuned on a development set.

6 Language Model Integration

Large n-gram language models (LMs) are critical to the performance of machine translation systems. Recent innovations have managed the complexity of LM integration using multi-pass architectures. Zhang and Gildea (2008) describes a coarse-to-fine approach that iteratively increases the order of the LM. Petrov et al. (2008) describes an additional coarse-to-fine hierarchy over language projections. Both of these approaches integrate LMs via bottom-up dynamic programs that employ beam search. As an alternative, Huang and Chiang (2007) describes a forest-based reranking algorithm called *cube growing*, which also employs beam search, but focuses computation only where necessary in a top-down pass through a parse forest.

In this section, we show that the coarse-to-fine idea of constraining each pass using marginal predictions of the previous pass also applies effectively to cube growing. Max marginal predictions from the parse can substantially reduce LM integration time.

6.1 Language Model Forest Reranking

Parsing produces a forest of derivations, where each edge in the forest holds its Viterbi (or one-best) derivation under the transducer grammar. In forest reranking via cube growing, edges in the forest produce k-best lists of derivations that are scored by both the grammar and an n-gram language model. Using ALNF, each edge must first generate a k-best list of derivations that are *not* scored by the language model. These derivations are then flattened to remove the binarization introduced by ALNF, so that the resulting derivations are each rooted by an n-ary rule r from the original grammar. The leaves of r correspond to sub-edges in the chart, which are recursively queried for their best language-model-scored derivations. These sub-derivations are combined by r, and new n-grams at the edges of these derivations are scored by the language model.

The language-model-scored derivations for the edge are placed on a priority queue. The top of the priority queue is repeatedly removed, and its successors added back on to the queue, until k language-model-scored derivations have been discovered. These k derivations are then sorted and

Pruning strategy	Max beam	BLEU	TM score	LM score	Total score	Inside time	Outside time	LM time	Total time
No pruning	20	57.67	58,570	-17,202	41,368	99	0	247	346
CTF parsing	200	58.43	58,495	-16,929	41,556	53	0	186	239
CTF reranking	200	58.63	58,582	-16,998	41,584	98	64	79	241
CTF parse + rerank	2000	58.90	58,602	-16,980	41,622	53	52	148	253

Table 3: Time in minutes and performance for 300 sentences. We used a trigram language model trained on 220 million words of English text. The *no pruning* baseline used a fix beam size for forest-based language model reranking. *Coarse-to-fine parsing* included a coarse pruning pass using a symbol subset grammar. *Coarse-to-fine reranking* used max marginals to constrain the reranking pass. *Coarse-to-fine parse + rerank* employed both of these approximations.

supplied to parent edges upon request.[3]

6.2 Coarse-to-Fine Parsing

Even with this efficient reranking algorithm, integrating a language model substantially increased decoding time and memory use. As a baseline, we reranked using a small fixed-size beam of 20 derivations at each edge. Larger beams exceeded the memory of our hardware. Results appear in Table 3.

Coarse-to-fine parsing before LM integration substantially improved language model reranking time. By pruning the chart with max marginals from the coarse symbol subset grammar from Section 5, we were able to rerank with beams of length 200, leading to a 0.8 BLEU increase and a 31% reduction in total decoding time.

6.3 Coarse-to-Fine Forest Reranking

We realized similar performance and speed benefits by instead pruning with max marginals from the full grammar. We found that LM reranking explored many edges with low max marginals, but used few of them in the final decoder output. Following the coarse-to-fine paradigm, we restricted the reranker to edges with a max marginal above a fixed threshold. Furthermore, we varied the beam size of each edge based on the parse. Let Δ_m be the ratio of the max marginal for edge m to the global Viterbi derivation for the sentence. We used a beam of size $\lceil k \cdot 2^{\ln \Delta_m} \rceil$ for each edge.

Computing max marginals under the full grammar required an additional outside pass over the full parse forest, adding substantially to parsing time. However, soft coarse-to-fine pruning based on these max marginals also allowed for beams up to length 200, yielding a 1.0 BLEU increase over the baseline and a 30% reduction in total decoding time.

We also combined the coarse-to-fine parsing approach with this soft coarse-to-fine reranker. Tiling these approximate search methods allowed another 10-fold increase in beam size, further improving BLEU while only slightly increasing decoding time.

7 Conclusion

As translation grammars increase in complexity while innovations drive down the computational cost of language model integration, the efficiency of the parsing phase of machine translation decoding is becoming increasingly important. Our grammar normal form, CKY improvements, and symbol subset coarse-to-fine procedure reduced parsing time for large transducer grammars by 81%.

These techniques also improved forest-based language model reranking. A full decoding pass without any of our innovations required 511 minutes using only small beams. Coarse-to-fine pruning in both the parsing and language model passes allowed a 100-fold increase in beam size, giving a performance improvement of 1.3 BLEU while decreasing total decoding time by 50%.

Acknowledgements

This work was enabled by the Information Sciences Institute Natural Language Group, primarily through the invaluable assistance of Jens Voeckler, and was supported by the National Science Foundation (NSF) under grant IIS-0643742.

[3] Huang and Chiang (2007) describes the cube growing algorithm in further detail, including the precise form of the successor function for derivations.

References

Eugene Charniak and Sharon Caraballo. 1998. New figures of merit for best-first probabilistic chart parsing. In *Computational Linguistics*.

Eugene Charniak. 1997. Statistical techniques for natural language parsing. In *National Conference on Artificial Intelligence*.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *The Annual Conference of the Association for Computational Linguistics*.

Steve DeNeefe, Kevin Knight, Wei Wang, and Daniel Marcu. 2007. What can syntax-based MT learn from phrase-based MT? In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

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*.

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*.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *The Annual Conference of the Association for Computational Linguistics*.

Dan Klein and Chris Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the Association for Computational Linguistics*.

Adam Lopez. 2007. Hierarchical phrase-based translation with suffix arrays. In *The Conference on Empirical Methods in Natural Language Processing*.

Daniel Marcu, Wei Wang, Abdessamad Echihabi, and Kevin Knight. 2006. SPMT: Statistical machine translation with syntactified target language phrases. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *The Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *The Annual Conference of the Association for Computational Linguistics*.

Slav Petrov, Aria Haghighi, and Dan Klein. 2008. Coarse-to-fine syntactic machine translation using language projections. In *The Conference on Empirical Methods in Natural Language Processing*.

Xinying Song, Shilin Ding, and Chin-Yew Lin. 2008. Better binarization for the CKY parsing. In *The Conference on Empirical Methods in Natural Language Processing*.

Ashish Venugopal, Andreas Zollmann, and Stephan Vogel. 2007. An efficient two-pass approach to synchronous-CFG driven statistical MT. In *In Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference*.

Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. 2006. Left-to-right target generation for hierarchical phrase-based translation. In *The Annual Conference of the Association for Computational Linguistics*.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23:377–404.

Hao Zhang and Daniel Gildea. 2008. Efficient multi-pass decoding for synchronous context free grammars. In *The Annual Conference of the Association for Computational Linguistics*.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *North American Chapter of the Association for Computational Linguistics*.

Andreas Zollmann, Ashish Venugopal, and Stephan Vogel. 2006. Syntax augmented machine translation via chart parsing. In *The Statistical Machine Translation Workshop at the North American Association for Computational Linguistics Conference*.