Faster Phrase-Based Decoding by Refining Feature State

Kenneth Heafield **Michael Kayser** **Christopher D. Manning**
Computer Science Department Stanford University, Stanford, CA, 94305
{heafield,mkayser,manning}@stanford.edu

Abstract

We contribute a faster decoding algorithm for phrase-based machine translation. Translation hypotheses keep track of state, such as context for the language model and coverage of words in the source sentence. Most features depend upon only part of the state, but traditional algorithms, including cube pruning, handle state atomically. For example, cube pruning will repeatedly query the language model with hypotheses that differ only in source coverage, despite the fact that source coverage is irrelevant to the language model. Our key contribution avoids this behavior by placing hypotheses into equivalence classes, masking the parts of state that matter least to the score. Moreover, we exploit shared words in hypotheses to iteratively refine language model scores rather than handling language model state atomically. Since our algorithm and cube pruning are both approximate, improvement can be used to increase speed or accuracy. When tuned to attain the same accuracy, our algorithm is 4.0–7.7 times as fast as the Moses decoder with cube pruning.

1 Introduction

Translation speed is critical to making suggestions as translators type, mining for parallel data by translating the web, and running on mobile devices without Internet connectivity. We contribute a fast decoding algorithm for phrase-based machine translation along with an implementation in a new open-source (LGPL) decoder available at `http://kheafield.com/code/`.

Phrase-based decoders (Koehn et al., 2007; Cer et al., 2010; Wuebker et al., 2012) keep track of several types of state with translation hypotheses: coverage of the source sentence thus far, context for the language model, the last position for the distortion model, and anything else features need. Existing decoders handle state atomically: hypotheses that have exactly the same state can be recombined and efficiently handled via dynamic programming, but there is no special handling for partial agreement. Therefore, features are repeatedly consulted regarding hypotheses that differ only in ways irrelevant to their score, such as coverage of the source sentence. Our decoder bundles hypotheses into equivalence classes so that features can focus on the relevant parts of state.

We pay particular attention to the language model because it is responsible for much of the hypothesis state. As the decoder builds translations from left to right (Koehn, 2004), it records the last $N-1$ words of each hypothesis so that they can be used as context to score the first $N-1$ words of a phrase, where N is the order of the language model. Traditional decoders (Huang and Chiang, 2007) try thousands of combinations of hypotheses and phrases, hoping to find ones that the language model likes. Our algorithm instead discovers good combinations in a coarse-to-fine manner. The algorithm exploits the fact that hypotheses often share the same suffix and phrases often share the same prefix. These shared suffixes and prefixes allow the algorithm to coarsely reason over many combinations at once.

Our primary contribution is a new search algorithm that exploits the above observations, namely that state can be divided into pieces relevant to each feature and that language model state can be further subdivided. The primary claim is that our algorithm is faster and more accurate than the popular cube pruning algorithm.

2 Related Work

Our previous work (Heafield et al., 2013) developed language model state refinement for bottom-up decoding in syntatic machine translation. In bottom-up decoding, hypotheses can be extended to the left or right, so hypotheses keep track of both their prefix and suffix. The present phrase-based setting is simpler because sentences are constructed from left to right, so prefix information is unnecessary. However, phrase-based translation implements reordering by allowing hypotheses that translate discontiguous words in the source sentence. There are exponentially many ways to cover the source sentence and hypotheses carry this information as additional state. A main contribution in this paper is efficiently ignoring coverage when evaluating the language model. In contrast, syntactic machine translation hypotheses correspond to contiguous spans in the source sentence, so in prior work we simply ran the search algorithm in every span.

Another improvement upon Heafield et al. (2013) is that we previously made no effort to exploit common words that appear in translation rules, which are analogous to phrases. In this work, we explicitly group target phrases by common prefixes, doing so directly in the phrase table.

Coarse-to-fine approaches (Petrov et al., 2008; Zhang and Gildea, 2008) invoke the decoder multiple times with increasingly detailed models, pruning after each pass. The key difference in our work is that, rather than refining models in lock step, we effectively refine the language model on demand for hypotheses that score well. Moreover, their work was performed in syntactic machine translation while we address issues specific to phrase-based translation.

