Learning Linear Ordering Problems for Better Translation*

Roy Tromble
Google, Inc.
4720 Forbes Ave.
Pittsburgh, PA 15213
royt@google.com

Jason Eisner
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
jason@cs.jhu.edu

Abstract

We apply machine learning to the Linear Ordering Problem in order to learn sentence-specific reordering models for machine translation. We demonstrate that even when these models are used as a mere preprocessing step for German-English translation, they significantly outperform Moses' integrated lexicalized reordering model.

Our models are trained on automatically aligned bitext. Their form is simple but novel. They assess, based on features of the input sentence, how strongly each pair of input word tokens w_i, w_j would like to reverse their relative order. Combining all these pairwise preferences to find the best global reordering is NP-hard. However, we present a non-trivial $O(n^3)$ algorithm, based on chart parsing, that at least finds the best reordering within a certain exponentially large neighborhood. We show how to iterate this reordering process within a local search algorithm, which we use in training.

1 Introduction

Machine translation is an important but difficult problem. One of the properties that makes it difficult is the fact that different languages express the same concepts in different orders. A machine translation system must therefore rearrange the source language concepts to produce a fluent translation in the target language.

Phrase-based translation systems rely heavily on the target language model to ensure a fluent output order. However, a target n-gram language model alone is known to be inadequate. Thus, translation systems should also look at *how the source sentence prefers to reorder*. Yet past systems have traditionally used rather weak models of the reordering process. They may look only at the distance between neighboring phrases, or depend only on phrase unigrams. The decoders also rely on search error, in the form of limited reordering windows, for both efficiency and translation quality.

Demonstrating the inadequacy of such approaches, Al-Onaizan and Papineni (2006) showed that even *given* the words in the reference translation, *and* their alignment to the source words, a decoder of this sort charged with merely rearranging them into the correct target-language order could achieve a BLEU score (Papineni et al., 2002) of at best 69%—and that only when restricted to keep most words very close to their source positions.

This paper introduces a more sophisticated model of reordering based on the Linear Ordering Problem (LOP), itself an NP-hard permutation problem. We apply machine learning, in the form of a modified perceptron algorithm, to learn parameters of a linear model that constructs a matrix of weights from each source language sentence. We train the parameters on orderings derived from automatic word alignments of parallel sentences.

The LOP model of reordering is a complete ordering model, capable of assigning a different score to every possible permutation of the source-language sentence. Unlike the target language model, it uses information about the relative positions of the words in the source language, as well as the source words themselves and their parts of speech and contexts. It is therefore a language-pair specific model.

[1]This work is excerpted and adapted from the first author's Ph.D. thesis (Tromble, 2009). Some of the ideas here appeared in (Eisner and Tromble, 2006) without empirical validation. The material is based in part upon work supported by the National Science Foundation under Grant No. 0347822.

Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 1007–1016,
Singapore, 6-7 August 2009.

We apply the learned LOP model as a preprocessing step before both training and evaluation of a phrase-based translation system, namely Moses. Our methods for finding a good reordering under the NP-hard LOP are themselves of interest, adapting algorithms from natural language parsing and developing novel dynamic programs.

Our results demonstrate a significant improvement over translation using unreordered German. Using Moses with only distance-based reordering and a distortion limit of 6, our preprocessing improves BLEU from 25.27 to 26.40. Furthermore, that improvement is significantly greater than the improvement Moses achieves with its lexicalized reordering model, 25.55.

Collins et al. (2005) improved German-English translation using a statistical parser and several hand-written rules for preprocessing the German sentences. This paper presents a similar improvement using fully automatic methods.

2 A Linear Ordering Model

This section introduces a model of word reordering for machine translation based on the Linear Ordering Problem.

2.1 Formalization

The input sentence is $\boldsymbol{w} = w_1 w_2 \ldots w_n$. To distinguish duplicate tokens of the same word, we assume that each token is superscripted by its input position, e.g., $\boldsymbol{w} =$ *die*^{1} *Katze*^{2} *hat*^{3} *die*^{4} *Frau*^{5} *gekauft*^{6} (gloss: "the cat has the woman bought").

For a fixed $\boldsymbol{w}$, a permutation $\boldsymbol{\pi} = \pi_1 \pi_2 \ldots \pi_n$ is any reordering of the tokens in $\boldsymbol{w}$. The set Π_n of all such permutations has size $n!$. We would like to define a scoring model that assigns a high score to the permutation $\boldsymbol{\pi} =$ *die*^{4} *Frau*^{5} *hat*^{3} *gekauft*^{6} *die*^{1} *Katze*^{2} (gloss: "the woman has bought the cat"), since that corresponds well to the desired English order.

