On Hierarchical Re-ordering and Permutation Parsing for Phrase-based Decoding

Colin Cherry
National Research Council
colin.cherry@nrc-cnrc.gc.ca

Robert C. Moore
Google
bobmoore@google.com

Chris Quirk
Microsoft Research
chrisq@microsoft.com

Abstract

The addition of a deterministic permutation parser can provide valuable hierarchical information to a phrase-based statistical machine translation (PBSMT) system. Permutation parsers have been used to implement hierarchical re-ordering models (Galley and Manning, 2008) and to enforce inversion transduction grammar (ITG) constraints (Feng et al., 2010). We present a number of theoretical results regarding the use of permutation parsers in PBSMT. In particular, we show that an existing ITG constraint (Zens et al., 2004) does not prevent all non-ITG permutations, and we demonstrate that the hierarchical re-ordering model can produce analyses during decoding that are inconsistent with analyses made during training. Experimentally, we verify the utility of hierarchical re-ordering, and compare several theoretically-motivated variants in terms of both translation quality and the syntactic complexity of their output.

1 Introduction

Despite the emergence of a number of syntax-based techniques, phrase-based statistical machine translation remains a competitive and very efficient translation paradigm (Galley and Manning, 2010). However, it lacks the syntactically-informed movement models and constraints that are provided implicitly by working with synchronous grammars. Therefore, re-ordering must be modeled and constrained explicitly. Movement can be modeled with a distortion penalty or lexicalized re-ordering probabilities (Koehn et al., 2003; Koehn et al., 2007), while decoding can be constrained by distortion limits or by mimicking the restrictions of inversion transduction grammars (Wu, 1997; Zens et al., 2004).

Recently, we have begun to see deterministic permutation parsers incorporated into phrase-based decoders. These efficient parsers analyze the sequence of phrases used to produce the target, and assemble them into a hierarchical translation history that can be used to inform re-ordering decisions. Thus far, they have been used to enable a hierarchical re-ordering model, or HRM (Galley and Manning, 2008), as well as an ITG constraint (Feng et al., 2010). We discuss each of these techniques in turn, and then explore the implications of ITG violations on hierarchical re-ordering.

We present one experimental and four theoretical contributions. Examining the HRM alone, we present an improved algorithm for extracting HRM statistics, reducing the complexity of Galley and Manning's solution from $O(n^4)$ to $O(n^2)$. Examining ITG constraints alone, we demonstrate that the three-stack constraint of Feng et al. can be reduced to one augmented stack, and we show that another phrase-based ITG constraint (Zens et al., 2004) actually allows some ITG violations to pass. Finally, we show that in the presence of ITG violations, the original HRM can fail to produce orientations that are consistent with the orientations collected during training. We propose three HRM variants to address this situation, including an approximate HRM that requires no permutation parser, and compare them experimentally. The variants perform similarly to the original in terms of BLEU score, but differently in terms of how they permute the source sentence.

Proceedings of the 7th Workshop on Statistical Machine Translation, pages 200–209,
Montréal, Canada, June 7-8, 2012. ©2012 Association for Computational Linguistics

We begin by establishing some notation. We view the phrase-based translation process as producing a sequence of source/target blocks in their target order. For the purposes of this paper, we disregard the lexical content of these blocks, treating blocks spanning the same source segment as equivalent. The block $[s_i, t_i]$ indicates that the source segment $w_{s_i+1}, \ldots, w_{t_i}$ was translated as a unit to produce the i^{th} target phrase. We index between words; therefore, a block's length in tokens is $t - s$, and for a sentence of length n, $0 \leq s \leq t \leq n$. Empty blocks have $s = t$, and are used only in special cases. Two blocks $[s_{i-1}, t_{i-1}]$ and $[s_i, t_i]$ are *adjacent* iff $t_{i-1} = s_i$ or $t_i = s_{i-1}$. Note that we concern ourselves only with adjacency in the source. Adjacency in the target is assumed, as the blocks are in target order. Figure 1 shows an example block sequence, where adjacency corresponds to cases where block corners touch. In the shift-reduce permutation parser we describe below, the parsing state is encoded as a stack of these same blocks.

2 Hierarchical Re-ordering

Hierarchical re-ordering models (HRMs) for phrase-based SMT are an extension of lexicalized re-ordering models (LRMs), so we begin by briefly reviewing the LRM (Tillmann, 2004; Koehn et al., 2007). The goal of an LRM is to characterize how a phrase-pair tends to be placed with respect to the block that immediately precedes it. Both the LRM and the HRM track orientations traveling through the target from left-to-right as well as right-to-left. For the sake of brevity and clarity, we discuss only the left-to-right direction except when stated otherwise. Re-ordering is typically categorized into three orientations, which are determined by examining two sequential blocks $[s_{i-1}, t_{i-1}]$ and $[s_i, t_i]$:

- Monotone Adjacent (M): $t_{i-1} = s_i$
- Swap Adjacent (S): $t_i = s_{i-1}$
- Disjoint (D): otherwise

Figure 1 shows a simple example, where the first two blocks are placed in monotone orientation, followed by a disjoint "red", a swapped "dog" and a disjoint period. The probability of an orientation $O_i \in \{M, S, D\}$ is determined by a conditional distribution: $Pr(O_i|\textit{source phrase}_i, \textit{target phrase}_i)$.

