Grouping Language Model Boundary Words to Speed K–Best Extraction from Hypergraphs

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

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

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

Abstract

We propose a new algorithm to approximately extract top-scoring hypotheses from a hypergraph when the score includes an N–gram language model. In the popular cube pruning algorithm, every hypothesis is annotated with boundary words and permitted to recombine only if all boundary words are equal. However, many hypotheses share some, but not all, boundary words. We use these common boundary words to group hypotheses and do so recursively, resulting in a tree of hypotheses. This tree forms the basis for our new search algorithm that iteratively refines groups of boundary words on demand. Machine translation experiments show our algorithm makes translation 1.50 to 3.51 times as fast as with cube pruning in common cases.

1 Introduction

This work presents a new algorithm to search a packed data structure for high-scoring hypotheses when the score includes an N–gram language model. Many natural language processing systems have this sort of problem e.g. hypergraph search in hierarchical and syntactic machine translation (Mi et al., 2008; Klein and Manning, 2001), lattice rescoring in speech recognition, and confusion network decoding in optical character recognition (Tong and Evans, 1996). Large language models have been shown to improve quality, especially in machine translation (Brants et al., 2007; Koehn and Haddow, 2012). However, language models make search computationally expensive because they examine surface words without regard to the structure of the packed search space. Prior work, including cube pruning (Chiang, 2007), has largely treated the language model as a black box. Our new search algorithm groups hypotheses by common prefixes and suffixes, exploiting the tendency of the language model to score these hypotheses similarly. An example is shown in Figure 1. The result is a substantial improvement over the time-accuracy trade-off presented by cube pruning.

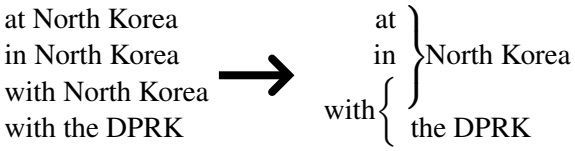

Figure 1: Hypotheses are grouped by common prefixes and suffixes.

The search spaces mentioned in the previous paragraph are special cases of a directed acyclic hypergraph. As used here, the difference from a normal graph is that an edge can go from one vertex to any number of vertices; this number is the *arity* of the edge. Lattices and confusion networks are hypergraphs in which every edge happens to have arity one. We experiment with parsing-based machine translation, where edges represent grammar rules that may have any number of non-terminals, including zero.

Hypotheses are paths in the hypergraph scored by a linear combination of features. Many features are *additive*: they can be expressed as weights on edges that sum to form hypothesis features. However, log probability from an N–gram language model is non-

additive because it examines surface strings across edge and vertex boundaries. Non-additivity makes search difficult because locally optimal hypotheses may not be globally optimal.

In order to properly compute the language model score, each hypothesis is annotated with its boundary words, collectively referred to as its *state* (Li and Khudanpur, 2008). Hypotheses with equal state may be recombined, so a straightforward dynamic programming approach (Bar-Hillel et al., 1964) simply treats state as an additional dimension in the dynamic programming table. However, this approach quickly becomes intractable for large language models where the number of states is too large.

Beam search (Chiang, 2005; Lowerre, 1976) approximates the straightforward algorithm by remembering a *beam* of up to k hypotheses[1] in each vertex. It visits each vertex in bottom-up order, each time calling a *beam filling algorithm* to select k hypotheses. The parameter k is a time-accuracy trade-off: larger k increases both CPU time and accuracy.

We contribute a new beam filling algorithm that improves the time-accuracy trade-off over the popular cube pruning algorithm (Chiang, 2007) discussed in §2.3. The algorithm is based on the observation that competing hypotheses come from the same imput, so their language model states are often similar. Grouping hypotheses by these similar words enables our algorithm to reason over multiple hypotheses at once. The algorithm is fully described in §3.

2 Related Work

2.1 Alternatives to Bottom-Up Search

Beam search visits each vertex in the hypergraph in bottom-up (topological) order. The hypergraph can also be searched in left-to-right order (Watanabe et al., 2006; Huang and Mi, 2010). Alternatively, hypotheses can be generated on demand with cube growing (Huang and Chiang, 2007), though we note that it showed little improvement in Moses (Xu and Koehn, 2012). All of these options are compatible with our algorithm. However, we only experiment with bottom-up beam search.

[1]We use K to denote the number of fully-formed hypotheses requested by the user and k to denote beam size.

2.2 Exhaustive Beam Filling

Originally, beam search was used with an exhaustive beam filling algorithm (Chiang, 2005). It generates every possible hypothesis (subject to the beams in previous vertices), selects the top k by score, and discards the remaining hypotheses. This is expensive: just one edge of arity a encodes $O(1 + a^k)$ hypotheses and each edge is evaluated exhaustively. In the worst case, our algorithm is exhaustive and generates the same number of hypotheses as beam search; in practice, we are concerned with the average case.

2.3 Baseline: Cube Pruning

Cube pruning (Chiang, 2007) is a fast approximate beam filling algorithm and our baseline. It chooses k hypotheses by popping them off the top of a priority queue. Initially, the queue is populated with hypotheses made from the best (highest-scoring) parts. These parts are an edge and a hypothesis from each vertex referenced by the edge. When a hypothesis is popped, several next-best alternatives are pushed. These alternatives substitute the next-best edge or a next-best hypothesis from one of the vertices.

