International Workshop on
Spoken Language Translation 2011
San Francisco, CA, USA
December 8-9, 2011

Left Language Model State for Syntactic Machine Translation

Kenneth Heafield[1], *Hieu Hoang*[2], *Philipp Koehn*[3],
Tetsuo Kiso[4], *Marcello Federico*[5]

[1] Carnegie Mellon University, USA
[2] Asia Online, Thailand
[3] University of Edinburgh, UK
[4] Nara Institute of Science and Technology, Japan
[5] FBK - Fondazione Bruno Kessler, Italy

heafield@cs.cmu.edu, hieu.hoang@asiaonline.net, pkoehn@inf.ed.ac.uk,
tetsuo-s@is.naist.jp, federico@fbk.eu

Abstract

Many syntactic machine translation decoders, including Moses, cdec, and Joshua, implement bottom-up dynamic programming to integrate N-gram language model probabilities into hypothesis scoring. These decoders concatenate hypotheses according to grammar rules, yielding larger hypotheses and eventually complete translations. When hypotheses are concatenated, the language model score is adjusted to account for boundary-crossing n-grams. Words on the boundary of each hypothesis are encoded in *state*, consisting of left state (the first few words) and right state (the last few words). We speed concatenation by encoding left state using data structure pointers in lieu of vocabulary indices and by avoiding unnecessary queries. To increase the decoder's opportunities to recombine hypothesis, we minimize the number of words encoded by left state. This has the effect of reducing search errors made by the decoder. The resulting gain in model score is smaller than for right state minimization, which we explain by observing a relationship between state minimization and language model probability. With a fixed cube pruning pop limit, we show a 3-6% reduction in CPU time and improved model scores. Reducing the pop limit to the point where model scores tie the baseline yields a net 11% reduction in CPU time.

1. Introduction

Language models are an important feature in syntactic machine translation. While most features can be summed over grammar rules that comprise a constituent, language models examine cross-constituent N-grams. A straightforward dynamic programming algorithm [1] accounts for these N-grams by combining hypotheses only if their first $N-1$ and last $N-1$ words are the same. This algorithm takes $O(V^{2N-2})$ time and space per constituent, where V is the vocabulary size. That is too expensive, so practical decoders implement approximate search by estimating the probability of sentence fragments for purposes of pruning and prioritization. We focus on the decoders [2, 3, 4] that build translations bottom-up by recursively concatenating sentence fragments, estimating their score after each rule application.

Cube pruning [5] is a commonly-implemented method to prioritize and prune grammar rule applications. Within a hypergraph node, each non-terminal has a set of possible values. Applying a rule consists of choosing a value for each non-terminal and scoring. Cube pruning estimates that the score under rule application will be the product (or sum in log space) of the scores of the rule itself and of each value. It then uses these estimates to prioritize rule applications, starting with the highest estimated score. This process continues until the *pop limit* is reached, which acts as a hard limit on the number of rule applications computed in each constituent.

The pop limit is a trade-off between search accuracy (in turn impacting translation quality) and speed. We aim to improve this trade-off both by making better search decisions and by decreasing the computational cost of a pop.

When values are substituted for non-terminals, the score is adjusted to account for N-grams across values. To do so correctly, each decoder hypothesis is annotated with *state*. State consists of *left state*, encoding at most the first N–1 words of the hypothesis, and *right state*, encoding at most the last N–1 words of the hypothesis. When hypotheses have equal state, the decoder recombines them, thus efficiently reasoning over many sentence fragments via dynamic programming. To increase recombination, it is desirable to encode less than $2N$–2 words where possible.

In this paper, we make three improvements related to state and concatenation:

1. Minimizing the number of words encoded by left state, enabling more recombination.
2. Encoding left state using pointers into the language model's data structure, making concatenation faster.
3. Avoiding queries that will not impact the estimated score, speeding concatenation.

2. Related Work

Some decoders [6, 7] avoid left state entirely by building translations left-to-right. Hypotheses may therefore recombine, for purposes of language modeling, when their right states are equal. Typically, these decoders use beam search, where the beam consists of approximately comparable hypotheses, such as those of equal length. These decoders have the advantage that more recombinations do happen, although they risk repeating work because constituents are evaluated in multiple different contexts. The purpose of our work here is not to decide whether one search algorithm or approximation is better, but simply to improve the commonly-implemented bottom-up strategy.

A faster alternative to bottom-up cube pruning is cube growing [8] that lazily generates hypotheses for each constituent instead of generating a fixed number. Like cube pruning, cube growing generates sentence fragments, recombines them into hypotheses, and ranks hypotheses according to estimated language model probabilities. The improvements we discuss here are therefore complementary, since the effect of our work is to improve recombination and ranking within each constituent.

