Language Model Rest Costs and Space-Efficient Storage

Kenneth Heafield[*,†] **Philipp Koehn**[†] **Alon Lavie**[*]

[*] Language Technologies Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
{heafield,alavie}@cs.cmu.edu

[†] School of Informatics
University of Edinburgh
10 Crichton Street
Edinburgh EH8 9AB, UK
pkoehn@inf.ed.ac.uk

Abstract

Approximate search algorithms, such as cube pruning in syntactic machine translation, rely on the language model to estimate probabilities of sentence fragments. We contribute two changes that trade between accuracy of these estimates and memory, holding sentence-level scores constant. Common practice uses lower-order entries in an N-gram model to score the first few words of a fragment; this violates assumptions made by common smoothing strategies, including Kneser-Ney. Instead, we use a unigram model to score the first word, a bigram for the second, etc. This improves search at the expense of memory. Conversely, we show how to save memory by collapsing probability and backoff into a single value without changing sentence-level scores, at the expense of less accurate estimates for sentence fragments. These changes can be stacked, achieving better estimates with unchanged memory usage. In order to interpret changes in search accuracy, we adjust the pop limit so that accuracy is unchanged and report the change in CPU time. In a German-English Moses system with target-side syntax, improved estimates yielded a 63% reduction in CPU time; for a Hiero-style version, the reduction is 21%. The compressed language model uses 26% less RAM while equivalent search quality takes 27% more CPU. Source code is released as part of KenLM.

1 Introduction

Language model storage is typically evaluated in terms of speed, space, and accuracy. We introduce a fourth dimension, rest cost quality, that captures how well the model scores sentence fragments for purposes of approximate search. Rest cost quality is distinct from accuracy in the sense that the score of a complete sentence is held constant. We first show how to improve rest cost quality over standard practice by using additional space. Then, conversely, we show how to compress the language model by making a pessimistic rest cost assumption[1].

Language models are designed to assign probability to sentences. However, approximate search algorithms use estimates for sentence fragments. If the language model has order N (an N-gram model), then the first $N-1$ words of the fragment have incomplete context and the last $N-1$ words have not been completely used as context. Our baseline is common practice (Koehn et al., 2007; Dyer et al., 2010; Li et al., 2009) that uses lower-order entries from the language model for the first words in the fragment and no rest cost adjustment for the last few words. Formally, the baseline estimate for sentence fragment w_1^k is

$$\left(\prod_{n=1}^{N-1} p_N(w_n|w_1^{n-1})\right)\left(\prod_{n=N}^{k} p_N(w_n|w_{n-N+1}^{n-1})\right)$$

where each w_n is a word and p_N is an N-gram language model.

The problem with the baseline estimate lies in lower order entries $p_N(w_n|w_1^{n-1})$. Commonly used Kneser-Ney (Kneser and Ney, 1995) smoothing,

[1] Here, the term rest cost means an adjustment to the score of a sentence fragment but not to whole sentences. The adjustment may be good or bad for approximate search.

including the modified version (Chen and Goodman, 1998), assumes that a lower-order entry will only be used because a longer match could not be found[2]. Formally, these entries actually evaluate $p_N(w_n|w_1^{n-1}, \text{did not find } w_0^n)$. For purposes of scoring sentence fragments, additional context is simply indeterminate, and the assumption may not hold.

As an example, we built 5-gram and unigram language models with Kneser-Ney smoothing on the same data. Sentence fragments frequently begin with "the". Using a lower-order entry from the 5-gram model, $\log_{10} p_5(\text{the}) = -2.49417$. The unigram model does not condition on backing off, assigning $\log_{10} p_1(\text{the}) = -1.28504$. Intuitively, the 5-gram model is surprised, by more than an order of magnitude, to see "the" without matching words that precede it.

To remedy the situation, we train N language models on the same data. Each model p_n is an n-gram model (it has order n). We then use p_n to score the nth word of a sentence fragment. Thus, a unigram model scores the first word of a sentence fragment, a bigram model scores the second word, and so on until either the n-gram is not present in the model or the first $N-1$ words have been scored. Storing probabilities from these models requires one additional value per n-gram in the model, except for N-grams where this probability is already stored.

Conversely, we can lower memory consumption relative to the baseline at the expense of poorer rest costs. Baseline models store two entries per n-gram: probability and backoff. We will show that the probability and backoff values in a language model can be collapsed into a single value for each n-gram without changing sentence probability. This transformation saves memory by halving the number of values stored per entry, but it makes rest cost estimates worse. Specifically, the rest cost pessimistically assumes that the model will back off to unigrams immediately following the sentence fragment.

