On the Expressivity of Linear Transductions

Markus Saers and **Dekai Wu**
Human Language Technology Center
Dept. of Computer Science and Engineering
Hong Kong University of Science and Technology
{masaers|dekai}@cs.ust.hk

Chris Quirk
Microsoft Research
One Microsoft Way, Redmond
Washington, USA
chrisq@microsoft.com

Abstract

We investigate the formal expressivity properties of linear transductions, the class of transductions generated by linear transduction grammars, linear inversion transduction grammars and preterminalized linear inversion transduction grammars. While empirical results such as those in previous work are of course an ultimate test of modeling adequacy for machine translation applications, it is equally important to understand the formal theoretical properties of any such new representation. An important part of the expressivity of a transduction is the possibility to align tokens between the two languages generated. We refer to the number of different alignments that are allowed under a transduction as its weak alignment capacity. This aspect of expressivity is quantified for linear transductions using preterminalized linear inversion transduction grammars, and compared to the expressivity of finite-state transductions, inversion transductions and syntax-directed transductions.

1 Introduction

We investigate the formal expressivity properties of linear transductions, the class of transductions generated by linear transduction grammars (Saers, 2011, LTGs), linear inversion transduction grammars (Saers et al., 2010, LITGs) and preterminalized LITGs (Saers and Wu, 2011, PLITGs). While empirical results such as those in previous work are of course an ultimate test of modeling adequacy for machine translation applications, it is equally important to understand the formal theoretical properties of any such new representation. In recent years, there has been a shift away from surface-based translation method such as phrase-based statistical machine translation (phrase-based SMT) (Marcu and Wong, 2002; Koehn et al., 2003) in favor of grammar-based SMT. Although most of the grammar based methods still rely on surface-based word alignments (Brown et al., 1993; Vogel et al., 1996) and language specific parsers, the grammar-based models themselves restrict reordering to a much higher degree than the surface-based methods, which typically allow any permutation of any segmentation of the input, and relies on heuristic search methods such as beam search, to restrict the exponential time to a tractable polynomial.

Although not always given the attention deserved, the expressivity of different grammar-based methods varies quite a lot. Unlike monolingual grammars, where all context-free grammars, for example, fall into one class of languages, the bilingual case is not as well-behaved. Syntax-directed transduction grammars (Lewis and Stearns, 1968; Aho and Ullman, 1972) form distinct transduction classes for all ranks above 3, while ranks 2 and 3 form a class of their own termed inversion transductions (Wu, 1997), and rank 1 forms the class termed linear transductions (Saers, 2011).

In this paper we will take a closer look at the expressive powers of transduction grammars in general, noting that the concept of generative capacity fails to capture all the relevant details. Instead, we will propose a division of the expressivity into a strong and weak *transductive capacity* and *align-*

Capacity		Required equality
Weak	Transductive	Sentence pairs
Strong		Biparse trees
Weak	Alignment	Token alignments
Strong		Compositional alignments

Table 1: Capacities of transduction grammars.

ment capacity. The argument for this division is given in our analysis of the expressive powers of transduction grammars in Section 2. Having established the analytical framework we will dive deeper into the *weak alignment capacity*, and how our definition of this new concept fits into the existing body of research (Section 3). After that we make a detailed analysis of the very same capacity for preterminalized linear inversion transduction grammars (Section 4). Finally, we offer some conclusions in Section 5.

2 Expressivity of transduction grammars

Noting that a transduction grammar generates two sentences instead of one, which a monolingual does, is technically correct, but also fails to capture the very essence of what a transduction grammar is. A transduction grammar not only generates two sentences, but also establishes a relation, not only between the sentences, but also between the parts of the sentences.