To construct a function that scores permutations of $\boldsymbol{w}$, we first construct a pairwise preference matrix $B_{\boldsymbol{w}} \in \mathbb{R}^{n \times n}$, whose entries are

$$B_{\boldsymbol{w}}[\ell, r] \stackrel{\text{def}}{=} \boldsymbol{\theta} \cdot \boldsymbol{\phi}(\boldsymbol{w}, \ell, r), \qquad (1)$$

Here $\boldsymbol{\theta}$ is a vector of weights. $\boldsymbol{\phi}$ is a vector of feature functions, each considering the entire word sequence $\boldsymbol{w}$, as well as any functions thereof, such as part of speech tags.

We will hereafter abbreviate $B_{\boldsymbol{w}}$ as B. Its integer indices ℓ and r are identified with the input tokens w_ℓ and w_r, and it can be helpful to write them that way; e.g., we will sometimes write $B[2, 5]$ as $B[\textit{Katze}^2, \textit{Frau}^5]$.

The idea behind our reordering model is that $B[\textit{Katze}^2, \textit{Frau}^5] > B[\textit{Katze}^5, \textit{Frau}^2]$ expresses a preference to keep *Katze*^{2} before *Frau*^{5}, whereas the opposite inequality would express a preference—other things equal—for permutations in which their order is reversed. Thus, we define[1]

$$\text{score}(\boldsymbol{\pi}) \stackrel{\text{def}}{=} \sum_{i,j:\ 1 \le i < j \le n} B[\pi_i, \pi_j] \qquad (2)$$

$$p(\boldsymbol{\pi}) \stackrel{\text{def}}{=} \frac{1}{Z} \exp(\gamma \cdot \text{score}(\boldsymbol{\pi})) \qquad (3)$$

$$\hat{\boldsymbol{\pi}} \stackrel{\text{def}}{=} \operatorname*{argmax}_{\boldsymbol{\pi} \in \Pi_n} \text{score}(\boldsymbol{\pi}) \qquad (4)$$

Note that i and j denote positions in $\boldsymbol{\pi}$, whereas π_i, π_j, ℓ, and r denote particular input tokens such as *Katze*^{2} and *Frau*^{5}.

2.2 Discussion

To the extent that the costs B generally discourage reordering, they will particularly discourage long-distance movement, as it swaps more pairs of words.

We point out that our model is somewhat peculiar, since it does not directly consider whether the permutation $\boldsymbol{\pi}$ keeps *die*^{4} and *Frau*^{5} adjacent or even close together, but only whether their order is reversed.

Of course, the model could be extended to consider adjacency, or more generally, the three-way cost of interposing k between i and j. See (Eisner and Tromble, 2006; Tromble, 2009) for such extensions and associated algorithms.

However, in the present paper we focus on the model in the simple form (2) that only considers pairwise reordering costs for all pairs in the sentence. Our goal is to show that these unfamiliar pairwise reordering costs are useful, when modeled with a rich feature set via equation (1). Even *in isolation* (as a preprocessing step), without considering any other kinds of reordering costs or language model, they can achieve useful reorderings

[1] For any $\ell < r$, we may assume without loss of generality that $B[r, \ell] = 0$, since if not, subtracting $B[r, \ell]$ from both $B[\ell, r]$ and $B[r, \ell]$ (exactly one of which appears in each $\text{score}(\boldsymbol{\pi})$) will merely reduce the scores of *all* permutations by this amount, leaving equations (3) and (4) unchanged. Thus, in practice, we take B to be an upper triangular matrix. We use equation (1) only to define $B[\ell, r]$ for $\ell < r$, and train $\boldsymbol{\theta}$ accordingly. However, we will ignore this point in our exposition.

of German that complement existing techniques and thus improve state-of-the-art systems. Our positive results in even this situation suggest that in future, pairwise reordering costs should probably be integrated into MT systems.

The probabilistic interpretation (3) of the score (2) may be useful when thus integrating our model with language models or other reordering models during translation, or simply when training our model to maximize likelihood or minimize expected error. However, in the present paper we will stick to purely discriminative training and decoding methods that simply try to maximize (2).

2.3 The Linear Ordering Problem

In the combinatorial optimization literature, the maximization problem (4) (with input B) is known as the Linear Ordering Problem. It has numerous practical applications in fields including economics, sociology, graph theory, graph drawing, archaeology, and task scheduling (Grötschel et al., 1984). Computational studies on real data have often used "input-output" matrices representing resource flows among economic sectors (Schiavinotto and Stützle, 2004).

Unfortunately, the problem is NP-hard. Furthermore, it is known to be APX-complete, meaning that there is no polynomial time approximation scheme unless P=NP (Mishra and Sikdar, 2004). However, there are various heuristic procedures for approximating it (Tromble, 2009). We now give an attractive, novel procedure, which uses a CKY-parsing-like algorithm to search various subsets of Π_n in polynomial time.