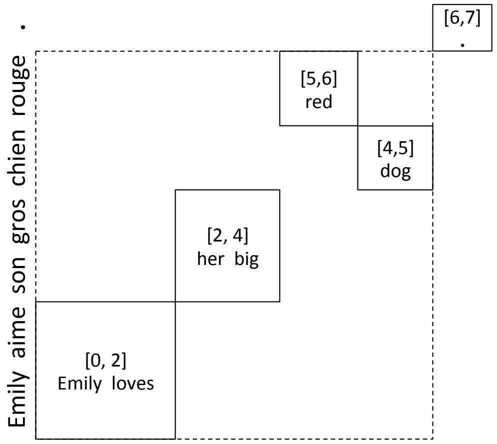

Figure 1: A French-to-English translation with 5 blocks.

To build this model, orientation counts can be extracted from aligned parallel text using a simple heuristic (Koehn et al., 2007).

The HRM (Galley and Manning, 2008) maintains similar re-ordering statistics, but determines orientation differently. It is designed to address the LRM's dependence on the previous block $[s_{i-1}, t_{i-1}]$. Consider the period [6,7] in Figure 1. If a different segmentation of the source had preceded it, such as one that translates "chien rouge" as a single [4,6] block, the period would have been in monotone orientation. Galley and Manning (2008) introduce a deterministic shift-reduce parser into decoding, so that the decoder always has access to the largest possible previous block, given the current translation history. The parser has two operations: *shift* places a newly translated block on the top of the stack. If the top two blocks are adjacent, then a *reduce* is immediately performed, replacing them with a single block spanning both. Table 1 shows the parser states corresponding to our running example. Whether "chien rouge" is translated using [5,6],[4,5] or [4,6] alone, the shift-reduce parser provides a consolidated previous block of [0,6] at the top of the stack (shown with dotted lines). Therefore, [6,7] is placed in monotone orientation in both cases.

The parser can be easily integrated into a phrase-based decoder's translation state, so each partial hypothesis carries its own shift-reduce stack. Time and memory costs for copying and storing stacks can be kept small by sharing tails across decoder states. The stack subsumes the coverage vector in that it contains strictly more information: every covered

Op	Stack
S	[0,2]
S	[0,2],[2,4]
R	[0,4]
S	[0,4],[5,6]
S	[0,4],[5,6],[4,5]
R	[0,4],[4,6]
R	[0,6]
S	[0,6],[6,7]
R	[0,7]

Table 1: Shift-reduce states corresponding to Figure 1.

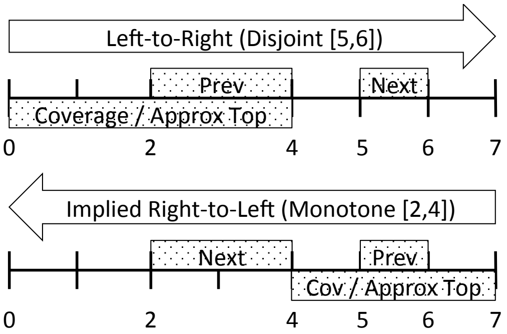
Left-to-Right (Disjoint [5,6])
Prev
Next
Coverage / Approx Top
0
2
4
5
6
7
Implied Right-to-Left (Monotone [2,4])
Next
Prev
Cov / Approx Top
0
2
4
5
6
7

Figure 2: Illustration of the coverage-vector stack approximation, as applied to right-to-left HRM orientation.

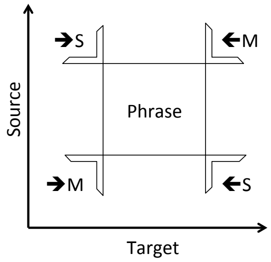

Figure 3: Relevant corners in HRM extraction. → indicates left-to-right orientation, and ← right-to-left.

word will be present in one of the stack's blocks. However, it can be useful to maintain both.

The top item of a parser's stack can be approximated using only the coverage vector. The approximate top is the largest block of covered words that contains the last translated block. This approximation will always be as large or larger than the true top of the stack, and it will often match the true top exactly. For example, in Figure 1, after we have translated [2,4], we can see that the coverage vector contains all of [0,4], making the approximate top [0,4], which is also the true top. In fact, this approximation is correct at every time step shown in Figure 1. Keep this approximation in mind, as we return to it in Sections 3.2 and 4.3.

