Dependency Tree Translation: Syntactically Informed Phrasal SMT

Chris Quirk, Arul Menezes[1], Colin Cherry[2]

November 22, 2004

[1]Microsoft Research, One Microsoft Way, Redmond, WA 98052

[2]University of Alberta, 114 St - 89 Ave, Edmonton Alberta, Canada T6G 2E1; work done while at Microsoft Research

Abstract

We describe a novel approach to statistical machine translation that combines syntactic information in the source language with recent advances in phrasal translation. We depend on a source-language dependency parser and a word-aligned parallel corpus. The only target language resource assumed is a word breaker. These are used to produce treelet ("phrase") translation pairs as well as several models, including a channel model, an order model, and a target language model. Together these models and the treelet translation pairs provide a powerful and promising approach to MT that incorporates the power of phrasal SMT with the linguistic generality available in a parser. We evaluate two decoding approaches, one inspired by dynamic programming and the other employing an A* search, comparing the results under a variety of settings.

Contents

Chapter 1

Introduction

Over the past decade, we have witnessed a revolution in the field of machine translation (MT) toward statistical or corpus-based methods. Statistical machine translation (SMT) systems have reached the point where they can provide end-to-end translation systems that operate in real-time. Along the way, lessons from the Example-Based MT (EBMT) community have been effectively leveraged to produce higher quality translations. State-of-the-art SMT systems are now consistently winning competitions, faring well in both automated and human evaluations.

Yet for all its successes, SMT still has many hurdles to overcome. Qualitative comparisons of SMT translations against more traditional transfer-based MT systems show strengths and weaknesses of each approach. While SMT excels at learning translations of domain-specific terminology and fixed phrases, simple grammatical generalizations are poorly captured and often mangled during translation ([34]). Transfer-based systems are complementary in many ways: while they succeed more often in producing grammatical and fluent translations, the time-consuming human input involved in producing such a system necessarily focuses on broad coverage generalizations. Thus they often fail in exactly the area where SMT succeeds: domain-specificity. A natural next step, then, is to incorporate the strengths of both approaches. Indeed, producing a hybrid system combining the power of statistical learning and modeling with the insight of linguistic analysis has recently been the subject of much research. One existing system developed at Microsoft Research ([20]), for instance, bridged the gap between the domain-specific learning of Example-based and SMT systems and the syntactically informed nature of transfer-based systems. However it did not incorporate the multiplicity of models and end-to-end search from which SMT systems derive much of their power.

This report proposes a novel means of employing a source language dependency parser in combination with a variety of statistical models to produce a dependency tree-based SMT system. A decoder for this system uses treelet translation pairs and a host of models to produce high quality translations.

1.1 Related work

1.1.1 Formative statistical work

The touchstone of SMT is a detailed journal report from a group of trailblazing researchers at IBM ([3]). In clear mathematical detail, this work explores the application of the expectation maximization (EM) algorithm ([6]) to a series of increasingly complex models intended to automatically induce a word alignment given a sentence aligned corpus. While there are limitations to this approach (e.g., a target word can be aligned to at most one source word), these models have proven incredibly powerful and general, and the detail provided in this paper is sufficient to allow reimplementation of the important ideas.

In a COLING paper from 1996, Vogel, Tillmann, and Ney ([36]) explored an improved alignment model incorporating an HMM. This has proven to be a successful and powerful replacement for Model 2, and is commonly used in word alignment. Very few other improvements to the original IBM models have proven so influential.

Och and Ney ([23]) have recently published a careful and detailed comparison of word alignment models on two standard datasets. This provides a reference for alignment error rates, and suggests training regimens that prove effective on both French-English Hansards data and German-English VERBMOBIL data. One unfortunate aspect of this evaluation is the relatively homogeneous typology of the language pairs involved: neither French-English nor the restricted language of VERBMOBIL demonstrate the broad differences that one might encounter in aligning Japanese or Warlpiri to English, for example. Regardless, this work is a valuable guide for applying the effective techniques of unsupervised word alignment. A recent workshop ([21]) demonstrates that these methods do indeed represent the state of the art.