Prior work [9] described and implemented algorithms to minimize left and right state in the context of a bottom-up chart decoder. For hypotheses shorter than N–1 words, they store the entire hypothesis in state. In our work, we apply state minimization to all hypotheses, including those shorter than N–1 words. For hypotheses longer than N–1 words, we minimize state in the same way that [9] does.

While [9] described an "inefficient implementation of the prefix- and suffix-lookup", we store the additional information with each n-gram entry, incurring minimal overhead and reusing lookups already performed in the normal course of scoring. Further, our work makes scoring faster by encoding left state using pointers instead of vocabulary identifiers and by exiting the scoring process early, two optimizations not performed in prior work.

Experiments in [9] tested a system with both left and right state minimization against a baseline without any state minimization. They related the impact of left and right state minimization to the sparsity of the language model. We agree with their assessment and make an additional observation: compared to right state minimization, left state minimization causes more recombination but less impact on single-best outputs. Observing and explaining this discrepancy is a key contribution of our work.

SRILM [10] is a commonly used language model toolkit based on reverse tries. It exposes two pieces of information to the decoder: the probability of an n-gram and, through a separate call that repeats lookups, the minimum number of words to encode in right state. There is no facility to determine the length of n-gram matched by the model, or whether a given n-gram extends to the left. These properties mean that decoders using SRILM can minimize right state, but typically store all N–1 words of left state.

Another toolkit, IRSTLM [11], provides the length of the n-gram that it matched with each query. Decoders can use this information to store at most n words in left or right state, depending on the position of the words being queried. However, this does not fully minimize state, as w_1^n may be matched by the model, but $w_1^n v$ may not be in the model for any word v. In this case w_1 may be safely omitted from right state but this information is hidden from the decoder. Similarly, were the toolkit to indicate that $v w_1^n$ does not appear for any word v (i.e. w_1^n does not extend left), then w_n could be omitted from left state.

In this work, we extend KenLM [12] to store and expose the necessary information. It implements two data structures, probing and trie. The probing data structure is a hash table from n-grams to probability and backoff and is byte-aligned for speed. The trie data structure is a reverse trie similar to SRILM and IRSTLM but with bit-level packing (i.e. it uses 31 bits to store probability since the sign bit is always negative). Since entries that do not extend right have zero backoff, a special backoff value flags n-grams that do not extend right; this information is provided to the decoder, as is the length of n-gram matched. We will show how to augment both data structures to indicate whether an entry extends to left.

3. Baseline

Here, we describe how the syntactic decoder typically incorporates language model probability into hypothesis scores.

The language model probability of sentence fragment w_1^K is estimated by

$$p(w_1^K) = \left(\prod_{i=1}^{N-1} p(w_i|w_1^{i-1})\right) \prod_{i=N}^{K} p(w_i|w_{i-N+1}^{i-1}) \qquad (1)$$

in which the probabilities of words w_1^{N-1} were estimated from incomplete context. The fragment may be shorter than N–1 words so in general the words w_1^k where $k = \min\{N-1, K\}$ have estimated probability. We refer to w_1^k as the left state of w_1^K.

When fragments w_J^0 (where $J \leq 0$) and w_1^K are concatenated to form w_J^K, the probabilities of w_1^k are adjusted to account for up to N–1 words of additional context, namely w_{2-N}^0. The fragment w_J^0 may be shorter than N–1 words, so in general this additional context is w_j^0 where $j = \max\{2-N, J\}$. We refer to w_j^0 as the right state of w_J^0. An example is shown in Figure 1. The probability of the combined fragment is given by

$$p(w_J^K) = p(w_J^0) p(w_1^K) c(w_j^0, w_1^k) \qquad (2)$$

where correction factor c updates estimates made in equation (1) to account for additional context

$$c(w_j^0, w_1^k) = \prod_{i=1}^{k} \frac{p(w_i|w_{\max\{j,i-N+1\}}^{i-1})}{p(w_i|w_1^{i-1})} \qquad (3)$$

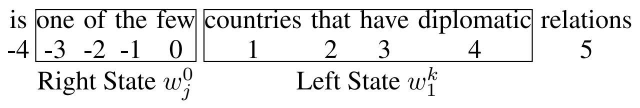

Figure 1: Concatenating two sentence fragments for a 5-gram language model. The right-looking state w_{-3}^0 of the first fragment is followed by the left-looking state w_1^4 of the second fragment.

The correction factor examines the first fragment's right state w_j^0 and the second fragment's left state w_1^k. To enable concatenation on either side, each fragment maintains both left state and right state, collectively referred to as state.