Our work follows a similar pattern of popping one queue entry then pushing multiple entries. However, our queue entries are a group of hypotheses while cube pruning's entries are a single hypothesis.

Hypotheses are usually fully scored before being placed in the priority queue. An alternative prioritizes hypotheses by their *additive score*. The additive score is the edge's score plus the score of each component hypothesis, ignoring the non-additive aspect of the language model. When the additive score is used, the language model is only called k times, once for each hypothesis popped from the queue.

Cube pruning can produce duplicate queue entries. Gesmundo and Henderson (2010) modified the algorithm prevent duplicates instead of using a hash table. We include their work in the experiments.

Hopkins and Langmead (2009) characterized cube pruning as A* search (Hart et al., 1968) with an inadmissible heuristic. Their analysis showed deep and unbalanced search trees. Our work can be interpreted as a partial rebalancing of these search trees.

2.4 Exact Algorithms

A number of exact search algorithms have been developed. We are not aware of an exact algorithm that tractably scales to the size of hypergraphs and language models used in many modern machine translation systems (Callison-Burch et al., 2012).

The hypergraph and language model can be compiled into an integer linear program. The best hypothesis can then be recovered by taking the dual and solving by Lagrangian relaxation (Rush and Collins, 2011). However, that work only dealt with language models up to order three.

Iglesias et al. (2011) represent the search space as a recursive transition network and the language model as a weighted finite state transducer. Using standard finite state algorithms, they intersect the two automatons then exactly search for the highest-scoring paths. However, the intersected automaton is too large. The authors suggested removing low probability entries from the language model, but this form of pruning negatively impacts translation quality (Moore and Quirk, 2009; Chelba et al., 2010). Their work bears some similarity to our algorithm in that partially overlapping state will be collapsed and efficiently handled together. However, the key advatage to our approach is that groups have a score that can be used for pruning *before* the group is expanded, enabling pruning without first constructing the intersected automaton.

2.5 Coarse-to-Fine

Coarse-to-fine (Petrov et al., 2008) performs multiple pruning passes, each time with more detail. Search is a subroutine of coarse-to-fine and our work is inside search, so the two are compatible. There are several forms of coarse-to-fine search; the closest to our work increases the language model order each iteration. However, by operating inside search, our algorithm is able to handle hypotheses at different levels of refinement and use scores to choose where to further refine hypotheses. Coarse-to-fine decoding cannot do this because it determines the level of refinement before calling search.

3 Our New Beam Filling Algorithm

In our algorithm, the primary idea is to group hypotheses with similar language model state. The following sections formalize what these groups are (partial state), that the groups have a recursive structure (state tree), how groups are split (bread crumbs), using groups with hypergraph edges (partial edge), prioritizing search (scoring) and best-first search (priority queue).

3.1 Partial State

An N–gram language model (with order N) computes the probability of a word given the $N-1$ preceding words. The left state of a hypothesis is the first $N-1$ words, which have insufficient context to be scored. Right state is the last $N-1$ words; these might become context for another hypothesis. Collectively, they are known as *state*. State minimization (Li and Khudanpur, 2008) may reduce the size of state due to backoff in the language model.

For example, the hypothesis "the few nations that have diplomatic relations with North Korea" might have left state "the few" and right state "Korea" after state minimization determined that "North" could be elided. Collectively, the state is denoted (the few $\dashv \diamond \vdash$ Korea). The diamond $\diamond$ is a stand-in for elided words. Terminators $\dashv$ and $\vdash$ indicate when left and right state are exhausted, respectively[2].

Our algorithm is based on *partial state*. Partial state is simply state with more inner words elided. For example, (the $\diamond$ Korea) is a partial state for (the few $\dashv \diamond \vdash$ Korea). Terminators $\dashv$ and $\vdash$ can be elided just like words. Empty state is denoted using the customary symbol for empty string, ϵ. For example, $(\epsilon \diamond \epsilon)$ is the empty partial state. The terminators serve to distinguish a completed state (which may be short due to state minimization) from an incomplete partial state.

3.2 State Tree

States (the few $\dashv \diamond \vdash$ Korea) and (the $\dashv \diamond \vdash$ Korea) have words in common, so the partial state (the $\diamond$ Korea) can be used to reason over both of them. Generalizing this notion to the set of hypotheses in a beam, we build a *state tree*. The root of the tree is the empty partial state $(\epsilon \diamond \epsilon)$ that reasons

[2] A corner case arises for hypotheses with less than $N-1$ words. For these hypotheses, we still attempt state minimization and, if successful, the state is treated normally. If state minimization fails, a flag is set in the state. For purposes of the state tree, the flag acts like a different terminator symbol.