Our baseline is cube pruning (Chiang, 2007; Huang and Chiang, 2007), which is both a way to organize search and an algorithm to search through cross products of sets. We adopt the same search organization (Section 3.1) but change how cross products are searched.

Chang and Collins (2011) developed an exact decoding algorithm based on Lagrangian relaxation. However, it has not been shown to tractably scale to 5-gram language models used by many modern translation systems.

3 Decoding

We begin by summarizing the high-level organization of phrase-based cube pruning (Koehn, 2004; Koehn et al., 2007; Huang and Chiang, 2007). Sections 3.2 and later show our contribution.

0 word	1 word	2 words	3 words
○○○	●○○ the	●●○ cat	●●● cat .
	○●○ cat	●●○ the cat	●●● the cat .
	○○● .	●●○ cat the	●●● cat the .
		●○● . the	●●● . the cat

Figure 1: Stacks to translate the French "le chat ." into English. Filled circles indicate that the source word has been translated. A phrase translates "le chat" as simply "cat", emphasizing that stacks are organized by the number of source words rather than the number of target words.

3.1 Search Organization

Phrase-based decoders construct hypotheses from left to right by appending phrases in the target language. The decoder organizes this search process using *stacks* (Figure 1). Stacks contain hypotheses that have translated the same number of source words. The zeroth stack contains one hypothesis with nothing translated. Subsequent stacks are built by extending hypotheses in preceding stacks. For example, the second stack contains hypotheses that translated two source words either separately or as a phrasal unit. Returning to Figure 1, the decoder can apply a phrase pair to translate "le chat" as "cat" or it can derive "the cat" by translating one word at a time; both appear in the second stack because they translate two source words. To generalize, the decoder populates the ith stack by pairing hypotheses in the $i - j$th stack with target phrases that translate source phrases of length j. Hypotheses remember which source word they translated, as indicated by the filled circles.

The reordering limit prevents hypotheses from jumping around the source sentence too much and dramatically reduces the search space. Formally, the decoder cannot propose translations that would require jumping back more than R words in the source sentence, including multiple small jumps.

In practice, stacks are limited to k hypotheses, where k is set by the user. Small k is faster but may prune good hypotheses, while large k is slower but more thorough, thereby comprising a time-accuracy trade-off. The central question in this paper is how to select these k hypotheses.

Populating a stack boils down to two steps. First, the decoder matches hypotheses with source phrases subject to three constraints: the total source length matches the stack being populated, none of the source words has already been trans-

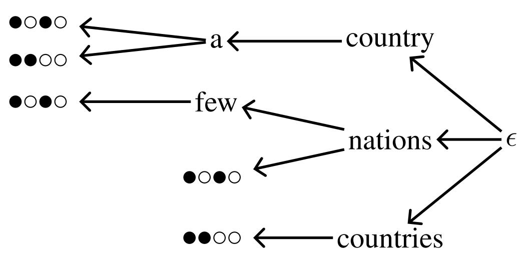

Figure 2: Hypothesis suffixes arranged into a trie. The leaves indicate source coverage and any other hypothesis state.

lated, and the reordering limit. Second, the decoder searches through these matches to select k high-scoring hypotheses for placement in the stack. We improve this second step.

The decoder provides our algorithm with pairs consisting of a hypothesis and a compatible source phrase. Each source phrase translates to multiple target phrases. The task is to grow these hypotheses by appending a target phrase, yielding new hypotheses. These new hypotheses will be placed into a stack of size k, so we are interested in selecting k new hypotheses that score highly.

Beam search (Lowerre, 1976; Koehn, 2004) tries every hypothesis with every compatible target phrase then selects the top k new hypotheses by score. This is wasteful because most hypotheses are discarded. Instead, we follow cube pruning (Chiang, 2007) in using a priority queue to generate k hypotheses. A key difference is that we generate these hypotheses iteratively.

3.2 Tries

For each source phrase, we collect the set of compatible hypotheses. We then place these hypotheses in a trie that emphasizes the suffix words because these matter most when appending a target phrase. Figure 2 shows an example. While it suffices to build this trie on the last $N-1$ words that matter to the language model, Li and Khudanpur (2008) have identified cases where fewer words are necessary because the language model will back off. The leaves of the trie are complete hypotheses and reveal information irrelevant to the language model, such as coverage of the source sentence and the state of other features.

