Efficient Left-to-Right Hierarchical Phrase-based Translation with Improved Reordering

Maryam Siahbani, Baskaran Sankaran, Anoop Sarkar
Simon Fraser University
Burnaby BC. CANADA
{msiahban,baskaran,anoop}@cs.sfu.ca

Abstract

Left-to-right (LR) decoding (Watanabe et al., 2006b) is a promising decoding algorithm for hierarchical phrase-based translation (Hiero). It generates the target sentence by extending the hypotheses only on the right edge. LR decoding has complexity $O(n^2b)$ for input of n words and beam size b, compared to $O(n^3)$ for the CKY algorithm. It requires a single language model (LM) history for each target hypothesis rather than two LM histories per hypothesis as in CKY. In this paper we present an augmented LR decoding algorithm that builds on the original algorithm in (Watanabe et al., 2006b). Unlike that algorithm, using experiments over multiple language pairs we show two new results: our LR decoding algorithm provides demonstrably more efficient decoding than CKY Hiero, four times faster; and by introducing new distortion and reordering features for LR decoding, it maintains the same translation quality (as in BLEU scores) obtained phrase-based and CKY Hiero with the same translation model.

1 Introduction

Hiero (Chiang, 2007) models translation using a lexicalized synchronous context-free grammar (SCFG) extracted from word aligned bitexts. Typically, CKY-style decoding is used for Hiero with time complexity $O(n^3)$ for source input with n words. Scoring the target language output using a language model within CKY-style decoding requires two histories per hypothesis, one on the left edge of each span and one on the right, due to the fact that the target side is not generated in left to right order, but rather built bottom-up from sub-spans. This leads to complex problems in efficient language model integration and requires state reduction techniques (Heafield et al., 2011; Heafield et al., 2013). The size of a Hiero SCFG grammar is typically larger than phrase-based models extracted from the same data creating challenges in rule extraction and decoding time especially for larger datasets (Sankaran et al., 2012).

In contrast, the LR-decoding algorithm could avoid these shortcomings such as faster time complexity, reduction in the grammar size and the simplified left-to-right language model scoring. It means LR decoding has the potential to replace CKY decoding for Hiero. Despite these attractive properties, we show that the original LR-Hiero decoding proposed by (Watanabe et al., 2006b) does not perform to the same level of the standard CKY Hiero with cube pruning (see Table 3). In addition, the current LR decoding algorithm does not obtain BLEU scores comparable to phrase-based or CKY-based Hiero models for different language pairs (see Table 4). In this paper we propose modifications to the LR decoding algorithm that addresses these limitations and provides, for the first time, a true alternative to the standard CKY Hiero algorithm that uses left-to-right decoding.

We introduce a new extended version of the LR decoding algorithm presented in (Watanabe et al., 2006b) which is demonstrably more efficient than the CKY Hiero algorithm. We measure the efficiency of the LR Hiero decoder in a way that is independent of the choice of system and programming language by measuring the number of language model queries. Although more efficient, the new LR decoding algorithm suffered from lower BLEU scores compared to CKY Hiero. Our analysis of left to right decoding showed that it has more potential for search errors due to early pruning of good hypotheses. This is unlike bottom-up decoding (CKY) which keeps best hypotheses for each span. To address this issue, we introduce two novel features into the Hiero SMT model that deal with reordering and distortion. Our experiments show that LR decoding with these features using prefix lexi-

Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1089–1099,
Seattle, Washington, USA, 18-21 October 2013.

calized target side rules equals the scores obtained by CKY decoding with prefix lexicalized target side rules and phrase-based translation system. It performs four times fewer language model queries on average, compare to CKY Hiero decoding with unrestricted Hiero rules: 6466.7 LM queries for CKY Hiero (with cube pruning) compared to 1500.45 LM queries in LR Hiero (with cube pruning). While translation quality suffers by only about 0.67 in BLEU score on average, across two different language pairs.

2 Left-to-Right Decoding for Hiero

Hierarchical phrase-based SMT (Chiang, 2005; Chiang, 2007) uses a synchronous context free grammar (SCFG), where the rules are of the form $X \rightarrow \langle \gamma, \alpha \rangle$, where X is a non-terminal, γ and α are strings of terminals and non-terminals.

Chiang (2007) places certain constraints on the extracted rules in order to simplify decoding. This includes limiting the maximum number of non-terminals (rule arity) to two and disallowing any rule with consecutive non-terminals on the foreign language side. It further limits the length of the initial phrase-pair as well as the number of terminals and non-terminals in the rule. For translating sentences longer than the maximum phrase-pair length, the decoder relies on additional glue rules $S \rightarrow \langle X, X \rangle$ and $S \rightarrow \langle SX, SX \rangle$ that allows monotone combination of phrases. The glue rules are used when no rules could match or the span length is larger than the maximum phrase-pair length.

2.1 Rule Extraction for LR Decoding

Left-to-right Hiero (Watanabe et al., 2006b) generates the target hypotheses left to right, but for synchronous context-free grammar (SCFG) as used in Hiero. The target-side rules are constrained to be prefix lexicalized. These constrained SCFG rules are defined as:

$$X \rightarrow \langle \gamma, \bar{b}\, \beta \rangle \tag{1}$$

where γ is a mixed string of terminals and non-terminals. $\bar{b}$ is a terminal sequence prefixed to the possibly empty non-terminal sequence β. For the sake of simplicity, We refer to these type of rules as *GNF rules*[1] in this paper.