While several groups have attempted to produce purely word-based string-to-string SMT systems (such as [35], [11]), the sheer computational complexity of the problem [15] was a difficult obstacle to overcome, and it proved difficult to capture local context in a word-to-word statistical model. Thus the resulting systems were often rather slow and produced only moderate quality translations.

1.1.2 Phrasal SMT

The field of statistical machine translation found a second wind with the incorporation of phrasal translations. Instead of attempting to model the translation of each word independently, the simple but powerful idea behind Phrasal SMT is to remember how chunks of words translate together. This captures an important intuition of foreign language learning: small idioms and common phrases are both idiosyncratic and important for both fluency and fidelity.

As a concrete example, consider the Spanish phrase *presuponer cosa*, which may translate as the English phrase *make assumption*, whereas the most likely translations of these words in isolation are *presuppose* and *thing*. While it is certainly possible to model such a translation as *presuponer* translating as *make* and *cosa* translating as *assumption* under restricted environments, the translation problem is greatly simplified if we use "phrases"[1], in addition to words, as our atomic translation units. Parameter estimation over phrases does become more difficult (i.e. maximum likelihood estimation quickly encounters sparse data issues), but the resultant translations are still significantly better even when these issues are not solved.

Those approaching the MT problem from a linguistic viewpoint might suspect that phrases aligned to constituent boundaries would be more helpful than arbitrary strings of words from the training set. However, recent experiments ([17]) suggest that this is not the case: a system that includes all aligned word sequences contiguous in the source and target outperforms a system only containing those sequences that were identified as constituents by an English Treebank parser.

Recent phrasal SMT systems ([17],[42]) are conceptually quite simple. Beginning with a word alignment, all contiguous source and target word sequences that are consistent with the alignment are gathered as possible phrasal translation pairs (also known as *alignment templates*); these pairs are collected into a single translation repository. Then a translation probability is associated with each distinct pair by maximum likelihood, IBM's Model 1 ([37]), or any number of other means. This translation model is used in combination with at least a target language model (to form a classic noisy channel model), or perhaps with a group of other models in a log-linear framework. The best scoring translation is found by a rather simple search: a *monotone* decoder assumes that source phrase order is preserved and uses Viterbi decoding to find the best path through the translation lattice ([37],[42]). More commonly, a small amount of phrase reordering is allowed; this phrasal movement is modeled in terms of offsets, much like the HMM alignment model, with the probabilities trained off the word-aligned corpus ([16]). The approach is clear, elegant, and effective.

Phrase-based SMT is not a panacea, however. The reordering model mentioned above is limited in terms of linguistic generalizations. For instance, when translating English to Japanese, we would hope for a system able to notice the large-scale typological distinctions. For instance, English SVO clauses generally become Japanese SOV clauses, and English post-modifying prepositional phrases become Japanese pre-modifying postpositional phrases. While the phrasal reordering model above might learn that reorderings are more common in English-Japanese than English-French, it does not learn that the subject is likely to stay in place where the object is likely to move before the verb; nor does it learn any generalization regarding prepositional/postpositional phrase movement. More often than not, a phrase-based decoder acts at the mercy of rote-memorized phrases and a target language model biased toward fluency, not necessarily accuracy.

[1]Throughout this work, we will use the term *phrase* to denote a contiguous sequence of words in a sentence. Note the distinction from a *constituent*, which carries the usual linguistic connotations.

In addition, phrasal SMT is currently limited to phrases that are contiguous in both source and target. Such a system cannot learn the generalization that English *not* may translate as French *ne . . . pas* except in the context of a specific verb (e.g., *not have* translating as *n'ai pas*). Very large datasets can correct these shortcomings for the more common discontiguous patterns by simply memorizing a wide variety of possibilities, but less common discontiguous phrases will simply be impossible to learn.

1.1.3 Syntactic SMT

For the above reasons and more, many researchers have tackled the incorporation of syntax into statistical machine translation. Yet syntactic information has many possible representations, and there are many ways to incorporate it into the translation pipeline. This is a brief summary of the more notable attempts; the interested reader is directed to the original sources for more detailed information.

Note that there has also been a body of work on incorporating syntactic information into the word alignment process (e.g., [5], [7], [9], [12]). Since this report does not attempt to address the word alignment problem directly, these papers will not be addressed directly.