3 Local Search

"Local search" refers to any hill-climbing procedure that iteratively improves a solution by making an optimal "local" change at each iteration.[2] In this case, we start with the identity permutation, find a "nearby" permutation with a better score (2), and repeat until we have reached a local maximum of the scoring objective.

This section describes a local search procedure that uses a very generous definition of "local." At each iteration, it finds the optimal permutation in a certain *exponentially large* neighborhood $N(\boldsymbol{\pi})$ of the current permutation $\boldsymbol{\pi}$.

[2]One can introduce randomness to obtain MCMC sampling or simulated annealing algorithms. Our algorithms extend naturally to allow this (cf. Tromble (2009)).

$$\begin{aligned} S &\rightarrow S_{0,n} \\ S_{i,k} &\rightarrow S_{i,j}\ S_{j,k} \\ S_{i-1,i} &\rightarrow \pi_i \end{aligned}$$

Figure 1: A grammar for a large neighborhood of permutations, given one permutation $\boldsymbol{\pi}$ of length n. The $S_{i,k}$ rules are instantiated for each $0 \leq i < j < k \leq n$, and the $S_{i-1,i}$ rules for each $0 < i \leq n$.

We say that two permutations are neighbors iff they can be aligned by an Inversion Transduction Grammar (ITG) (Wu, 1997), which is a familiar reordering device in machine translation. Equivalently, $\boldsymbol{\pi}' \in N(\boldsymbol{\pi})$ iff $\boldsymbol{\pi}$ can be transformed into $\boldsymbol{\pi}'$ by swapping various adjacent substrings of $\boldsymbol{\pi}$, as long as these swaps are properly nested. Zens and Ney (2003) used a normal form to show that the size of the ITG neighborhood $N(\boldsymbol{\pi})$ is a large Schröder number, which grows exponentially in n. Asymptotically, the ratio between the size of the neighborhood for $n+1$ and the size for n approaches $3 + 2\sqrt{2} \approx 5.8$.

We show that equation (2) can be optimized within $N(\boldsymbol{\pi})$ in $O(n^3)$ time, using dynamic programming. The algorithm is based on CKY parsing. However, a novelty is that the grammar weights must themselves be computed by $O(n^3)$ dynamic programming.

Our grammar is shown in Figure 1. Parsing the "input sentence" $\boldsymbol{\pi}$ with this grammar simply constructs all binary trees that yield the string $\boldsymbol{\pi}$. There is essentially only one nonterminal, S, but we split it into $O(n^2)$ position-specific nonterminals such as $S_{i,j}$, which can only yield the span $\pi_{i+1}\pi_{i+2}\ldots\pi_j$. An example parse is shown in Figure 2.

The important point is that we will place a score on each binary grammar rule. The score of the rule $S_{i,k} \rightarrow S_{i,j}\ S_{j,k}$ is $\max(0, \Delta_{i,j,k})$, where $\Delta_{i,j,k}$ is the benefit to swapping the substrings $\pi_{i+1}\pi_{i+2}\ldots\pi_j$ and $\pi_{j+1}\pi_{j+2}\ldots\pi_k$. The rule is considered to be a "swap rule" if its score is positive, showing that a swap will be beneficial (independent of the rest of the tree). If the parse in Figure 2 is the parse with the highest *total* score, and its swap rules are $S_{0,5} \rightarrow S_{0,1}\ S_{1,5}$ and $S_{3,5} \rightarrow S_{3,4}\ S_{4,5}$, then our best permutation in the neighborhood of $\boldsymbol{\pi}$ must be the (linguistically desirable) permutation *die*^{4}*Frau*^{5}*hat*^{3}*gekauft*^{6}*die*^{1}*Katze*^{2}, obtained from

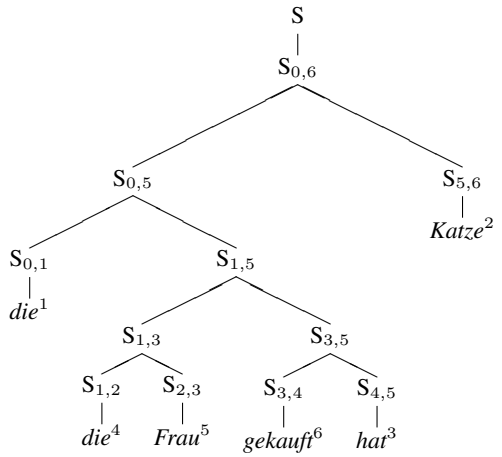

Figure 2: One parse of the current permutation $\boldsymbol{\pi}$. In this example, $\boldsymbol{\pi}$ has somehow gotten the input words into alphabetical order (owing to previous hill-climbing steps). We are now trying to further improve this order.