We do not use a shift-reduce parser that consumes source words from right-to-left;[1] therefore, we apply the above approximation to handle the right-to-left HRM. Before doing so, we re-interpret the decoder state to simulate a right-to-left decoder. The last block becomes $[s_i, t_i]$ and the next block becomes $[s_{i-1}, t_{i-1}]$, and the coverage vector is inverted so that covered words become uncovered and vice versa. Taken all together, the approximate test for right-to-left adjacency checks that any gap between $[s_{i-1}, t_{i-1}]$ and $[s_i, t_i]$ is *uncovered* in the original coverage vector.[2] Figure 2 illustrates how a monotone right-to-left orientation can be (correctly) determined for $[2, 4]$ after placing $[5, 6]$ in Figure 1.

Statistics for the HRM can be extracted from word-aligned training data. Galley and Manning (2008) propose an algorithm that begins by running standard phrase extraction (Och and Ney, 2004) without a phrase-length limit, noting the corners of each phrase found. Next, the left-to-right and right-to-left orientation for each phrase of interest (those within the phrase-length limit) can be determined by checking to see if any corners noted in the previous step are adjacent, as shown in Figure 3.

2.1 Efficient Extraction of HRM statistics

The time complexity of phrase extraction is bounded by the number of phrases to be extracted, which is determined by the sparsity of the input word alignment. Without a limit on phrase length, a sentence pair with n words in each language can have as many as $O(n^4)$ phrase-pairs.[3] Because it relies on unrestricted phrase extraction, the corner collection step for determining HRM orientation is also $O(n^4)$.

By leveraging the fact that the first step collects corners, not phrase-pairs, we can show that HRM extraction can actually be done in $O(n^2)$ time, through a process we call *corner propagation*. Instead of running unrestricted phrase-extraction, corner propagation begins by extracting all minimal

[1] This would require a second, right-to-left decoding pass.

[2] Galley and Manning (2008) present an under-specified approximation that is consistent with what we present here.

[3] Consider a word-alignment with only one link in the center of the grid.

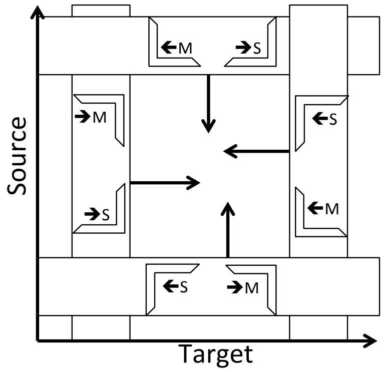

Figure 4: Corner Propagation: Each of the four passes propagates two types of corners along a single dimension.

phrase-pairs; that is, those that do not include unaligned words at their boundaries. The complexity of this step is $O(n^2)$, as the number of minimal phrases is bounded by the minimum of the number of monolingual phrases in either language. We note corners for each minimal pair, as in the original HRM extractor. We then carry out four non-nested propagation steps to handle unaligned words, traversing the source (target) in forward and reverse order, with each unaligned row (column) copying corners from the previous row (column). Each pass takes $O(n^2)$ time, for a total complexity of $O(n^2)$. This process is analogous to the growing step in phrase extraction, but computational complexity is minimized because each corner is considered independently. Pseudo-code is provided in Algorithm 1, and the propagation step is diagrammed in Figure 4. In our implementation, corner propagation is roughly two-times faster than running unrestricted phrase-extraction to collect corners.

Note that the trickiest corners to catch are those that are diagonally separated from their minimal block (they result from unaligned growth in both the source and target). These cases are handled correctly because each corner type is touched by two propagators, one for the source and one for the target (see Figure 4). For example, the top-right-corner array $A^{\urcorner}$ is populated by both propagate-right and propagate-up. Thus, one propagator can copy a corner along one dimension, while the next propagator copies the copies along the other dimension, moving the original corner diagonally.

Algorithm 1 Corner Propagation

```
Initialize target-source indexed binary arrays $A^{\urcorner}[m][n]$, $A_{\lrcorner}[m][n]$, $A^{\ulcorner}[m][n]$ and $A_{\llcorner}[m][n]$ to record corners found in minimal phrase-pairs.
{Propagate Right}
for $i$ from 2 to $m$ s.t. $target[i]$ is unaligned do
    for $j$ from 1 to $n$ do
        $A^{\urcorner}[i][j] =$ True if $A^{\urcorner}[i-1][j]$ is True
        $A_{\lrcorner}[i][j] =$ True if $A_{\lrcorner}[i-1][j]$ is True
{Propagate Up}
for $j$ from 2 to $n$ s.t. $source[j]$ is unaligned do
    for $i$ from 1 to $m$ do
        $A^{\ulcorner}[i][j] =$ True if $A^{\ulcorner}[i][j-1]$ is True
        $A^{\urcorner}[i][j] =$ True if $A^{\urcorner}[i][j-1]$ is True
{Propagate Left and Down are similar}
return $A^{\urcorner}$, $A_{\lrcorner}$, $A^{\ulcorner}$ and $A_{\llcorner}$
```

3 ITG-Constrained Decoding

Phrase-based decoding places no implicit limits on re-ordering; all $n!$ permutations are theoretically possible. This is undesirable, as it leads to intractability (Knight, 1999). Therefore, re-ordering is limited explicitly, typically using a distortion limit. One particularly well-studied re-ordering constraint is the ITG constraint, which limits source permutations to those achievable by a binary bracketing synchronous context-free grammar (Wu, 1997). ITG constraints are known to stop permutations that generalize 3142 and 2413,[4] and can drastically limit the re-ordering space for long strings (Zens and Ney, 2003). There are two methods to incorporate ITG constraints into a phrase-based decoder, one using the coverage vector (Zens et al., 2004), and the other using a shift-reduce parser (Feng et al., 2010). We begin with the latter, returning to the coverage-vector constraint later in this section.