The two modifications can be used independently or simultaneously. To measure the impact of their different rest costs, we experiment with cube pruning (Chiang, 2007) in syntactic machine translation. Cube pruning's goal is to find high-scoring sentence fragments for the root non-terminal in the parse tree. It does so by going bottom-up in the parse tree, searching for high-scoring sentence fragments for each non-terminal. Within each non-terminal, it generates a fixed number of high-scoring sentence fragments; this is known as the *pop limit*. Increasing the pop limit therefore makes search more accurate but costs more time. By moderating the pop limit, improved accuracy can be interpreted as a reduction in CPU time and vice-versa.

[2]Other smoothing techniques, including Witten-Bell (Witten and Bell, 1991), do not make this assumption.

2 Related Work

Vilar and Ney (2011) study several modifications to cube pruning and cube growing (Huang and Chiang, 2007). Most relevant is their use of a class-based language model for the first of two decoding passes. This first pass is cheaper because translation alternatives are likely to fall into the same class. Entries are scored with the maximum probability over class members (thereby making them no longer normalized). Thus, paths that score highly in this first pass may contain high-scoring paths under the lexicalized language model, so the second pass more fully explores these options. The rest cost estimates we describe here could be applied in both passes, so our work is largely orthogonal.

Zens and Ney (2008) present rest costs for phrase-based translation. These rest costs are based on factors external to the sentence fragment, namely output that the decoder may generate in the future. Our rest costs examine words internal to the sentence fragment, namely the first and last few words. We also differ by focusing on syntactic translation.

A wide variety of work has been done on language model compression. While data structure compression (Raj and Whittaker, 2003; Heafield, 2011) and randomized data structures (Talbot and Osborne, 2007; Guthrie and Hepple, 2010) are useful, here we are concerned solely with the values stored by these data structures. Quantization (Whittaker and Raj, 2001; Federico and Bertoldi, 2006) uses less bits to store each numerical value at the expense of model quality, including scores of full sentences, and is compatible with our approach. In fact, the lower-order probabilities might be quantized further than normal since these are used solely for rest cost

purposes. Our compression technique reduces storage from two values, probability and backoff, to one value, theoretically halving the bits per value (except N-grams which all have backoff 1). This makes the storage requirement for higher-quality modified Kneser-Ney smoothing comparable to stupid backoff (Brants et al., 2007). Whether to use one smoothing technique or the other then becomes largely an issue of training costs and quality after quantization.

3 Contribution

3.1 Better Rest Costs

As alluded to in the introduction, the first few words of a sentence fragment are typically scored using lower-order entries from an N-gram language model. However, Kneser-Ney smoothing (Kneser and Ney, 1995) conditions lower-order probabilities on backing off. Specifically, lower-order counts are adjusted to represent the number of unique extensions an n-gram has:

$$a(w_1^n) = \begin{cases} |\{w_0 : c(w_0^n) > 0\}| & \text{if } n < N \\ c(w_1^n) & \text{if } n = N \end{cases}$$

where $c(w_1^n)$ is the number of times w_1^n appears in the training data[3]. This adjustment is also performed for modified Kneser-Ney smoothing. The intuition is based on the fact that the language model will base its probability on the longest possible match. If an N-gram was seen in the training data, the model will match it fully and use the smoothed count. Otherwise, the full N-gram was not seen in the training data and the model resorts to a shorter n-gram match. Probability of this shorter match is based on how often the n-gram is seen in different contexts. Thus, these shorter n-gram probabilities are not representative of cases where context is short simply because additional context is unknown at the time of scoring.

In some cases, we are able to determine that the model will back off and therefore the lower-order probability makes the appropriate assumption. Specifically, if vw_1^n does not appear in the model for any word v, then computing $p(w_n|vw_1^{n-1})$ will always back off to w_1^{n-1} or fewer words[4]. This criterion is the same as used to minimize the length of left language model state (Li and Khudanpur, 2008) and can be retrieved for each n-gram without using additional memory in common data structures (Heafield et al., 2011).