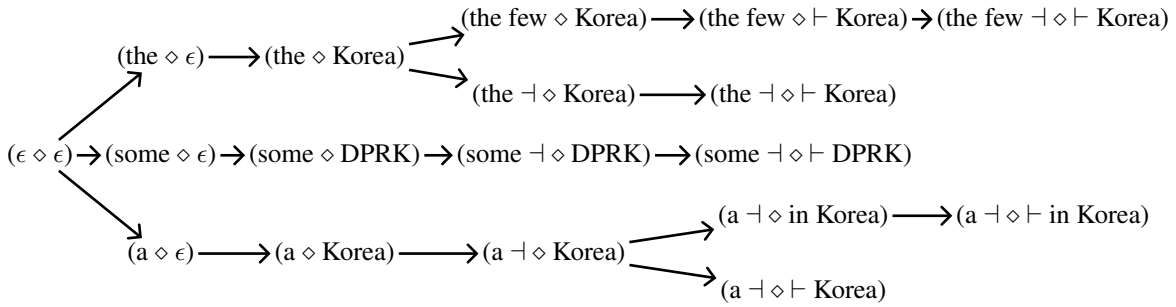

Figure 2: A state tree containing five states: (the few ⊣ ⋄ ⊢ Korea), (the ⊣ ⋄ ⊢ Korea), (some ⊣ ⋄ ⊢ DPRK), (a ⊣ ⋄ ⊢ in Korea), and (a ⊣ ⋄ ⊢ Korea). Nodes of the tree are partial states. The branching order is the first word, the last word, the second word, and so on. If the left or right state is exhausted, then branching continues with the remaining state. For purposes of branching, termination symbols ⊣ and ⊢ act like normal words.

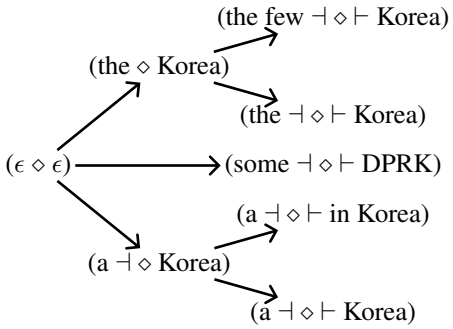

Figure 3: The optimized version of Figure 2. Nodes immediately reveal the longest shared prefix and suffix among hypotheses below them.

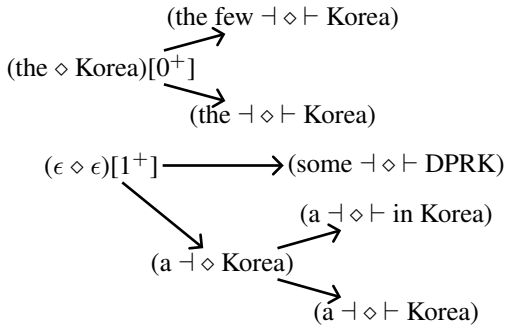

Figure 4: Visiting the root node partitions the tree into best child (the ⋄ Korea)[0^+] and bread crumb ($\epsilon \diamond \epsilon$)[$1^+$]. The data structure remains intact for use elsewhere.

over all hypotheses. From the root, the tree branches by the first word of state, the last word, the second word, the second-to-last word, and so on. If left or right state is exhausted, then branching continues using the remaining state. The branching order prioritizes the outermost words because these can be used to update the language model probability. The decision to start with left state is arbitrary. An example tree is shown in Figure 2.

As an optimization, each node determines the longest shared prefix and suffix of the hypotheses below it. The node reports these words immediately, rendering some other nodes redundant. This makes our algorithm faster because it will then only encounter nodes when there is a branching decision to be made. The original tree is shown in Figure 2 and the optimized version is shown in Figure 3. As a side effect of branching by left state first, the algorithm did not notice that states (the ⋄ Korea) and (a ⊣ ⋄ Korea) both end with Korea. We designed the tree building algorithm for speed and plan to experiment with alternatives as future work.

The state tree is built lazily. A node initially holds a flat array of all the hypotheses below it. When its children are first needed, the hypotheses are grouped by the branching word and an array of child nodes is built. In turn, these newly created children each initially hold an array of hypotheses. CPU time is saved because nodes containing low-scoring nodes may never construct their children.

Each node has a score. For leaves, this score is copied from the underlying hypothesis (or best hypothesis if some other feature prevented recombination). The score of an internal node is the maximum score of its children. As an example, the root node's score is the same as the highest-scoring hypothesis in the tree. Children are sorted by score.

3.3 Bread Crumbs

The state tree is explored in a best-first manner. Specifically, when the algorithm visits a node, it considers that node's best child. The best child reveals more words, so the score may go up or down when the language model is consulted. Therefore, simply following best children may lead to a poor hypothesis. Some backtracking mechanism is required, for which we use bread crumbs. Visiting a node results in two items: the best child and a bread crumb. The bread crumb encodes the node that was visited and how many children have already been considered. Figure 4 shows an example.

More formally, each node has an array of children sorted by score, so it suffices for the bread crumb to keep an index in this array. An index of zero denotes that no child has been visited. Continuing the example from Figure 3, $(\epsilon \diamond \epsilon)[0^+]$ denotes the root partial state with children starting at index 0 (i.e. all of them). Visiting $(\epsilon \diamond \epsilon)[0^+]$ yields best child (the $\diamond$ Korea)$[0^+]$ and bread crumb $(\epsilon \diamond \epsilon)[1^+]$. Later, the search algorithm may return to $(\epsilon \diamond \epsilon)[1^+]$, yielding best child (some $\dashv \diamond \vdash$ DPRK)$[0^+]$ and bread crumb $(\epsilon \diamond \epsilon)[2^+]$. If there is no remaining sibling, visiting yields only the best child.