One very simple means of incorporating syntax into SMT is via reranking: a baseline system is used to produce an n-best list of translations, and then a group of models that may include syntactic models, is used to rerank the output. A recent workshop investigated employing a variety of syntactically motivated features in this approach, but found very little positive impact ([25]). In fact, the only feature that had a strong positive impact was a new statistical model. Yet this is a rather tenuous means of introducing syntactic information: an n-best list of even 16,000 translations captures only a minute fragment of the translation possibilities of a 20 word sentence, and reranking provides the syntactic model no opportunity to boost or prune large sections of that search space.

Inversion Transduction Grammars ([38],[39]) were one of the earliest attempts to incorporate a notion of constituency into SMT. The fundamental idea is to consider alignment and translations as simultaneous parses of source and target language. Two types of binary branching rules are allowed. Either the source and target constituents are produced in the same order, or the source and target constituents are produced in reverse order. It turns out that the reordering constraints allowed by these parses are actually quite broad and cover most common situations ([41],[43]). To make this process computationally efficient, however, some severely limiting simplifying assumptions are made. First, the parsing process is modeled with a single non-terminal label. This reduces translation complexity to a low order polynomial (n^6) at the cost of modeling power. In effect one simply learns a very high level preference regarding how often nodes should switch without any information regarding the contexts in which such a switch should occur. Also the translation model acts only at the level of a single lexical item at a time; phrasal combinations are not modeled directly. In the light of recent phrasal SMT successes, this is a rather severe limitation.

In an attempt to improve the problems with fluency in an SMT system, an alternate approach is to employ a parser in the target language ([40], etc). On the training data, one can learn probabilities on a set of operations to convert a target language tree to a source language string. These operations can be combined with a tree-based language model to produce a noisy channel translation search. This approach does seem to positively impact fluency ([4]), but fails to impact translation quality significantly. In a sense, this is not so surprising: the parser is applied to MT-output, which is notoriously unlike native language, and no additional insight is gained during source language analysis.

Recent work on Multitext Grammars and Generalized Multitext Grammars ([19]) generalizes the ITG line of research by allowing non-contiguous translations and loosening the reordering constraints. While the theory is elegant and intriguing, no published results exist. In addition the published details on parameter estimation and decoding in this framework do not describe how to incorporate phrasal information. Still, this work is very young and we may soon see progress in this area.

Yet another approach toward the translation problem is dependency tree translation via head-transduction ([1]). As a first approximation, assume the existence of a source dependency parse for the input sentence. A collection of transducers then apply to each level of the dependency tree to produce a target dependency tree, and the translation is then simply read from the tree. Instead of using a more standard dependency parser, their dependency structures are induced directly from the corpus. Thus many of the difficult monolingual behaviors (e.g., *ne* ... *pas* in French,) may have a dependency structure that is more suitable for translation, though the constituency boundaries produced by the process may be more suspect. These transducers are also limited in scope, only relying on very local context, so the end result is a fundamentally word-based, not phrase-based, decoder. The transducer induction process may also be complicated by data sparsity problems: instead of factoring the translation model into several different components (lexical selection, ordering, etc.), a single transducer is trained. A variety of methods for training these transducers have been proposed (e.g., [14]), but few attempt to address the problem of limited context or add a larger number of models into the decoding process.

Another line of research has formed at the confluence of dependency transducers and Multitext Grammars: Synchronous Dependency Insertion Grammars ([8]). This provides a formalism for both alignment and translation, the latter incorporating a standard noisy channel model with a dependency-tree-based language model to predict the probability of each child given its head.

Another means of employing dependency information is translating via paths in the dependency tree ([18]). This is one of the first methods that seems to incorporate larger memorized patterns like phrasal SMT in combination with a dependency analysis. However the modeling here is really quite simple: in the interest of translation speed, only a direct MLE translation model is used. Thus the decoding process does not balance fidelity against fluency using a target language model; in our experience, the absence of a target model is usually fatal. The resulting translations do seem to benefit from the use of treelets, but overall the approach does not come close to quality of a phrasal SMT decoder.