Feng et al. (2010) describe an ITG constraint that is implemented using the same permutation parser used in the HRM. To understand their method, it is important to note that the set of ITG-compliant permutations is exactly the same as those that can be reduced to a single-item stack using the shift-reduce permutation parser (Zhang and Gildea, 2007). In fact, this manner of parsing was introduced to SMT

[4] 2413 is shorthand notation that denotes the block sequence [1,2],[3,4],[0,1],[2,3] as diagrammed in Figure 5a.

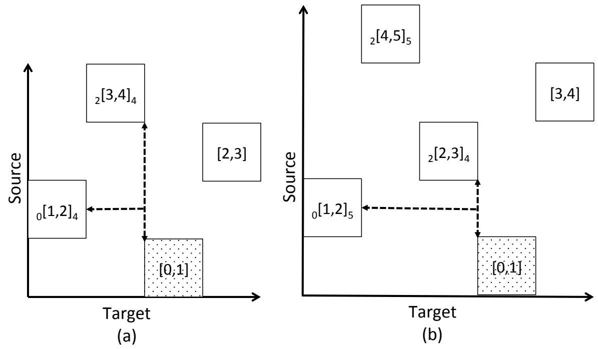

Figure 5: Two non-ITG permutations. Violations of potential adjacency are indicated with dotted spans. Bounds for the one-stack constraint are shown as subscripts.

in order to binarize synchronous grammar productions (Zhang et al., 2006). Therefore, enforcing an ITG constraint in the presence of a shift-reduce parser amounts to ensuring that every shifted item can eventually be reduced. To discuss this constraint, we introduce a notion of *potential adjacency*, where two blocks are potentially adjacent if any words separating them have not yet been covered. Formally, blocks $[s,t]$ and $[s',t']$ are potentially adjacent iff one of the following conditions holds:

· they are adjacent ($t' = s$ or $t = s'$)
· $t' < s$ and $[t', s]$ is uncovered
· $t < s'$ and $[t, s']$ is uncovered

Recall that a reduction occurs when the top two items of the stack are adjacent. To ensure that reductions remain possible, we only shift items onto the stack that are potentially adjacent to the current top. Figure 5 diagrams two non-ITG permutations and highlights where potential adjacency is violated. Note that no reductions occur in either of these examples; therefore, each block $[s_i, t_i]$ is also the top of the stack at time i. Potential adjacency can be confirmed with some overhead using the stack and coverage vector together, but Feng et al. (2010) present an elegant three-stack solution that provides potentially adjacent regions in constant time, without a coverage vector. We improve upon their method later this section. From this point on, we abbreviate potential adjacency as PA.

We briefly sketch a proof that maintaining potential adjacency maintains reducibility, by showing that non-PA shifts produce irreducible stacks, and that PA shifts are reducible. It is easy to see that every non-PA shift leads to an irreducible stack. Let $[s', t']$ be an item to be shifted onto the stack, and $[s, t]$ be the current top. Assume that $t' < s$ and the two items are not PA (the case where $t < s'$ is similar). Because they are not PA, there is some index k in $[t', s]$ that has been previously covered. Since it is covered, k exists somewhere in the stack, buried beneath $[s, t]$. Because k cannot be re-used, no series of additional shift and reduce operations can extend $[s', t']$ so that it becomes adjacent to $[s, t]$. Therefore, $[s, t]$ will never participate in a reduction, and parsing will close with at least two items on the stack. Similarly, one can easily show that every PA shift is reducible, because the uncovered space $[t', s]$ can be filled by extending the new top toward the previous top using strictly adjacent shifts.

3.1 A One-stack ITG Constraint

As mentioned earlier, Feng et al. (2010) provide a method to track potential adjacency that does not require a coverage vector. Instead, they maintain three stacks, the original stack and two others to track potentially adjacent regions to the left and right respectively. These regions become available to the decoder only when the top of the original stack is adjacent to one of the adjacency stacks.

We show that the same goal can be achieved with even less book-keeping by augmenting the items on the original stack to track the regions of potential adjacency around them. The intuition behind this technique is that on a shift, the new top inherits all of the constraints on the old top, and the old top becomes a constraint itself. Each stack item now has four fields, the original block $[s, t]$, plus a left and right adjacency bound, denoted together as $_\ell[s, t]_r$, where ℓ and r are indices for the maximal span containing $[s, t]$ that is uncovered except for $[s, t]$. If the top of the stack is $_\ell[s, t]_r$, then shifted items must fall inside one of the two PA regions, $[\ell, s]$ or $[t, r]$. The region shifted into determines new item's bounds.