Each source phrase translates to a set of target phrases. Because these phrases will be appended to a hypothesis, the first few words matter the most to the language model. We therefore arrange the target phrases into a prefix trie. An example is shown in Figure 3. Similar to the hypothesis trie, the depth may be shorter than $N-1$ in cases where the language model will provably back off (Li and Khudanpur, 2008). The trie can also be short because the target phrase has fewer than $N-1$ words. We currently store this trie data structure directly in the phrase table, though it could also be computed on demand to save memory. Empirically, our phrase table uses less RAM than Moses's memory-based phrase table.

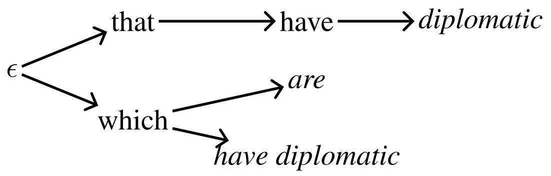

Figure 3: Target phrases arranged into a trie. Set in *italic*, leaves reveal parts of the phrase that are irrelevant to the language model.

As an optimization, a trie reveals multiple words when there would otherwise be no branching. This allows the search algorithm to make decisions only when needed.

Following Heafield et al. (2013), leaves in the trie take the score of the underlying hypothesis or target phrase. Non-leaf nodes take the maximum score of their descendants. Children of a node are sorted by score.

3.3 Boundary Pairs

The idea is that the decoder reasons over pairs of nodes in the hypothesis and phrase tries before devling into detail. In this way, it can determine what the language model likes and, conversely, quickly discard combinations that the model does not like.

A boundary pair consists of a node in the hypothesis trie and a node in the target phrase trie. For example, the decoder starts at the root of each trie with the boundary pair (ϵ, ϵ). The score of a boundary pair is the sum of the scores of the underlying trie nodes. However, once some words have been revealed, the decoder calls the language model to compute a score adjustment. For example, the boundary pair (country, that) has score adjustment

$$\log \frac{p(\text{that} \mid \text{country})}{p(\text{that})}$$

times the weight of the language model. This has the effect of cancelling out the estimate made when the phrase was scored in isolation, replacing it with a more accurate estimate based on available context. These score adjustments are efficient to compute because the decoder retained a pointer to "that" in the language model's data structure (Heafield et al., 2011).

3.4 Splitting

Refinement is the notion that the boundary pair (ϵ, ϵ) divides into several boundary pairs that reveal specific words from hypotheses or target phrases. The most straightforward way to do this is simply to split into all children of a trie node. Continuing the example from Figure 2, we could split (ϵ, ϵ) into three boundary pairs: $(\text{country}, \epsilon)$, $(\text{nations}, \epsilon)$, and $(\text{countries}, \epsilon)$. However, it is somewhat inefficient to separately consider the low-scoring child $(\text{countries}, \epsilon)$. Instead, we continue to split off the best child $(\text{country}, \epsilon)$ and leave a note that the zeroth child has been split off, denoted $(\epsilon[1^+], \epsilon)$. The index increases each time a child is split off.

The the boundary pair $(\epsilon[1^+], \epsilon)$ no longer counts $(\text{country}, \epsilon)$ as a child, so its score is lower.

Splitting alternates sides. For example, $(\text{country}, \epsilon)$ splits into $(\text{country}, \text{that})$ and $(\text{country}, \epsilon[1^+])$. If one side has completely revealed words that matter to the language model, then splitting continues with the other side. This procedure ensures that the language model score is completely resolved before considering irrelevant differences, such as coverage of the source sentence.

3.5 Priority Queue

Search proceeds in a best-first fashion controlled by a priority queue. For each source phrase, we convert the compatible hypotheses into a trie. The target phrases were already converted into a trie when the phrase table was loaded. We then push the root (ϵ, ϵ) boundary pair into the priority queue. We do this for all source phrases under consideration, putting their root boundary pairs into the same priority queue. The algorithm then loops by popping the top boundary pair. It the top boundary pair uniquely describes a hypothesis and target phrase, then remaining features are evaluated and the new hypothesis is output to the decoder's stack. Otherwise, the algorithm splits the boundary pair and pushes both split versions. Iteration continues until k new hypotheses have been found.