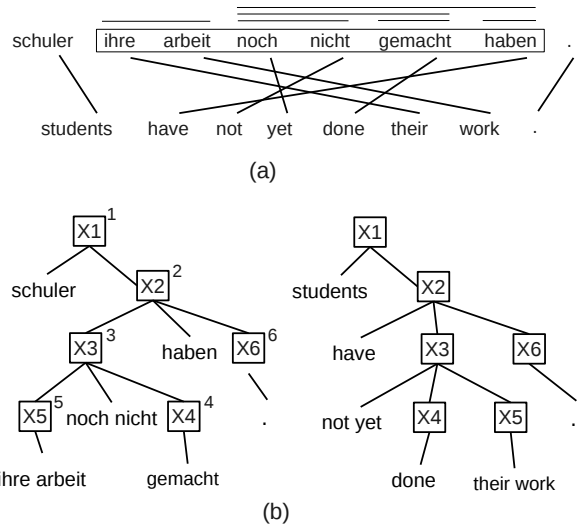


Figure 1: (a): A word-aligned German-English sentence pair. The bars above the source words indicate phrase-pairs having at least two words. (b): its corresponding left-to-right target derivation tree. Superscripts on the source non-terminals show the indices of the rules (see Fig 2) used in derivation.

Rule extraction is similar to Hiero, except any rules violating GNF form on the target side are excluded. Rule extraction considers each smaller source-target phrase pair within a larger phrase pair and replaces the spans with non-terminal X, yielding hierarchical rules. Figure 1(a) shows a word-aligned German-English sentence with a phrase pair $\langle ihre\ arbeit\ noch\ nicht\ gemacht\ haben,\ have\ not\ yet\ done\ their\ work \rangle$ that will lead to a SCFG rule. Given other smaller phrases (marked by bars above the source side), we extract a GNF rule[2]:

$$X \rightarrow \langle X_1\ noch\ nicht\ X_2\ haben, have\ not\ yet\ X_2\ X_1 \rangle \tag{2}$$

In order to avoid data sparsity and for better generalization, Watanabe et al. (2006b) adds four *glue* rules for each lexical rule $\langle \bar{f}, \bar{e} \rangle$ which are analogous to the glue rules defined in (Chiang, 2007) (see above) except that these glue rules for LR decoding

[1] Griebach Normal Form (GNF), although the synchronous grammar is not in this normal form, rather only the target side is prefix lexicalized as if it were in GNF form.

[2] LR-Hiero rule extraction excludes non-GNF rules such as $X \rightarrow \langle X_1\ noch\ nicht\ gemacht\ X_2, X_2\ not\ yet\ done\ X_1 \rangle$.

allow reordering as well.

$$\begin{array}{ll} X \rightarrow \langle \bar{f} X_1, \bar{e} X_1 \rangle & X \rightarrow \langle X_1 \bar{f} X_2, \bar{e} X_1 X_2 \rangle \\ X \rightarrow \langle X_1 \bar{f}, \bar{e} X_1 \rangle & X \rightarrow \langle X_1 \bar{f} X_2, \bar{e} X_2 X_1 \rangle \end{array} \quad (3)$$

It might appear that the restriction that target-side rules be GNF is a severe restriction on the coverage of possible hypotheses compared to the full set of rules permitted by the Hiero extraction heuristic. However there is some evidence in the literature that discontinuous spans on the source side in translation rules is a lot more useful than discontinuous spans in the target side (which is disallowed in the GNF). For instance, (Galley and Manning, 2010) do an extensive study of discontinuous spans on source and target side and show that source side discontinuous spans are very useful but removing discontinuous spans on the target side only lowers the BLEU score by 0.2 points (using the Joshua SMT system on Chinese-English). Removing discontinuous spans means that the target side rules have the form: uX, Xu, XuX, XXu, or uXX of which we disallow Xu, XuX, XXu. Zhang and Zong (2012) also conduct a study on discontinuous spans on source and target side of Hiero rules and conclude that source discontinuous spans are always more useful than discontinuities on the target side with experiments on four language pairs (zh-en, fr-en, de-en and es-en). As we shall also see in our experimental results (see Table 4) we can get close to the BLEU scores obtained using the full set of Hiero rules by using only target lexicalized rules in our LR decoder.

2.2 LR-Hiero Decoding

LR-Hiero decoding uses a top-down depth-first search, which strictly grows the hypotheses in target surface ordering. Search on the source side follows an Earley-style search (Earley, 1970), the dot jumps around on the source side of the rules based on the order of nonterminals on the target side. This search is integrated with beam search or cube pruning to efficiently find the k-best translations.

Several important details about the algorithm of LR-Hiero decoding are implicit and unexplained in (Watanabe et al., 2006b). In this section we describe the LR-Hiero decoding algorithm in more detail than the original description in (Watanabe et al., 2006b). We explain our own modified algorithm for LR decoding with cube pruning in Section 2.3.

