Machine Translation as Tree Labeling

Mark Hopkins
Department of Linguistics
University of Potsdam, Germany
hopkins@ling.uni-potsdam.de

Jonas Kuhn
Department of Linguistics
University of Potsdam, Germany
kuhn@ling.uni-potsdam.de

Abstract

We present the main ideas behind a new syntax-based machine translation system, based on reducing the machine translation task to a tree-labeling task. This tree labeling is further reduced to a sequence of decisions (of four varieties), which can be discriminatively trained. The optimal tree labeling (i.e. translation) is then found through a simple depth-first branch-and-bound search. An early system founded on these ideas has been shown to be competitive with Pharaoh when both are trained on a small subsection of the Europarl corpus.

1 Motivation

Statistical machine translation has, for a while now, been dominated by the phrase-based translation paradigm (Och and Ney, 2003). In this paradigm, sentences are translated from a source language to a target language through the repeated substitution of contiguous word sequences ("phrases") from the source language for word sequences in the target language. Training of the phrase translation model builds on top of a standard statistical word alignment over the training corpus for identifying corresponding word blocks, assuming no further linguistic analysis of the source or target language. In decoding, these systems then typically rely on n-gram language models and simple statistical reordering models to shuffle the phrases into an order that is coherent in the target language.

There are limits to what such an approach can ultimately achieve. Machine translation based on a deeper analysis of the syntactic structure of a sentence has long been identified as a desirable objective in principle (consider (Wu, 1997; Yamada and Knight, 2001)). However, attempts to retrofit syntactic information into the phrase-based paradigm have not met with enormous success (Koehn et al., 2003; Och et al., 2003)[1], and purely phrase-based machine translation systems continue to outperform these syntax/phrase-based hybrids.

In this work, we try to make a fresh start with syntax-based machine translation, discarding the phrase-based paradigm and designing a machine translation system from the ground up, using syntax as our central guiding star. Evaluation with BLEU and a detailed manual error analysis of our nascent system show that this new approach might well have the potential to finally realize some of the promises of syntax.

2 Problem Formulation

We want to build a system that can learn to translate sentences from a source language to a destination language. As our first step, we will assume that the system will be learning from a corpus consisting of triples $\langle f, e, a \rangle$, where: (i) f is a sentence from our source language, which is parsed (the words of the sentence and the nodes of the parse tree may or may not be annotated with auxiliary information), (ii) e is a gold-standard translation of sentence f (the words of sentence e may or may not be annotated with auxiliary information), and (iii) a is an automatically-generated word alignment (e.g. via GIZA++) between source sentence f and destination sentence e.

[1] (Chiang, 2005) also reports that with his hierarchical generalization of the phrase-based approach, the addition of parser information doesn't lead to any improvements.

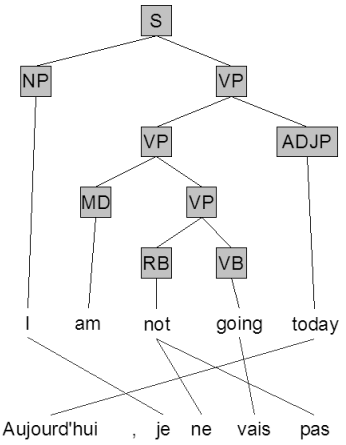

Figure 1: Example translation object.

Let us refer to these triples as *translation objects*.

The learning task is: using the training data, produce a scoring function P that assigns a score to every translation object $\langle f, e, a \rangle$, such that this scoring function assigns a high score to good translations, and a low score to poor ones. The decoding task is: given scoring function P and an arbitrary sentence f from the source language, find translation object $\langle f, e, a \rangle$ that maximizes $P(\langle f, e, a \rangle)$.

To facilitate matters, we will map translation objects to an alternate representation. In (Galley et al., 2003), the authors give a semantics to every translation object by associating each with an annotated parse tree (hereafter called a *GHKM tree*) representing a specific theory about how the source sentence was translated into the destination sentence.

In Figure 1, we show an example translation object and in Figure 2, we show its associated GHKM tree. The GHKM tree is simply the parse tree f of the translation object, annotated with rules (hereafter referred to as *GHKM rules*). We will not describe in depth the mapping process from translation object to GHKM tree. Suffice it to say that the alignment induces a set of intuitive translation rules. Essentially, a rule like: "not 1 $\rightarrow$ ne 1 pas" (see Figure 2) means: if we see the word "not" in English, followed by a phrase already translated into French, then translate the entire thing as the word "ne" + the translated phrase + the word "pas." A parse tree node gets labeled with one of these rules if, roughly speaking, its span is still contiguous when projected (via the alignment) into the target language.