The stack is initialized with a special $_0[0, 0]_n$ item, and we then shift unannotated blocks onto the stack. As we shift $[s', t']$ onto the stack, rules derive bounds ℓ' and r' for the new top based on the old top $_\ell[s, t]_r$:

- Shift-left ($t' \leq s$): $\ell' = \ell$, $r' = s$
- Shift-right ($t \leq s'$): $\ell' = t$, $r' = r$

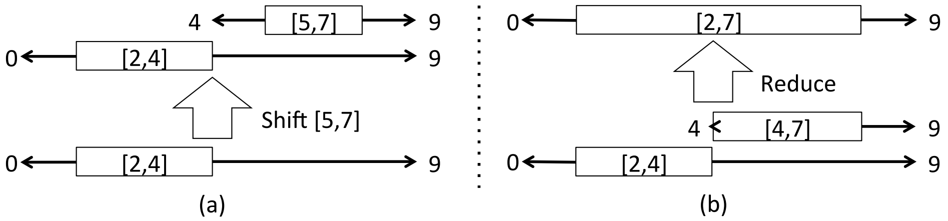

Figure 6: Two examples of boundaries for the one-stack solution for potential adjacency. Stacks are built from bottom to top, blocks indicate [s,t] blocks, while tails are left and right adjacency boundaries.

Meanwhile, when reducing a stack with ${}_{\ell'}[s', t']_{r'}$ at the top and ${}_{\ell}[s, t]_{r}$ below it, the new top simply copies ℓ and r. The merged item is larger than $[s, t]$, but it is PA to the same regions. Figure 6 diagrams a shift-right and a reduce, while Figure 5 annotates bounds for blocks during its ITG violations.

3.2 The Coverage-Vector ITG Constraint is Incomplete

The stack-based solution for ITG constraints is elegant, but there is also a proposed constraint that uses only the coverage vector (Zens et al., 2004). This constraint can be stated with one simple rule: if the previously translated block is $[s_{i-1}, t_{i-1}]$ and the next block to be translated is $[s_i, t_i]$, one must be able to travel along the coverage vector from $[s_{i-1}, t_{i-1}]$ to $[s_i, t_i]$ without transitioning from an uncovered word to a covered word. Feng et al. (2010) compare the two ITG constraints, and show that they perform similarly, but not identically. They attribute the discrepancy to differences in *when* the constraints are applied, which is strange, as the two constraints need not be timed differently.

Let us examine the coverage-vector constraint more carefully, assuming that $t_i < s_{i-1}$ (the case where $t_{i-1} < s_i$ is similar). The constraint consists of two phases: first, starting from s_{i-1}, we travel to the left toward t_i, consuming covered words until we reach the first uncovered word. We then enter into the second phase, and the path must remain uncovered until we reach t_i. The first step over covered positions corresponds to finding the left boundary of the largest covered block containing $[s_{i-1}, t_{i-1}]$, which is an approximation to the top of the stack (Section 2). The second step over uncovered positions corresponds to determining whether $[s_i, t_i]$ is PA to the approximate top. That is, the coverage-vector ITG constraint checks for potential adjacency using the same top-of-stack approximation as the right-to-left HRM.

This implicit approximation implies that there may well be cases where the coverage-vector constraint makes the wrong decision. Indeed this is the case, which we prove by example. Consider the irreducible sequence 25314, illustrated in Figure 5b. This non-ITG permutation is allowed by the coverage-vector approximation, but not by the stack-based constraint. Both constraints allow the placement of the first three blocks $[1, 2]$, $[4, 5]$ and $[2, 3]$. After adding $[0, 1]$, the stack-based solution detects a PA-violation. Meanwhile, the vector-based solution checks the path from 2 to 1 for a transition from uncovered to covered. This short path touches only covered words. Similarly, as we add $[3, 4]$, the path from 1 to 3 is also completely covered. The entire permutation is accepted without complaint. The proof provided by Zens et al. (2004) misses this case, as it accounts for phrasal generalizations of the 2413 ITG-forbidden substructure, but it does not account for generalizations where the substructure is interrupted by a discontiguous item, such as in 25{3}14, where 2413 is revealed not by merging items but by deleting 3.

4 Inconsistencies in HRM parsing

We have shown that the HRM and the ITG constraints for phrase-based decoding use the same deterministic shift-reduce parser. The entirety of the ITG discussion was devoted to preventing the parser from reaching an irreducible state. However, up until now, work on the HRM has not addressed the question of irreducibility (Galley and Manning, 2008; Nguyen et al., 2009).