Algorithm 1: LR-Hiero Decoding

Input sentence: $\mathbf{f} = f_0 f_1 \dots f_n$
$\mathcal{F}$ = FutureCost($\mathbf{f}$) (Precompute future cost for spans)
for $i = 0, \dots, n$ **do**
  $S_i = \{\}$ (Create empty stacks)
$h_0 = (\langle s \rangle, [[0, n]], \emptyset, \mathcal{F}_{[0,n]})$ (Initial hypothesis 4-tuple)
Add h_0 to S_0 (Push initial hyp into first Stack)
for $i = 0, \dots, n-1$ **do**
  for each h in S_i **do**
    $[u, v]$ = pop(h_s) (Pop first uncovered span from list)
    R = GetSpanRules($[u, v]$) (Extract rules matching the entire span $[u, v]$)
    for $r \in R$ **do**
      h' = GrowHypothesis($h, r, [u, v], \mathcal{F}$) (New hypothesis)
      Add h' to S_l, where $l = |h'_{cov}|$ (Add new hyp to stack)
return $\arg\max(S_n)$

GrowHypothesis$(h, r, [u, v], \mathcal{F})$
  $h' = (h'_t = \emptyset, h'_s = h_s, h'_{cov} = \emptyset, h'_c = 0)$
  $r_X = \{X_j, X_k, \dots | j \lhd k \lhd \dots\}$ (Get NTs in surface order)
  for each X in reverse(r_X) **do**
    push($h'_s, span(X)$) (Push uncovered spans to LIFO list)
  h'_t = Concatenate(h_t, r_t)
  h'_{cov} = UpdateCoverage(h_{cov}, r_s)
  h'_c = ComputeCost($g(h'), \mathcal{F}_{\neg h'_{cov}}$)
  return h'

Algorithm 1 shows the pseudocode for LR decoding. Decoding the example in Figure 1(b) is explained using a walk-through shown in Figure 2. Each partial hypothesis h is a 4-tuple (h_t, h_s, h_{cov}, h_c): consisting of a translation prefix h_t, a (LIFO-ordered) *list* h_s of uncovered spans, source words coverage set h_{cov} and the hypothesis cost h_c. The initial hypothesis is a null string with just a sentence-initial marker $\langle s \rangle$ and the list h_s containing a span of the whole sentence, $[0, n]$. The hypotheses are stored in stacks $S_0, \dots, S_n$, where each stack corresponds to a coverage vector of same size, covering same number of source words (Koehn et al., 2003).

At the beginning of beam search the initial hy-

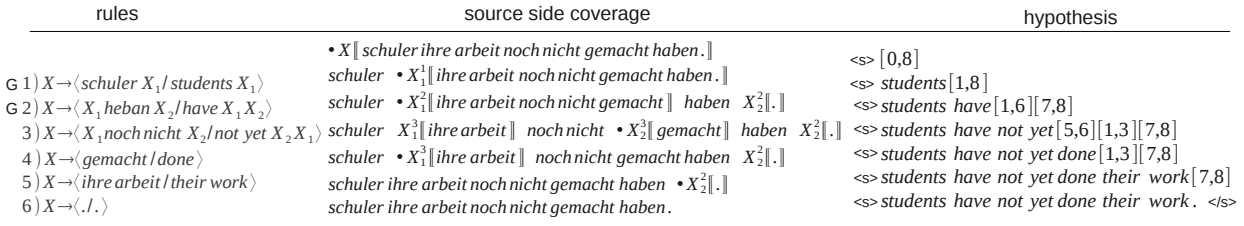


Figure 2: Illustration of the LR-Hiero decoding process in Figure 1. (a) Rules pane show the rules used in the derivation (glue rules are marked by G) (b) Decoder state using Earley dot notation (superscripts show rule#) (c) Hypotheses pane showing translation prefix and ordered list of yet-to-be-covered spans.

pothesis h_0 is added to the decoder stack S_0 (line 6 in Algoorithm 1). Hypotheses in each decoder stack are expanded iteratively, generating new hypotheses, which are added to the latter stacks corresponding to the number of source words covered. In each step it pops from the LIFO list h_s, the span $[u, v]$ of the next hypothesis h to be processed.

All rules that match the entire span $[u, v]$ are then obtained efficiently via pattern matching (Lopez, 2007). *GetSpanRules* addresses possible ambiguities in matched rules to the given span $[u, v]$. For example, given a rule r, with source side r_s : $\langle X_1 \; the \; X_2 \rangle$ and source phrase p : $\langle$*ok, the more the better*$\rangle$. There is ambiguity in matching r to p. *GetSpanRules* returns a distinct matched rule for each possible matching.

The *GrowHypothesis* routine creates a new candidate by expanding given hypothesis h using rule r and computes the complete hypothesis score including language model score. Since the target-side rules are in GNF, the translation prefix of the new hypothesis is obtained by simply concatenating the terminal prefixes of h and r in same order (line 20). *UpdateCoverage* updates source word coverage set using the source side of r. The h_s list is built by pushing the non-terminal spans of rule r in a reverse order (lines 17 and 18). The reverse ordering maintains the left-to-right generation of the target side.

In the walk-through in Figure 2, the derivation process starts by expanding the initial hypothesis h_0 (first item in the right pane of Fig 2) with the rule (rule #1 in left pane) to generate a new partial candidate having a terminal prefix of $\langle s \rangle$ *students* (second item in right pane). The second item in the middle pane shows the current position of the parser employing Earley's dot notation, indicating that the first word has already been translated. Now the decoder considers the second hypothesis and pops the span $[1, 8]$. It then matches the rule (#2) and pushes the spans $[1, 6]$ and $[7, 8]$ into the list h_s in the reverse order of their appearance in the target-side rule. At each step the new hypothesis is added to the decoder stack S_l depending on the number of covered words in the new hypothesis (line 13 in Algorithm 1).