3.6 Overall Algorithm

We build hypotheses from left-to-right and manage stacks just like cube pruning. The only difference is how the k elements of these stacks are selected.

When the decoder matches a hypothesis with a compatible source phrase, we immediately evaluate the distortion feature and update future costs, both of which are independent of the target phrase. Our future costs are exactly the same as those used in Moses (Koehn et al., 2007): the highest-scoring way to cover the rest of the source sentence. This includes the language model score within target phrases but ignores the change in language model score that would occur were these phrases to be appended together. The hypotheses compatible with each source phrase are arranged into a trie. Finally, the priority queue algorithm from the preceding section searches for options that the language model likes.

4 Experiments

The primary claim is that our algorithm performs better than cube pruning in terms of the trade-off between time and accuracy. We compare our new decoder implementation with Moses (Koehn et al., 2007) by translating 1677 sentences from Chinese to English. These sentences are a deduplicated subset of the NIST Open MT 2012 test set and were drawn from Chinese online text sources, such as discussion forums. We trained our phrase table using a bitext of 10.8 million sentence pairs, which after tokenization amounts to approximately 290 million words on the English side. The bitext contains data from several sources, including news articles, UN proceedings, Hong Kong government documents, online forum data, and specialized sources such as an idiom translation table. We also trained our language model on the English half of this bitext using unpruned interpolated modified Kneser-Ney smoothing (Kneser and Ney, 1995; Chen and Goodman, 1998).

The system has standard phrase table, length, distortion, and language model features. We plan to implement lexicalized reordering in future work; without this, the test system is 0.53 BLEU (Papineni et al., 2002) point behind a state-of-the-art system. We set the reordering limit to $R = 15$. The phrase table was pre-pruned by applying the same heuristic as Moses: select the top 20 target phrases by score, including the language model.

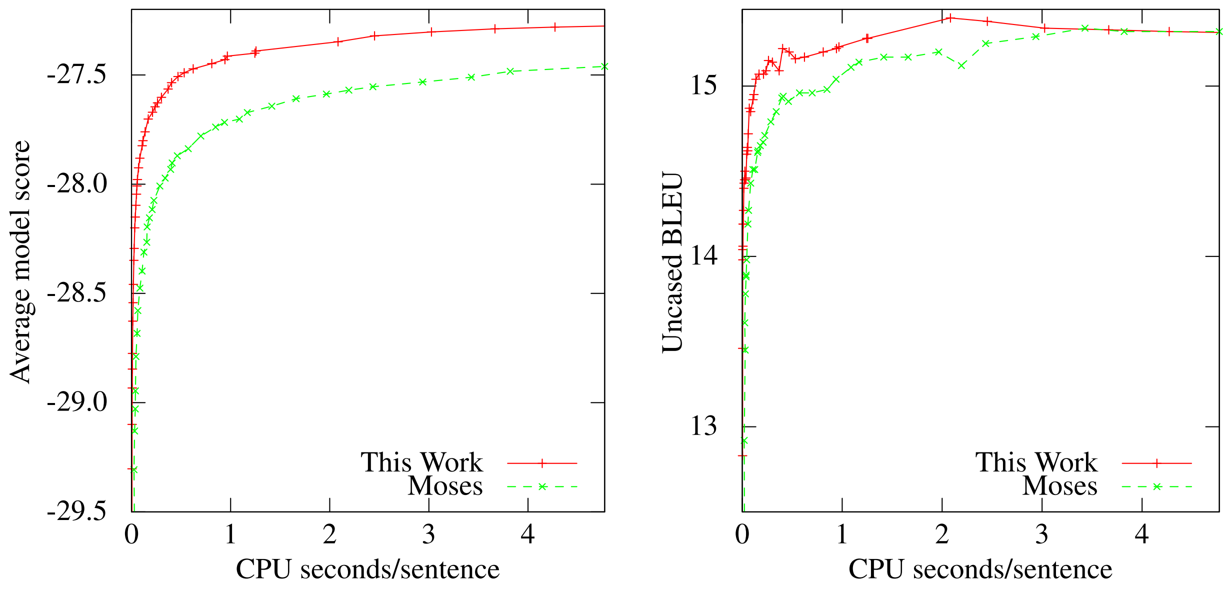

Figure 4: Performance of our decoder and Moses for various stack sizes k.