Irreducible derivations do occur during HRM decoding, and when they do, they can create inconsistencies with respect to HRM extraction from word-

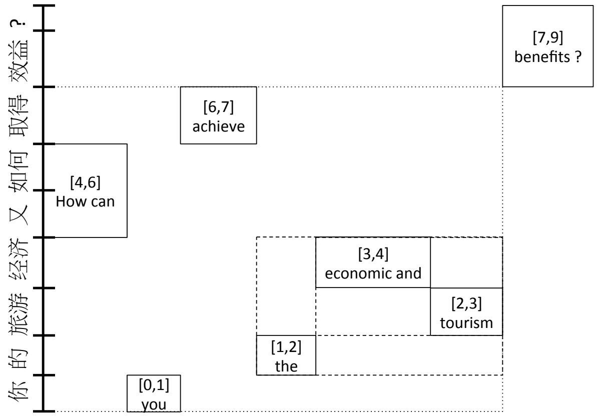

Figure 7: An example irreducible derivation, drawn from our Chinese-to-English decoder's k-best output.

Last translated block		2-red	*-red	approx
How can	[4, 6]	[4,6]	[4,6]	[4,6]
you	[0, 1]	[0,1]	[0,1]	[0,1]
achieve	[6, 7]	[6,7]	[6,7]	**[4,7]**
the	[1, 2]	[1,2]	[1,2]	**[0,2]**
economic and	[3, 4]	[3,4]	[3,4]	**[3,7]**
tourism	[2, 3]	**[1,4]**	[0,7]	[0,7]
benefits?	[7, 9]	**[7,9]**	[0,9]	[0,9]

Table 2: Top of stack at each time step in Figure 7, under 2-reduction (as in the original HRM), *-reduction, and the coverage-vector approximation.

aligned training data. In Figure 7, we show an irreducible block sequence, extracted from a Chinese-English decoder. The parser can perform a few small reductions, creating a [1,4] block indicated with a dashed box, but translation closes with 5 items on the stack. One can see that [7,9] is assigned a disjoint orientation by the HRM. However, if the same translation and alignment were seen during training, the unrestricted phrase extractor would find a phrase at [0,7], indicated with a dotted box, and [7,9] would be assigned monotone orientation. This inconsistency penalizes this derivation, as "benefits ?" is forced into an unlikely disjoint orientation. One potential implication is that the decoder will tend to avoid irreducible states, as those states will tend to force unlikely orientations, resulting in a hidden, soft ITG-constraint. Indeed, our decoder does not select this hypothesis, but instead a (worse) translation that is fully reducible. The impact of these inconsistencies on translation quality can only be determined empirically. However, to do so, we require alternatives that address these inconsistencies. We describe three such variants below.

4.1 ITG-constrained decoding

Perhaps the most obvious way to address irreducible states is to activate ITG constraints whenever decoding with an HRM. Irreducible derivations will disappear from the decoder, along with the corresponding inconsistencies in orientation. Since both techniques require the same parser, there is very little overhead. However, we will have also limited our decoder's reordering capabilities.

4.2 Unrestricted shift-reduce parsing

The deterministic shift-reduce parser used throughout this paper is actually a special case of a general class of permutation parsers, much in the same way that a binary ITG is a special case of synchronous context-free grammar. Zhang and Gildea (2007) describe a family of k-reducing permutation parsers, which can reduce the top k items of the stack instead of the top 2. For $k \geq 2$ we can generalize the adjacency requirement for reduction to a *permutation* requirement. Let $\{[s_i, t_i]|\text{i=1...k}\}$ be the top k items of a stack; they are a permutation iff:

$$\max_i(t_i) - \min_i(s_i) = \sum_i [t_i - s_i]$$

That is, every number between the max and min is present somewhere in the set. Since two adjacent items always fulfill this property, we know the original parser is 2-reducing. k-reducing parsers reduce by moving progressively deeper in the stack, looking for the smallest $2 \leq i \leq k$ that satisfies the permutation property (see Algorithm 2). As in the original parser, a k-reduction is performed every time the top of the stack changes; that is, after each shift and each successful reduction.

If we set $k = \infty$, the parser will find the smallest possible reduction without restriction; we refer to this as a *-reducing parser. This parser will never reach an irreducible state. In the worst case, it reduces the entire permutation as a single n-reduction after the last shift. This means it will exactly mimic unrestricted phrase-extraction when predicting orientations, eliminating inconsistencies without restricting our re-ordering space. The disadvantage is

Algorithm 2 k-reduce a stack

```
input stack $\{[s_i, t_i] | i = 1 \dots l\}$; $i = 1$ is the top
input max reduction size $k$, $k \geq 2$
set $s' = s_1$; $t' = t_1$; $size = t_1 - s_1$
for $i$ from 2 to $\min(k, l)$ do
    set $s' = \min(s', s_i)$; $t' = \max(t', t_i)$
    set $size = size + (t_i - s_i)$
    if $t' - s' == size$ then
        pop $\{[s_j, t_j] | j = 1 \dots i\}$ from the stack
        push $[s', t']$ onto the stack;
        return true // successful reduction
return false // failed to reduce
```

that reduction is no longer a constant-time operation, but is instead $O(n)$ in the worst case (consider Algorithm 2 with $k = \infty$ and $l = n$ items on the stack).[5] As a result, we will carefully track the impact of this parser on decoding speed.