For pruning we use an estimate of the future cost[3] of the spans uncovered by current hypothesis together with the hypothesis cost. The future cost is precomputed (line 2 Algorithm 1) in a way similar to the phrase-based models (Koehn et al., 2007) using only the terminal rules of the grammar. The ComputeCost method (line 22 in Algorithm 1) uses the usual log-linear model and scores a hypothesis based on its different feature scores $g(h')$ and the future cost of the *yet to be covered* spans ($\mathcal{F}_{\neg h'_{cov}}$). Time complexity of left to right Hiero decoding with beam search is $O(n^2 b)$ in practice where n is the length of source sentence and b is the size of beam (Huang and Mi, 2010).

2.3 LR-Hiero Decoding with Cube Pruning

The Algorithm 1 presented earlier does an exhaustive search as it generates all possible partial translations for a given stack that are reachable from the hypotheses in previous stacks. However only a few of these hypotheses are retained, while majority of them are pruned away. The cube pruning technique (Chiang, 2007) avoids the wasteful generation of poor hypotheses that are likely to be pruned away by efficiently restricting the generation to only high scoring partial translations.

We modify the cube pruning for LR-decoding that takes into account the next uncovered span to

[3] Watanabe et al. (2006b) also use a similar future cost, even though it is not discussed in the paper (p.c.).

Algorithm 2: LR-Hiero Decoding with Cube Pruning

1: Input sentence: $\mathbf{f} = f_0 f_1 \dots f_n$
2: $\mathcal{F}$ = FutureCost($\mathbf{f}$) (Precompute future cost for spans)
3: $S_0 = \{\}$ (Create empty initial stack)
4: $h_0 = (\langle s \rangle, [[0, n]], \emptyset, \mathcal{F}_{[0,n]})$ (Initial hypothesis 4-tuple)
5: Add h_0 to S_0 (Push initial hyp into first Stack)
6: **for** $i = 1, \dots, n$ **do**
7: $cubeList = \{\}$ (MRL is max rule length)
8: **for** $p = max(i - \text{MRL}, 0), \dots, i - 1$ **do**
9: $\{G\}$ = Grouped(S_p) (Group based on the first uncovered span)
10: **for** $g \in \{G\}$ **do**
11: $[u, v]$ = g_{span}
12: R = GetSpanRules($[u, v]$)
13: **for** $R_s \in R$ **do**
14: $cube = [g_{hyps}, R_s]$
15: Add $cube$ to $cubeList$
16: S_i = Merge($cubeList, \mathcal{F}$) (Create stack S_i and add new hypotheses to it, see Figure 3)
17: **return** $\arg\max(S_n)$

18: **Merge**($CubeList, \mathcal{F}$)
19: $heapQ = \{\}$
20: **for** each (H, R) in $cubeList$ **do**
21: $[u, v]$ = span of rule R
22: h' = GrowHypothesis($h_1, r_1, [u, v], \mathcal{F}$) (from Algorithm 1)
23: push($heapQ, (h'_c, h', [H, R])$
24: $hypList = \{\}$
25: **while** $|heapQ| > 0$ and $|hypList| < K$ **do**
26: $(h'_c, h', [H, R])$ = pop($heapQ$)
27: push($heapQ, GetNeighbours([H, R])$
28: Add h' to $hypList$
29: **return** $hypList$

be translated indicated by the Earley's dot notation. The Algorithm 2 shows the pseudocode for LR-decoding using cube pruning. The structure of stacks and hypotheses and computing the future cost is similar to Algorithm 1 (lines 1-5). To fill stack S_i, it iterates over previous stacks (line 8 in Algorithm 2) [4]. All hypotheses in each stack S_p (covering p words on the source-side) are first partitioned into a set of groups, $\{G\}$, based on their first uncovered span (line 9) [5]. Each group g is a 2-tuple (g_{span}, g_{hyps}), where g_{hyps} is a list of hypotheses which share the same first uncovered span g_{span}. Rules matching the span g_{span} are obtained from routine *GetSpanRules*, which are then grouped based on unique source side rules (i.e. each R_s contains rules that share the same source side s but have different target sides). Each g_{hyps} and possible R_s[6] create a cube which is added to $cubeList$.

In LR-Hiero, each hypothesis is developed with only one uncovered span, therefore each cube always has just two dimensions: (1) hypotheses with the same number of covered words and similar first uncovered span, (2) rules sharing the same source side. In Figure 3(a), each group of hypotheses, g_{hyps}, is shown in a green box (in stacks), and each rectangle on the top is a cube. Figure 3 is using the example in Figure 2.

