The Prague Bulletin of Mathematical Linguistics
NUMBER 98 OCTOBER 2012 133–142

Extending Hiero Decoding in Moses with Cube Growing

Wenduan Xu[a], Philipp Koehn[b]

[a] Computer Laboratory, University of Cambridge
[b] School of Informatics, University of Edinburgh

Abstract

Hierarchical phrase-based (Hiero) models have richer expressiveness than phrase-based models and have shown promising translation quality gains for many language pairs whose syntactic divergences, such as reordering, could be better captured. However, their expressiveness comes at a high computational cost in decoding, which is induced by huge dynamic programs associated with language model integrated decoding, where the search space is lexically exploded and exact search often becomes intractable. Cube pruning and growing are two approximate search algorithms to make decoding much more efficient. In this article, we describe an extension to the Hiero decoder of the Moses toolkit by providing cube growing as an alternative to cube pruning, with an additional parameter similar to Jane's cube growing implementation that is not present in the original one. We also report experimental results on a full-scale NIST MT08 Chinese-English translation task.

1. Introduction

Cube pruning for machine translation (MT) decoding performs lazy computation along multi-hyperedges in parallel. However, it still computes a full k-best list for each node in the hypergraph. Based on this observation, Huang and Chiang (2007) further propose a lazy variant of cube pruning, known as *cube growing*, derived from k-best parsing in Huang and Chiang (2005), which turns the k-best selection problem into a depth-first, top-down recursive k-best generation procedure, and only generates as many hypotheses as needed at each hypergraph node to obtain the k^{th} best hypothesis of the root node.

In Hiero decoding, cube growing is a two-pass procedure. In the first pass translation model only monotonic (`-LM`) decoding, a translation hypergraph is generated.

Corresponding author: pkoehn@inf.ed.ac.uk

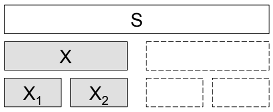


Figure 1. A toy derivation in a hypergraph. The hyperedge associated with the SCFG rule $X \rightarrow X_1X_2$ *is shaded.*

This pass could be treated as an initial bottom-up rule look-up only phrase, even though more housekeeping work is carried out in this first pass as preparation for the second-pass top-down main cube growing procedure. The main cube growing is then applied to the resulting hypergraph to generate the *k*-best hypotheses of the goal node (the *k*-best translations of a given complete sentence) in a top-down manner.

The original cube growing algorithm by Huang and Chiang (2007) is applied to a tree-to-string translation model, and the authors have left its extension to the Hiero model as part of their future work. We adapted the cube growing algorithm to the Hiero model and implemented it as an alternative search algorithm in addition to cube pruning in Moses (Koehn et al., 2007). Moreover, inspired by the cube growing implementation by Vilar et al. (2010)[1], we have introduced an additional parameter not present in the original cube growing algorithm to boost its search and translation quality.

2. Cube Growing for Hiero Decoding

We first provide an overview of the main structure of the cube growing algorithm, then follow this up with an example concentrating on a critical detail of it. We depict one possible derivation of the goal node of a hypergraph in Figure 1. Suppose we are interested in finding the first-best hypothesis of S (for the sake of discussion, we ignore the immediate right child of S, and concentrate only on the immediate left child node X). First, the main cube growing procedure is called on the S node, it will then request the first-best hypothesis of the X node. However, this required first-best hypothesis has not been generated yet, and the X node will call the main cube growing procedure on itself, which causes two further recursive calls of the main cube growing procedure on the two leaf nodes X_1 and X_2 (assuming it is the first time we visit these two leaf nodes, the first-best hypotheses of them are unknown either). Using a subprocedure of cube growing, the two leaf nodes will return the recursive calls to node X with their respective first-best hypothesis (assume for now the two leaf-nodes used a black-box subprocedure to return their first-best hypotheses to X). Upon receiving the first-best hypotheses of both X_1 and X_2, node X will use the same black-box subprocedure to

[1]To the best of our knowledge, Jane was the only open-source decoder which implements cube growing.

0.6	0.4	0.3
0.1	0.5	0.6
0.7	0.5	0.2

Figure 2. Language model costs for all hypotheses associated with a hyperedge, lower costs are better.

determine if the candidate first-best hypothesis generated from the given two first-best hypotheses of X_1 and X_2 is good enough to be passed up to node S. If not, more recursive calls would be made to both of its child nodes until it is confident to pass a hypothesis as its first-best hypothesis to S. Similarly, node S would use the same black-box subprocedure to determine its first-best hypothesis.