4.3 Coverage vector approximation

One final option is to adopt the top-of-stack approximation for left-to-right orientations, in addition to its current use for right-to-left orientations, eliminating the need for any permutation parser. The next block $[s_i, t_i]$ is adjacent to the approximate top of the stack only if any space between $[s_i, t_i]$ and the previous block $[s_{i-1}, t_{i-1}]$ is covered. But before committing fully to this approximation, we should better understand it. Thus far, we have implied that this approximation can fail to predict correct orientations, but we have not specified when these failures occur. We now show that incorrect orientations can only occur while producing a non-ITG permutation.

Let $[s_{i-1}, t_{i-1}]$ be the last translated block, and $[s_i, t_i]$ be the next block. Recall that the approximation determines the top of the stack using the largest block of covered words that contains $[s_{i-1}, t_{i-1}]$. The approximate top always contains the true top, because they both contain $[s_{i-1}, t_{i-1}]$ and the approximate top is the largest block that does so. Therefore, the approximation errs on the side of adjacency, meaning it can only make mistakes when assigning an M or S orientation; if it assigns a D, it is always correct. Let us consider the false M case (the false S case is similar). If we assign a false M, then $t_{i-1} < s_i$ and s_i is adjacent to the approximate top; therefore, all positions between t_{i-1} and s_i are covered. However, since the M is false, the true top of the stack must end at some $t' : t_{i-1} \leq t' < s_i$. Since we know that every position between t' and s_i is covered, $[s_i, t_i]$ cannot be PA to the true top of the stack, and we must be in the midst of making a non-ITG permutation. See Figure 8 for an illustration of the various indices involved. As it turns out, both the approximation and the 2-reducing parser assign incorrect orientations only in the presence of ITG violations. However, the approximation may be preferable, as it requires only a coverage vector.

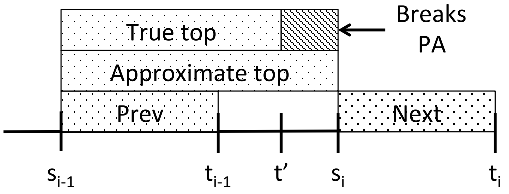

Figure 8: Indices for when the coverage approximation predicts a false M.

4.4 Qualitative comparison

Each solution manages its stack differently, and we illustrate the differences in terms of the top of the stack at time i in Table 2. The *-reducing parser is the gold standard, so we highlight deviations from its decisions in bold. As one can see, the original 2-reducing parser does fine before and during an ITG violation, but can create false disjoint orientations after the violation is complete, as the top of its stack becomes too small due to missing reductions. Conversely, the coverage-vector approximation makes errors inside the violation: the approximate top becomes too large, potentially creating false monotone or swap orientations. Once the violation is complete, it recovers nicely.

5 Experiments

We compare the LRM, the HRM and the three HRM variants suggested in Section 4 on a Chinese-to-English translation task. We measure the impact on translation quality in terms of BLEU score (Papineni et al., 2002), as well as the impact on permutation

[5] Zhang and Gildea (2007) provide an efficient algorithm for *-reduction that uses additional book-keeping so that the number of permutation checks as one traverses the entire sequence is linear in aggregate; however, we implement the simpler, less efficient version here to simplify decoder integration.

	BLEU			NIST 08 Complexity Counts						Speed
Method	nist04	nist06	nist08	> 2	4	5	6	7	≥ 8	sec/sent
LRM	38.00	33.79	27.12	241	146	40	32	12	11	3.187
HRM 2-red	38.53	34.20	27.57	176	113	31	20	8	4	3.353
HRM apprx	38.58	34.09	27.60	280	198	41	26	13	2	3.231
HRM *-red	38.39	34.22	27.41	328	189	71	34	20	14	3.585
HRM itg	38.70	34.26	27.33	0	0	0	0	0	0	3.274

Table 3: Chinese-to-English translation results, comparing the LRM and 4 HRM variants: the original 2-reducing parser, the coverage vector approximation, the *-reducing parser, and an ITG-constrained decoder.

complexity, as measured by the largest k required to k-reduce the translations.

5.1 Data

The system was trained on data from the NIST 2009 Chinese MT evaluation, consisting of more than 10M sentence pairs. The training corpora were split into two phrase tables, one for Hong Kong and UN data, and one for all other data. The dev set was taken from the NIST 05 evaluation set, augmented with some material reserved from other NIST corpora; it consists of 1.5K sentence pairs. The NIST 04, 06, and 08 evaluation sets were used for testing.