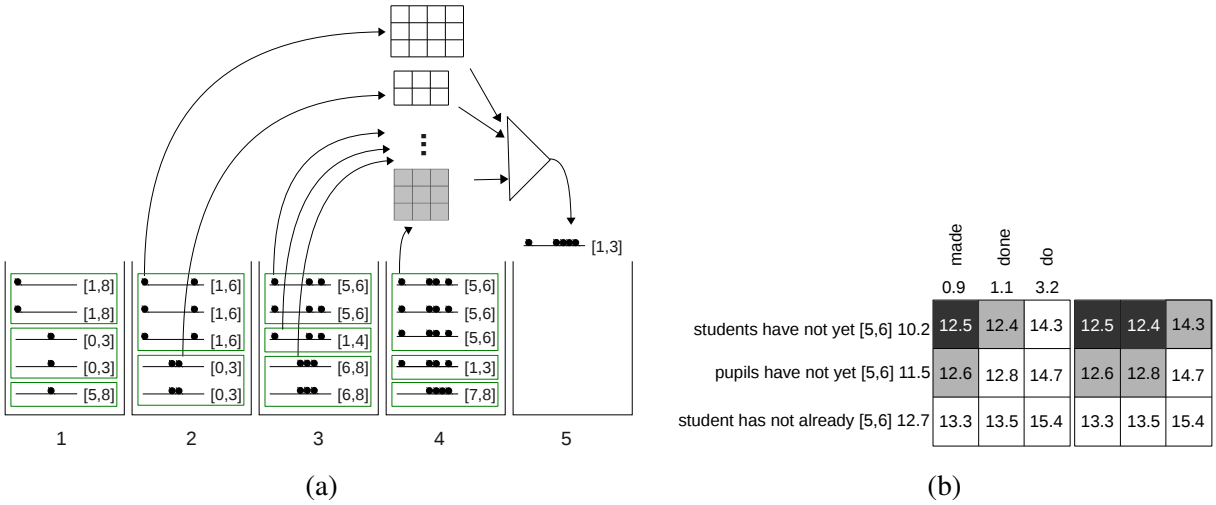
The *Merge* routine is the core function of cube pruning which generates the best hypotheses from all cubes (Chiang, 2007). For each possible cube, (H, R), the best hypothesis is generated by calling $GrowHypothesis(h_1, r_1, \textit{span}, \mathcal{F})$ where h_1 and r_1 are the best hypothesis and rule in H and R respectively (line 22). Figure 3 (b) shows a more detailed view of a cube (shaded cube in Figure 3(a)). Rows are hypotheses and columns are rules which are sorted based on their scores.

The first best hypotheses, h', along with their score, h'_c and corresponding cube, (H, R) are placed in a priority queue, $heapQ$ (triangle in Figure 3). Iteratively the best hypothesis is popped from the queue (line 26) and its neighbours in the cube are added to the priority queue (using $GetNeighbours([H, Q])$). It continues to generate all K best hypotheses. Using cube pruning technique, each stack is filled with K best hypotheses without generating all possible hypotheses in each cube.

[4] As the length of rules are limited (at most MRL), we can ignore stacks with index less than $i - \text{MRL}$

[5] The beam search decoder in Phrase-based system (Huang and Chiang, 2007; Koehn et al., 2007; Sankaran et al., 2010) groups the hypotheses in a given stack based on their coverage vector. But this idea does not work in LRHiero decoding in which the expansion of each hypothesis is restricted to its first uncovered span. We have also tried another way of grouping hypotheses: group by all uncovered spans, h_s. Our experiments did not show any significant difference between the final results (BLEU score), therefore we decided to stick to the simpler idea: using first uncovered span for grouping.

[6] Note that, just rules whose number of terminals in their source side is equal to $i - p$ can be used.



(a) (b)

Figure 3: Example of generating hypotheses in cube pruning using Figure 2: (a) Hypotheses in previous stacks are grouped based on their first uncovered span, and build cubes (grids on top). Cubes are in different sizes because of different number of rules and group sizes. Cubes are fed to a priority queue (triangle) and new hypotheses are iteratively popped from the queue and added to the current stack, S_5. (b) Generating hypotheses from a cube. The top side of the grid denotes the target side of rules sharing the same source side (R_s) along with their scores. Left side of the grid shows the hypotheses in a same group, their first uncovered span and their scores. Hypothesis generated from row 1 and column 1 is added to the queue at first. Once it is popped from the queue, its neighbours (in the grid) are subsequently added to the queue.

Figure 3 (b) shows the derivation of the two best hypotheses from the cube. The best hypothesis of this cube which is likely created from the best hypothesis and rule (left top most entry) is popped at first step. Then, *GetNeighbours* calls *GrowHypothesis* to generate next potential best hypotheses of this cube (neighbours of the popped entry which are shaded in Figure 3(b)). These hypotheses are added to the priority queue. In the next iteration, the best hypothesis is popped from all candidates in the queue and algorithm continues.

3 Features

We use the following standard SMT features for the log-linear model of LR-Hiero: relative-frequency translation probabilities $p(f|e)$ and $p(e|f)$, lexical translation probabilities $p_l(f|e)$ and $p_l(e|f)$, a language model probability, word count and phrase count. In addition we also use the glue rule count and the two reordering penalty features employed by Watanabe et al. (2006b; 2006a). These features compute the *height* and *width* (span size of the entire subtree) of all subtrees which are *backtraced* in the derivation of a hypothesis. A non-terminal X_i is pushed into the LIFO list of a partial hypothesis; it's *backtrace* refers to the set of NTs that must be popped before X_i.

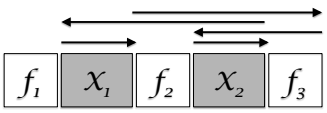
In Figure 1(b), X_2 has two subtrees X_3 and X_6, where X_3 should be processed before X_6. The subtree rooted at X_3 in Figure 1(b) has a height of 2 and span $[1, 6]$ having a width of 5. Similarly, X_4 should be backtraced before X_5 and has height and width of 1. Backtracing applies only for rules having at least two non-terminals. Thus the total height and width penalty for this derivation are 3 and 6 respectively.