We also include commonly-implemented right state minimization in the baseline. If a fragment ends with w_j^0 such that $w_j^0 v$ is not in the model for any word v and the backoff penalty $b(w_j^0) = 1$, then w_j may be omitted from right state. Minimization applies recursively, so that w_{j+1} and so on may be similarly omitted if they meet the criterion.

This baseline is commonly implemented in syntactic decoders. Prior work [9] went further by minimizing left state of hypotheses longer than N–1 words for the Joshua decoder. However, the implementation is sufficiently inefficient that the Joshua documentation instead recommends using SRILM without any state minimization because this takes less total CPU and memory. We show how to implement left state minimization in a way that makes decoding faster while using the same amount of memory.

4. Improvements

This section explains three improvements that we have made over the baseline.

4.1. Minimizing Left State

When a fragment begins with words w_1^k, we consider minimizing left state to omit w_k and possibly more words. Generally, the word w_k may be omitted from left state if vw_1^k does not explicitly appear in the language model for any v. The rest of the section deals with this more formally along with some corner cases that arise.

An N-gram language model is a sparse set of n-grams for $1 \leq n \leq N$. In order to minimize words encoded by left state, we make use of the *substring property* of language models

Property 1. *If the n-gram w_1^n appears in a language model, then so do substrings w_i^j for all $1 \leq i \leq j \leq n$.*

For unpruned models, the substring property follows naturally from estimation: when w_1^n occurs in the corpus, all of its substrings are also extracted. However, models pruned by SRILM [10] violate this property. In that case, we re-add the missing entries, estimating their probability using the normal backoff procedure explained in the following paragraph. This results in a model that returns the same probabilities and, if default pruning settings were used, typically has 1.4% more n-grams[1].

In addition to the substring property, we exploit the backoff procedure to minimize left state. When queried for $p(w_n|w_1^{n-1})$, the language model finds longest matching entry w_f^n then evaluates

$$p(w_n|w_1^{n-1}) = p(w_n|w_f^{n-1}) \prod_{i=1}^{f-1} b(w_i^{n-1}) \qquad (4)$$

where conditional probability p and backoffs b were estimated when the model was built. The backoff procedure leads to a useful proposition.

Proposition 1. *If w_0^n is not in the model for all w_0 then for all contexts w_j^0 with $j \leq 0$,*

$$\frac{p(w_n|w_j^{n-1})}{p(w_n|w_1^{n-1})} = \prod_{i=\max\{j,n-N+1\}}^{0} b(w_i^{n-1}) \qquad (5)$$

Proof. Let $j \leq 0$ and w_j^0 be words. By hypothesis, w_0^n is not in the model. Applying the substring property, w_i^n is not in the model for each $i \leq 0$. When queried for $p(w_n|w_j^{n-1})$, the model finds longest matching entry w_f^n according to the backoff procedure. Because w_i^n is not in the model for each $i \leq 0$, we have that $f > 0$. When queried for $p(w_n|w_1^{n-1})$, the model finds the longest entry w_e^n. Since w_f^n is in the model and $f > 0$, we have that $e \leq f$ by construction of e. Similarly, w_e^n is in the model so $e \geq f$ by construction of f. Thus $e = f$. Since p is a function, the probability term $p(w_n|w_f^{n-1})$ is the same for both $p(w_n|w_j^{n-1})$ and $p(w_n|w_1^{n-1})$ (recall $j \leq 0$). Dividing the backoff terms in (4) yields the remaining product. $\square$

Equation (5) is noticeably independent of w_n, so if the language model can establish that vw_1^n is not in the model for all words v, it may safely omit w_n from left state. This is established in one of two ways. First, if the query does not find w_1^n in the model ($f > 1$ in the backoff procedure), then by the substring property neither is any w_0^n. Second, if w_1^n is in the model ($f = 1$ in the backoff procedure) then we check if any leftward extension w_0^n exists. In the common reverse trie data structure, this check is trivial since the entry for w_1^n points to all entries w_0^n. Other data structures may precompute this information at model building time then store an additional bit for each n-gram with $n < N$; we abuse the otherwise-constant probability sign bit for this purpose in KenLM's byte-aligned probing data structure.

Backoff weights present a subtle issue. State may be short simply because the hypothesis has fewer than N–1

[1]These added n-grams reduce recombination opportunities in a minor way. We tested this by decoding once without the added entries and once with the added entries. In both cases, right state minimization was enabled but left state minimization was disabled because it requires the substring property. On our 1357-sentence Chinese-English test set, the single-best outputs were identical, including model scores.

words. If state minimization does not further reduce state size, then the backoff weights in equation (5) should not be assessed. We refer to these hypotheses as *small*. Otherwise, the hypothesis is *large*, indicating that the left state does not encode the entire hypothesis. This may be a hypothesis longer than N–1 words or a hypothesis shorter than N–1 words from which left state minimization has omitted a word. In the large case, backoffs should be charged according to equation (5) while these are not charged in the small case. We therefore distinguish the two cases by storing a $flag$ in state. The flag enables us to improve upon [9] who did not minimize state of sentence fragments shorter than N–1 words.