1.1.4 Trends, strengths, and weaknesses

Many, though not all, attempts in syntactic SMT have relied on a context-free constituency analysis instead of a dependency analysis. While this may seem like a natural starting point due to its well-understood nature and commonly available tools, we feel that this is not the most effective representation for syntax in MT. Dependency analysis, in contrast to constituency analysis, tends to bring more semantically related elements together (e.g., verbs become adjacent to all their arguments, not just objects). In addition, dependency trees are better suited to heavily lexicalized operations, which have proven quite effective in phrasal SMT.

A vast majority of syntactic SMT approaches have focused on word-to-word translation, despite the proven effectiveness of phrasal SMT. Conversely, phrasal SMT has not yet incorporated notions of constituency into the translation search. In addition, syntactic SMT decoders have not yet incorporated the more powerful log-linear framework. Instead, they have used more traditional noisy-channel models, which probably limits possible translation quality.

Chapter 2

Fundamentals

The primary contribution of this work is the pervasive use of a dependency tree representation of the source language in every aspect of the translation system[1] and in combination with the application of the best modeling and decoding advances from the string-based phrasal SMT community to these dependency tree representations, with particular emphasis on a novel tree-based ordering model. Additional important contributions include efficient, high-quality decoders that address the unique challenges of tree-based decoding.

2.1 Overview

At a high level, our system employs a source-language dependency parser, a target language word segmentation component, and an unsupervised word alignment component to learn translations from a parallel sentence aligned corpus. We begin by parsing the the source side to obtain dependency trees. We then apply a word-alignment component (currently string-based) to the bitext.

The word alignments are then used to project the source dependency parses onto the target sentences, to produce an aligned parallel dependency corpus. From this corpus we extract a treelet translation model incorporating source and target treelet pairs, where a *treelet* is defined to be a connected subgraph of the dependency tree. Note that the concept of a treelet is a generalization of a *subtree*, which must include all descendants of the root node. Like subtrees, though, every treelet still has a unique root. In the system described in this report we also allow one unusual variation on treelets: treelets with missing roots. This allows us to model important phenomena, such as "*not* *" $\mapsto$ "'*ne* * *pas*"'.

In addition, we train a variety of statistical models on this aligned dependency tree corpus, including a channel model, an order model, and an agreement model.

[1]With the notable exception, in this report, of the alignment component. We intend to address this in a followup paper.

To translate an input sentence, we first produce a dependency parse of that sentence. We then employ a decoder to find the best combination and ordering of the treelet translation pairs according to a host of models that are combined in a log-linear framework as in [27] or [24].

This approach offers the following advantages over string-based SMT systems. Instead of limiting learned phrases to word sequences contiguous in the source and in the target, we allow translation by all possible chunks that are form connected subgraphs (treelets) in the source and target dependency trees. This is a powerful extension: not only are the vast majority of phrases contiguous in the surface also treelets of the tree, but we also gain very powerful combinations not previously available to a phrasal SMT component. For instance, combinations such as verb-subject, verb-preposition, article-noun, adjective-noun etc. will be learned regardless of the number of intervening words.

Another major advantage is to allow reordering along source language constituency bounds, and to allow our movement models to incorporate information from the source analysis. Such models have a great deal of power: they model directly the probability that the object of a preposition in English should move before the postposition in Japanese, or the probability that a pre-modifying adjective in English becomes a post-modifying adjective in French. This does, however, imply that we are somewhat constrained by the source language analysis. We generally assume the concept of *phrasal cohesion* ([10]): given a source language subtree rooted at s, the translation of this subtree should be a subtree in the target. Yet in practice, this assumption does not prove to be a significant limitation: since our treelet translation pairs can be arbitrary subgraphs of the source and target dependency graphs, we can easily translate fixed phrases that do not lay precisely on constituency bounds. Also, as in [18], we do not rely on a target dependency analysis; instead we project the dependencies from source to target. Thus constructions that might seem divergent in a monolingually-motivated target languages analysis may in fact be represented in our projected dependency trees by very parallel structures.

The remainder of this chapter is devoted to the process of deriving the aligned parallel dependency tree corpus, which acts as a foundation for the remainder of our training. Chapter 3 discusses our models, Chapter 4 addresses decoding, and Chapter 5 is devoted to methods for deriving weights for each model. In Chapter we present experimental results in a variety of settings, and conclude by discussing some of the important lessons learned as well as future work.