However, the height and width features do not distinguish between a rule that reorders the non-terminals in source and target from one that preserves the ordering. Rules #2 and #3 in Figure 2 are treated equally although they have different orderings. The decoder is thus agnostic to this difference and would not be able to exploit this effectively to control reordering and instead would rely on the partial LM score. This issue is exacerbated for glue rules, where the decoder has to choose from different possibilities without any way to favour one over the others. Instead of the rule #2, the decoder could use its reordered version $\langle X_1\ haben\ X_2, have\ X_2\ X_1 \rangle$ leading to a poor translation.

The features we introduce can be used to learn if the model should favour monotone translations at the cost of re-orderings or vice versa and hence can easily adapt to different language pairs. Further, our experiments (see Section 4) suggest that the features h and w are not sufficient by themselves to model reordering for language pairs exhibiting very different syntactic structure.

3.1 Distortion Features

Our distortion features are inspired by their namesake in phrase-based system, with some modifications to adapt the idea for the discontiguous phrases in LR-Hiero grammar.



(a)

$r : \langle f_1 X_1 f_2 X_2 f_3, tX_2X_1 \rangle \quad I = [\vdash, f_1, f_2, f_3, X_2, X_1, \dashv]$

$r : \langle X_1 \textit{ noch nicht } X_2 / \textit{not yet } X_2 X_1 \rangle$

$_1\textit{ihre}_2 \textit{arbeit}_3 \textit{noch}_4 \textit{nicht}_5 \textit{gemacht}_6$

$I = [(1,1),(3,5),(5,6),(1,3),(6,6)]$

(b)

Figure 4: (a) Distortion feature computation using a rule r. (b) Example of distortion computation for applying r_3 on phrase $\langle$*ihre arbeit noch nicht gemacht haben*$\rangle$. subscripts between words show the indices which are used to build I. Distortion would be: $d = 2 + 0 + 5 + 3$.

Consider a rule $r = \langle \gamma, \bar{b} \ \beta \rangle$, with the source term γ being a mixed string of terminals and non-terminals. Representing the non-terminal spans and each sequence of terminals in γ as distinct *items*, our distortion feature counts the total length of *jumps* between the items during Earley parsing.

Figure 4 (a) explains the computation of our distortion feature for an example rule r. Let $I = [I_0, \ldots, I_k]$ be the *items* denoting the terminal sequences and non-terminal spans with I_0 and I_k being dummy items ($\vdash$ and $\dashv$ in Fig) marking the left and right indices of the rule r in input sentence f. Other items are arranged by their realization order on the target-side with the terminal sequences preceding non-terminal spans. The items for the example rule are shown in Figure 4 (a). The distortion feature is computed as follows:

$$d(r) = \sum_{j=1}^{k} |I_j^{\mathcal{L}} - I_{j-1}^{\mathcal{R}}| \tag{4}$$

where superscripts refer to position of left ($\mathcal{L}$) and right ($\mathcal{R}$) edge of each item in the source sentence f. These are then aggregated across the rules of a derivation D as: $d = \sum_{r \in D} d(r)$. For each item I_j, we count the jump from the end of previous item to the beginning of the current. In Figure 4 (a) the jumps are indicated by the arrows above the rule. Figure 4 (b) shows an example of distortion computation for r_3 and phrase $\langle$*ihre arbeit noch nicht gemacht haben*$\rangle$ from Figure 2.

Since the glue rules are likely to be used in the top levels (possibly with large distortion) of the derivation, we would want the decoder to learn the distortion for regular and glue rules separately. We thus use two distortion features for the two rule types and we call them d_p and d_g.

These features do not *directly* model the source-target reordering, but only capture the source-side jumps. Furthermore they apply for both monotone and reordering rules. We now introduce a new feature for exclusively modelling the reordering.

3.2 Reordering Feature

This feature simply counts the number of *reordering rules*, where the non-terminals in source and target sides are reordered. Thus $r_{\langle\rangle} = \text{rule}(D, \langle\rangle)$, where $\text{rule}(D, \langle\rangle)$ is the number of reordering rules in D. Similar to width and height, this feature is applied for rule having at least two non-terminals. This feature is applied to regular and glue rules.

4 Experiments

We conduct different types of experiments to evaluate LR-Hiero decoding developed by cube pruning and integrating new features into LR-Hiero system for two language pairs: German-English (de-en) and Czech-English (cs-en).Table 1 shows the dataset details.

4.1 System Setup

In our experiments we use four baselines as well as our implementation of LR-Hiero (written in Python):

	Corpus	Train/Dev/Test
cs-en	Europarl(v7), CzEng(v0.9); News commentary	7.95M/3000/3003
de-en	Europarl(v7); News commentary	1.5M/2000/2000

Table 1: Corpus statistics in number of sentences

Model	cs-en	de-en
Phrase-based	233.0	77.2
Hiero	1,961.6	858.5
LR-Hiero	230.5	101.3

Table 2: Model sizes (millions of rules). We do not count glue rules for LR-Hiero which are created at runtime as needed.