The arguments we have made apply recursively. If w_n can be omitted from state, we consider omitting w_{n-1} as well. At first glance, computing the backoffs in equation (5) seems to depend on w_{n-1}. However, if w_{n-1} can be omitted from left state, then w_0^{n-1} does not appear in the model for all w_0. By the substring property, w_i^{n-1} does not appear in the model for $j \leq i \leq 0$. If an n-gram does not appear in the model, the backoff penalty is 1: $b(w_i^{n-1}) = 1$. These imply that equation (5) evaluates to 1 independent of the specific value of w_{n-1}. Therefore, we may safely omit w_{n-1} if it qualifies (namely w_0^{n-1} is not in the model for any w_0).

After minimization, left state encodes a string w_1^k for some $0 \leq k \leq N-1$ plus the aforementioned $flag$. The next section explains how this state is encoded.

4.2. Encoding Left State

After minimization, the left state encodes a string w_1^k for some $k < N$ and a $flag$ that indicates if backoff should be charged. The purpose of this state is to make computing correction factor c correct and efficient for any context w_j^0. Repeating equation (3),

$$c(w_j^0, w_1^k) = \prod_{i=1}^{k} \frac{p(w_i | w_{\max\{j,i-N+1\}}^{i-1})}{p(w_i|w_1^{i-1})} \tag{6}$$

We want to encode each w_1^i in a way that makes computing the term

$$\frac{p(w_i | w_{\max\{j,i-N+1\}}^{i-1})}{p(w_i|w_1^{i-1})} \tag{7}$$

efficient.

The N-gram language model is stored in a data structure. In the reverse trie data structure implemented by [10, 11], the entry for w_1^i points to an array of entries w_0^i, and these in turn point to the entries w_{-1}^i, etc. In order to query an n-gram w_j^i, the model typically visits w_i, w_{i-1}^i, w_{i-2}^i, and so on until either an entry does not exist or it reaches w_j^i. We speed this process by storing a pointer $d(w_1^i)$ to the record for w_1^i, so that the data structure need not visit $w_i, w_{i-1}^i, \ldots, w_2^i$. It could also avoid revisiting w_1^i by encoding both $p(w_i|w_1^{i-1})$ and pointers[2] to the records w_0^n, however this would increase the memory requirement of left state from 8 bytes per word to 20 bytes per word.

To summarize, left state w_1^k with backoff flag $flag$ is encoded as

$$(\{d(w_1^i)\}_{i=1}^k, flag) \tag{8}$$

In order for data structure pointer $d(w_1^i)$ to be defined, w_1^i must appear in the model. This is the case because w_1^i is a substring of w_1^k, which itself must appear in the model because otherwise w_k would have been omitted by state minimization.

Decoders frequently compare states to assess if they can be recombined. For purposes of comparison, it is sufficient to examine k, $flag$, and $d(w_1^k)$. While we have used the term pointer, $d(w_1^k)$ may actually be an index into the array of k-grams, in which case it is only guaranteed to be unique within the same order. The order of n-gram to which a pointer corresponds is implied by its position in state, so the data structure will know which array to use.

4.3. Exiting Early

When two fragments w_j^0 and w_1^k are concatenated, the model checks for cross-fragment n-grams and adjusts the score accordingly. If some of these n-grams are not present, then it may be able to avoid looking up longer n-grams that will not be found. This section formalizes that notion.

To compute the correction factor c

$$c(w_j^0, w_1^k) = \prod_{i=1}^{k} \frac{p(w_i | w_{\max\{j,i-N+1\}}^{i-1})}{p(w_i|w_1^{i-1})} \tag{9}$$

the model executes a loop in the natural way going from $i = 1$ to k. These queries exhibit a left-to-right pattern wherein w_i is appended to the right to the string considered by the previous iteration. Just as we maintain right state for the right edge of a hypothesis, we maintain updated right state after each w_i is appended. The right state is minimized as described in Section 3, so after some iteration m, it may no longer encode w_0 (and preceding words). In that case, the subsequent queries will not extend to incorporate w_0, so

$$\frac{p(w_i | w_{\max\{j,i-N+1\}}^{i-1})}{p(w_i|w_1^{i-1})} = 1 \tag{10}$$

for all $i > m$. Therefore, these queries may be avoided because they will not change the concatenated fragment's probability.