Where it is unknown if the model will back off, we use a language model of the same order to produce a rest cost. Specifically, there are N language models, one of each order from 1 to N. The models are trained on the same corpus with the same smoothing parameters to the extent that they apply. We then compile these into one data structure where each n-gram record has three values:

1. Probability p_n from the n-gram language model
2. Probability p_N from the N-gram language model
3. Backoff b from the N-gram language model

For N-grams, the two probabilities are the same and backoff is always 1, so only one value is stored. Without pruning, the n-gram model contains the same n-grams as the N-gram model. With pruning, the two sets may be different, so we query the n-gram model in the normal way to score every n-gram in the N-gram model. The idea is that p_n is the average conditional probability that will be encountered once additional context becomes known. We also tried more complicated estimates by additionally interpolating upper bound, lower bound, and p_N with weights trained on cube pruning logs; none of these improved results in any meaningful way.

Formalizing the above, let w_1^k be a sentence fragment. Choose the largest s so that vw_1^s appears in the model for some v; equivalently w_1^s is the left state described in Li and Khudanpur (2008). The

[3]Counts are not modified for n-grams bound to the beginning of sentence, namely those with $w_1 = <s>$.

[4]Usually, this happens because w_1^n does not appear, though it can also happen that w_1^n appears but all vw_1^n were removed by pruning or filtering.

baseline estimate is

$$p_b(w_1^k) = \left(\prod_{n=1}^{s} p_N(w_n|w_1^{n-1})\right) \cdot \left(\prod_{n=s+1}^{N+1} p_N(w_n|w_1^{n-1})\right) \cdot \left(\prod_{n=N}^{k} p_N(w_n|w_{n-N+1}^{n-1})\right) \quad (1)$$

while our improved estimate is

$$p_r(w_1^k) = \left(\prod_{n=1}^{s} p_n(w_n|w_1^{n-1})\right) \cdot \left(\prod_{n=s+1}^{N+1} p_N(w_n|w_1^{n-1})\right) \cdot \left(\prod_{n=N}^{k} p_N(w_n|w_{n-N+1}^{n-1})\right) \quad (2)$$

The difference between these equations is that p_n is used for words in the left state i.e. $1 \leq n \leq s$. We have also abused notation by using p_N to denote both probabilities stored explicitly in the model and the model's backoff-smoothed probabilities when not present. It is not necessary to store backoffs for p_n because s was chosen such that all queried n-grams appear in the model.

This modification to the language model improves rest costs (and therefore quality or CPU time) at the expense of using more memory to store p_n. In the next section, we do the opposite: make rest costs worse to reduce storage size.

3.2 Less Memory

Many language model smoothing strategies, including modified Kneser-Ney smoothing, use the backoff algorithm shown in Figure 1. Given an n-gram w_1^n, the backoff algorithm bases probability on as much context as possible. Equivalently, it finds the minimum f so that w_f^n is in the model then uses $p(w_n|w_f^{n-1})$ as a basis. Backoff penalties b are charged because a longer match was not found, forming the product

$$p(w_n|w_1^{n-1}) = p(w_n|w_f^{n-1}) \prod_{j=1}^{f-1} b(w_j^{n-1}) \quad (3)$$

Notably, the backoff penalties $\{b(w_j^{n-1})\}_{j=1}^{n-1}$ are independent of w_n, though which backoff penalties are charged depends on f and therefore w_n.

backoff $\leftarrow 1$
for $f = 1 \rightarrow n$ **do**
  if w_f^n is in the model **then**
    return $p(w_n|w_f^{n-1}) \cdot$ backoff
  else
    if w_f^{n-1} is in the model **then**
      backoff $\leftarrow$ backoff $\cdot\, b(w_f^{n-1})$
    end if
  end if
end for

Figure 1: The baseline backoff algorithm to compute $p(w_n|w_1^{n-1})$. It always terminates with a probability because even unknown words are treated as a unigram.

for $f = 1 \rightarrow n$ **do**
  if w_f^n is in the model **then**
    return $q(w_n|w_f^{n-1})$
  end if
end for

Figure 2: The run-time pessimistic backoff algorithm to find $q(w_n|w_1^{n-1})$. It assumes that q has been computed at model building time.

In order to save memory, we propose to account for backoff in a different way, defining q

$$q(w_n|w_1^{n-1}) = \frac{p(w_n|w_f^{n-1}) \prod_{j=f}^{n} b(w_j^n)}{\prod_{j=f}^{n-1} b(w_j^{n-1})}$$

where again w_f^n is the longest matching entry in the model. The idea is that q is a term in the telescoping series that scores a sentence fragment, shown in equation (1) or (2). The numerator pessimistically charges all backoff penalties, as if the next word w_{n+1} will only match a unigram. When w_{n+1} is scored, the denominator of $q(w_{n+1}|w_1^n)$ cancels out backoff terms that were wrongly charged. Once these terms are canceled, all that is left is p, the correct backoff penalties, and terms on the edge of the series.