The index serves to restrict the array of children to those with that index or above. Formally, let d map from a node or bread crumb to the set of leaves descended from it. The descendants of a node n are those of its children

$$d(n) = \bigsqcup_{i=0}^{|n|-1} d(n[i])$$

where $\sqcup$ takes the union of disjoint sets and $n[i]$ is the ith child. In a bread crumb with index c, only descendents by the remaining children are considered

$$d(n[c^+]) = \bigsqcup_{i=c}^{|n|-1} d(n[i])$$

It follows that the set of descendants is *partitioned* into two disjoint sets

$$d(n[c^+]) = d(n[c]) \bigsqcup d(n[c+1^+])$$

3.4 Partial Edge

The beam filling algorithm is tasked with selecting hypotheses given a number of hypergraph edges. Hypergraph edges are strings comprised of words and references to vertices (in parsing, terminals and non-terminals). A hypergraph edge is converted to a *partial edge* by replacing each vertex reference with the root node from that vertex. For example, the hypergraph edge "is v ." referencing vertex v becomes partial edge "is $(\epsilon \diamond \epsilon)[0^+]$."

Partial edges allow our algorithm to reason over a large set of hypotheses at once. Visiting a partial edge divides that set into two as follows. A heuristic chooses one of the non-leaf nodes to visit. Currently, this heuristic picks the node with the fewest words revealed. As a tie breaker, it chooses the leftmost node. The chosen node is visited (partitioned), yielding the best child and bread crumb as described in the previous section. These are substituted into separate copies of the partial edge. Continuing our example with the vertex shown in Figure 3, "is $(\epsilon \diamond \epsilon)[0^+]$." partitions into "is (the $\diamond$ Korea)$[0^+]$." and "is $(\epsilon \diamond \epsilon)[1^+]$."

3.5 Scoring

Every partial edge has a score that determines its search priority. Initially, this score is the sum of the edge's score and the scores of each bread crumb (defined below). As words are revealed, the score is updated to account for new language model context.

Each edge score includes a log language model probability and possibly additive features. Whenever there is insufficient context to compute the language model probability of a word, an estimate r is used. For example, edge "is v ." incorporates estimate

$$\log r(\text{is}) r(.)$$

into its score. The same applies to hypotheses: (the few $\dashv \diamond \vdash$ Korea) includes estimate

$$\log r(\text{the}) r(\text{few} \mid \text{the})$$

because the words in left state are those with insufficient context.

In common practice (Chiang, 2007; Hoang et al., 2009; Dyer et al., 2010), the estimate is taken from the language model: $r = p$. However, querying the language model with incomplete context leads

Kneser-Ney smoothing (Kneser and Ney, 1995) to assume that backoff has occurred. An alternative is to use average-case rest costs explicitly stored in the language model (Heafield et al., 2012). Both options are used in the experiments[3].

The score of a bread crumb is the maximum score of its descendants as defined in §3.3. For example, the bread crumb $(\epsilon \diamond \epsilon)[1^+]$ has a lower score than $(\epsilon \diamond \epsilon)[0^+]$ because the best child (the $\diamond$ Korea)$[0^+]$ and its descendants no longer contribute to the maximum.

The score of partial edge "is $(\epsilon \diamond \epsilon)[0^+]$." is the sum of scores from its two parts: edge "is v ." and bread crumb $(\epsilon \diamond \epsilon)[0^+]$. The edge's score includes estimated log probability $\log r(\text{is})r(.)$ as explained earlier. The bread crumb's score comes from its highest-scoring descendent (the few $\dashv \diamond \vdash$ Korea) and therefore includes estimate $\log r(\text{the})r(\text{few} \mid \text{the})$.

Estimates are updated as words are revealed. Continuing the example, "is $(\epsilon \diamond \epsilon)[0^+]$." has best child "is (the $\diamond$ Korea)$[0^+]$." In this best child, the estimate $r(.)$ is updated to $r(. \mid \text{Korea})$. Similarly, $r(\text{the})$ is replaced with $r(\text{the} \mid \text{is})$. Updates examine only words that have been revealed: $r(\text{few} \mid \text{the})$ remains unrevised.

Updates are computed efficiently by using pointers (Heafield et al., 2011) with KenLM. To summarize, the language model computes

$$\frac{r(w_n|w_1^{n-1})}{r(w_n|w_i^{n-1})}$$

in a single call. In the popular reverse trie data structure, the language model visits w_i^n while retrieving w_1^n, so the cost is the same as a single query. Moreover, when the language model earlier provided estimate $r(w_n|w_i^{n-1})$, it also returned a data-structure pointer $t(w_i^n)$. Pointers are retained in hypotheses, edges, and partial edges for each word with an estimated probability. When context is revealed, our algorithm queries the language model with new context w_1^{i-1} and pointer $t(w_i^n)$. The language model uses this pointer to immediately retrieve denominator $r(w_n|w_i^{n-1})$ and as a starting point to retrieve numerator $r(w_n|w_1^{n-1})$. It can therefore avoid looking up $r(w_n), r(w_n|w_{n-1}), \ldots, r(w_n|w_{i+1}^{n-1})$ as would normally be required with a reverse trie.