A monolingual grammar has a weak and strong *generative capacity*, corresponding to the sentences and sentences paired with analyses (parse trees) respectively, and not much attention has been paid to how these two concepts generalize to the bilingual case. In this paper we will argue that a finer distinction has to be made for transduction grammars. Instead of a weak and strong generative capacity, transduction grammars are characterized by a weak and strong *transductive capacity*, as well as a weak and strong *alignment capacity*. We believe that it is possible and indeed imperative to separate these concepts in order to correctly characterize different transduction grammars. Table 1 shows a summary of the different capacities, and the type of entities used to establish equivalence.

Transduction grammars are generative grammars that, from the start symbol, generate sentence pairs. As such it is tempting to apply the monolingual nomenclature and characterize them in terms of a weak generative capacity—the sentence pairs generated, and a strong generative capacity—the analyzed sentence pairs. This does, however, miss one crucial point: transduction grammars relate not only the full sentences to each other, but also their parts. In fact: each node in a biparse tree conveys two messages: "my yields are related" and "my immediate children should be independently ordered such that ..." This means that each node in the tree constitutes one point where the two sentences are aligned—an *alignment point*. By retaining only the alignment information of a tree we get a *compositional alignment*, which is related to the strong alignment capacity, and by retaining only the leaves of the compositional alignment we get a *token alignment*, which is related to the weak alignment capacity. Both of these alignments are given when a biparse tree of a sentence pair is known. Since they not only generate a sentence pair, but also this alignment information, we will use the term weak and strong *transductive capacity* to refer to the grammars capacity to generate sentence pairs and biparse trees. This serves two purposes; first, the strong transductive capacity of a transduction grammar includes its capacity to generate alignments, making it markedly different from the strong generative capacity of a monolingual grammar; second, a sentence pair where the pairing implies that a specific relation holds between the two sentence is markedly different from merely asserting membership.

3 Weak alignment capacity

In this section we operationalize the concept of weak alignment capacity for transduction grammars, and fit it to three classes of grammars: finite-state transduction grammar, inversion transduction grammars and syntax-directed transduction grammars.

The weak alignment capacity of a transduction grammar is tightly related to token alignments, so we will start by defining what we mean with the term token alignment. Since all transduction grammars considered in this paper can be put in a normal form where at most one token is produced in either language with any one rule, we will consider only grammars which have been normalized in this way. It has the benefit of sparing us from considering multiple segmentations of the same sentence pair when determining the weak alignment capacity of a grammar. The token-to-token alignments are matrices with dimensions equal to the two sentences being generated. Since the grammar may contain singletons (biterminals where one of the sides are empty) we may encounter sentence pairs where the two sentences are of different length. Singletons in themselves tell us nothing of the relation between the two languages, and add no information to the concept of weak alignment capacity, prompting us to exclude them from the alignments matrices. This leaves us with a bijection between the related tokens, which we will use as our operationalization of *token alignment*. The weak alignment capacity is simply the set of token alignments that a transduction grammar can generate.

Turning to the specific grammar classes, we will start with the most efficient grammar type: the finite-state transduction grammars (FSTGs). These are the grammar form of finite-state transducers, and generate finite-state transductions. Since they are inherently monotonic, the only token alignments that can be generated are perfectly straight diagonals. Given a grammar that can generate infinitely long sentence pairs, the set of these token alignments (one for every possible sentence length) is also infinitely large, which means that we cannot compare the absolute sizes of the sets. Instead we will observe how the number of token alignment grows as a function of their length, and for easy comparison to following grammars, we will express it as a recurrence formula:

$$a_1^{\mathfrak{F}} = 1, \qquad a_n^{\mathfrak{F}} = a_{n-1}^{\mathfrak{F}} + 1$$

Moving on to inversion transduction grammars (ITGs), we know from previous work (Wu, 1997; Huang et al., 2009) that the number of token alignments up to and including length n is equal to the n^{th} large Schröder numbers (Schröder, 1870), which can be expressed as:

$$a_1^{\mathfrak{I}} = 1, \quad a_2^{\mathfrak{I}} = 2, \quad a_n^{\mathfrak{I}} = \tfrac{6n-9}{n}\, a_{n-1}^{\mathfrak{I}} - \tfrac{n-3}{n}\, a_{n-2}^{\mathfrak{I}}$$