5.2 System

We use a phrase-based translation system similar to Moses (Koehn et al., 2007). In addition to our 8 translation model features (4 for each phrase table), we have a distortion penalty incorporating the minimum possible completion cost described by Moore and Quirk (2007), a length penalty, a 5-gram language model trained on the NIST09 Gigaword corpus, and a 4-gram language model trained on the target half of the parallel corpus. The LRM and HRM are represented with six features, with separate weights for M, S and D in both directions (Koehn et al., 2007). We employ a gap constraint as our only distortion limit (Chang and Collins, 2011). This restricts the maximum distance between the start of a phrase and the earliest uncovered word, and is set to 7 words. Parameters are tuned using a batch-lattice version of hope-fear MIRA (Chiang et al., 2008; Cherry and Foster, 2012). We re-tune parameters for each variant.

5.3 Results

Our results are summarized in Table 3. Speed and complexity are measured on the NIST08 test set, which has 1357 sentences. We measure permutation complexity by parsing the one-best derivations from each system with an external *-reducing parser, and noting the largest k-reduction for each derivation. Therefore, the $>$2 column counts the number of non-ITG derivations produced by each system.

Regarding quality, we have verified the effectiveness of the HRM: each HRM variant outperforms the LRM, with the 2-reducing HRM doing so by 0.4 BLEU points on average. Unlike Feng et al. (2010), we see no consistent benefit from adding hard ITG constraints, perhaps because we are building on an HRM-enabled system. In fact, all HRM variants perform more or less the same, with no clear winner emerging. Interestingly, the approximate HRM is included in this pack, which implies that groups wishing to augment their phrase-based decoder with an HRM need not incorporate a shift-reduce parser.

Regarding complexity, the 2-reducing HRM produces about half as many non-ITG derivations as the *-reducing system, confirming our hypothesis that a 2-reducing HRM acts as a sort of soft ITG constraint. Both the approximate and *-reducing decoders produce more violating derivations than the LRM. This is likely due to their encouragement of more movement overall. The largest reduction we observed was $k = 11$.

Our speed tests show that all of the systems translate at roughly the same speed, with the LRM being fastest and the *-reducing HRM being slowest. The *-reducing system is less than 7% slower than the 2-reducing system, alleviating our concerns regarding the cost of *-reduction.

6 Discussion

We have presented a number of theoretical contributions on the topic of phrase-based decoding with an on-board permutation parser. In particular, we have shown that the coverage-vector ITG constraint is actually incomplete, and that the original HRM can produce inconsistent orientations in the presence of ITG violations. We have presented three HRM variants that address these inconsistencies, and we have compared them in terms of both translation quality and permutation complexity. Though our results indicate that a permutation parser is actually unnecessary to reap the benefits of hierarchical re-ordering, we are excited about the prospects of further exploring the information provided by these on-board parsers. In particular, we are interested in using features borrowed from transition-based parsing while decoding.

References

Yin-Wen Chang and Michael Collins. 2011. Exact decoding of phrase-based translation models through lagrangian relaxation. In *EMNLP*, pages 26–37, Edinburgh, Scotland, UK., July.

Colin Cherry and George Foster. 2012. Batch tuning strategies for statistical machine translation. In *HLT-NAACL*, Montreal, Canada, June.

David Chiang, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *EMNLP*, pages 224–233.

Yang Feng, Haitao Mi, Yang Liu, and Qun Liu. 2010. An efficient shift-reduce decoding algorithm for phrase-based machine translation. In *COLING*, pages 285–293, Beijing, China, August.

Michel Galley and Christopher D. Manning. 2008. A simple and effective hierarchical phrase reordering model. In *EMNLP*, pages 848–856, Honolulu, Hawaii, October.

Michel Galley and Christopher D. Manning. 2010. Accurate non-hierarchical phrase-based translation. In *HLT-NAACL*, pages 966–974, Los Angeles, California, June.

Kevin Knight. 1999. Squibs and discussions: Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615, December.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *HLT-NAACL*, pages 127–133.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *ACL*, pages 177–180, Prague, Czech Republic, June.

Robert C. Moore and Chris Quirk. 2007. Faster beam-search decoding for phrasal statistical machine translation. In *MT Summit XI*, September.

Vinh Van Nguyen, Akira Shimazu, Minh Le Nguyen, and Thai Phuong Nguyen. 2009. Improving a lexicalized hierarchical reordering model using maximum entropy. In *MT Summit XII*, Ottawa, Canada, August.

Franz Josef Och and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4), December.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318.

Christoph Tillmann. 2004. A unigram orientation model for statistical machine translation. In *HLT-NAACL*, pages 101–104, Boston, USA, May.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403.

Richard Zens and Hermann Ney. 2003. A comparative study on reordering constraints in statistical machine translation. In *ACL*, pages 144–151.

Richard Zens, Hermann Ney, Taro Watanabe, and Eiichiro Sumita. 2004. Reordering constraints for phrase-based statistical machine translation. In *COLING*, pages 205–211, Geneva, Switzerland, August.

Hao Zhang and Daniel Gildea. 2007. Factorization of synchronous context-free grammars in linear time. In *Proceedings of SSST, NAACL-HLT 2007 / AMTA Workshop on Syntax and Structure in Statistical Translation*, pages 25–32, Rochester, New York, April.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *HLT-NAACL*, pages 256–263, New York City, USA, June.