[3]We also tested upper bounds (Huang et al., 2012; Carter et al., 2012) but the result is still approximate due to beam pruning and initial experiments showed degraded performance.

3.6 Priority Queue

Our beam filling algorithm is controlled by a priority queue containing partial edges. The queue is populated by converting all outgoing hypergraph edges into partial edges and pushing them onto the queue. After this initialization, the algorithm loops. Each iteration begins by popping the top-scoring partial edge off the queue. If all nodes are leaves, then the partial edge is converted to a hypothesis and placed in the beam. Otherwise, the partial edge is partitioned as described in §3.3. The two resulting partial edges are pushed onto the queue. Looping continues with the next iteration until the queue is empty or the beam is full. After the loop terminates, the beam is given to the root node of the state tree; other nodes will be built lazily as described in §3.2.

Overall, the algorithm visits hypergraph vertices in bottom-up order. Our beam filling algorithm runs in each vertex, making use of state trees in vertices below. The top of the tree contains full hypotheses. If a K-best list is desired, packing and extraction works the same way as with cube pruning.

4 Experiments

Performance is measured by translating the 3003-sentence German-English test set from the 2011 Workshop on Machine Translation (Callison-Burch et al., 2011). Two translation models were built, one hierarchical (Chiang, 2007) and one with target syntax. The target-syntax system is based on English parses from the Collins (1999) parser. Both were trained on Europarl (Koehn, 2005). The language model interpolates models built on Europarl, news commentary, and news data provided by the evaluation. Interpolation weights were tuned on the 2010 test set. Language models were built with SRILM (Stolcke, 2002), modified Kneser-Ney smoothing (Kneser and Ney, 1995; Chen and Goodman, 1998), default pruning, and order 5. Feature weights were tuned with MERT (Och, 2003), beam size 1000, 100-best output, and cube pruning. Systems were built with the Moses (Hoang et al., 2009) pipeline.

Measurements were collected by running the decoder on all 3003 sentences. For consistency, all

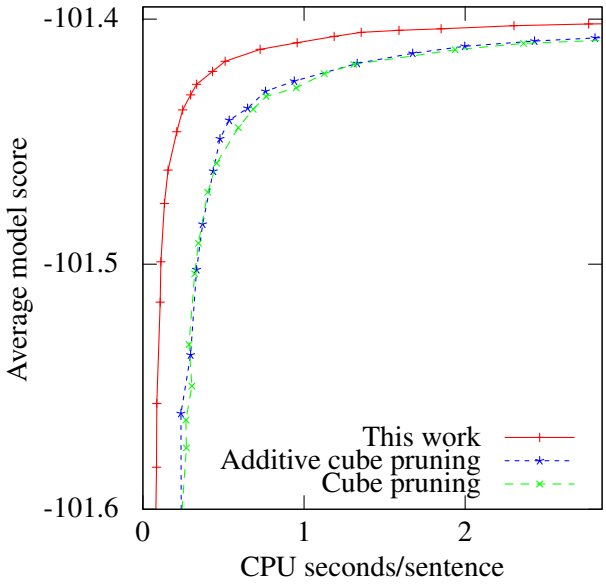

Figure 5: Hierarchial system in Moses with our algorithm, cube pruning with additive scores, and cube pruning with full scores (§2.3). The two baselines overlap.

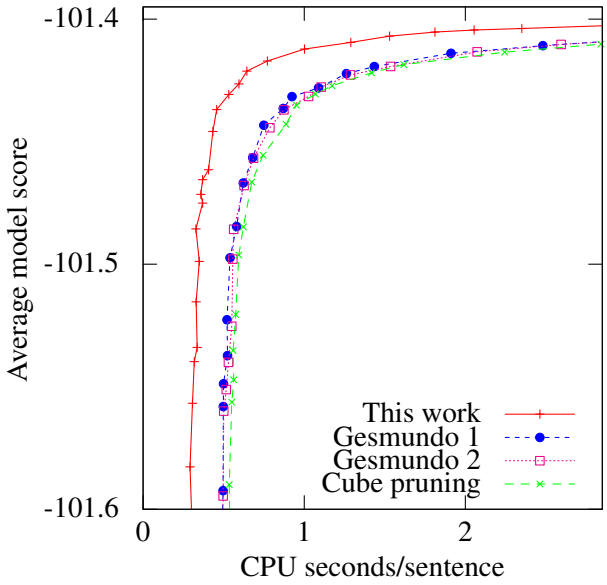

Figure 6: Hierarchical system in cdec with our algorithm, similarly-performing variants of cube pruning defined in Gesmundo and Henderson (2010), and the default.

relevant files were forced into the operating system disk cache before each run. CPU time is the total user and system time taken by the decoder minus loading time. Loading time was measured by running the decoder with empty input. In particular, CPU time includes the cost of parsing. Our test system has 32 cores and 64 GB of RAM; no run came close to running out of memory. While multi-threaded experiments showed improvements as well, we only report single-threaded results to reduce noise and to compare with cdec (Dyer et al., 2010). Decoders were compiled with the optimization settings suggested in their documentation.