Formally, what is a GHKM tree? Define a *rule element* as a string or an indexed variable (e.g. x_1, x_4, x_{32}). A *GHKM rule of rank* k (where k is a non-negative integer) is a pair $\langle R_s, R_d \rangle$, where *source list* R_s and *destination list* R_d are both lists of rule elements, such that each variable of $X_k \triangleq \{x_1, x_2, ..., x_k\}$ appears exactly once in R_s and exactly once in R_d. Moreover, in R_s, the variables appear in ascending order. In Figure 2, some of the tree nodes are annotated with GHKM rules. For clarity, we use a simplified notation. For instance, rule $\langle \langle x_1, x_2, x_3 \rangle, \langle x_3, \text{","}, x_1, x_2 \rangle \rangle$ is represented as "1 2 3 $\rightarrow$ 3 , 1 2". We have also labeled the nodes with roman numerals. When we want to refer to a particular node in later examples, we will refer to it, e.g., as $t_{(i)}$ or $t_{(vii)}$.

A *rule node* is a tree node annotated with a GHKM rule (for instance, nodes $t_{(i)}$ or $t_{(v)}$ of Figure 2, but not node $t_{(iv)}$). A tree node t_2 is *reachable* from tree node t_1 iff node t_2 is a proper descendant of node t_1 and there is no rule node (not including nodes t_1, t_2) on the path from node t_1 to node t_2.

Define the *successor list* of a tree node t as the list of rule nodes and leaves reachable from t (ordered in left-to-right depth-first search order). For Figure 2, the successor list of node $t_{(i)}$ is $\langle t_{(ii)}, t_{(v)}, t_{(xiii)} \rangle$, and the successor list of node $t_{(v)}$ is $\langle t_{(vii)}, t_{(viii)} \rangle$. The *rule node successor list* of a tree node is its successor list, with all non-rule nodes removed.

Define the *signature* of a parse tree node t as the result of taking its successor list, replacing the jth rule node with variable x_j, and replacing every non-rule node with its word label (observe that all non-rule nodes in the successor list are parse tree leaves, and therefore they have word labels). For Figure 2, the signature of node $t_{(i)}$ is $\langle x_1, x_2, x_3 \rangle$, and the signature of node $t_{(v)}$ is $\langle \text{"am"}, x_1 \rangle$.

Notice that the signature of every rule node in Figure 2 coincides with the source list of its GHKM rule. This is no accident, but rather a requirement. Define a *GHKM tree node* as a parse tree node whose children are all GHKM tree nodes, and whose GHKM rule's source list is equivalent to its signature (if the node is a rule node).

Given these definitions, we can proceed to define how a GHKM tree expresses a translation theory. Suppose we have a list $S = \langle s_1, ..., s_k \rangle$ of strings. Define the *substitution* of string list S into rule ele-

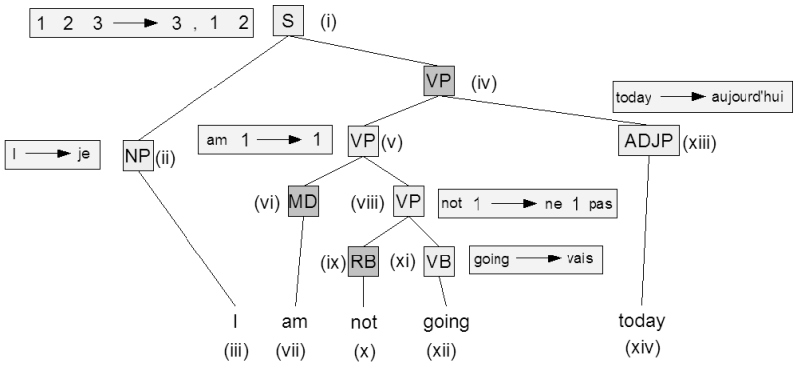

Figure 2: GHKM tree equivalent of example translation object. The light gray nodes are rule nodes of the GHKM tree.

ment r as:

$$r[S] = \begin{cases} s_i & \text{if } r \text{ is indexed var } x_i \\ r & \text{otherwise} \end{cases}$$

Notice that this operation always produces a string. Define the substitution of string list S into rule element list $R = \langle r_1, ..., r_j \rangle$ as:

$$R[S] = \text{concat}(r_1[S], r_2[S], ..., r_j[S])$$

where concat$(s_1, ..., s_k)$ is the spaced concatenation of strings s_1, ..., s_k (e.g., concat("hi", "there") = "hi there"). This operation also produces a string.