Finally, we have the arbitrary rank syntax-directed transduction grammars (SDTGs), which are capable of generating any permutation (Lewis and Stearns, 1968; Aho and Ullman, 1972). The number of permutations are $n!$, which we can also formulate as a recurrence formula:

$$a_1^{\mathfrak{T}} = 1, \qquad a_n^{\mathfrak{T}} = n a_{n-1}^{\mathfrak{T}}$$

It should be clear that these series grow at different paces, and that:

$$a_n^{\mathfrak{F}} < a_n^{\mathfrak{I}} < a_n^{\mathfrak{T}}$$

4 Weak alignment capacity of PLITGs

In this section we investigate the weak alignment capacity of preterminalized linear inversion transduction grammars (PLITGs). The grammar class was introduced in Saers and Wu (2011), as a way to treat phrasal bilexicon induction as a transduction grammar induction problem, treating the parallel corpus as a linear transduction. Linear transductions have been studied previously through linear inversion transduction grammars (Saers et al., 2010) and linear transduction grammars (Saers, 2011). It is reasonable to believe that the findings in this section apply equally to these grammar formalisms, since the equivalence of their weak transductive capacity has been established.

Definition 1. *A* PLITG *over languages L_1 and L_2 is a tuple $G = \langle N, P, \Sigma, \Delta, S, R\rangle$, where N is a finite, nonempty set of nonterminal symbols, P is a finite, nonempty set of preterminal symbols , Σ and Δ are the alphabets of L_1 and L_2 respectively, $S \in N$ is the designated start symbol and R is a finite, nonempty set of preterminal linear inversion transduction rules on the forms:*

$$\begin{array}{lll} A \to [BY], & A \to \langle BY\rangle, & A \to \epsilon/\epsilon, \\ A \to [YB], & A \to \langle YB\rangle, & X \to a/x \end{array}$$

where $A, B \in N$, $X, Y \in P$, $a \in \Sigma^*$ and $x \in \Delta^*$.

In this paper, we are working with a normal form, where a and x may only consist of zero or one tokens, and one of them must be nonempty for any rule.

To illustrate how the application of PLITG rules corresponds to the generation of a token alignment, consider the rule $A \to [YB]$. Whenever it is applied, we know that A corresponds to some token alignment, and the rule states that that token alignment will have a biterminal generated by Y in its top left corner, and the rest of it will be the token alignment generated by B. Graphically, we can view it as:

$$[A] \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & B \end{bmatrix}$$

The other three nonterminal rules can be viewed in the same way, but with the 1 in one of the other three corners.

We will denote the number of permutations in a linear transduction as $a_n^{\mathfrak{L}}$, and using a PLITG to generate them, we can easily determine the value for low ns by enumeration:

$$a_1^{\mathfrak{L}} = 1 : \begin{bmatrix} 1 \end{bmatrix}$$

$$a_2^{\mathfrak{L}} = 2 : \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$a_3^{\mathfrak{L}} = 6 : \begin{matrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \end{matrix}$$

At this point, the enumeration approach starts to be cumbersome, but so far we have managed to cover all possible permutations. We saw that PLITGs are limited to adding new preterminals (and thus biterminals, and thus alignment points) in the corners of the token alignments. Since we have, at $n = 3$, all possible corner configurations, we will work with generic corner configurations from now on.

The six different corner configurations can be classified into two different classes: those with two corners aligned (the maximum number of aligned corners in a token alignment) which we will call type A, and those with one corner aligned which we will call type B. The six different corner configurations can be found in Figure 2.