Search accuracy is measured by average model score; higher is better. Only relative comparisons are meaningful because model scores have arbitrary scale and include constant factors. Beam sizes start at 5 and rise until a time limit determined by running the slowest algorithm with beam size 1000.

4.1 Comparison Inside Moses

Figure 5 shows Moses performance with this work and with cube pruning. These results used the hierarchical system with common-practice estimates (§3.5). The two cube pruning variants are explained in §2.3. Briefly, the queue can be prioritized using additive or full scores. Performance with additive scores is roughly the same as using full scores with half the beam size.

Our algorithm is faster for every beam size tested. It is also more accurate than additive cube pruning with the same beam size. However, when compared with full scores cube pruning, it is less accurate for beam sizes below 300. This makes sense because our algorithm starts with additive estimates and iteratively refines them by calling the language model. Moreover, when beams are small, there are fewer chances to group hypotheses. With beams larger than 300, our algorithm can group more hypotheses, overtaking both forms of cube pruning.

Accuracy improvements can be interpreted as speed improvements by asking how much time each algorithm takes to achieve a set level of accuracy. By this metric, our algorithm is 2.04 to 3.37 times as fast as both baselines.

4.2 Comparison Inside cdec

We also implemented our algorithm in cdec (Dyer et al., 2010). Figure 6 compares with two enhanced versions of cube pruning (Gesmundo and Henderson, 2010) and the cdec baseline. The model scores

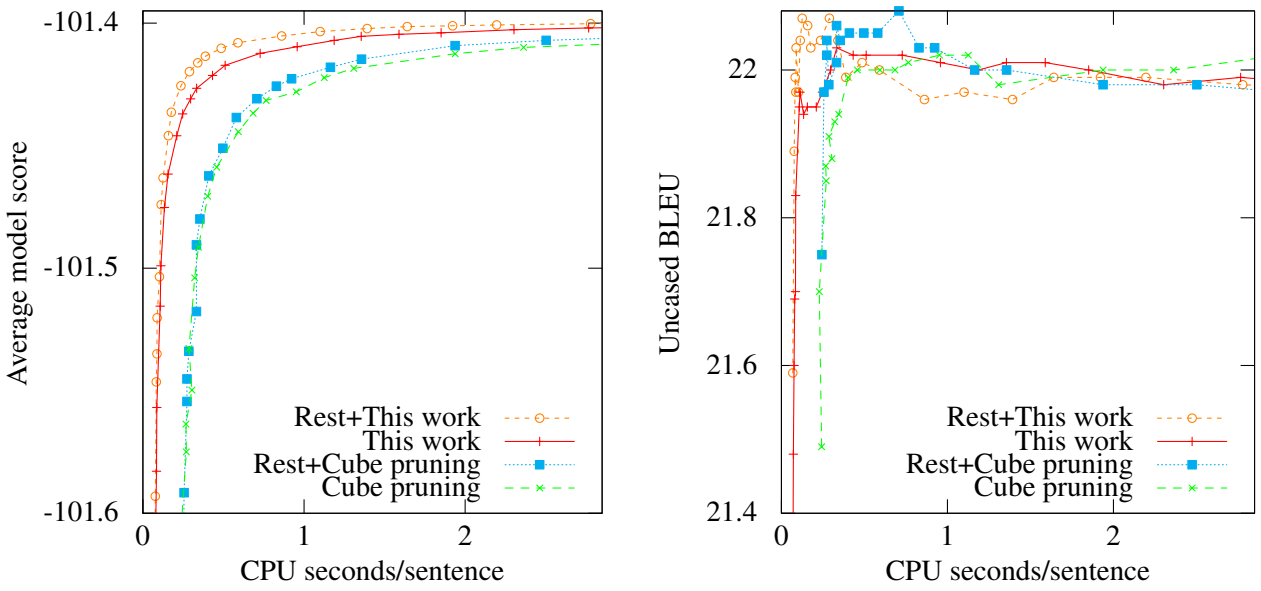

Figure 7: Effect of rest costs on our algorithm and on cube pruning in Moses. Noisy BLEU scores reflect model errors.

are comparable with Moses[4].

Measuring at equal accuracy, our algorithm makes cdec 1.56 to 2.24 times as fast as the best baseline. At first, this seems to suggest that cdec is faster. In fact, the opposite is true: comparing Figures 5 and 6 reveals that cdec has a higher parsing cost than Moses[5], thereby biasing the speed ratio towards 1. In subsequent experiments, we use Moses because it more accurately reflects search costs.

4.3 Average-Case Rest Costs

Previous experiments used the common-practice probability estimate described in §3.5. Figure 7 shows the impact of average-case rest costs on our algorithm and on cube pruning in Moses. We also looked at uncased BLEU (Papineni et al., 2002) scores, finding that our algorithm attains near-peak BLEU in less time. The relationship between model score and BLEU is noisy due to model errors.