$\boldsymbol{\pi}$ by two swaps.

How do we find this solution? Clearly the benefit (positive or negative) to swapping $\pi_{i+1}\pi_{i+2}\ldots\pi_j$ with $\pi_{j+1}\pi_{j+2}\ldots\pi_k$ is

$$\Delta_{i,j,k} = \sum_{\ell=i+1}^{j}\sum_{r=j+1}^{k} B[\pi_r, \pi_\ell] - B[\pi_\ell, \pi_r] \quad (5)$$

We can evaluate all $O(n^3)$ possible swaps in total time $O(n^3)$, using the dynamic programming recurrence

$$\begin{aligned}\Delta_{i,j,k} &= \Delta_{i,j,k-1} + \Delta_{i+1,j,k} - \Delta_{i+1,j,k-1} \quad (6)\\ &\quad + B[\pi_k, \pi_{i+1}] - B[\pi_{i+1}, \pi_k]\end{aligned}$$

with the base case $\Delta_{i,j,k} = 0$ if $i = j$ or $j = k$. This gives us the weights for the grammar rules, and then we can use weighted CKY parsing to find the highest-scoring (Viterbi) parse in $O(n^3)$ time. Extracting our new and improved permutation $\boldsymbol{\pi'} \in N(\boldsymbol{\pi})$ from this parse is a simple $O(n)$-time algorithm.

Figure 3 gives pseudocode for our local search algorithm, showing how to compute the quantities (6) during parsing rather than beforehand. $\beta[i,k]$ holds the weight of the best permutation (in the neighborhood) of the subsequence $\pi_{i+1}\pi_{i+1}\ldots\pi_k$.[3]

[3]The use of β is intended to suggest an analogy to inside probability—or more precisely, the Viterbi approximation to inside probability (since we are maximizing rather than summing over parses).

The next two sections describe how to use our local search algorithm to discriminatively learn the weights of the parameters from Section 2, equation (1).

4 Features

Our objective function (2) works only to the extent that we can derive a good pairwise preference matrix $B_{\boldsymbol{w}}$. We do this by using a rich feature set in equation (1).

We adapt the features of McDonald et al. (2005), introduced there for dependency parsing, to the task of machine translation reordering. Because both models construct features for pairs of words given the entire sentence, there is a close correspondence between the two tasks, although the output is quite different.

Each feature $\phi(\boldsymbol{w}, \ell, r)$ in equation (1) is a binary feature that fires when $(\boldsymbol{w}, \ell, r)$ has some conjunction of properties. The properties that are considered include the words w_ℓ and w_r, the parts of speech of $\{w_{\ell-1}, \ldots, w_{r+1}\}$, and the distance $r - \ell$. Table 1 shows the feature templates.

We also tried features based on a dependency parse of the German, with the idea of using LOP features to reorder the dependents of each word, and thus model syntactic movement. This did produce better monolingual reorderings (as in Table 2), but it did not help final translation into English (Table 3), so we do not report the details here.

5 Learning to Reorder

Ideally, we would have a large corpus of desirable reorderings of source sentences—in our case, German sentences permuted into target English word order—from which to train the parameters of our model. Unfortunately, the alignments between German and English sentences are only infrequently one-to-one. Furthermore, human-aligned parallel sentences are hard to come by, and never in the quantity we would like.

Instead, we make do with automatically-generated word alignments, and we heuristically derive an English-like word order for the German sentence based on the alignment. We used GIZA++ (Och and Ney, 2003) to align approximately 751,000 sentences from the German-English portion of the Europarl corpus (Koehn, 2005), in both the German-to-English and English-to-German directions. We combined the

```
1: procedure LocalSearchStep(B, π, n)
2:     for i ← 0 to n − 1 do
3:         β[i, i + 1] ← 0
4:         for k ← i + 1 to n do
5:             Δ[i, i, k] ← Δ[i, k, k] ← 0
6:         end for
7:     end for
8:     for w ← 2 to n do
9:         for i ← 0 to n − w do
10:            k ← i + w
11:            β[i, k] ← −∞
12:            for j ← i + 1 to k − 1 do
13:                Δ[i, j, k] ← Δ[i, j, k − 1] + Δ[i + 1, j, k] − Δ[i + 1, j, k − 1] + B[π_k, π_{i+1}] − B[π_{i+1}, π_k]
14:                β[i, k] ← max(β[i, k], β[i, j] + β[j, k] + max(0, Δ[i, j, k]))
15:            end for
16:        end for
17:    end for
18:    return β[0, n]
19: end procedure
```

Figure 3: Pseudocode for computing the score of the best permutation in the neighborhood of $\boldsymbol{\pi}$ under the Linear Ordering Problem specified by the matrix B. Computing the best neighbor is a simple matter of keeping back pointers to the choices of $\max$ and ordering them as implied.

alignments using the "grow-diag-final-and" procedure provided with Moses (Koehn et al., 2007).