By rotating an A alignment 90° it becomes the other A alignment, which means that anything that holds for A_1 also holds for A_2 if rotated. The same rotation argument can be made for the B alignments. This is important when we want to count the total number of token alignments that a PLITG can generate—the counts have to be equal for all A alignments of a certain length, and all B alignments of that length. Using the equalities:

$$A_1 = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 0 \end{bmatrix},$$

$$B_1 = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix},$$

$$B_3 = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 0 \end{bmatrix}, \quad B_4 = \begin{bmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$$

Figure 2: The six different corner configurations that can be generated with a PLITG.

$$c(A_1, n) = c(A_2, n) = c(A, n)$$

and

$$c(B_1, n) = \cdots = c(B_4, n) = c(B, n)$$

The total number of permutations of length n can be calculated as:

$$a_n^{\mathfrak{L}} = 2c(A, n) + 4c(B, n) \qquad (1)$$

where $c(X, n)$ is the number of alignments of type X and length n.

Let us start by determining $c(A, n)$. This quantity depends on the number of different permutations of length $n-1$. Permutations of type A_1 can be formed from shorter permutations of type A_1 like this:

$$\begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$$

but not from permutations of type A_2. In fact, any A permutation can be formed from exactly one shorter A permutation. For any A permutation there is also a shorter B permutation it could be formed from:

$$\begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$$

$$\begin{aligned}
a_n^{\mathfrak{L}} &= 2c(A,n) + 4c(B,n)\\
&= 2(c(A,n-1) + c(B,n-1)) + 4(c(A,n-1) + 3c(B,n-1))\\
&= 2c(A,n-1) + 2c(B,n-1) + 4c(A,n-1) + 12c(B,n-1)\\
&= 6c(A,n-1) + 14c(B,n-1)\\
&= 8c(A,n-1) + 16c(B,n-1) - 2(c(A,n-1) + c(B,n-1))\\
&= 4(2c(A,n-1) + 4c(B,n-1)) - 2(c(A,n-1) + c(B,n-1))\\
&= 4a_{n-1}^{\mathfrak{L}} - 2(c(A,n-2) + c(B,n-2) + c(A,n-2) + 3c(B,n-2))\\
&= 4a_{n-1}^{\mathfrak{L}} - 2(2c(A,n-2) + 4c(B,n-2))\\
&= 4a_{n-1}^{\mathfrak{L}} - 2a_{n-2}^{\mathfrak{L}}
\end{aligned}$$

Figure 1: Derivation of the recurrence formula for the number of permutations a PLITG can generate.

It is clear that for any A permutation there is exactly one shorter B permutation it could be formed from. From this we have:

$$c(A,n) = c(A,n-1) + c(B,n-1) \quad (2)$$

The B class permutations can also be built from shorter A and B permutations. Specifically, there is one shorter type of A permutations that can be used to build a B permutation:

$$\begin{bmatrix} 0 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & \dots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \dots & 0 & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix}$$

Building B permutations from shorter B permutations is easier, and there are a total of three different types of B permutation to use:

$$\begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 1 & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \dots & 0 & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & \dots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix}$$

The fourth type of B permutations would result in an A type permutation. From this we have:

$$c(B,n) = c(A,n-1) + 3c(B,n-1) \quad (3)$$

Plugging equations (2) and (3) into equation (1) and performing the derivations steps shown in Figure 1, we get:

$$a_n^{\mathfrak{L}} = 4a_{n-1}^{\mathfrak{L}} - 2a_{n-2}^{\mathfrak{L}}$$

Comparing this recurrence formula,[1] to the ones for finite-state transduction grammars ($a^{\mathfrak{F}}$), inversion transduction grammars ($a^{\mathfrak{I}}$) and syntax-directed transduction grammars ($a^{\mathfrak{T}}$) , we have:

$$\begin{aligned}
&a_1^{\mathfrak{F}} = 1, \quad a_n^{\mathfrak{F}} = a_{n-1}^{\mathfrak{F}} + 1\\
&a_1^{\mathfrak{L}} = 1, \quad a_2^{\mathfrak{L}} = 2, \quad a_n^{\mathfrak{L}} = 4a_{n-1}^{\mathfrak{L}} - 2a_{n-2}^{\mathfrak{L}}\\
&a_1^{\mathfrak{I}} = 1, \quad a_2^{\mathfrak{I}} = 2, \quad a_n^{\mathfrak{I}} = \tfrac{6n-9}{n}\, a_{n-1}^{\mathfrak{I}} - \tfrac{n-3}{n}\, a_{n-2}^{\mathfrak{I}}\\
&a_1^{\mathfrak{T}} = 1, \quad a_n^{\mathfrak{T}} = n a_{n-1}^{\mathfrak{T}}
\end{aligned}$$