2.2 Word alignment

As in most modern SMT systems, we depend heavily on a word-aligned bitext. This is the foundation of our translation information: we extract treelet translation pairs as well as positional information from this alignment. We are rather indifferent about the source of this alignment: whether it comes a purely surface-based alignment using the IBM models or a more syntactic source such as that of Lin and Cherry ([5]), we employ it the same way. This is a potential downfall, since we do not necessarily enforce any agreement between the word alignment and the phrasal information.

We currently follow the common practice of deriving many-to-many alignments by running the IBM models in both directions (both source-to-target and target-to-source) and combining the results heuristically ([26]). However, we differ in that we use alignments from the union of the two sets of alignments if and only if they are motivated by contiguity in the source dependency tree. We accept alignments from the union using the following heuristics (in order):

1. Accept all alignments that are unique on both sides (i.e. the only alignment in the union from the given source word is to the given target word, and vice versa).

2. Accept all alignments that are unique on one side (i.e. the only alignment in the union from the given source word is to that target word, but the target word has other non-unique alignments, or vice versa).

3. Accept those many-to-one alignments that are adjacent to existing alignments in the source dependency tree (i.e. accept an alignment (s_i, t_k) if we've already accepted an alignment (s_j, t_k), and either $h_s(i) = j$ or $h_s(j) = i$).

4. Accept all one-to-many alignments.

2.3 Source dependency trees

To gather syntactic information, we require a source language dependency parser that produces unlabeled, ordered dependency trees and annotates each source word with a part-of-speech (POS). Given some input set of tokens $s_1..s_n$, a dependency parse is a head function h: for any s_i, $h(i)$ is the index of the head of s_i, or 0 if s_i is the root. The dependencies do not cross[2] — given any nonzero i, k such that $h(i) = k$, then for all j between i and k, $h(j)$ is also between i and k. The dependency structure is directed, unlabeled, and ordered.

An isomorphic representation is a tree structure. The root of the dependency structure is the root of the tree. Each node in the tree has a list of its pre-modifying dependents and post-modifying dependents in order. We will switch between these two representations as convenient, though we most often use the tree structure.

2.4 Projecting dependency trees

Given a pair of parallel word aligned sentences and a source dependency tree, we use the alignment to project the source structure onto the target sentence.

[2]Note that dependency grammars may, in general, allow crossing dependencies. We do not allow them in our system, however.

Let us first consider a sentence pair where each source word is uniquely aligned to a single target word and vice versa. In this case, the induced dependency function h_t is simple to construct using the source dependency function h_s and the set of aligned pairs A. We follow a rule where:

$$h_t(i) = j \iff \exists (s_k, t_i), (s_l, t_j) \in A \text{ such that } h_s(k) = l$$

While this will create a target dependency tree that is maximally in agreement with the source tree, it may create a dependency tree that does not read off in the same order as the target string (since our alignments do not enforce phrasal cohesion). For instance, consider the following dependency tree (denoted by bracketing) and word alignment (denoted by indices): [3]

(((tired$_1$) men$_2$) and$_3$ (dogs$_4$))
hommes$_2$ y$_3$ chiens$_4$ fátigues$_1$

Using the algorithm described above, we end up with the following dependency tree:

((hommes (fátigues)) et (chiens))

If we read off these leaves in order, we do not get the original input string. In a second pass, we correct this situation by reattaching out-of-order nodes higher in the tree. For instance, if we reattach *fátigues* to *et*, then the string reads off in the correct order. For each node that is the wrong order relative to its siblings, we reattach it to the lowest of its ancestors that allows it to be in the correct place relative to all its siblings and parent.

Now consider a sentence pair where we have many-to-one alignments. The projection rule defined above for one-to-one alignments applies without modification:

$$h_t(i) = j \iff \exists (s_k, t_i), (s_l, t_j) \in A \text{ such that } h_s(k) = l$$

Since the heuristics for refining alignments guarantee that all sets of source nodes aligned to the same target node are connected subgraphs of the source dependency tree, the projected target dependencies would correspond exactly to the source dependencies if each connected subgraph aligned to the same target node is condensed into a single node.