Initially, it seems that this optimization could have been implemented in the baseline. However, baseline implementations typically evaluate equation (9) by storing the denominator

$$\prod_{i=1}^{k} p(w_i|w_1^{i-1}) \tag{11}$$

[2]Actually, w_1^n points to the beginning of the block of entries for each w_0^n. To determine the end pointer, the code consults the entry immediately following w_1^n that points to the beginning of the next block. So, avoiding a visit to w_1^n would would require storing $p(w_n|w_1^{n-1})$, the begin pointer, and the end pointer.

in state then issuing k queries to form the numerator. Because the denominator atomically incorporates all k terms, the system cannot exit early. Alternatively, each term could be stored separately in state, costing additional memory but permitting early exit. We are able to make less queries and avoid storing the denominator entirely by modifying the language model to return relative scores

$$\frac{p(w_i|w_{\max\{j,i-N+1\}}^{i-1})}{p(w_i|w_1^{i-1})} \quad (12)$$

5. Experiments

In this section, we perform several experiments to measure the specific impact of our changes followed by measuring the overall performance impact. We instrument and modify the Moses chart [2] decoder for these experiments, although improvements have also been ported to cdec [3].

5.1. Test Data

In the experiments, we refer to five systems covering three language pairs:

Chinese-English A hierarchical system trained on 2.1 million parallel news sentences. The language model is an interpolation of models built on the Xinhua and AFP portions of English Gigaword version 4 [13] plus the English side of the parallel data. The test set is 1357 sentences from the NIST 2008 evaluation. The uncased BLEU [14] score is 29.63%.

German-English Two systems, one hierarchical and one with target-side syntax. Rules were extracted from Europarl [15]. The language model interpolates English Europarl, news commentary, and monolingual news released for the Workshop on Machine Translation [16]. The official 3003-sentence test set is used. On this test set, the hierarchical and target syntax systems score 21.10% and 20.14% BLEU, respectively.

English-German A hierarchical system and a target-syntax system, both trained on Europarl. The language model consists of the German monolingual data released for the 2011 Workshop on Machine Translation: Europarl, news commentary, and monolingual news data from each year. As in the evaluation, the test set is the same 3003 sentences used for German-English but translated in the opposite direction. The uncased BLEU scores on this test set are 14.95% for hierarchical and 14.69% for target syntax.

These systems were designed for participation in the NIST Open MT Evaluation (Chinese-English) and Workshop on Machine Translation (German-English and English-German) using constrained data and the normal Moses pipeline. For target syntax models, the Collins parser [17] was used to parse the target side. Language models were built with order

		Right Length					
		0	**1**	**2**	**3**	**4**	**Sum**
Left Length	**0**	0.3	1.1	3.0	3.2	3.2	10.7
	1	0.3	1.7	3.8	3.5	3.5	12.9
	2	0.6	2.9	9.5	8.8	8.5	30.3
	3	0.4	2.1	6.9	7.7	7.5	24.7
	4	0.3	1.6	5.1	5.6	6.7	19.2
Sum		1.9	9.4	28.3	28.8	29.5	97.8

Table 1: Percentage of hypotheses by length of left state and length of right state in the Chinese-English task with a cube pruning pop limit of 1000. On hypotheses considered by the decoder, left state minimization is more aggressive than right state minimization. Excluded from the table, 2.2% of hypotheses were shorter than 4 words, consisting of 0.1% unigrams, 1.1% bigrams, and 1.1% trigrams. Sums were computed prior to rounding.

$N = 5$ using SRILM [10], modified Kneser-Ney smoothing, and the default pruning[3].

5.2. Recombination

We measure the effectiveness of left state minimization in two ways: the extent to which it minimizes state length and the number of recombinations that happen as a result.

In Table 1, we bin hypotheses by their state lengths and report the percentage of hypotheses falling into each bin. Left state minimization is generally more aggressive: 19.2% of left states are full length (4) compared to 29.5% of right states. The trend continues for shorter lengths as well. One reason for this difference is that the decoder applies the glue rule from left to right, generating hypotheses bound to the beginning of sentence. Hypotheses bound to the beginning of sentence cannot extend further left, and so their left state length is zero. Baseline decoders already implement special-case handling for the beginning of sentence that allows hypotheses to recombine even if their first N–1 words differ, effectively dropping left state to zero length as well.