Finally, define the *translation* of GHKM tree node t as:

$$\tau(t) \triangleq R_d[\langle \tau(t_1), ..., \tau(t_k) \rangle]$$

where $\langle t_1, ..., t_k \rangle$ is the rule node successor list of GHKM tree node t.

For Figure 2, the rule node successor list of node $t_{(viii)}$ is $\langle t_{(xi)} \rangle$. So:

$$\begin{aligned} \tau(t_{(viii)}) &= \langle \text{“ne”}, x_1, \text{“pas”} \rangle [\langle \tau(t_{(xi)}) \rangle] \\ &= \langle \text{“ne”}, x_1, \text{“pas”} \rangle [\langle \text{“vais”} \rangle] \\ &= \text{“ne vais pas”} \end{aligned}$$

A similar derivation gives us:

$$\tau(t_{(i)}) = \text{“aujourd'hui , je ne vais pas”}$$

In this way, every GHKM tree encodes a translation. Given this interpretation of a translation object, the task of machine translation becomes something concrete: label the nodes of a parsed source sentence with a good set of GHKM rules.

3 Probabilistic Approach

To achieve this "good" labeling of GHKM rules, we will define a probabilistic generative model P of GHKM trees, which will serve as our scoring function. We would like to depart from the standard probabilistic approach of most phrase-based translators, which employ very simple probability models to enable polynomial-time decoding. Instead, we will use an alternative probabilistic approach (an *assignment process*), which sacrifices polynomial-time guarantees in favor of a more flexible and powerful model. This sacrifice of guaranteed polynomial-time decoding does not entail the sacrifice of good running time in practice.

3.1 Assignment Processes

An assignment process builds a sequence of variable assignments (called an *assignment history*) by repeatedly iterating the following steps. First, it requests a variable name (say x_{22}) from a so-named *variable generator*. It takes this variable name and the assignment history built so far and compresses this information into a set of features (say $\{f_2, f_6, f_{80}\}$) using a feature function. These features are then mapped to a probability distribution by a function (say p_7) requested from a so-named *distribution generator*. The iteration ends by assigning to the chosen variable a value (say $v4$) drawn from this distribution. In the above running example, the iteration assigns v_4 to x_{22}, which was drawn according to distribution $p_7(\{f_2, f_6, f_{80}\})$. The process ends when the variable generator produces the reserved token $STOP$ instead of a variable name.

Var	Assignment	Distribution	Features
x_{23}	$true$	p_4	$\{\}$
x_7	"the"	p_{10}	$\{f_{12}, f_{102}\}$
x_8	blue	p_2	$\{f_5, f_{55}\}$
x_{51}	red	p_2	$\{f_5, f_{15}, f_{50}\}$
x_{19}	7.29	p_5	$\{f_2\}$
x_{30}	$false$	p_4	$\{f_2, f_5, f_7\}$
x_1	"man"	p_{10}	$\{f_1, f_2, f_{12}\}$
x_{102}	blue	p_2	$\{f_1, f_{55}, f_{56}\}$

Figure 3: A example assignment history generated by an assignment process.

At this point, the assignment history built so far (like the example in Figure 3) is returned.

Formally, define a *variable signature* as a pair $\Sigma = \langle X, V \rangle$, where X is a set of variable names and V is a set of values. Define a *variable assignment* of signature $\langle X, V \rangle$ as a pair $\langle x, v \rangle$, for variable $x \in X$ and value $v \in V$. Define an *assignment history* of signature Σ as an ordered list of variable assignments of Σ. The notation $H(\Sigma)$ represents the set of all assignment histories of signature Σ.

We define a *feature function* of signature $\Sigma = \langle X, V \rangle$ as a function f that maps every pair of set $X \times H(\Sigma)$ to a set of assignments (called *features*) of an auxiliary variable signature Σ_f.