- **Hiero:** we used Kriya, our open-source implementation of Hiero in Python, which performs comparably to other open-source Hiero systems (Sankaran et al., 2012). Kriya can obtain statistically significantly equal BLEU scores when compared with Moses (Koehn et al., 2007) for several language pairs (Razmara et al., 2012; Callison-Burch et al., 2012).

- **Hiero-GNF:** where we use Hiero decoder with the restricted LR-Hiero grammar (GNF rules).

- **LR-Hiero:** our implementation of LR-Hiero (Watanabe et al., 2006b) in Python.

- **phrase-based:** Moses (Koehn et al., 2007)

- **LR-Hiero+CP:** LR-Hiero decoding with cube pruning.

We use a 5-gram LM trained on the Gigaword corpus and use KenLM (Heafield, 2011) for LM scoring during decoding. We tune weights by minimizing BLEU loss on the dev set through MERT (Och, 2003) and report BLEU scores on the test set. We use comparable pop limits in each of the decoders: 1000 for Moses and LR-Hiero and 500 with cube pruning for CKY Hiero and LR-Hiero+CP. Other extraction and decoder settings such as maximum phrase length, etc. were identical across settings so that the results are comparable.

Table 2 shows how the LR-Hiero grammar is much smaller than CKY-based Hiero.

Model	cs-en #queries / time(ms)	de-en #queries / time(ms)
Hiero	5,679.7 / 16.12	7,231.62 / 20.33
Hiero-GNF	4,952.5 / 14.71	5,858.74 / 18.23
LR-Hiero (1000)	46,333.21 / 163.6	83,518.63 / 328.11
LR-Hiero (500)	24,141.03 / 97.61	42,783.12 / 192.23
LR-Hiero+CP	**1,303.2 / 4.2**	**1,697.7 / 5.67**

Table 3: Comparing average number and time of language model queries.

4.2 Time Efficiency Comparison

To evaluate the performance of LR-Hiero decoding with cube pruning (LR-Hiero+CP), we compare it with three baselines: (i) CKY Hiero, (ii) CKY Hiero-GNF, and (iii) LR-Hiero (without cube pruning) with two different beam size 500 and 1000. When it comes to instrument timing results, there are lots of system level details that we wish to abstract away from, and focus only on the number of "edges" processed by the decoder. In comparison of parsing algorithms, the common practice is to measure the number of edges processed by different algorithms for the same reason (Moore and Dowding, 1991). By analogy to parsing algorithm comparisons, we compare the different decoding algorithms with respect to the number of calls made to the language model (LM) since that directly corresponds to the number of hypotheses considered by the decoder. A decoder is more time efficient if it can consider fewer translation hypotheses while maintaining the same BLEU score. All of the baselines use the same wrapper to query the language model, and we have instrumented the wrapper to count the statistics we need and thus we can say this is a fair comparison. For this experiment we use a sample set of 50 sentences taken from the test sets.

Table 3 shows the results in terms of average number of language model queries and times in milliseconds.

4.3 Reordering Features

To evaluate the new reordering features proposed to LR-Hiero (Section 3.2), LR-Hiero+CP with new features is compared to all baselines. Table 4 shows the BLEU scores of different models in two language pairs. The baseline (Watanabe et al., 2006b) model uses all the features mentioned therein but is

Model	cs-en	de-en
Phrase-based	20.32	24.71
CKY Hiero	20.64	25.52
CKY Hiero-GNF	20.04	24.84
LR-Hiero	18.30	23.47
LR-Hiero + reordering feats	20.20	24.90
LR-Hiero + CP + reordering feats	20.15	24.83
CKY Hiero-GNF + reordering feats	20.52	25.09
CKY Hiero + reordering feats	**20.77**	**25.72**

Table 4: BLEU scores. The rows are grouped such that each group use the same model. The last row in part 2 of table shows LR-Hiero+CP using our new features in addition to the baseline Watanabe features (line *LR-Hiero baseline*). The last part shows CKY Hiero using new reordering features. The reordering features used are d_p, d_g and $r_{\langle\rangle}$. LR-Hiero+CP has a beam size of 500 while LR-Hiero has a beam size of 1000, c.f. with the LM calls shown in Table 3.

worse than both phrase-based and CKY-Hiero baselines by up to 2.3 BLEU points.

All the reported results are obtained from a single optimizer run. However we observed insignificant changes in different tuning runs in our experiments. We find a gain of about 1 BLEU point when we add a single distortion feature d and a further gain of 0.3 BLEU (not shown due to lack of space) when we split the distortion feature for the two rule types (d_p and d_g). The last line in part two of Table 4 shows a consistent gain of 1.6 BLEU over the LR-Hiero baseline for both language pairs. It shows that LR-Hiero maintains the BLEU scores obtained by "phrase-based" and "CKY Hiero-GNF".

We performed statistical significance tests using two different tools: Moses bootstrap resampling and MultEval (Clark et al., 2011). The difference between "LR-Hiero+CP+reordering feat" and three baselines: "phrase-based", "CKY Hiero-GNF", "LR-Hiero+reordering feat" are not statistically significant even for p-value of 0.1 for both tools.

To investigate the impact of proposed reordering features with other decoder or models. We add these features to both Hiero and Hiero-GNF[7]. The last part of Table 4 shows the performance CKY decoder with different models (full Hiero and GNF) with the new reordering features in terms of BLEU score. The results show that these features are helpful in both models. Although, they do not make a big difference in Hiero with full model, they can alleviate the lack of non-GNF rules in Hiero-GNF.