For each of these German sentences, we derived the English-like reordering of it, which we call German$'$, by the following procedure. Each German token was assigned an integer key, namely the position of the leftmost of the English tokens to which it was aligned, or 0 if it was not aligned to any English tokens. We then did a stable sort of the German tokens based on these keys, meaning that if two German tokens had the same key, their order was preserved.

This is similar to the oracle ordering used by Al-Onaizan and Papineni (2006), but differs in the handling of unaligned words. They kept unaligned words with the closest preceding aligned word.[4]

Having found the German$'$ corresponding to each German sentence, we randomly divided the sentences into 2,000 each for development and evaluation, and the remaining approximately 747,000 for training.

We used the averaged perceptron algorithm (Freund and Schapire, 1998; Collins, 2002) to train the parameters of the model. We ran the algorithm multiple times over the training sentences, measuring the quality of the learned parameters by reordering the held-out development set after each iteration. We stopped when the BLEU score on the development set failed to improve for two consecutive iterations, which occurred after fourteen passes over the data.

Each perceptron update should compare the true German$'$ to the German$'$ that would be predicted by the model (2). As the latter is NP-hard to find, we instead substitute the local maximum found by local search as described in Section 3, starting at the identity permutation, which corresponds to the original German word order.

During training, we iterate the local search as described earlier. However, for decoding, we only do a single step of local search, thus restricting reorderings to the ITG neighborhood of the original German. This restriction turns out to improve performance slightly, even though it reduces the quality of our approximation to the LOP problem (4). In other words, it turns out that reorderings found outside the ITG neighborhood tend to be poor German$'$ even if our LOP-based objective function thinks that they are good German$'$.

This is not to say that the gold standard German$'$ is always in the ITG neighborhood of the original German—often it is not. Thus, it might be better in future work to still allow the local search to take more than one step, but to penalize the second step. In effect, score($\boldsymbol{\pi}$) would then include a feature indicating whether $\boldsymbol{\pi}$ is in the neighborhood of the original German.

[4] We tried two other methods for deriving English word order from word alignments. The first alternative was to align only in one direction, from English to German, with null alignments disallowed, so that every German word was aligned to a single English word. The second alternative used BerkeleyAligner (Liang et al., 2006; DeNero and Klein, 2007), which shares information between the two alignment directions to improve alignment quality. Neither alternative produced improvements in our ultimate translation quality.

$t_{\ell-1}$	w_ℓ	t_ℓ	$t_{\ell+1}$	t_b	t_{r-1}	w_r	t_r	t_{r+1}
	×	×				×	×	
	×	×				×		
	×					×	×	
	×	×					×	
		×				×	×	
	×					×		
		×					×	
	×	×						
						×	×	
	×							
						×		
		×						
							×	
		×		×			×	
		×	×		×		×	
		×	×				×	
		×	×				×	×
		×					×	×
×		×					×	×
×		×					×	
×		×			×		×	
		×			×		×	

Table 1: Feature templates for $B[\ell, r]$ (w_ℓ is the ℓth word, t_ℓ its part of speech tag, and b matches any index such that $\ell < b < r$). Each of the above is also conjoined with the distance between the words, $r - \ell$, to form an additional feature template. Distances are binned into 1, 2, 3, 4, 5, > 5, and > 10.

The model is initialized at the start of training using log-odds of the parameters. Let $\Phi_m = \{(\boldsymbol{w}, \ell, r) \mid \phi_m(\boldsymbol{w}, \ell, r) = 1\}$ be the set of word pairs in the training data for which feature m fires. Let $\vec{\Phi}_m$ be the subset of Φ_m for which the words stay in order, and $\overleftarrow{\Phi}_m$ the subset for which the words reverse order. Then in this model,

$$\theta_m = \log\left(\left|\vec{\Phi}_m\right| + \frac{1}{2}\right) - \log\left(\left|\overleftarrow{\Phi}_m\right| + \frac{1}{2}\right). \quad (7)$$

This model is equivalent to smoothed naïve Bayes if converted to probabilities. The learned model significantly outperforms it on the monolingual reordering task.

Table 2 compares the model after perceptron training to the model at the start of training, measuring BLEU score of the predicted German′ against the observed German′. In addition to these BLEU scores, we can measure precision and recall of pairs of reordered words against the observed German′. On the held out test set, the predicted German′ achieves a recall of only 21%, but a precision of 64%. Thus, the learned model is too conservative, but makes moderately good decisions when it does reorder.