We define an *assignment process* of signature $\Sigma = \langle X, V \rangle$ as a tuple $\langle f, P, g_x, g_p \rangle$, where: (i) f is a feature function of Σ, (ii) $P = \{p_1, ..., p_k\}$ is a finite set of k functions (called the *feature-conditional distributions*) that map each feature set in $range(f)$ to a probability distribution over V, (iii) g_x is a function (called the *variable generator*) mapping each assignment history in the set $H(\Sigma)$ to either a variable name in X or the reserved token $STOP$, and (iv) g_p is a function (called the *distribution generator*) mapping each assignment history in the set $H(\Sigma)$ to a positive integer between 1 and k.

An assignment process probabilistically generates an assignment history of signature Σ in the following way:

1. $h \leftarrow$ empty list
2. Do until $g_x(h) = STOP$:
   (a) Let $x = g_x(h)$ and let $j = g_p(h)$.
   (b) Draw value v probabilistically from distribution $p_j(f(x, h))$.
   (c) Append assignment $\langle x, v \rangle$ to history h.
3. Return history h.

3.2 Training

Given all components of an assignment process of signature Σ except for the set P of feature-conditional distributions, the training task is to learn P from a training corpus of assignment histories of signature Σ. This can be achieved straightforwardly by taking the feature vectors generated by a particular distribution and using them to discriminatively learn the distribution. For instance, say that our corpus consists of the single history given in Figure **??**. To learn distribution p_2, we simply take the three variable assignments produced by p_2 and feed these feature vectors to a generic discriminative learner. We prefer learners that produce distributions (rather than hard classifiers) as output, but this is not required.

3.3 Decoding

Notice that an assignment process of signature Σ induces a probability distribution over the set $H(\Sigma)$ of all assignment histories of Σ. The decoding question is: given a partial assignment history h, what is the most probable completion of the history, according to this induced distribution? We will use the natural naive search space for this question. The nodes of this search space are the assignment histories of $H(\Sigma)$. The children of the search node representing history h are those histories that can be generated from h in one iteration of the assignment process. The value of a search node is the probability of its assignment history (according to the assignment process). To decode, we begin at the node representing history h, and search for the highest-value descendant that represents a complete assignment history (i.e. an assignment history terminated by the $STOP$ token).

This is, potentially, a very large and intractible search space. However, if most assignment decisions can be made with relative confidence, then the great majority of search nodes have values which are inferior to those of the best solutions. The standard search technique of *depth-first branch-and-bound search* takes advantage of search spaces with this particular characteristic by first finding greedy good-quality solutions and using their values to optimally prune a significant portion of the search space.

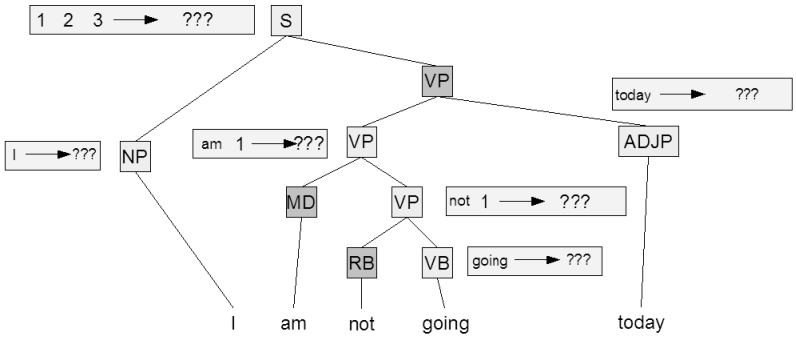

Figure 4: Partial GHKM tree, after rule nodes have been identified (light gray). Notice that once we identify the rule node, the rule left-hand sides are already determined.

Depth-first branch-and-bound search has the following advantage: it finds a good (suboptimal) solution in linear time and continually improves on this solution until it finds the optimal. Thus it can be run either as an optimal decoder or as a heuristic decoder, since we can interrupt its execution at any time to get the best solution found so far. Additionally, it takes only linear space to run.

4 Generative Model

We now return to where we left off at the end of Section 2, and devise an assignment process that produces a GHKM tree from an unlabeled parse tree. This will give us a quality measure that we can use to produce a "good" labeling of a given parse tree with GHKM rules (i.e., the probability of such a labeling according to the assignment process).