Table 2 contains the number of permutations for $n \le 20$, and Figure 3 contains a plot in logarithmic space of the numbers for $n \le 40$. It is clear that linear transductions contain several orders of magnitude fewer permutations for any non-toy sentences than inversion transductions, but it is also clear that a very large number of permutations are represented—clearly more than for finite-state transduction grammars. Comparing the weak alignment

[1]This is the integer series A006012 in *The On-Line Encyclopedia of Integer Series* (`www.oeis.org/A006012`).

n	Linear transductions	Inversion transductions	Syntax-directed transductions
1	1	1	1
2	2	2	2
3	6	6	6
4	20	22	24
5	68	90	120
6	232	394	720
7	792	1,806	5,040
8	2,704	8,558	40,320
9	9,232	41,586	362,880
10	31,520	206,098	3,628,800
11	107,616	1,037,718	39,916,800
12	367,424	5,293,446	479,001,600
13	1,254,464	27,297,738	6,227,020,800
14	4,283,008	142,078,746	87,178,291,200
15	14,623,104	745,387,038	1,307,674,368,000
16	49,926,400	3,937,603,038	20,922,789,888,000
17	170,459,392	20,927,156,706	355,687,428,096,000
18	581,984,768	111,818,026,018	6,402,373,705,728,000
19	1,987,020,288	600,318,853,926	121,645,100,408,832,000
20	6,784,111,616	3,236,724,317,174	2,432,902,008,176,640,000

Table 2: The number of permutations in different transductions for the first 20 values of n.

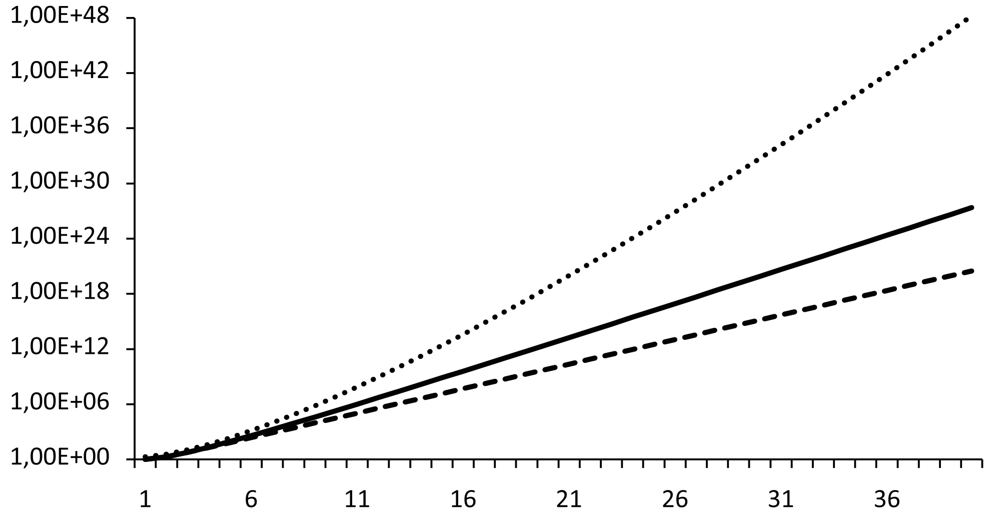

Figure 3: Number of permutations in syntax-directed transductions (dotted line), inversion transductions (solid line) and linear transductions (dashed line) as a function of the length of the sentence pair.