[4]The glue rule builds hypotheses left-to-right. In Moses, glued hypotheses start with <s> and thus have empty left state. In cdec, sentence boundary tokens are normally added last, so intermediate hypotheses have spurious left state. Running cdec with the Moses glue rule led to improved time-accuracy performance. The improved version is used in all results reported. We accounted for constant-factor differences in feature definition i.e. whether <s> is part of the word count.

[5]In-memory phrase tables were used with both decoders. The on-disk phrase table makes Moses slower than cdec.

Average-case rest costs impact our algorithm more than they impact cube pruning. For small beam sizes, our algorithm becomes more accurate, mostly eliminating the disadvantage reported in §4.1. Compared to the common-practice estimate with beam size 1000, rest costs made our algorithm 1.62 times as fast and cube pruning 1.22 times as fast.

Table 1 compares our best result with the best baseline: our algorithm and cube pruning, both with rest costs inside Moses. In this scenario, our algorithm is 2.59 to 3.51 times as fast as cube pruning.

4.4 Target-Syntax

We took the best baseline and best result from previous experiments (Moses with rest costs) and ran the target-syntax system. Results are shown in Figure 8. Parsing and search are far more expensive. For beam size 5, our algorithm attains equivalent accuracy 1.16 times as fast. Above 5, our algorithm is 1.50 to 2.00 times as fast as cube pruning. Moreover, our algorithm took less time with beam size 6900 than cube pruning took with beam size 1000.

A small bump in model score occurs around 15 seconds. This is due to translating "durchzogenen" as "criss-crossed" instead of passing it through, which incurs a severe penalty (-100). The only rule capable of doing so translates "X durchzogenen" as "criss-crossed *PP*"; a direct translation rule was not

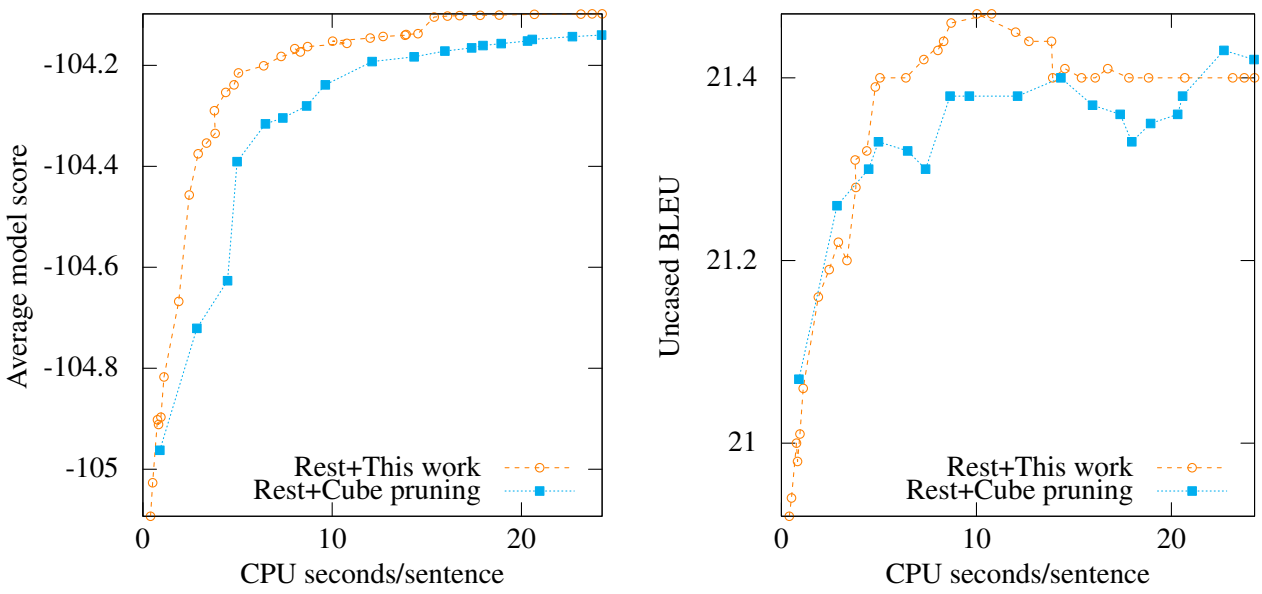

Figure 8: Performance of Moses with the target-syntax system.

extracted due to reordering. An appropriate prepositional phrase (PP) was pruned with smaller beam sizes because it is disfluent.

4.5 Memory

Peak virtual memory usage was measured before each process terminated. Compared with cube pruning at a beam size of 1000, our algorithm uses 160 MB more RAM in Moses and 298 MB less RAM in cdec. The differences are smaller with lower beam sizes and minor relative to 12-13 GB total size, most of which is the phrase table and language model.