We now turn to the critical subprocedure and concentrating on cube growing along one hyperedge. In short, cube growing uses an estimated minimal language model cost, termed the language model intersection cost heuristic to determine whether a hypothesis is good enough to be enumerated into the final *k*-best list of a hyperedge. To define the language model intersection cost heuristic, assume for the moment that all language model intersection costs for a hyperedge are known, as depicted in each cell of Figure 2, and the language model heuristic for this hyperedge is just $\min\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7\} = 0.1$. In general, with $\mathrm{LM}(\mathbf{h}_e)$ denoting the language model cost of a hypothesis $\mathbf{h}$ from the hyperedge e, the language model intersection cost heuristic for this hyperedge is

$$\tau = \min_{\mathbf{h}_e \in \mathbf{H}} \mathrm{LM}(\mathbf{h}_e),$$

where $\mathbf{H}$ is the set of all possible hypotheses of this hyperedge. We note that this language model heuristic is a lower bound on language model intersection costs along this hyperedge, and we could estimate the total cost of any hypothesis $\mathbf{h}_e$ generated from this hyperedge as $\beta = \lambda(\mathbf{h}_e) + \tau$, where $\lambda(\mathbf{h}_e)$ is the total cost disregarding the language model cost. Also, β is always an underestimate if τ is the true minimal language model intersection cost along the hyperedge e. Furthermore, we denote the *true* total cost of a hypothesis $\mathbf{h}_e$ as $\alpha = \lambda(\mathbf{h}_e) + \mathrm{LM}(\mathbf{h}_e)$.

Return to our example and focus on cube growing along the hyperedge $X \to X_1 X_2$ as illustrated in Figure 3, assuming we know in advance that $\tau = 0.1$ for this hyperedge. In Figure 3(a) (top cube) the first recursive calls for X_1 and X_2 have returned, yielding a total cost of 1.0 for both of the first-best hypotheses of X_1 and X_2. A candidate first-best item of X is then generated, as shown in the top cube of Figure 3(a), its α cost is $1.0 + 1.0 + 0.7 = 2.7$, with 0.7 as the language model cost. This item is first put into a priority queue (*PriorityQueue*), and then is immediately popped out

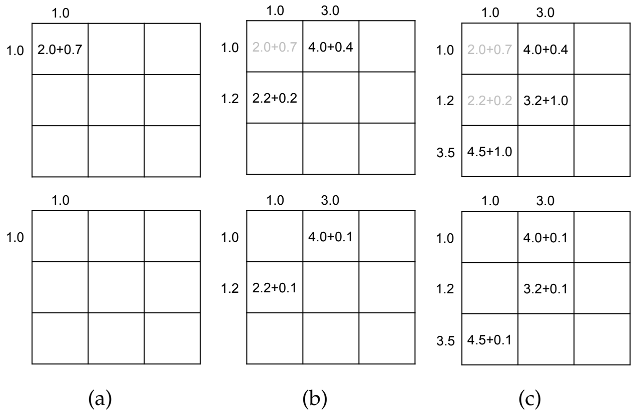


Figure 3. Example illustrating cube growing along one hyperedge. The two axes of all the "cubes" correspond to hypotheses for X_1 and X_2 respectively. On the top row, black numbers indicate α costs of hypotheses in the PriorityQueue, and grey numbers indicate hypotheses in the PriorityQueue-temp. For each cube on the top row, the corresponding cube on the bottom row represents another priority queue which contains the same hypotheses as the cube above it, but with β costs as the priorities.

of the queue as it is the only item in the queue and pushed into a buffer priority queue (*PriorityQueue-temp*) for further comparison with more items (as we are not certain this item is the true first-best item of X due to non-monotonicity). More recursive calls cause more candidate items being generated at the node X and the top cube of Figure 3(b) shows two more candidate items with α costs $2.2 + 0.2 = 2.4$ and $4.0 + 0.4 = 4.4$ and these two items are again put into the *PriorityQueue*. Moreover, as we are given that $\tau = 0.1$ for this hyperedge, we have the estimated total costs (the β costs) for these two items as $2.2 + 0.1 = 2.3$ and $4.0 + 0.1 = 4.1$. For each cube on the top row, we record β costs of hypotheses in the *PriorityQueue* into a corresponding monotonic grid on the bottom row of Figure 3 (in a practical implementation, we could implement this monotonic grid as another priority queue).