$[0,1,2,3]$	$[0,1,3,2]$	$[0,2,1,3]$	$[0,2,3,1]$	$[0,3,1,2]$	$[0,3,2,1]$
$[1,0,2,3]$	$*\,[1,0,3,2]$	$[1,2,0,3]$	$[1,2,3,0]$	$*\,[1,3,0,2]$	$[1,3,2,0]$
$[2,0,1,3]$	$*\,[2,0,3,1]$	$[2,1,0,3]$	$[2,1,3,0]$	$*\,[2,3,0,1]$	$[2,3,1,0]$
$[3,0,1,2]$	$[3,0,2,1]$	$[3,1,0,2]$	$[3,1,2,0]$	$[3,2,0,1]$	$[3,2,1,0]$

Figure 4: All permutation vectors of length four. Those a PLITG cannot generate are marked with an asterisk.

capacity of PLITGs to the other grammar classes we already know, we have:

$$a_n^{\mathfrak{F}} < a_n^{\mathfrak{L}} < a_n^{\mathfrak{I}} < a_n^{\mathfrak{T}}$$

That the weak reordering capacity of one grammar class is strictly contained within another is not surprising, since these grammar classes have already been studied from what we in this paper term weak transductive capacity, and found to have the same relationship to each other (Aho and Ullman, 1972; Saers, 2011). One difference to earlier work is that we not only have the relationship between the classes, but also a quantification of their differences.

It is also clear that finite-state transductions are the odd one out, as the number of alignments is significantly smaller than for any of the other grammar classes. We attribute this to the fact that a FSTG restricts the possible bispans that can make up a parse, whereas the others do not. Enumerating the bispans that can make up a constituent for SDTGs, ITGs and PLITGs (parsing the sentence pair $\langle \mathbf{e}, \mathbf{f} \rangle$) we have:

$$\{\langle s,t,u,v\rangle | 0 \leq s \leq t \leq |\mathbf{e}|, 0 \leq u \leq v \leq |\mathbf{f}|\}$$

Whereas the same set for a right-linear FSTG is:

$$\{\langle s,t,u,v\rangle | 0 \leq s \leq t = |\mathbf{e}|, 0 \leq u \leq v = |\mathbf{f}|\}$$

Changing the inequalities to equalities makes the set two orders of magnitude smaller for FSTGS than for the other three classes.

Merely counting the number of token alignments that can be generated offers, however, little in terms of actual understanding of, for example, the limitations of linear transductions over inversion transductions. To delve a little deeper, we will also look at the permutation vectors[2] of length four. Both ITGs and PLIGs are capable of generating all permutation vectors up to length three, so length four is where they start to diverge. Consulting Table 2 (or the above recurrence formulas), we see that ITGs are capable of generating 22 of the 24 possible permutation vectors of length four, whereas PLITGs only generate 20 of them. From previous work we know that ITGs fail to generate the so called *inside-out* alignments, represented by the permutation vectors $[2,4,1,3]$ and $[3,1,4,2]$ (Wu, 1997). Let us now turn to PLITGs.

We know that all the token alignments that a PLITG can generate will have one of the corner configurations $A_1, A_2, B_1, \ldots, B_4$. Converting them into underspecified permutation vector form we have the following set of permutation vectors:

$$\begin{aligned} A_0 &= [0,?,?,3]\,, & A_1 &= [3,?,?,0]\,, \\ B_0 &= [0,?,?,?]\,, & B_1 &= [3,?,?,?]\,, \\ B_3 &= [?,?,?,3]\,, & B_2 &= [?,?,?,0] \end{aligned}$$

Since we wish to use these permutation vector templates as conditions to test all possible permutations against, we will start by noting that the A permutations have the property that they satisfy the constraints of two B vectors, such that $A_1 = B_1 \wedge B_4$ and $A_2 = B_2 \wedge B_3$. Any permutation vector that matches a B vector can thus be generated by a PLITG.[3] Having made this initial analysis we can inspect the full set of 24 possible permutation vectors, and be sure to find four vectors that neither ends nor begins with a 1 or 4. These are the four that cannot be generated by a PLITG. Figure 4 shows the results of this.