	Rest+This work			**Rest+Cube pruning**		
k	**CPU**	**Model**	**BLEU**	**CPU**	**Model**	**BLEU**
5	0.068	-1.698	21.59	0.243	-1.667	21.75
10	0.076	-1.593	21.89	0.255	-1.592	21.97
50	0.125	-1.463	22.07	0.353	-1.480	22.04
75	0.157	-1.446	22.06	0.408	-1.462	22.05
100	0.176	-1.436	22.03	0.496	-1.451	22.05
500	0.589	-1.408	22.00	1.356	-1.415	22.00
750	0.861	-1.405	21.96	1.937	-1.409	21.98
1000	1.099	-1.403	21.97	2.502	-1.407	21.98

Table 1: Numerical results from the hierarchical system for select beam sizes k comparing our best result with the best baseline, both in Moses with rest costs enabled. To conserve space, model scores are shown with 100 added.

5 Conclusion

We have described a new search algorithm that achieves equivalent accuracy 1.16 to 3.51 times as fast as cube pruning, including two implementations and four variants. The algorithm is based on grouping similar language model feature states together and dynamically expanding these groups. In doing so, it exploits the language model's ability to estimate with incomplete information. Our implementation is available under the LGPL as a standalone from `http://kheafield.com/code/` and distributed with Moses and cdec.

Acknowledgements

This research work was supported in part by the National Science Foundation under grant IIS-0713402, by a NPRP grant (NPRP 09-1140-1-177) from the Qatar National Research Fund (a member of the Qatar Foundation), and by computing resources provided by the NSF-sponsored XSEDE program under grant TG-CCR110017. The statements made herein are solely the responsibility of the authors. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreements 287576 (CASMACAT), 287658 (EU BRIDGE), 287688 (MateCat), and 288769 (ACCEPT).

References

Yehoshua Bar-Hillel, Micha Perles, and Eli Shamir. 1964. *On Formal Properties of Simple Phrase Structure Grammars*. Hebrew University Students' Press.

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.

Chris Callison-Burch, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia. 2012. Findings of the 2012 workshop on statistical machine translation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 10–51, Montréal, Canada, June. Association for Computational Linguistics.

Simon Carter, Marc Dymetman, and Guillaume Bouchard. 2012. Exact sampling and decoding in high-order hidden Markov models. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1125–1134, Jeju Island, Korea, July.

Ciprian Chelba, Thorsten Brants, Will Neveitt, and Peng Xu. 2010. Study on interaction between entropy pruning and Kneser-Ney smoothing. In *Proceedings of Interspeech*, pages 2242–2245.

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. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270, Ann Arbor, Michigan, June.

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.

Andrea Gesmundo and James Henderson. 2010. Faster cube pruning. In *Proceedings of the International Workshop on Spoken Language Translation (IWSLT)*, pages 267–274.

Peter Hart, Nils Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July.

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. 2012. Language model rest costs and space-efficient storage. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Jeju Island, Korea.

Hieu Hoang, Philipp Koehn, and Adam Lopez. 2009. A unified framework for phrase-based, hierarchical, and syntax-based statistical machine translation. In *Proceedings of the International Workshop on Spoken Language Translation*, pages 152–159, Tokyo, Japan.

Mark Hopkins and Greg Langmead. 2009. Cube pruning as heuristic search. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 62–71, Singapore, August.

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.

Liang Huang and Haitao Mi. 2010. Efficient incremental decoding for tree-to-string translation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 273–283, Cambridge, MA, October.

Zhiheng Huang, Yi Chang, Bo Long, Jean-Francois Crespo, Anlei Dong, Sathiya Keerthi, and Su-Lin Wu. 2012. Iterative Viterbi A* algorithm for k-best sequential decoding. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1125–1134, Jeju Island, Korea, July.

Gonzalo Iglesias, Cyril Allauzen, William Byrne, Adrià de Gispert, and Michael Riley. 2011. Hierarchical phrase-based translation representations. In *Proceedings of the 2011 Conference on Empirical Methods in*

Natural Language Processing, pages 1373–1383, Edinburgh, Scotland, UK, July. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2001. Parsing and hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies*, Beijing, China, October.

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 and Barry Haddow. 2012. Towards effective use of training data in statistical machine translation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 317–321, Montréal, Canada, June. Association for Computational Linguistics.

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.

Bruce Lowerre. 1976. *The Harpy Speech Recognition System*. Ph.D. thesis, Carnegie Mellon University.

Haitao Mi, Liang Huang, and Qun Liu. 2008. Forest-based translation. In *Proceedings of ACL-08: HLT*, pages 192–199, Columbus, Ohio, June.

Robert C. Moore and Chris Quirk. 2009. Less is more: Significance-based n-gram selection for smaller, better language models. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 746–755, August.

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.

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.

Alexander Rush and Michael Collins. 2011. Exact decoding of syntactic translation models through lagrangian relaxation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 72–82, Portland, Oregon, USA, June.

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

Xiang Tong and David A. Evans. 1996. A statistical approach to automatic OCR error correction in context. In *Proceedings of the Fourth Workshop on Very Large Corpora*, pages 88–100, Copenhagen, Denmark, April.

Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. 2006. 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*, pages 777–784, Sydney, Australia, July.

Wenduan Xu and Philipp Koehn. 2012. Extending hiero decoding in Moses with cube growing. *The Prague Bulletin of Mathematical Linguistics*, 98:133–142.