Now *PriorityQueue-temp* contains a single item $\mathbf{h}_e\langle 1 \rangle$ with an α cost of 2.7 depicted in grey (top cube of Figure 3(b)) and *PriorityQueue* contains two items with $\alpha(\mathbf{h}_e\langle 2,1\rangle) = 2.4$, $\alpha(\mathbf{h}_e\langle 1,2\rangle) = 4.4$, depicted in black (top cube of Figure 3(b)) and $\beta(\mathbf{h}_e\langle 2,1\rangle) = 2.3$, $\beta(\mathbf{h}_e\langle 1,2\rangle) = 4.1$ (bottom cube of Figure 3(b)). As in Huang and Chiang (2007), we define

$$\text{bound} = \min_{\mathbf{h}_e \in \text{PriorityQueue}} \beta\left(\mathbf{h}_e\right),$$

and in this example, $\mathrm{bound} = \min\{\beta(h_e\,(2,1)), \beta(h_e\,(1,2))\} = \min\{2.3, 4.1\} = 2.3$ with β costs as recorded in the bottom cube of Figure 3(b). Because $\alpha\,(h_e\,(1)) >$ bound, i.e., $2.7 > 2.3$, assuming lower cost is better, the candidate first-best item cannot be popped out of *PriorityQueue-temp* as the first-best hypothesis of this hyperedge (because bound tells us the α costs of any future items along this hyperedge could be anything greater than 2.3, hence $h_e\,(1)$ with an α cost of 2.7 may not be the true first-best, assuming lower cost is better). As such, we continue popping the minimal α cost hypothesis $h_e\,(2,1)$ from the *PriorityQueue* and put it into the *PriorityQueue-temp*, as shown in the top cube of Figure 3(c). More recursive calls give us the neighbours of this popped out hypothesis, with α costs $4.5 + 1.0 = 5.5$ and $3.2 + 1.0 = 4.2$ respectively. These two new neighbours are put into the *PriorityQueue* and their β costs are recorded as in the bottom cube of Figure 3(c). Now there are three items in the *PriorityQueue* (indicated by the three black numbers in the top cube of Figure 3(c)). Taking the minimal β costs of the three items (as depicted in the bottom cube of Figure 3(c)) in the *PriorityQueue*, we get $\mathrm{bound} = \min\{4.6, 3.3, 4.1\} = 3.3$. This time, α costs of both of the items in the *PriorityQueue-temp* are less than *bound*, and therefore could be enumerated into the final *k*-best lists, with $h_e\,(2,1)$ as the first-best hypothesis with α cost 2.4 and $h_e\,(1)$ as the second-best hypothesis with α cost 2.7. This procedure continues along this hyperedge and eventually generates all the *k*-best hypotheses required and in this case, the generated *k*-best items would be the true *k*-best hypotheses because 0.1 is a true lower bound of the β costs (equivalently, underestimate of α costs) along this hyperedge.

3. Language Model Cost Heuristic

In the previous section, we have explained the overall structure and described the details of the cube growing algorithm along a single hyperedge. The main recipe for cube growing along one hyperedge is to use the lower bound of the language model costs of all the hypotheses from that hyperedge to guide the search (i.e., we use the lower bound to determine if a hypothesis could be popped out of *PriorityQueue-temp* and enumerated to the final *k*-best list). For a single hyperedge, if the bound is a true lower bound, then there will be no search errors and an exact *k*-best list would be obtained for that hyperedge. Analogous to cube pruning, the version of cube growing used in MT decoding is a multi-hyperedge version, and thus we need to establish whether the lower bound is still valid if we apply the above procedure to the multi-hyperedge case where the *PriorityQueue* contains hypotheses from multiple hyperedges of one consequent node in a hypergraph. Fortunately, the validity of this lower bound across multiple hyperedges of a given consequent node is formally obtained and a complete proof is given by Huang and Chiang (2007). For completeness, we show the pseudocode of cube growing for Hiero decoding in Algorithm 1.