Next consider a sentence pair with one-to-many alignments. The projection rule described above would make the target nodes siblings. While this is a valid representation, it fails to capture our intuition that the a set of target words aligned to a source word should probably move as a unit. Thus for

[3]Notice that either the English dependency parse is incorrect, or the target sentence is not a correct translation of the source sentence. This source dependency tree would correspond to the translation *hommes fátigues et chiens*, for example.

each set of target words aligned to the same source word, we make the right-most word be the head, and each previous word becomes a dependent on that word:[4]

$$\begin{aligned} h_t(i) = j \quad \Longleftrightarrow \quad & \Big(\exists (s_k, t_i), (s_l, t_j) \in A \text{ such that } h_s(k) = l \wedge \\ & \forall (s_k, t_m) \in A \; (m < i)\Big) \vee \\ & \big(\exists (s_k, t_i), (s_k, t_j) \in A \wedge i < j\big) \end{aligned}$$

Finally, we must deal with the case of unaligned nodes. Most source unaligned nodes do not present a problem, with the exception of the root. If the root is unaligned, then we start the projection process at each dependent of the root, producing a forest of dependency structures. We then pick one such tree to be the root (currently the right-most structure), and make all other trees be dependents of that root tree.

To attach unaligned target nodes into the dependency structure, we first project the dependency structure as defined above for all the nodes that are aligned. Now assume we have an unaligned node in position j. Let $i < j$ and $k > j$ be the target positions closest to j such that $h(i) = k$ or $h(k) = i$. If this is well defined, then we attach t_j to the lower of the two (e.g., if $h(i) = k$, then we attach t_j to t_i). However, it is possible that all the nodes to the left of position i are unaligned (or conversely that all the nodes to the right are unaligned). In that case, we attach to the left-most (or right-most) node that was aligned.

2.5 Extracting treelet translation pairs

We now have an aligned pair of dependency trees, from which we extract *treelet translation pairs*: pairs of source and target treelets along with word-level alignment linkages. We begin by enumerating all possible treelets of the source dependency tree. For each source treelet, we then look at the union of all target nodes aligned to the source nodes. If this set of target nodes is a treelet (i.e. a connected subgraph of the target dependency tree), then we keep the treelet translation pair along with its word alignment and ordering information. For instance, from the aligned dependency tree pair above (*tired men and dogs* aligned to *hommes et chiens fátigues*), we would learn the following treelet translation pairs:

- $(\textit{tired}_1) \mapsto (\textit{fátigues}_1)$
- $(\textit{men}_1) \mapsto (\textit{hommes}_1)$
- $(\textit{and}_1) \mapsto (\textit{et}_1)$
- $(\textit{dogs}_1) \mapsto (\textit{chiens}_1)$

[4]In the future we plan to use a more consistent and informative method of selecting the head, such as using the node with the strongest lexical alignment probability.

- ((*men*$_1$) *and*$_2$) $\mapsto$ ((*hommes*$_1$) *et*$_2$)
- (*and*$_1$ (*dogs*$_2$)) $\mapsto$ (*et*$_1$ (*chiens*$_2$))
- ((*men*$_1$) *and*$_2$ (*dogs*$_3$)) $\mapsto$ ((*hommes*$_1$) *et*$_2$ (*chiens*$_3$))
- (((*tired*$_1$) *men*$_2$) *and*$_3$ (*dogs*$_4$)) $\mapsto$ ((*hommes*$_2$) *et*$_3$ (*chiens*$_4$) (*fâtigues*$_1$))
- ((*men*$_1$) * (*dogs*$_2$)) $\mapsto$ ((*hommes*$_1$) * (*chiens*$_2$))

As mentioned above, we also allow the special case of treelets with wildcard roots. This allows us to learn translations such as ((*doesn't*$_1$) *) $\mapsto$ ((*ne*$_1$) * (*pas*$_1$) from examples such as *doesn't recalculate* and *ne recalcule pas*. Note that we do not learn a treelet translation pair for the source treelet ((*tired*) *men*) because the set of target nodes aligned to those words, *fâtigues* and *hommes*, do not form a treelet: they are not a *connected* subgraph of the target dependency tree.