Moses (Koehn et al., 2007) revision d6df825 was compiled with all optimizations recommended in the documentation. We use the in-memory phrase table for speed. Tests were run on otherwise-idle identical machines with 32 GB RAM; the processes did not come close to running out of memory. The language model was compiled into KenLM probing format (Heafield, 2011) and placed in RAM while text phrase tables were forced into the disk cache before each run. Timing is based on CPU usage (user plus system) minus loading time, as measured by running on empty input; our decoder is also faster at loading. All results are single-threaded. Model score is comparable across decoders and averaged over all 1677 sentences; higher is better. The relationship between model score and uncased BLEU (Papineni et al., 2002) is noisy, so peak BLEU is not attained by the highest search accuracy.

Figure 4 shows the results for pop limits k ranging from 5 to 10000 while Table 1 shows select results. For Moses, we also set the stack size to k to disable a second pruning pass, as is common. Because Moses is slower, we also ran our decoder with higher beam sizes to fill in the graph. Our decoder is more accurate, but mostly faster. We can interpret accuracy improvments as speed improvements by asking how much time is required to attain the same accuracy as the baseline. By this metric, our decoder is 4.0 to 7.7 times as fast as Moses, depending on k.

	Model		**CPU**		**BLEU**	
Stack	**Moses**	**This**	**Moses**	**This**	**Moses**	**This**
10	-29.96	-29.70	0.019	0.004	12.92	13.46
100	-28.68	-28.54	0.057	0.016	14.19	14.40
1000	-27.87	-27.80	0.463	0.116	14.91	14.95
10000	-27.46	-27.39	4.773	1.256	15.32	15.28

Table 1: Results for select stack sizes k.

5 Conclusion

We have contributed a new phrase-based search algorithm based on the principle that the language model cares the most about boundary words. This leads to two contributions: hiding irrelevant state from features and an incremental refinement algorithm to find high-scoring combinations. This algorithm is implemented in a new fast phrase-based decoder, which we release as open-source under the LGPL at `kheafield.com/code/`.

Acknowledgements

This work was supported by the Defense Advanced Research Projects Agency (DARPA) Broad Operational Language Translation (BOLT) program through IBM. This work used Stampede provided by the Texas Advanced Computing Center (TACC) at The University of Texas at Austin under XSEDE allocation TG-CCR140009. XSEDE is supported by NSF grant number OCI-1053575. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of DARPA or the US government.

References

Daniel Cer, Michel Galley, Daniel Jurafsky, and Christopher D. Manning. 2010. Phrasal: A statistical machine translation toolkit for exploring new model features. In *Proceedings of the NAACL HLT 2010 Demonstration Session*, pages 9–12, Los Angeles, California, June. Association for Computational Linguistics.

Yin-Wen Chang and Michael Collins. 2011. Exact decoding of phrase-based translation models through lagrangian relaxation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, Edinburgh, Scotland, UK, 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.

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, Philipp Koehn, and Alon Lavie. 2013. Grouping language model boundary words to speed k-best extraction from hypergraphs. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Atlanta, Georgia, USA, June.

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 ACL*, Prague, Czech Republic, June.

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, Ondřej 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. 2004. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Machine translation: From real users to research*, pages 115–124. Springer, September.

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.

Bruce T. Lowerre. 1976. *The Harpy speech recognition system*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

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.

Slav Petrov, Aria Haghighi, and Dan Klein. 2008. Coarse-to-fine syntactic machine translation using language projections. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 108–116, Honolulu, HI, USA, October.

Joern Wuebker, Matthias Huck, Stephan Peitz, Malte Nuhn, Markus Freitag, Jan-Thorsten Peter, Saab Mansour, and Hermann Ney. 2012. Jane 2: Open source phrase-based and hierarchical statistical machine translation. In *Proceedings of COLING 2012: Demonstration Papers*, pages 483–492, Mumbai, India, December.

Hao Zhang and Daniel Gildea. 2008. Efficient multipass decoding for synchronous context free grammars. In *Proceedings of ACL-08: HLT*, pages 209–217, Columbus, Ohio.