Algorithm 1 Cube Growing for Hiero Decoding, adapted from the algorithm in Huang and Chiang (2007)

```
 1: procedure DecodeChartSpanTopDown($X, k$)
 2:     if PriorityQueue uninitialised then
 3:         $\mathsf{PriorityQueue}\,(X) \leftarrow \emptyset$
 4:         $\mathsf{LazyCreateCube}\,(\mathsf{PriorityQueue}, \mathbf{1}, e)$ foreach $e \in \mathsf{BS}\,(X)$
 5:         *PriorityQueue-temp* $\leftarrow \emptyset$
 6:     while $|H_{\text{top-k}}| < k$ and $|\textit{PriorityQueue-temp}\,(X)| + |H_{\text{top-k}}| < j$ do
 7:         if $|\mathsf{PriorityQueue}| > 0$ then
 8:             $\mathbf{h}_e\,(\mathbf{u}) \leftarrow \textit{PriorityQueue.pop-min}_{\preceq}$
 9:             *PriorityQueue-temp*.$\mathsf{push}\,(\mathbf{h}_e\,(\mathbf{u}))$
10:             $\mathsf{LazyCreateNeighbours}\,(\mathsf{PriorityQueue}, \mathbf{h}_e\,(\mathbf{u}))$
11:             $\mathsf{bound} \leftarrow \min\{\beta\,(\mathbf{h}_e) \mid \mathbf{h}_e \in \mathsf{PriorityQueue}\}$
12:             $\mathsf{GenerateHypothesis}\,(\textit{PriorityQueue-temp}, H_{\text{top-k}}, \mathsf{bound})$
13:     $\mathsf{GenerateHypothesis}\,(\textit{PriorityQueue-temp}, H_{\text{top-k}}, +\infty)$

14: procedure LazyCreateNeighbours($\mathsf{PriorityQueue}, \mathbf{h}_e\,(\mathbf{u})$)
15:     $\mathsf{LazyCreateCube}\,(\mathsf{PriorityQueue}, \mathbf{u}+\mathbf{b}_i, e)$ foreach $i$ in $1 \ldots |e|$

16: procedure LazyCreateCube($\mathsf{PriorityQueue}, \mathbf{u}, e$)
17:     $e$ is $X \leftarrow X_1 \ldots X_{|e|}$
18:     for $\leftarrow 1 \ldots |e|$ do
19:         $\mathsf{DecodeSpanTopDown}\,(X_i, \mathbf{u}_i)$
20:         if $|H_{\text{top-k}}\,(X_i)| < \mathbf{u}_i$ then
21:             return
22:         *PriorityQueue*.$\mathsf{push}\,(\mathbf{h}_e\,(\mathbf{u}))$

23: procedure GenerateHypothesis(*PriorityQueue-temp*, $H_{\text{top-k}}$, bound)
24:     while $|\textit{PriorityQueue-temp}| > 0$ and $\min\,(\textit{PriorityQueue-temp}) < \mathsf{bound}$ do
25:         $H_{\text{top-k}}.\mathsf{push}\,(\textit{PriorityQueue-temp.pop-min})$
```

4. Estimating Language Model Cost Lower Bounds

As we have briefly described before, a first initial pass of `-LM` decoding is needed for the main cube growing procedure. It is in the first pass that we generate the translation hypergraph and try to estimate the language model lower bounds for each hyperedge. In a practical implementation, it is computationally too expensive to compute the true lower bound of language model intersection costs for each hyperedge, and following Huang and Chiang (2007), we adopt a less computationally demanding approach in our implementation.

In the `-LM` pass of decoding a sentence, we first generate the `-LM` top-*k* best translations for the complete input sentence, these *k*-best translations correspond to k complete derivations in the translation hypergraph and they may share some identical hyperedges. By following backpointers of these top-*k* `-LM` derivations, we can calculate language model intersection costs for all the hyperedges involved in those `-LM` *k*-best derivations using `-LM` hypotheses. For a hyperedge shared among multiple derivations, the minimal intersection cost is then taken as the language model lower bound estimate for that hyperedge, and for a hyperedge not shared by more than one derivation, its only intersection cost will be used as the language model lower bound estimate. Moreover, a hyperedge may be found not represented in the first pass at all, in that case we then take the language model cost of the first-best hypothesis at that hyperedge as the lower bound estimate. More detailed study on the miss rate of this procedure can be found in Vilar and Ney (2009).

While this procedure is not guaranteed to cover all hyperedges that would be used in the top-down main cube growing procedure, it is expected that the resulting estimates from these top-*k* `-LM` derivations would be sufficient to guide the search.

5. Cube Growing Practicalities

In the original cube growing algorithm, a parameter analogous to `popLimit` in cube pruning is used to control its computational cost (the variable j as shown on line 6 of Algorithm 1). This parameter is an upper bound on the number of hypotheses allowed in *PriorityQueue-temp*. Inspired by the cube growing implementation in Jane (Vilar et al., 2010), we introduce an additional parameter which is used as a lower bound on the hypothesis count in *PriorityQueue-temp*. However, our specific implementation of the lower bound parameter is different. In our implementation, the lower bound must be surpassed each time we start to add hypotheses into *PriorityQueue-temp*, while in Jane it is only used as an one-off lower bound to accumulate hypotheses when it is the first time hypotheses are being added into a *PriorityQueue-temp*. In our initial experiments, we have found that by using this lower bound parameter, it consistently improves the search and translation quality of our cube growing implementation in Moses, and thus it is used by default in our experiment.