The simplest assignment process would have a variable for each node of the parse tree, and these variables would all be assigned by the same feature-conditional distribution over the space of all possible GHKM rules. The problem with such a formulation is that such a distribution would be inachievably difficult to learn. We want an assignment process in which all variables can take only a very small number of possible values, because it will be much easier to learn distributions over such variables. This means we need to break down the process of constructing a GHKM rule into simpler steps.

Our assignment process will begin by sequentially assigning a set of boolean variables (which we will call *rule node indicator variables*), one for each node in the parse tree. For parse tree node t, we denote its corresponding rule node indicator variable x_t^r. Variable x_t^r is assigned $true$ iff the parse tree node t will be a rule node in the GHKM tree.

In Figure 3.3, we show a partial GHKM tree after these assignments are made. The key thing to observe is that, after this sequence of boolean decisions, the LHS of every rule in the tree is already determined! To complete the tree, all we need to do is to fill in their right-hand sides.

Again, we could create variables to do this directly, i.e. have a variable for each rule whose domain is the space of possible right-hand sides for its established left-hand sides. But this is still a wide-open decision, so we will break it down further.

For each rule, we will begin by choosing the *template* of its RHS, which is a RHS in which all sequences of variables are replaced with an empty slot into which variables can later be placed. For instance, the template of $\langle$"ne", x_1, "pas"$\rangle$ is $\langle$"ne", X, "pas"$\rangle$ and the template of $\langle x_3, ",", x_1, x_2\rangle$ is $\langle X, ",", X\rangle$, where X represents the empty slots.

Once the template is chosen, it simply needs to be filled with the variables from the LHS. To do so, we process the LHS variables, one by one. By default, they are placed to the right of the previously placed variable (the first variable is placed in the first slot). We repeatedly offer the option to push the variable to the right until the option is declined or it is no longer possible to push it further right. If the variable was not pushed right at all, we repeatedly offer the option to push the variable to the left until the option is declined or it is no longer possible to push it further left. Figure 4 shows this generative story in action for the rule RHS $\langle x_3, ",", x_1, x_2\rangle$.

These are all of the decisions we need to make

Decision to make	Decision	RHS so far
RHS template?	X , X	X , X
default placement of var 1		1 , X
push var 1 right?	yes	X , 1
default placement of var 2		X , 1 2
push var 2 left?	no	X , 1 2
default placement of var 3		X , 1 2 3
push var 3 left?	yes	X , 1 3 2
push var 3 left?	yes	X , 3 1 2
push var 3 left?	yes	3 , 1 2

Figure 5: Trace of the generative story for the right-hand side of a GHKM rule.

in order to label a parse tree with GHKM rules. Notice that, aside from the template decisions, all of the decisions are binary (i.e. feasible to learn discriminatively). Even the template decisions are not terribly large-domain, if we maintain a separate feature-conditional distribution for each LHS template. For instance, if the LHS template is $\langle$"not", $X\rangle$, then RHS template $\langle$"ne", X, "pas"$\rangle$ and a few other select candidates should bear most of the probability mass.

5 Evaluation

In this section, we evaluate a preliminary English-to-German translation system based on the ideas outlined in this paper. We first present a quantiative comparison with the phrase-based approach, using the BLEU metric; then we discuss two concrete translation examples as a preliminary qualitative evaluation. Finally, we present a detailed manual error analysis.

Our data was a subset of the Europarl corpus consisting of sentences of lengths ranging from 8 to 17 words. Our training corpus contained 50000 sentences and our test corpus contained 300 sentences. We also had a small number of reserved sentences for development. The English sentences were parsed using the Bikel parser (Bikel, 2004), and the sentences were aligned with GIZA++ (Och and Ney, 2000). We used the WEKA machine learning package (Witten and Frank, 2005) to train the distributions (specifically, we used model trees).

For comparison, we also trained and evaluated Pharaoh (Koehn, 2005) on this limited corpus, using Pharaoh's default parameters. Pharaoh achieved a BLEU score of 11.17 on the test set, whereas our system achieved a BLEU score of 11.52. What is notable here is not the scores themselves (low due to the size of the training corpus). However our system managed to perform comparably with Pharaoh in a very early stage of its development, with rudimentary features and without the benefit of an n-gram language model.

Let's take a closer look at the sentences produced by our system, to gain some insight as to its current strengths and weaknesses.

Starting with the English sentence (note that all data is lowercase):

> i agree with the spirit of those amendments .