Proposition 1. *The terms of q telescope. Formally, let w_1^k be a sentence fragment and f take the minimum value so that w_f^k is in the model. Then,*

$$q(w_1^k) = p(w_1^k) \prod_{j=f}^{k} b(w_j^k)$$

Proof. By induction on k. When $k = 1$, $f = 1$ since the word w_1 is either in the vocabulary or mapped to <unk> and treated like a unigram.

$$q(w_1) = \frac{p(w_1) \prod_{j=1}^{1} b(w_j^1)}{\prod_{j=1}^{0} b(w_j^0)} = p(w_1)b(w_1)$$

For $k > 1$,

$$\begin{aligned} q(w_1^k) &= q(w_1^{k-1})q(w_k|w_1^{k-1}) \\ &= \frac{q(w_1^{k-1})p(w_k|w_f^{k-1}) \prod_{j=f}^{k} b(w_j^k)}{\prod_{j=f}^{k-1} b(w_j^{k-1})} \end{aligned}$$

where f has the lowest value such that w_f^k is in the model. Applying the inductive hypothesis to expand $q(w_1^{k-1})$, we obtain

$$\frac{p(w_1^{k-1}) \left(\prod_{j=e}^{k-1} b(w_j^{k-1}) \right) p(w_k|w_f^{k-1}) \prod_{j=f}^{k} b(w_j^k)}{\prod_{j=f}^{k-1} b(w_j^{k-1})}$$

where e has the lowest value such that w_e^{k-1} is in the model. The backoff terms cancel to yield

$$p(w_1^{k-1}) \left(\prod_{j=e}^{f-1} b(w_j^{k-1}) \right) p(w_k|w_f^{k-1}) \prod_{j=f}^{k} b(w_j^k)$$

By construction of e, w_j^{k-1} is not in the model for all $j < e$. Hence, $b(w_j^{k-1}) = 1$ implicitly for all $j < e$. Multiplying by 1,

$$p(w_1^{k-1}) \left(\prod_{j=1}^{f-1} b(w_j^{k-1}) \right) p(w_k|w_f^{k-1}) \prod_{j=f}^{k} b(w_j^k)$$

Recognizing the backoff equation (3) to simplify,

$$p(w_1^{k-1})p(w_k|w_1^{k-1}) \prod_{j=f}^{k} b(w_j^k)$$

Finally, the conditional probability folds as desired

$$q(w_1^k) = p(w_1^k) \prod_{j=f}^{k} b(w_j^k)$$

□

We note that entries ending in </s> have backoff 1, so it follows from Proposition 1 that sentence-level scores are unchanged.

$$q(\text{<s> } w_1^k \text{ </s>}) = p(\text{<s> } w_1^k \text{ </s>})$$

Proposition 1 characterizes q as a pessimistic rest cost on sentence fragments that scores sentences in exactly the same way as the baseline using p and b. To save memory, we simply store q in lieu of p and b. Compared with the baseline, this halves number of values from two to one float per n-gram, except N-grams that already have one value. The impact of this reduction is substantial, as seen in Section 4.3. Run-time scoring is also simplified as shown in Figure 2 since the language model locates the longest match w_f^n then returns the value $q(w_n|w_1^{n-1}) = q(w_n|w_f^{n-1})$ without any calculation or additional lookup. Baseline language models either retrieve backoffs values with additional lookups (Stolcke, 2002; Federico et al., 2008) or modify the decoder to annotate sentence fragments with backoff information (Heafield, 2011); we have effectively moved this step to preprocessing. The disadvantage is that q is not a proper probability and it produces worse rest costs than does the baseline.

Language models are actually applied at two points in syntactic machine translation: scoring lexical items in grammar rules and during cube pruning. Grammar scoring is an offline and embarrassingly parallel process where memory is not as tight (since the phrase table is streamed) and fewer queries are made, so slow non-lossy compression and even network-based sharding can be used. We therefore use an ordinary language model for grammar scoring and only apply the compressed model during cube pruning. Grammar scoring impacts grammar pruning (by selecting only top-scoring grammar rules) and the order in which rules are tried during cube pruning.

3.3 Combined Scheme

Our two language model modifications can be trivially combined by using lower-order probabilities on the left of a fragment and by charging all backoff penalties on the right of a fragment. The net result is a language model that uses the same memory as the baseline but has better rest cost estimates.