6. Experiment

We evaluated our implementation with a full-scale NIST MT08 Chinese-English test set, which consists of 1,357 Chinese sentences. The Hiero model is trained with part of the GALE 2008 data, which has about 50M words for each language. A sentence length limit of 80 words and a symbol limit of 5 are applied. A 5-gram language model interpolated from three separate language models trained on the English side of the parallel corpus is used.

To compare with the cube pruning baseline, we decoded the test set with 21 combinations of lower and upper bounds (cf., Section 5) for the *PriorityQueue-temp* size, with each combination having equal lower and upper bounds. The first 19 bounds range from 10 to 100 in steps of 5, the last two bounds are 150 and 250, respectively. First pass `-LM` *k*-best size for cube growing is set to 200 in all experiments. BLEU and model scores against average hypothesis count plots are shown in Figure 4, with Figure 4(c) showing an enlarged part of the model cost plot. In a nutshell, cube growing maintains much more consistent runtime requirements than the cube pruning baseline and achieves more significant speedups when the `popLimit` values of cube pruning are relatively high, while maintaining similar levels of translation and search quality.

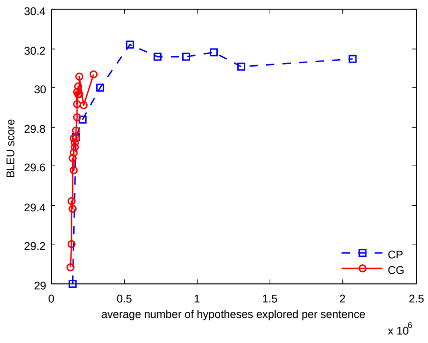
When competing with cube pruning at low `popLimit` values, cube growing has no clear advantage of speed. This is due to the fact that cube growing always requires a first bottom-up pass to generate `-LM` hypotheses and compute language model lower bound estimates. This first pass already dominates the overall runtime for cube growing with low *PriorityQueue-temp* size bounds and would have compromised the speedups gained in the top-down cube growing pass.

7. Conclusion

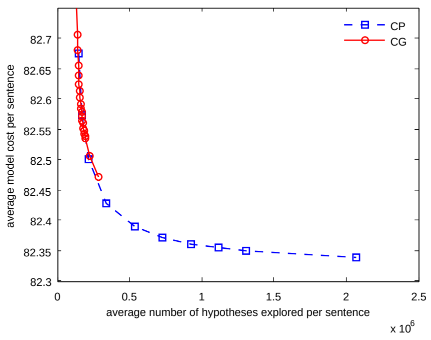
We described our implementation of the cube growing decoding algorithm originally proposed for a tree-to-string translation model (Huang and Chiang, 2007) as an alternative search algorithm for Hiero decoding in Moses and inspired by Jane (Vilar et al., 2010), a new parameter is also introduced into the original algorithm. Our experiment shows that cube growing provides a competitive alternative to cube pruning in terms of decoding speed, while maintaining the same level of translation and search quality. As future work, we would like to extend cube growing to support more syntax-based models in Moses.

Acknowledgements

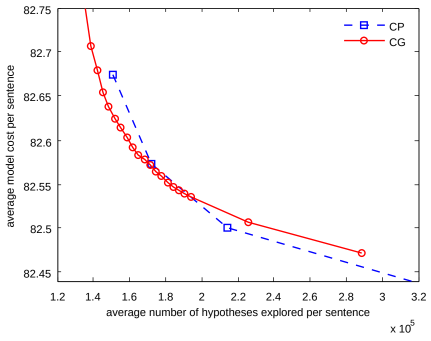
The authors would like to thank David Vilar and Huang Liang for helpful discussions and the anonymous reviewers for their constructive feedback.



(a) BLEU score vs. average hypothesis count.



(b) Average model cost (lower is better) vs. average hypothesis count.



(c) Enlarged (b). Average model cost (lower is better) vs. average hypothesis count.

Figure 4. Translation and search quality comparisons of cube pruning and cube growing.

Bibliography

Huang, Liang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64. Association for Computational Linguistics, 2005.

Huang, L. and D. Chiang. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, volume 45, page 144, 2007.

Koehn, P., H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics, 2007.

Vilar, D. and H. Ney. On lm heuristics for the cube growing algorithm. In *Proceedings of the Annual Conference of the European Association for Machine Translation*, pages 242–249. Association for Computational Linguistics, 2009.

Vilar, D., D. Stein, M. Huck, and H. Ney. Jane: Open source hierarchical translation, extended with reordering and lexicon models. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 262–270. Association for Computational Linguistics, 2010.

Address for correspondence:
Philipp Koehn
pkoehn@inf.ed.ac.uk
Informatics Forum
10 Crichton Street, Edinburgh, EH8 9AB, United Kingdom