The first thing to notice is that a PLITG is unable to generate the inside-out alignments, which is ex-

[2]A permutation vector is a vector where each number corresponds to the row where the one in the corresponding token alignment was found.

[3]This is only valid for permutation vectors of length four, since we know that the underspecified part, being of length three, constitutes a valid permutation vector.

pected since linear transductions are a proper subset of inversion transductions. The other two that cannot be generated are $[1, 0, 3, 2]$ (which we call *serial inversion*) and $[2, 3, 0, 1]$ (which we call *constituent swapping*). Whereas there are some evidence that the inside-out alignments are irrelevant to natural language translation (Huang et al., 2009; Søgaard, 2010), no such results exist for serial inversion and constituent swapping. On the contrary, we intuitively expect these phenomena to be frequent between natural languages. We consider this to be a serious problem with linear transductions, but empirical studies will have give the final say on how much it hurts performance.

5 Conclusions

In this paper we have presented an analysis of the weak reordering capacity of linear transductions, and compared it to that of finite-state transduction grammars, inversion transduction grammars and syntax-directed transduction grammars. We have showed that it is possible to quantify the exact number of permutations contained within a linear transduction as a function of the length of the sentence pair, and compared it to similar measures for the other transduction types. As linear transductions are a proper subset of inversion transductions, the number of permutations is lower but still high. Whether the permutations lost by moving from inversion transductions to linear transductions are needed or not has to be further studied.

We believe that this kind of analysis is useful for understanding the modeling restrictions of different kinds of grammar-based approaches to translation. A firm understanding of the models used is imperative to interpreting the empirical results they yield.

Acknowledgments

This work was funded by the Defense Advanced Research Projects Agency (DARPA) under GALE Contract Nos. HR0011-06-C-0023 and HR0011-06-C-0023, and the Hong Kong Research Grants Council (RGC) under research grants GRF621008, GRF612806, DAG03/04.EG09, RGC6256/00E, and RGC6083/99E. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*. Prentice-Halll, Englewood Cliffs, NJ.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Liang Huang, Hao Zhang, Daniel Gildea, and Kevin Knight. 2009. Binarization of synchronous context-free grammars. *Computational Linguistics*, 35(4):559–595, December.

Philipp Koehn, Franz Joseph Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference*, Edmonton, Canada, May/June.

Philip M. Lewis and Richard E. Stearns. 1968. Syntax-directed transduction. *Journal of the Association for Computing Machinery*, 15(3):465–488.

Daniel Marcu and Daniel Wong. 2002. A phrase-based,joint probability model for statistical machine translation. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 133–139, Philadelphia, Pennsylvania, July. Association for Computational Linguistics.

Markus Saers and Dekai Wu. 2011. Principled induction of phrasal bilexica. In *Proceedings of the 15th Annual Conference of the European Association for Machine Translation*, Leuven, Belgium, May.

Markus Saers, Joakim Nivre, and Dekai Wu. 2010. Word alignment with stochastic bracketing linear inversion transduction grammar. In *HLT/NAACL2010*, pages 341–344, Los Angeles, California, June. Association for Computational Linguistics.

Markus Saers. 2011. *Translation as Linear Transduction: Models and Algorithms for Efficient Learning in Statistical Machine Translation*. Ph.D. thesis, Uppsala University, Department of Linguistics and Philology.

Ernst Schröder. 1870. Vier combinatorische probleme. *Zeitschrift für Mathematik und Physik*, 15:361–376.

Anders Søgaard. 2010. Can inversion transduction grammars generate hand alignments? In *Proceedings of the 14th Annual Conference of the European Association for Machine Translation*, St. Raphael, France.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. HMM-based word alignment in statistical translation. In *Proceedings of the 16th International Conference on Computational Linguistics*, volume 1, pages 836–841, Copenhagen, Denmark, August.

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