To measure recombination, we decoded with left and right minimization enabled but kept track of the highest-scoring sentence fragment carried by the hypothesis. When two hypotheses recombined, we compared their highest-scoring fragments to decide whether the recombination would have been permitted without state minimization. In the Chinese-English system with a cube pruning pop limit of 1000, recombinations averaged 176,191 per sentence. Of these, 101,409 (57%) agreed on the first and last N–1 words or were covered by the decoder's special case rules that effectively make state length zero at the beginning or end of sentence. 26,111 (15%) additional recombinations were licensed by right state minimization alone. Left state min-

[3]We also tried limited experiments without pruning. The primary result that left state minimization changes the model score less than right state minimization still held.

	Hierarchical			Syntax	
	zh-en	**de-en**	**en-de**	**de-en**	**en-de**
None	-82.5435	-101.392	-79.2035	-104.244	-13.1425
Left	-82.5417	-101.390	-79.2028	-104.244	-13.1415
Right	-82.5368	-101.388	-79.1982	-104.242	-13.1389
Both	-82.5340	-101.387	-79.1972	-104.241	-13.1370

Table 2: Impact of state minimization on average model score. A higher model score is better. Model scores are scale-invariant and include a constant that indicates translation difficulty. Right state minimization increases the score more than does left state minimization. All experiments used a cube pruning pop limit of 1000.

imization permitted another 33,537 (19%) recombinations. Finally, 15,133 (9%) recombinations depended on both right and left state minimization. Compared with right state minimization, left state minimization is more aggressive and leads to more recombinations.

5.3. Model Score

The decoder's objective is to maximize model score, the dot product between feature weights and feature values. Given that increased recombination allows the decoder to reason over more sentence fragments simultaneously, we expect the model score to increase on average[4]. Table 2 shows that the model score does improve with left state minimization, but that the improvement is smaller than the impact of right state minimization. This is surprising because, in Section 5.2, we found that left state minimization is more aggressive and leads to more recombinations when compared to right state minimization. In the next section, we examine why this is the case.

5.4. Behavior Under Concatenation

A priori, it seems that left and right state minimization are symmetric and similarly improve search quality. However, the experiments reported in Table 2 show that left state minimization produces smaller improvement in single-best model score compared with right state minimization. This led us to observe that left and right state differ in how and when backoff penalties are assessed.

Backoff penalties are a function of context as shown in equation (4). When left state is minimized, backoff penalties are likely to be assessed, but context is indeterminate, so assessing some of the penalty is delayed until concatenation in equation (5). Therefore, short left state predicts that a hypothesis will fare poorly in concatenation relative to its score. These are precisely the same hypotheses that recombine as a result of left state minimization. Left state minimization

[4]An increase in model score is not strictly guaranteed: a hypothesis that would have been pruned without recombination might instead be expanded, grow into additional hypotheses, and cause a component of the baseline's single-best hypothesis to be pruned.

For large hypotheses ($flag$ = large):

		Right Length			
		1	**2**	**3**	**4**
Left Length	**0**	-0.741	-1.062	-1.357	-1.701
	1	-0.269	-0.429	-0.588	-0.836
	2	-0.129	-0.236	-0.362	-0.567
	3	0.007	-0.061	-0.128	-0.314
	4	0.220	0.202	0.169	0.037

For small hypotheses ($flag$ = small):

		Right Length			
		1	**2**	**3**	**4**
Left Length	**1**	0.017	-0.068	-0.174	-0.359
	2	0.096	0.046	-0.045	-0.239
	3	0.159	0.130	0.061	-0.117

Table 3: Mean $\log_{10}$ correction factors on the Chinese-English system with pop limit 1000, reported separately by state lengths and by the left state flag. Zero length right state is not shown in the table because estimates were already based on empty context so the correction factor is zero. Small hypotheses can only have lengths 1 through 3. Short left state predicts that scores will decline during concatenation while short right state predicts that scores will increase.

minimally impacts single-best translations because the hypotheses that do recombine are likely to score poorly and be pruned. By contrast, when right state is minimized, the context is known (except in short hypotheses) and penalties have already been included in the hypothesis score.

To measure the effect, we collected the geometric[5] mean correction c for each tuple $(r, l, flag)$ of right state length r, left state length l, and $flag$ distinguishing small hypotheses. Table 3 shows the results in log format. We also wanted to know how well these values generalize across language pairs; Table 4 shows the figures for large hypotheses in the German-English task. Short left state consistently predicts lower (more negative) correction factors while short right state usually predicts positive correction factors.

5.5. Language Model Lookups

We measured the impact of our speed improvements by counting lookups and by timing the decoder. To translate 1357 sentences with a pop limit of 1000, the baseline Chinese-English system made a total of 760,788,076 pops. Turning on left state minimization led to a slight increase in pops at 761,124,349. Each pop entails evaluating the language model probability of the target side of a rule application, consisting of a string of terminals and non-terminals. To do so, it executes queries: one query for each terminal and one query for each word in the left state of a non-terminal.