Our system produces:

> ich stimme die geist dieser
> I vote the.FEM spirit.MASC these
> änderungsanträge zu .
> change-proposals to .

The GHKM tree is depicted in Figure 5. The key feature of this translation is how the English phrase "agree with" is translated as the German "stimme ... zu" construction. Such a feat is difficult to produce consistently with a purely phrase-based system, as phrases of arbitrary length can be placed between the words "stimme" and "zu", as we can see happening in this particular example. By contrast, Pharaoh opts for the following (somewhat less desirable) translation:

> ich stimme mit dem geist dieser
> I vote with the.MASC spirit.MASC these
> änderungsanträge .
> change-proposals .

A weakness in our system is also evident here. The German noun "Geist" is masculine, thus our system uses the wrong article (a problem that Pharaoh, with its embedded n-gram language model, does not encounter).

In general, it seems that our system is superior to Pharaoh at figuring out the proper way to arrange the words of the output sentence, and inferior to Pharaoh at finding what the actual translation of those words should be.

Consider the English sentence:

> we shall submit a proposal along these lines before the end of this year .

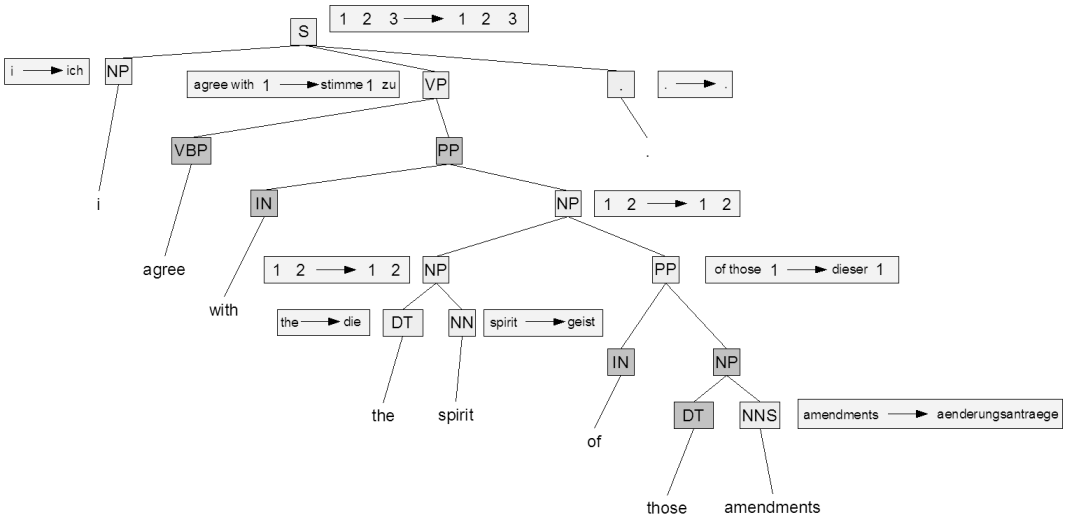

Figure 6: GHKM tree output for the first test sentence.

Here we have an example of a double verb: "shall submit." In German, the second verb should go at the end of the sentence, and this is achieved by our system (translating "shall" as "werden", and "submit" as "vorlegen").

wir	werden	eine	vorschlag	in	dieser
we	will	a.FEM	proposal.MASC	in	these
haushaltslinien	vor	die	ende		
budget-lines	before	the.FEM	end.NEUT		
dieser	jahres	vorlegen .			
this.FEM	year.NEUT	submit	.		

Pharaoh does not manage this (translating "submit" as "unterbreiten" and placing it mid-sentence).

werden	wir	unterbreiten	eine	vorschlag	in dieser
will	we	submit	a	proposal	in these
haushaltslinien	vor	ende	dieser	jahr	.
budget-lines	before	end	this.FEM	year.NEUT	.

It is worth noting that while our system gets the word order of the output system right, it makes several agreement mistakes and (like Pharaoh) doesn't get the translation of "along these lines" right.

To have a more systematic basis for comparison, we did a manual error analysis for 100 sentences from the test set. A native speaker of German (in the present pilot study one of the authors) determined the editing steps required to transform the system output into an acceptable translation – both in terms of fluency and adequacy of translation. In order to avoid a bias for our system, we randomized the presentation of output from one of the two systems.