4 Experiments

To measure the impact of different rest costs, we use the Moses chart decoder (Koehn et al., 2007) for the WMT 2011 German-English translation task (Callison-Burch et al., 2011). Using the Moses pipeline, we trained two syntactic German-English systems, one with target-side syntax and the other hierarchical with unlabeled grammar rules (Chiang, 2007). Grammar rules were extracted from Europarl (Koehn, 2005) using the Collins parser (Collins, 1999) for syntax on the English side. The language model interpolates, on the WMT 2010 test set, separate models built on Europarl, news commentary, and the WMT news data for each year. Models were built and interpolated using SRILM (Stolcke, 2002) with modified Kneser-Ney smoothing (Kneser and Ney, 1995; Chen and Goodman, 1998) and the default pruning settings. In all scenarios, the primary language model has order 5. For lower-order rest costs, we also built models with orders 1 through 4 then used the n-gram model to score n-grams in the 5-gram model. Feature weights were trained with MERT (Och, 2003) on the baseline using a pop limit of 1000 and 100-best output. Since final feature values are unchanged, we did not re-run MERT in each condition. Measurements were collected by running the decoder on the 3003-sentence test set.

4.1 Rest Costs as Prediction

Scoring the first few words of a sentence fragment is a prediction task. The goal is to predict what the probability will be when more context becomes known. In order to measure performance on this task, we ran the decoder on the hierarchical system with a pop limit of 1000. Every time more context became known, we logged[5] the prediction error (estimated log probability minus updated log probability) for both lower-order rest costs and the baseline. Table 1 shows the results.

		Lower			Baseline		
n	Mean	Bias	MSE	Var	Bias	MSE	Var
1	-3.21	.10	.84	.83	-.12	.87	.86
2	-2.27	.04	.18	.17	-.14	.23	.24
3	-1.80	.02	.07	.07	-.09	.10	.09
4	-1.29	.01	.04	.04	-.10	.09	.08

Table 1: Bias (mean error), mean squared error, and variance (of the error) for the lower-order rest cost and the baseline. Error is the estimated log probability minus the final probability. Statistics were computed separately for the first word of a fragment ($n = 1$), the second word ($n = 2$), etc. The lower-order estimates are better across the board, reducing error in cube pruning. All numbers are in log base ten, as is standard for ARPA-format language models. Statistics were only collected for words with incomplete context.

Cube pruning uses relative scores, so bias matters less, though positive bias will favor rules with more arity. Variance matters the most because lower variance means cube pruning's relative rankings are more accurate. Our lower-order rest costs are better across the board in terms of absolute bias, mean squared error, and variance.

4.2 Pop Limit Trade-Offs

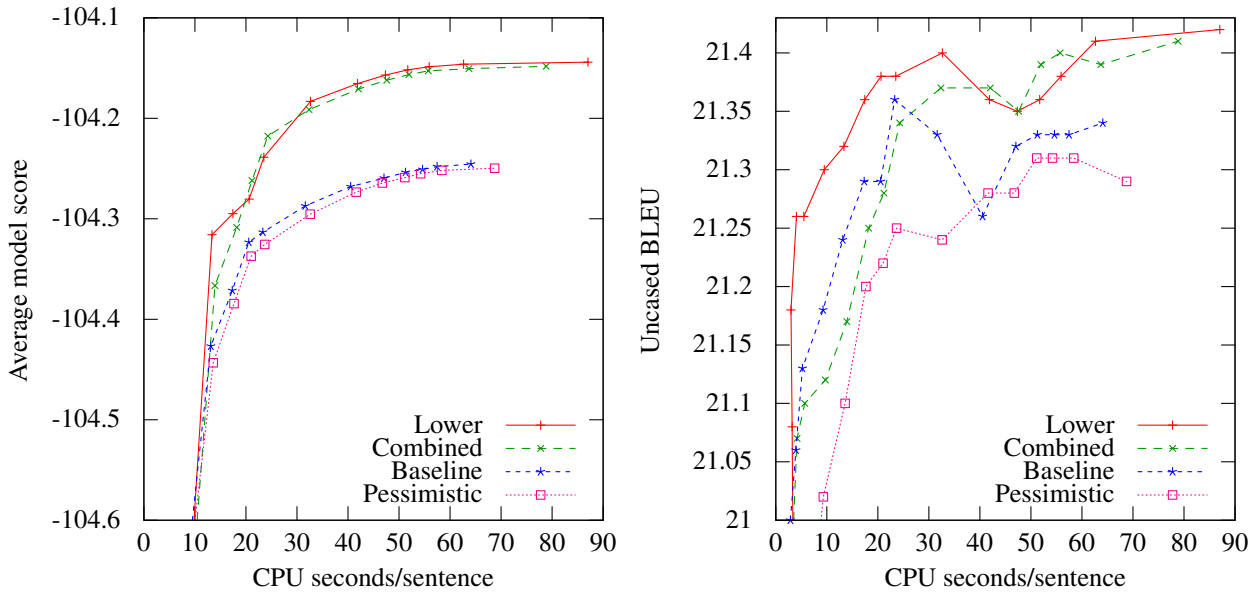
The cube pruning pop limit is a trade-off between search accuracy and CPU time. Here, we measure how our rest costs improve (or degrade) that trade-off. Search accuracy is measured by the average model score of single-best translations. Model scores are scale-invariant and include a large constant factor; higher is better. We also measure overall performance with uncased BLEU (Papineni et al., 2002). CPU time is the sum of user and system time used by Moses divided by the number of sentences (3003). Timing includes time to load, though files were forced into the disk cache in advance. Our test machine has 64 GB of RAM and 32 cores. Results are shown in Figures 3 and 4.