[5]The language model feature is actually log language model probability, so it is natural to take the arithmetic mean in log space.

For large hypotheses ($flag$ = large):

		Right Length			
		1	**2**	**3**	**4**
Left Length	**0**	-0.780	-1.074	-1.347	-1.573
	1	-0.181	-0.306	-0.455	-0.569
	2	0.023	-0.028	-0.100	-0.213
	3	0.184	0.199	0.190	0.111
	4	0.390	0.481	0.509	0.409

Table 4: Mean $\log_{10}$ correction factors on the German-English hierarchical system with pop limit 1000. Entries show the same general trend as in Table 3, but the magnitude is larger and sometimes long right states fare better than do short right states.

n	**Baseline**	**Pointer**	**Reduction**
1	3,709,029,243	1,988,280,862	46%
2	3,305,400,924	2,128,356,570	35%
3	2,425,086,263	1,692,042,948	30%
4	998,098,720	752,425,908	24%
5	229,849,561	213,076,869	7%

Table 5: Number of language model data structure lookups for each n-gram length made by the Chinese-English system to translate 1357 sentences. Storing a data structure pointer in left state means queries can skip over shorter orders.

Left state minimization and early exit led to a 30% reduction in queries. Surviving queries perform data structure lookups, typically starting with unigrams and proceeding to larger n-grams. Storing pointers in left state allows the model to skip over lower-order n-grams. Table 5 shows the combined effect of reduced queries and data structure pointers.

There is a substantial reduction in lookups, but they are also highly repetitive. In particular, left state pointers only skip over n-grams that have recently been queried and are likely to be cached by hardware. Across all five systems we experiment with, CPU time was reduced by 3-6%. We also tried varying the pop limit to 50, 100, 200, 500, and 1000, finding that the relative improvement still fell within this range. In the next section, we build on these performance gains by using the pop limit to interpret our model score gains as speed gains.

5.6. Pop Limit Tradeoff

We have made use of average model score, but this number is difficult to interpret. By reducing the pop limit until average model score equals that of the baseline, we can interpret search improvements as speed improvements. Figure 2 shows the trade-off between CPU time and average model score as moderated by the pop limit on the Chinese-English task. With a pop limit of 1000, the baseline decoder yields average model score -82.5368 using 19.6 CPU seconds per sentence. Reducing the pop limit to 910 and applying left state minimization, the decoder yields average model score -82.5367 (better than the baseline) using 17.3 CPU seconds per sentence. This is a net 11% reduction in CPU time. Since the quality improvement has been traded for speed, we expect to see no change in translation quality. Using uncased BLEU [14] as a proxy for translation quality, the baseline with pop limit 1000 scored 29.63% while left state minimization with pop limit 910 scored 29.67%, insignificantly better.

In terms of memory consumption, Moses's preexisting method for tracking state is less efficient than our encoding, so there was a small (about 1%) reduction in memory usage across the test systems using the same pop limit. There was no increase in language model storage size. Reverse tries already encode left extension information while we repurposed the probability sign bit to store this information in KenLM's probing data structure.

Timing measurements were performed on a 32-core machine with 64 GB of RAM. The language model and grammar were converted into binary format in advance then faulted into the operating system disk cache before each run.

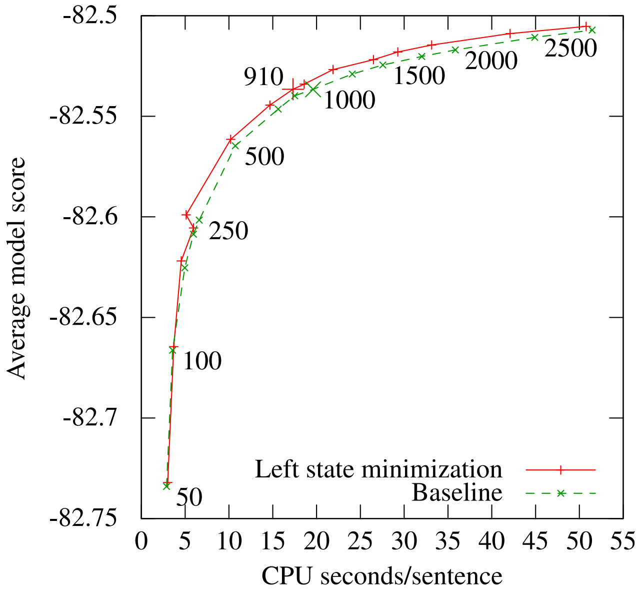

Figure 2: CPU seconds per sentence (user plus system) and average model score on the Chinese-English task. The labels correspond to pop limits. Two points are enlarged: the baseline with pop limit 1000 and left state minimization with pop limit 910, which produces a higher average model score using less time.