Nguyen and Vogel (2013) integrate traditional phrase-based features: distortion and lexicalized reordering into Hiero as well. They show that such features can be useful to boost the translation quality of CKY Hiero with the full rule set. Nguyen and Vogel (2013) compute the distortion feature in a different way, only applicable to CKY. The distortion for each cell is computed after the translation for non-terminal sub-spans is complete. In LR-decoding, we compute distortion for rules even though we are yet to translate some of the sub-spans. Thus our approach computes the distortion incrementally for the untranslated sub-spans which are later added. Unlike (Nguyen and Vogel, 2013), our distortion feature can be applied to both LR and CKY-decoding (Table 4). We have also introduced another reordering feature (Section 3.2) not proposed previously.

5 Conclusion and Future Work

We provided a detailed description of left-to-right Hiero decoding, many details of which were only implicit in (Watanabe et al., 2006b). We presented an augmented LR decoding algorithm that builds on the original algorithm in (Watanabe et al., 2006b) but unlike that algorithm, using experiments over multiple language pairs we showed two new results: (i) Our LR decoding algorithm provides demonstrably more efficient decoding than CKY Hiero and the original LR decoding algorithm in (Watanabe et al., 2006b). And, (ii) by introducing new distortion and reordering features for LR decoding we show that it maintains the BLEU scores obtained by phrase-based and CKY Hiero-GNF.

CKY Hiero uses standard Hiero-style translation rules capturing better reordering model than prefix lexicalized target-side translation rules used in LR-Hiero. Our LR-decoding algorithm is 4 times faster in terms of LM calls while translation quality suffers by about 0.67 in BLEU score on average.

Unlike Watanabe et al. (2006b), our new features can easily adapt to the reordering requirements of different language pairs. We also introduce the use

[7]Feature $r_{\langle\rangle}$ is defined for SCFG rules and cannot be adopted to phrase-based translation systems; and Moses uses distortion feature therefore we omit Moses from this experiment.

of future cost in decoding algorithm which is an essential part in decoding. We have shown in this paper that left-to-right (LR) decoding can be considered as a potential faster alternative to CKY decoding for Hiero-style machine translation systems.

In future work, we plan to apply lexicalized reordering models to LR-Hiero. It has been shown to be useful for Hiero in some languages therefore it is promising to improve translation quality in LR-Hiero which suffers from lack of modeling power of non-GNF target side rules. We also plan to extend the glue rules in LR-Hiero to provide a better reordering model. We believe such an extension would be very effective in reducing search errors and capturing better reordering models in language pairs involving complex reordering requirements like Chinese-English.

Acknowledgments

This research was partially supported by an NSERC, Canada (RGPIN: 264905) grant and a Google Faculty Award to the third author. The authors wish to thank Taro Watanabe and Marzieh Razavi for their valuable discussions and suggestions, and the anonymous reviewers for their helpful comments.

References

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.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *In ACL*, pages 263–270.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33.

Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. 2011. Better hypothesis testing for statistical machine translation: controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, HLT '11, pages 176–181, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, February.

Michel Galley and Christopher D. Manning. 2010. Accurate non-hierarchical phrase-based translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 966–974, Los Angeles, California, June. Association for Computational Linguistics.

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*, pages 183–190, San Francisco, California, USA, 12.

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, 6.

Kenneth Heafield. 2011. KenLM: Faster and smaller language model queries. In *In Proc. of the Sixth Workshop on Statistical Machine Translation*.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *In ACL 07*.

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. Association for Computational Linguistics.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proc. of NAACL*.

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 *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Adam Lopez. 2007. Hierarchical phrase-based translation with suffix arrays. In *EMNLP-CoNLL*, pages 976–985.

Robert C. Moore and John Dowding. 1991. Efficient bottom-up parsing. In *HLT*. Morgan Kaufmann.

Thuylinh Nguyen and Stephan Vogel. 2013. Integrating phrase-based reordering features into chart-based decoder for machine translation. In *Proc. of ACL*.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 160–167, Stroudsburg, PA, USA. Association for Computational Linguistics.

Majid Razmara, Baskaran Sankaran, Ann Clifton, and Anoop Sarkar. 2012. Kriya - the sfu system for translation task at wmt-12. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, WMT '12, pages 356–361, Stroudsburg, PA, USA. Association for Computational Linguistics.

Baskaran Sankaran, Ajeet Grewal, and Anoop Sarkar. 2010. Incremental decoding for phrase-based statistical machine translation. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, WMT '10, pages 216–223, Stroudsburg, PA, USA. Association for Computational Linguistics.

Baskaran Sankaran, Majid Razmara, and Anoop Sarkar. 2012. Kriya - an end-to-end hierarchical phrase-based mt system. *The Prague Bulletin of Mathematical Linguistics (PBML)*, (97):83–98, apr.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2006a. NTT statistical machine translation for iwslt 2006. In *Proceedings of IWSLT 2006*, pages 95–102.

Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. 2006b. Left-to-right target generation for hierarchical phrase-based translation. In *Proc. of ACL*.

Jiajun Zhang and Chenqqing Zong. 2012. A Comparative Study on Discontinuous Phrase Translation. In *NLPCC 2012*, pages 164–175.