Lower-order rest costs perform better in both systems, reaching plateau model scores and BLEU with less CPU time. The gain is much larger for tar-

[5]Logging was only enabled for this experiment.

Pop	Baseline			Lower Order			Pessimistic			Combined		
	CPU	Model	BLEU	CPU	Model	BLEU	CPU	Model	BLEU	CPU	Model	BLEU
2	3.29	-105.56	20.45	3.68	-105.44	20.79	3.74	-105.62	20.01	3.18	-105.49	20.43
10	5.21	-104.74	21.13	5.50	-104.72	21.26	5.43	-104.77	20.85	5.67	-104.75	21.10
50	23.30	-104.31	21.36	23.51	-104.24	21.38	23.68	-104.33	21.25	24.29	-104.22	21.34
500	54.61	-104.25	21.33	55.92	-104.15	21.38	54.23	-104.26	21.31	55.74	-104.15	21.40
700	64.08	-104.25	21.34	87.02	-104.14	21.42	68.74	-104.25	21.29	78.84	-104.15	21.41

(a) Numerical results for select pop limits.



(b) Model and BLEU scores near the plateau.

Figure 3: Target-syntax performance. CPU time and model score are averaged over 3003 sentences.

get syntax, where a pop limit of 50 outperforms the baseline with pop limit 700. CPU time per sentence is reduced to 23.5 seconds from 64.0 seconds, a 63.3% reduction. The combined setting, using the same memory as the baseline, shows a similar 62.1% reduction in CPU time. We attribute this difference to improved grammar rule scoring that impacts pruning and sorting. In the target syntax model, the grammar is not saturated (i.e. less pruning will still improve scores) but we nonetheless prune for tractability reasons. The lower-order rest costs are particularly useful for grammar pruning because lexical items are typically less than five words long (and frequently only word).

The hierarchical grammar is nearly saturated with respect to grammar pruning, so improvement there is due mostly to better search. In the hierarchical system, peak BLEU 22.34 is achieved under the lower-order condition with pop limits 50 and 200, while other scenarios are still climbing to the plateau. With a pop limit of 1000, the baseline's average model score is -101.3867. Better average models scores are obtained from the lower-order model with pop limit 690 using 79% of baseline CPU, the combined model with pop limit 900 using 97% CPU, and the pessimistic model with pop limit 1350 using 127% CPU.