6. Conclusion

Exposing more information from the language model enables us to improve the run time of a syntactic decoder by optimizing left language model state handling. Initially, we were surprised to see that left state minimization had less impact that right state minimization. Biases in accounting for backoff penalties explain the difference and, in future work, we plan to investigate correcting for these biases using rest costs. Code has already been released as part of KenLM under the LGPL; both Moses and cdec currently use this functionality.

7. Acknowledgements

This work was supported by the EuroMatrixPlus project funded by the European Commission (7th Framework Programme), by a National Science Foundation Graduate Research Fellowship under Grant No. 0750271, and by the DARPA GALE program. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575. Specifically, benchmarks were run on Trestles at the San Diego Supercomputer Center under allocation TG-CCR110017.

8. References

[1] Y. Bar-Hillel, M. Perles, and E. Shamir, *On Formal Properties of Simple Phrase Structure Grammars*. Hebrew University Students' Press, 1964.

[2] H. Hoang, P. Koehn, and A. Lopez, "A Unified Framework for Phrase-Based, Hierarchical, and Syntax-Based Statistical Machine Translation," in *Proceedings of the International Workshop on Spoken Language Translation*, Tokyo, Japan, 2009, pp. 152–159.

[3] C. Dyer, A. Lopez, J. Ganitkevitch, J. Weese, F. Ture, P. Blunsom, H. Setiawan, V. Eidelman, and P. Resnik, "cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models," in *Proceedings of the ACL 2010 System Demonstrations*, 2010, pp. 7–12.

[4] Z. Li, C. Callison-Burch, C. Dyer, S. Khudanpur, L. Schwartz, W. Thornton, J. Weese, and O. Zaidan, "Joshua: An open source toolkit for parsing-based machine translation," in *Proceedings of the Fourth Workshop on Statistical Machine Translation*. Athens, Greece: Association for Computational Linguistics, March 2009, pp. 135–139.

[5] D. Chiang, "Hierarchical phrase-based translation," *Computational Linguistics*, vol. 33, pp. 201–228, June 2007.

[6] T. Watanabe, H. Tsukada, and H. Isozaki, "Left-to-right target generation for hierarchical phrase-based translation," in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, Sydney, Australia, July 2006, pp. 777–784.

[7] L. Huang and H. Mi, "Efficient incremental decoding for tree-to-string translation." in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, Cambridge, MA, October 2010, pp. 273–283.

[8] L. Huang and D. Chiang, "Forest rescoring: Faster decoding with integrated language models," in *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, Prague, Czech Republic, 2007.

[9] Z. Li and S. Khudanpur, "A scalable decoder for parsing-based machine translation with equivalent language model state maintenance," in *Proceedings of the Second ACL Workshop on Syntax and Structure in Statistical Translation (SSST-2)*, Columbus, Ohio, June 2008, pp. 10–18.

[10] A. Stolcke, "SRILM - an extensible language modeling toolkit," in *Proceedings of the Seventh International Conference on Spoken Language Processing*, 2002, pp. 901–904.

[11] M. Federico, N. Bertoldi, and M. Cettolo, "IRSTLM: an open source toolkit for handling large scale language models," in *Proceedings of Interspeech*, Brisbane, Australia, 2008.

[12] K. Heafield, "KenLM: Faster and smaller language model queries," in *Proceedings of the Sixth Workshop on Statistical Machine Translation*. Edinburgh, Scotland: Association for Computational Linguistics, July 2011.

[13] R. Parker, D. Graff, J. Kong, K. Chen, and K. Maeda, "English gigaword fourth edition," Linguistic Data Consortium (LDC), Philadelphia, 2009, lDC2009T13.

[14] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: a method for automatic evaluation of machine translation," in *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, PA, July 2002, pp. 311–318.

[15] P. Koehn, "Europarl: A parallel corpus for statistical machine translation," in *Proceedings of MT Summit*, 2005.

[16] C. Callison-Burch, P. Koehn, C. Monz, and O. Zaidan, "Findings of the 2011 workshop on statistical machine translation," in *Proceedings of the Sixth Workshop on Statistical Machine Translation*. Edinburgh, Scotland: Association for Computational Linguistics, July 2011, pp. 22–64. [Online]. Available: http://www.aclweb.org/anthology/W11-2103

[17] M. Collins, "Head-driven statistical models for natural language parsing," Ph.D. dissertation, University of Pennsylvania, 1999.