We also keep treelet counts to be used in maximum likelihood estimation. We use a variable threshold to limit the size of extracted treelets to control the combinatorial explosion. In practice, we find that extracting all treelets containing up to four source nodes provides an effective trade-off.

Chapter 3

Models

Given a candidate translation **t** for a source sentence **s**, the role of the translation model is to define a distribution $\mathbf{Pr}(\mathbf{t}|\mathbf{s})$. Given such a model, the translation process is a search through candidate space for a **t** that maximizes translation probability.

As in ([27]), we have generalized beyond the simple noisy-channel framework to a more general framework of log-linear models. Let us walk through this generalization in more detail, starting with the classic noisy channel approach:

$$\begin{aligned}
\mathbf{t}^* &= \operatorname*{argmax}_{\mathbf{t}}(\mathbf{Pr}(\mathbf{t}|\mathbf{s})) \\
&= \text{argmax}\left(\frac{\mathbf{Pr}(\mathbf{s}|\mathbf{t})\mathbf{Pr}(\mathbf{t})}{\mathbf{Pr}(\mathbf{s})}\right) \\
&= \text{argmax}\left(\mathbf{Pr}(\mathbf{s}|\mathbf{t})\mathbf{Pr}(\mathbf{t})\right) \\
&= \text{argmax}\left(\exp(\log(\mathbf{Pr}(\mathbf{s}|\mathbf{t})) + \log(\mathbf{Pr}(\mathbf{t})))\right) \\
&= \text{argmax}\left(\log(\mathbf{Pr}(\mathbf{s}|\mathbf{t})) + \log(\mathbf{Pr}(\mathbf{t}))\right)
\end{aligned}$$

The above equation suggests that we should uniformly weight the contributions of the channel model and the target language model. Yet these probability factors are seldom completely comparable in practice: they may differ significantly in scale or in variance. As an attempt to compensate for this fact, we may introduce scaling factors:

$$\text{argmax}(\mathbf{Pr}(\mathbf{t}|\mathbf{s})) = \text{argmax}\left(\lambda_1 \log(\mathbf{Pr}(\mathbf{s}|\mathbf{t})) + \lambda_2 \log(\mathbf{Pr}(\mathbf{t}))\right)$$

Here we may notice a striking similarity to the framework of log-linear models. Such a framework is composed of a vector of feature functions F and a vector of scaling factors or weights Λ whose dot product is the overall score for any given candidate. In this framework, the best **t** given a fixed **s** is:

$$\operatorname*{argmax}_{\mathbf{t}}\left(\sum_i \lambda_i f_i(\mathbf{t},\mathbf{s})\right)$$

By treating our component probability models as feature functions, this framework allows us to incorporate a variety of different models, each of which attempts to capture a different aspect of the translation process. In the following sections we describe a set of models that have proven effective in our system. In Chapter 5 we describe methods for training the Λ weights to maximize an objective function.

3.1 Channel models

We first attempt to model $\mathbf{Pr}(\mathbf{s}|\mathbf{t})$, also known as the channel model. In actual fact, we incorporate three distinct channel models, one based on a maximum likelihood estimate of replacement, and two based on word-to-word alignment probabilities.

Given a source treelet σ and a target treelet τ, the maximum likelihood probability estimate is simply:

$$\mathbf{Pr}_{MLE}(\sigma|\tau) = \frac{C(\sigma, \tau)}{C(\cdot, \tau)}$$

Here $C(\sigma, \tau)$ represents the count of times that we saw the treelets σ and τ aligned in any training sentence pair, and $C(\cdot, \tau)$ represents the number of times we saw the treelet τ in any target side sentence.

While the MLE model effectively captures idiomatic and other non-literal phrasal translations, it suffers from sparsity of data. Therefore we also include word-to-word models. These models do not typically suffer from data sparsity, although they have the downside of preferring more literal translations.