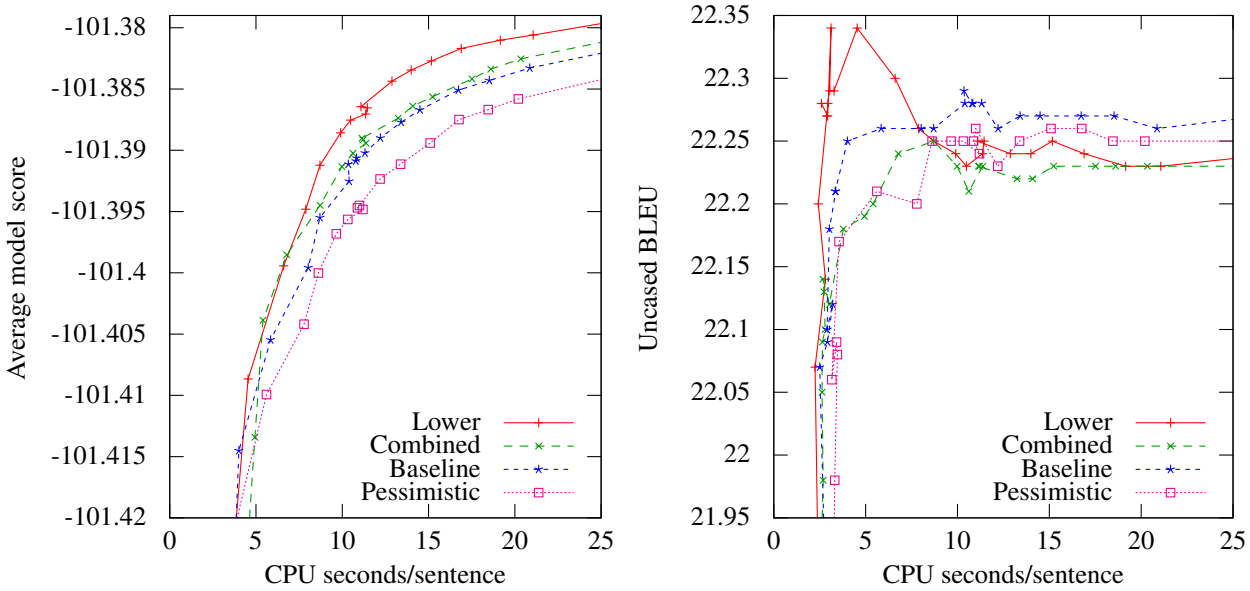
Pessimistic compression does worsen search, requiring 27% more CPU in the hierarchical system to achieve the same quality. This is worthwhile to fit large-scale language models in memory, especially if the alternative is a remote language model.

4.3 Memory Usage

Our rest costs add a value (for lower-order probabilities) or remove a value (pessimistic compression) for each n-gram except those of highest order ($n = N$). The combined condition adds one value

Pop	Baseline CPU	Baseline Model	Baseline BLEU	Lower Order CPU	Lower Order Model	Lower Order BLEU	Pessimistic CPU	Pessimistic Model	Pessimistic BLEU	Combined CPU	Combined Model	Combined BLEU
2	2.96	-101.85	21.19	2.44	-101.80	21.63	2.71	-101.90	20.85	3.05	-101.84	21.37
10	2.80	-101.60	21.90	2.42	-101.58	22.20	2.95	-101.63	21.74	2.69	-101.60	21.98
50	3.02	-101.47	22.18	3.11	-101.46	22.34	3.46	-101.48	22.08	2.67	-101.47	22.14
690	10.83	-101.39	22.28	11.45	-101.39	22.25	10.88	-101.40	22.25	11.19	-101.39	22.23
900	13.41	-101.39	22.27	14.00	-101.38	22.24	13.38	-101.39	22.25	14.09	-101.39	22.22
1000	14.50	-101.39	22.27	15.17	-101.38	22.25	15.09	-101.39	22.26	15.23	-101.39	22.23
1350	18.52	-101.38	22.27	19.16	-101.38	22.23	18.46	-101.39	22.25	18.61	-101.38	22.23
5000	59.67	-101.38	22.24	61.41	-101.38	22.22	59.76	-101.38	22.27	61.38	-101.38	22.22

(a) Numerical results for select pop limits.



(b) Model and BLEU scores near the plateau.

Figure 4: Hierarchical system performance. All values are averaged over 3003 sentences.

and removes another, so it uses the same memory as the baseline. The memory footprint of adding or removing a value depends on the number of such n-grams, the underlying data structure, and the extent of quantization. Our test language model has 135 million n-grams for $n < 5$ and 56 million 5-grams. Memory usage was measured for KenLM data structures (Heafield, 2011) and minimal perfect hashing (Guthrie and Hepple, 2010). For minimal perfect hashing, we assume the Compress, Hash and Displace algorithm (Belazzougui et al., 2008) with 8-bit signatures and 8-bit quantization. Table 2 shows the results. Storage size of the smallest model is reduced by 26%, bringing higher-quality smoothed models in line with stupid backoff models that also store one value per n-gram.

Structure	Baseline	Change	%
Probing	4,072	517	13%
Trie	2,647	506	19%
8-bit quantized trie	1,236	140	11%
8-bit minimal perfect hash	540	140	26%

Table 2: Size in megabytes of our language model, excluding operating system overhead. Change is the cost of adding an additional value to store lower-order probabilities. Equivalently, it is the savings from pessimistic compression.