Ordering	p_2	p_3	p_4	**BLEU**
German	57.4	38.3	27.7	49.65
Log-odds	57.4	38.4	27.8	49.75
Perceptron	**58.6**	**40.3**	**29.8**	**51.51**

Table 2: Monolingual BLEU score on development data, measured against the "true" German′ ordering that was derived from automatic alignments to known English translations. The table evaluates three candidate orderings: the original German, German reordered using the log-odds initialized model, and German reordered using the perceptron-learned model. In addition to the BLEU score, the table shows bigram, trigram, and 4-gram precisions. The unigram precisions are always 100%, because the correct words are given.

6 Reordering as Preprocessing

This section describes experiments using the model introduced in Section 2 and learned in Section 5 to preprocess German sentences for translation into English. These experiments are similar to those of Collins et al. (2005).

We used the model learned in Section 5 to generate a German′ ordering of the training, development, and test sets. The training sentences are the same that the model was trained on, and the development set is the same that was used as the stopping criterion for the perceptron. The test set was unused in training.

We used the resulting German′ as the input to the Moses training pipeline. That is, Moses recomputed alignments of the German′ training data to the English sentences using GIZA++, then constructed a phrase table. Moses used the development data for minimum error-rate training (Och, 2003) of its small number of parameters. Finally, Moses translated the test sentences, and we measured performance against the English reference sentences. This is the standard Moses pipeline, except German has been replaced by German′.

Table 3 shows the results of translation, both starting with unreordered German, and starting with German′, reordered using the learned Linear Ordering Problems. Note that Moses may itself re-

System	Input	Moses Reord.	p_1	p_2	p_3	p_4	BLEU	METEOR	TER
baseline	German	Distance	59.6	31.4	18.8	11.6	25.27	54.03	60.60
(a)	German	Lexical	60.0	32.0	19.3	12.1	25.55	54.18	59.76
(b)	German′	Distance	60.4	32.7	20.2	12.8	26.40	**54.91**	**58.63**
(a)+(b)	German′	Lexical	59.9	32.4	20.0	12.8	**26.44**	54.61	59.23

Table 3: Machine translation performance of several systems, measured against a single English reference translation. The results vary both the preprocessing—either none, or reordered using the learned Linear Ordering Problems—and the reordering model used in Moses. Performance is measured using BLEU, METEOR (Lavie et al., 2004), and TER (Snover et al., 2006). (For TER, smaller values are better.)

order whatever input that it receives, during translation into English. Thus, the results in the table also vary the reordering model used in Moses, set to either a single-parameter distance-based model, or to the lexicalized bidirectional msd model. The latter model has six parameters for each phrase in the phrase table, corresponding to monotone, swapped, or discontinuous ordering relative to the previous phrase in either the source or target language.

How should we understand the results? The baseline system is Moses phrase-based translation with no preprocessing and only a simple distance-based reordering model. There are two ways to improve this: (a) ask Moses to use the lexicalized bidirectional msd reordering model that is provided with Moses and is integrated with the rest of translation, or (b) keep the simple distance-based model within Moses, but preprocess its training and test data with our linear reordering model. Note that the preprocessing in (b) will obviously change the phrasal substrings that are learned by Moses, for better or for worse.

First, remarkably, (b) is significantly better than (a) on BLEU, with $p < 0.0001$ according to a paired permutation test.

Second, combining (a) with (b) produced no improvement over (b) in BLEU score (the difference between 26.40 and 26.44 is not significant, even at $p < 0.2$, according to the same paired permutation test). Lexicalized reordering in Moses even degraded translation performance according to METEOR and TER. The TER change is significant according to the paired permutation test at $p < 0.001$. (We did not perform a significance test for METEOR.)

Our word-based model surpasses the lexicalized reordering in Moses largely because of long-distance movement. The 518 sentences (26%) in the test set for which our model moves a word more than six words away from its starting position account for more than 67% of the improvement in BLEU from (a) to (b).

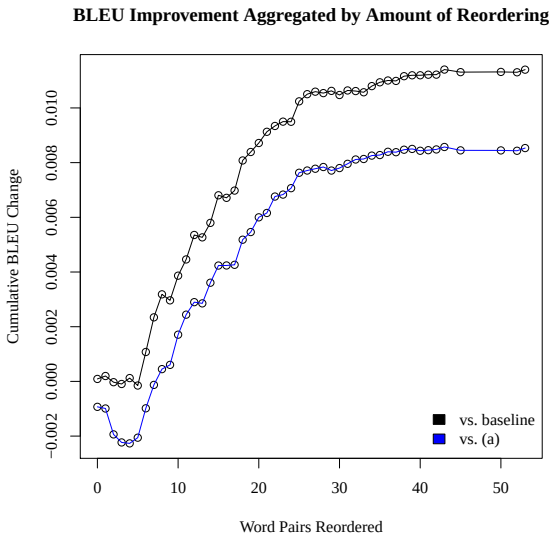

Figure 4: Cumulative change in BLEU score of (b) relative to the baseline and (a), aggregated by the number of reordered word pairs in each sentence. For those sentences where our model reorders fewer than five word pairs, the BLEU score of translation degrades.