We defined the following basic types of edits, with further subdistinctions depending on the word type: ADD, DELETE, CHANGE and MOVE. A special type TRANSLATE-untranslated was assumed for untranslated source words in the output. For the CHANGE, more fine-grained distinctions were made.[2] A single MOVE operation was assumed to displace an entire phrase; the distance of the movement in terms of the number of words was calculated. The table in Figure 7 shows the edits required for correcting the output of the two systems on 100 sentences.

We again observe that our system, which is at an early stage of development and contrary to the Pharaoh system does not include an n-gram language model trained on a large corpus, already yields promising results. The higher proportion of CHANGE operations, in particular CHANGE-inflection and CHANGE-function-word edits is presumably a direct consequence of providing a language model or not. An interesting observation is that our system currently tends to overtranslate, i.e., redundantly produce several translations for a word, which leads to the need of DELETE operations. The Pharaoh system had a tendency to undertranslate, often with crucial words missing.

[2] CHANGE-inflection: keeping the lemma and category the same, e.g. *taken* → *takes*; CHANGE-part-of-speech: choosing a different derivational form, e.g., *judged* → *judgement*; CHANGE-function-word: e.g., *in* → *from*; CHANGE-content-word: e.g., *opinion* → *consensus*.

	TL-MT	Pharaoh
ADD-function-word	40	49
ADD-content-word	17	35
ADD-punctuation	12	13
ADD (total)	69	97
DELETE-function-word	37	18
DELETE-content-word	22	10
DELETE-punctuation	13	15
DELETE-untranslated	2	1
DELETE (total)	74	44
CHANGE-content-word	24	19
CHANGE-function-word	44	26
CHANGE-inflection	101	80
CHANGE-part-of-speech	4	10
CHANGE (total)	173	135
TRANSLATE-untranslated	34	1
MOVE (distance)		
1	16	17
2	12	16
3	13	11
4	3	6
≥ 5	7	5
MOVE (total)	51	55
TOTAL # EDITS	401	332
edits-per-word ratio	0.342	0.295

Figure 7: Edits required for an acceptable system output, based on 100 test sentences.

6 Discussion

In describing this pilot project, we have attempted to give a "big picture" view of the essential ideas behind our system. To avoid obscuring the presentation, we have avoided many of the implementation details, in particular our choice of features. There are exactly four types of decisions that we need to train: (1) whether a parse tree node should be a rule node, (2) the RHS template of a rule, (3) whether a rule variable should be pushed left, and (4) whether a rule variable should be pushed right. For each of these decisions, there are a number of possible features that suggest themselves. For instance, recall that in German, typically the second verb of a double verb (such as "shall submit" or "can do") gets placed at the end of the sentence or clause. So when the system is considering whether to push a rule's noun phrase to the left, past an existing verb, it would be useful for it to consider (as a feature) whether that verb is the first or second verb of its clause.

This system was designed to be very flexible with the kind of information that it can exploit as features. Essentially any aspect of the parse tree, or of previous decisions that have been taken by the assignment process, can be used. Furthermore, we can mark-up the parse tree with any auxiliary information that might be beneficial, like noun gender or verb cases. The current implementation has hardly begun to explore these possibilities, containing only features pertaining to aspects of the parse tree.

Even in these early stages of development, the system shows promise in using syntactic information flexibly and effectively for machine translation. We hope to develop the system into a competitive alternative to phrase-based approaches.

References

Daniel M. Bikel. 2004. Intricacies of Collins' parsing model. *Computational Linguistics*, 30(4):479–511.

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

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2003. What's in a translation rule? In *Proc. NAACL*.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the Human Language Technology Conference 2003 (HLT-NAACL 2003)*, Edmonton, Canada.

Philipp Koehn. 2005. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Proceedings of the Sixth Conference of the Association for Machine Translation in the Americas*, pages 115–124.

F. J. Och and H. Ney. 2000. Improved statistical alignment models. In *Proc. ACL*, pages 440–447, Hongkong, China, October.

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

F. J. Och, D. Gildea, S. Khudanpur, A. Sarkar, K. Yamada, A. Fraser, S. Kumar, L. Shen, D. Smith, K. Eng, Viren Jain, Z.Jin, and D. Radev. 2003. Syntax for statistical machine translation. Technical report, Center for Language and Speech Processing, Johns Hopkins University, Baltimore. Summer Workshop Final Report.

Ian H. Witten and Eibe Frank. 2005. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

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

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 523–530.