5 Conclusion

Our techniques reach plateau-level BLEU scores with less time or less memory. Efficiently storing lower-order probabilities and using them as rest costs improves both cube pruning (21% CPU reduction in a hierarchical system) and model filtering (net 63% CPU time reduction with target syntax) at the expense of 13-26% more RAM for the language model. This model filtering improvement is surprising both in the impact relative to changing the pop limit and simplicity of implementation, since it can be done offline. Compressing the language model to halve the number of values per n-gram (except N-grams) results in a 13-26% reduction in RAM with 26% over the smallest model, costing 27% more CPU and leaving overall sentence scores unchanged. This compression technique is likely to have more general application outside of machine translation, especially where only sentence-level scores are required. Source code is being released[6] under the LGPL as part of KenLM (Heafield, 2011).

Acknowledgements

This work was supported by the National Science Foundation under grants DGE-0750271, IIS-0713402, and IIS-0915327; by the EuroMatrixPlus project funded by the European Commission (7th Framework Programme), and by the DARPA GALE program. Benchmarks were run on Trestles at the San Diego Supercomputer Center under allocation TG-CCR110017. Trestles is part of the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

References

Djamal Belazzougui, Fabiano C. Botelho, and Martin Dietzfelbinger. 2008. Hash, displace, and compress. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming (ICALP '08)*, pages 385–396.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Language Learning*, pages 858–867, June.

Chris Callison-Burch, Philipp Koehn, Christof Monz, and Omar Zaidan. 2011. Findings of the 2011 workshop on statistical machine translation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 22–64, Edinburgh, Scotland, July. Association for Computational Linguistics.

Stanley Chen and Joshua Goodman. 1998. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University, August.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33:201–228, June.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Chris Dyer, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. 2010. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, ACLDemos '10, pages 7–12.

Marcello Federico and Nicola Bertoldi. 2006. How many bits are needed to store probabilities for phrase-based translation? In *Proceedings of the Workshop on Statistical Machine Translation*, pages 94–101, New York City, June.

Marcello Federico, Nicola Bertoldi, and Mauro Cettolo. 2008. IRSTLM: an open source toolkit for handling large scale language models. In *Proceedings of Interspeech*, Brisbane, Australia.

David Guthrie and Mark Hepple. 2010. Storing the web in memory: Space efficient language models with constant time retrieval. In *Proceedings of EMNLP 2010*, Los Angeles, CA.

Kenneth Heafield, Hieu Hoang, Philipp Koehn, Tetsuo Kiso, and Marcello Federico. 2011. Left language model state for syntactic machine translation. In *Proceedings of the International Workshop on Spoken Language Translation*, San Francisco, CA, USA, December.

Kenneth Heafield. 2011. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, Edinburgh, UK, July. Association for Computational Linguistics.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, Prague, Czech Republic.

[6] http://kheafield.com/code/kenlm/

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 181–184.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, June.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of MT Summit*.

Zhifei Li and Sanjeev Khudanpur. 2008. 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)*, pages 10–18, Columbus, Ohio, June.

Zhifei Li, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan. 2009. Joshua: An open source toolkit for parsing-based machine translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 135–139, Athens, Greece, March. Association for Computational Linguistics.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 160–167, Morristown, NJ, USA. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evalution of machine translation. In *Proceedings 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, PA, July.

Bhiksha Raj and Ed Whittaker. 2003. Lossless compression of language model structure and word identifiers. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 388–391.

Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *Proceedings of the Seventh International Conference on Spoken Language Processing*, pages 901–904.

David Talbot and Miles Osborne. 2007. Randomised language modelling for statistical machine translation. In *Proceedings of ACL*, pages 512–519, Prague, Czech Republic.

David Vilar and Hermann Ney. 2011. Cardinality pruning and language model heuristics for hierarchical phrase-based translation. *Machine Translation*, pages 1–38, November. DOI 10.1007/s10590-011-9119-4.

Ed Whittaker and Bhiksha Raj. 2001. Quantization-based language model compression. In *Proceedings of EUROSPEECH*, pages 33–36, September.

Ian H. Witten and Timothy C. Bell. 1991. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094.

Richard Zens and Hermann Ney. 2008. Improvements in dynamic programming beam search for phrase-based statistical machine translation. In *Proceedings of the International Workshop on Spoken Language Translation (IWSLT)*, Honolulu, Hawaii, October.