Figure 4 shows another view of the BLEU improvement. It shows that, compared to the baseline, our preprocessing has basically no effect for sentences where it does only a little reordering, changing the relative order of fewer than five pairs of words. Compared to Moses with lexicalized reordering, these same sentences actually hurt performance. This more than accounts for the difference between the BLEU scores of (b) and (a)+(b).

Going beyond preprocessing, our model could also be integrated into a phrase-based decoder. We briefly sketch that possibility here.

Phrase-based decoders keep a source coverage vector with every partial translation hypothesis. That coverage vector allows us to incorporate the scores from a LOP matrix B directly. Whenever the decoder extends the hypothesis with a new source phrase, covering $w_{i+1}w_{i+2}\ldots w_j$, it adds

$$\sum_{\ell=i+1}^{j-1}\sum_{r=\ell+1}^{j} B[\ell,r] + \sum_{\ell=i+1}^{j}\sum_{r\in\mathcal{U}} B[\ell,r].$$

The first term represents the phrase-internal score, and the second the score of putting the words in the phrase before all the remaining uncovered words $\mathcal{U}$.

7 Comparison to Prior Work

Preprocessing the source language to improve translation is a common technique. Xia and McCord (2004) improved English-French translation using syntactic rewrite rules derived from Slot Grammar parses. Collins et al. (2005) reported an improvement from 25.2% to 26.8% BLEU on German-English translation using six handwritten rules to reorder the German sentences based on automatically-generated phrase-structure trees. Our work differs from these approaches in providing an explicit model that scores all possible reorderings. In this paper, our model was trained and used only for 1-best preprocessing, but it could potentially be integrated into decoding as well, where it would work together with the translation model and target language model to find a congenial translation.

Costa-jussà and Fonollosa (2006) improved Spanish-English and Chinese-English translation using a two-step process, first reordering the source language, then translating it, both using different versions of a phrase-based translation system. Many others have proposed more explicit reordering models (Tillmann, 2004; Kumar and Byrne, 2005; Koehn et al., 2005; Al-Onaizan and Papineni, 2006). The primary advantage of our model is that it directly accounts for interactions between distant words, leading to better treatment of long-distance movement.

Xiong et al. (2006) proposed a constituent reordering model for a bracketing transduction grammar (BTG) (Wu, 1995), which predicts the probability that a pair of subconstituents will reorder when combined to form a new constituent. The features of their model look only at the *first* source and target word of each constituent, making it something like a sparse version of our model. However, because of the target word features, their reordering model cannot be separated from their translation model.

8 Conclusions and Future Work

We have presented an entirely new model of reordering for statistical machine translation, based on the Linear Ordering Problem, and shown that it can substantially improve translation from German to English.

The model is demonstrably useful in this preprocessing setting—which means that it can be very simply added as a preprocessing step to any MT system. German-to-English is a particularly attractive use case, because the word orders are sufficiently different as to require a good reordering model that requires long-distance reordering. Our preprocessing here gave us a BLEU gain of 0.9 point over the best Moses-based result. English-to-German would obviously be another potential win, as would translating between English and Japanese, for example.

As mentioned in Section 6, our model could also be integrated into a phrase-based, or a syntax-based decoder. That possibility remains future work, but it is likely to lead to further improvements, because it allows the translation system to consider multiple possible reorderings under the model, as well as to tune the weight of the model relative to the other parts of the system during MERT.

Tromble (2009) covers this integration in more detail, and proposes several other ways of integrating our reordering model into machine translation. It also experiments with numerous other parameter estimation procedures, including some that use the probabilistic interpretation of our model from (3). It presents numerous additional neighborhoods for search in the Linear Ordering Problem.

We mentioned several possible extensions to the model, such as going beyond the scoring model of equation (2), or considering syntax-based features. Another extension would try to reorder not words but phrases, following (Xiong et al., 2006), or segment choice models (Kuhn et al., 2006), which assume a single segmentation of the words into phrases. We would have to define the pairwise preference matrix B over phrases rather than

words (Eisner and Tromble, 2006). This would have the disadvantage of complicating the feature space, but might be a better fit for integration with a phrase-based decoder.

Finally, we gave a novel algorithm for approximately solving the Linear Ordering Problem, interestingly combining dynamic programming with local search. Another novel contribution is that we showed how to parameterize a function that constructs a specific Linear Ordering Problem instance from an input sentence $\boldsymbol{w}$, and showed how to learn those parameters from a corpus of parallel sentences, using the perceptron algorithm. Likelihood-based training using equation (3) would also be possible, with modifications to our algorithm, notably the use of normal forms to avoid counting some permutations multiple times (Tromble, 2009).

It would be interesting to compare the speed and accuracy of our dynamic-programming local-search method with an exact algorithm for solving the LOP, such as integer linear programming with branch and bound (cf. Charon and Hudry (2006)). Exact solutions can generally be found in practice for $n \leq 100$.

References

Yaser Al-Onaizan and Kishore Papineni. 2006. Distortion models for statistical machine translation. In *COLING-ACL*, pages 529–536, Sydney, July.

Irène Charon and Olivier Hudry. 2006. A branch-and-bound algorithm to solve the linear ordering problem for weighted tournaments. *Discrete Applied Mathematics*, 154(15):2097–2116, October.

Michael Collins, Philipp Koehn, and Ivona Kučerová. 2005. Clause restructuring for statistical machine translation. In *ACL*, pages 531–540, Ann Arbor, Michigan, June.

Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, pages 1–8, Philadelphia, July.

Marta R. Costa-jussà and José A. R. Fonollosa. 2006. Statistical machine reordering. In *EMNLP*, pages 70–76, Sydney, July.

John DeNero and Dan Klein. 2007. Tailoring word alignments to syntactic machine translation. In *ACL*, pages 17–24, Prague, June.

Jason Eisner and Roy W. Tromble. 2006. Local search with very large-scale neighborhoods for optimal permutations in machine translation. In *Workshop on computationally hard problems and joint inference in speech and language processing*, New York, June.

Yoav Freund and Robert E. Schapire. 1998. Large margin classification using the perceptron algorithm. In *COLT*, pages 209–217, New York. ACM Press.

Martin Grötschel, Michael Jünger, and Gerhard Reinelt. 1984. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, November–December.

Philipp Koehn, Amittai Axelrod, Alexandra Birch Mayne, Chris Callison-Burch, Miles Osborne, and David Talbot. 2005. Edinburgh system description for the 2005 IWSLT speech translation evaluation. In *IWSLT*, Pittsburgh, October.

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 Demo and Poster Sessions*, pages 177–180, Prague, June.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT Summit X*, pages 79–86, Phuket, Thailand, September.

Roland Kuhn, Denis Yuen, Michel Simard, Patrick Paul, George Foster, Eric Joanis, and Howard Johnson. 2006. Segment choice models: Feature-rich models for global distortion in statistical machine translation. In *HLT-NAACL*, pages 25–32, New York, June.

Shankar Kumar and William Byrne. 2005. Local phrase reordering models for statistical machine translation. In *HLT-EMNLP*, pages 161–168, Vancouver, October.

Alon Lavie, Kenji Sagae, and Shyamsundar Jayaraman. 2004. The signicance of recall in automatic metrics for MT evaluation. In Robert E. Frederking and Kathryn B. Taylor, editors, *Machine Translation: From Real Users to Research*, pages 134–143. AMTA, Springer, September–October.

Percy Liang, Ben Taskar, and Dan Klein. 2006. Alignment by agreement. In *HLT-NAACL*, pages 104–111, New York, June.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Spanning tree methods for discriminative training of dependency parsers. Technical Report MS-CIS-05-11, UPenn CIS.

Sounaka Mishra and Kripasindhu Sikdar. 2004. On approximability of linear ordering and related NP-optimization problems on graphs. *Discrete Applied Mathematics*, 136(2–3):249–269, February.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, March.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *ACL*, pages 160–167, Sapporo, July.

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, Philadelphia, July.

Tommaso Schiavinotto and Thomas Stützle. 2004. The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4):367–402, December.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *AMTA*.

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

Roy Wesley Tromble. 2009. *Search and Learning for the Linear Ordering Problem with an Application to Machine Translation*. Ph.D. thesis, Johns Hopkins University, Baltimore, April. `http://nlp.cs.jhu.edu/~royt/`

Dekai Wu. 1995. An algorithm for simultaneously bracketing parallel texts by aligning words. In *ACL*, pages 244–251, Cambridge, Massachusetts, June.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404, September.

Fei Xia and Michael McCord. 2004. Improving a statistical MT system with automatically learned rewrite patterns. In *COLING*, pages 508–514, Geneva, August.

Deyi Xiong, Qun Liu, and Shouxun Lin. 2006. Maximum entropy based phrase reordering model for statistical machine translation. In *COLING-ACL*, pages 521–528, Sydney, July.

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