The word-to-word model uses the sum of IBM Model 1 scores over all possible alignments within the treelet ([37]); we use an unnormalized Model 1 score to help bias toward larger treelets. Given a source treelet σ and a target treelet τ, we estimate probabilities in the following manner:

$$\mathbf{Pr}_{M1}(\sigma|\tau) = \prod_{\mathbf{s}\in\sigma} \sum_{\mathbf{t}\in\tau} \mathbf{Pr}(\mathbf{s}|\mathbf{t})$$

In addition, we incorporate Model 1 probability estimates in the forward direction:

$$\mathbf{Pr}_{M1}(\tau|\sigma) = \prod_{\mathbf{t}\in\tau} \sum_{\mathbf{s}\in\sigma} \mathbf{Pr}(\mathbf{t}|\mathbf{s})$$

Given a set of treelet translation pairs that cover a given input dependency tree and produce a target dependency tree, we model the probability of source given target as the product of the individual treelet translation probabilities. That is, we assume a uniform probability distribution over the decompositions of a tree intro treelets.

3.2 Order model

We model the likelihood of seeing the ordering of a *nuclear subtree*: that is, a node and all its children, but not its indirect descendants. In formal terms, given a target node t, we wish to model the probability of seeing the pre-modifiers in a given order $\mathbf{Pr}(r(t)|t,\mathbf{s})$ and the post-modifiers in a given order $\mathbf{Pr}(o(t)|t,\mathbf{s})$. The reader will notice that we have included the full source sentence $\mathbf{s}$ in the conditioning information for these probability distributions. We attempt to mine the source dependency tree for movement information.

Our work in order modeling was most directly inspired by the Amalgam project ([31]), an attempt to reconstruct surface strings from a deep analysis. Ordering was an important component in their pipeline of operations: after having generated a set of surface nodes from an abstract representation, it was necessary to pick a surface order that was representative of the grammatical roles and constraints given by the input structure and the target language grammatical constraint. We face a corresponding issue: given a source language dependency structure and a set of partially ordered target nodes, we must find the most likely total ordering of those target nodes.

The order models we use attempt to predict the ordering of each set of constituents independently: the order model probability of a whole dependency tree is the product of the ordering probabilities at each head. Consider the following source and target dependency trees:

((To$_1$) create$_2$ ((the$_3$) ((InkColor$_4$) property$_5$) definition$_6$))
((Pour$_1$) créer$_2$ ((la$_3$) définition$_6$ (de$_0$ (propriété$_5$ (ColourEncre$_4$)))))

To model the probability of a whole candidate, we first model the ordering of the modifiers of *creér*: *pour* and *définition*. We then proceed down the dependency tree to model the ordering of each of the modifiers relative to their head. Thus the probability of the above target dependency tree would be decomposed as follows, where the heads are identified in italic:

$$\begin{aligned}
&\mathbf{Pr}(\langle \text{pour}, \textit{créer}, \text{définition}\rangle \mid \ldots) \cdot \\
&\mathbf{Pr}(\langle \text{la}, \textit{définition}, \text{de}\rangle \mid \ldots) \cdot \\
&\mathbf{Pr}(\langle \textit{de}, \text{propriété}\rangle \mid \ldots) \cdot \\
&\mathbf{Pr}(\langle \textit{propriété}, \text{ColourEncre}\rangle \mid \ldots)
\end{aligned}$$

(The conditioning information has been omitted for brevity.)

We model ordering at each head in one of two ways, as described in the next two sections.

3.2.1 Head-relative position model

One method of predicting the order of a set of modifiers is to model in terms of position relative to the head of the structure. That is, we decompose the order at each head into a series of placement decisions, one for each child. For instance, we have:

$$\begin{aligned}&\mathbf{Pr}(\langle \text{pour}, \textit{créer}, \text{définition}\rangle \mid \ldots) = \\ &\quad \mathbf{Pr}(\text{POS}(\text{pour}) = -1 \mid \ldots) \cdot \\ &\quad \mathbf{Pr}(\text{POS}(\text{définition}) = +1 \mid \ldots)\end{aligned}$$

We train this model using decision trees (DTs), which allow incorporation a variety of conditioning information. The following items are currently used:

- Lexical item of the node being ordered
- Lexical item of the head
- Lexical item and part-of-speech of the input node aligned to the node being ordered
- Lexical item and part-of-speech of the input node aligned to the head
- Head-relative position of the input node aligned to the node being ordered

In addition, it is possible to include information about siblings in the conditioning information to produce a Markov ordering model. However, we found that this additional information has little impact in practice; thus we do not include it in the final model. The target feature is simply the head-relative position in the target side.

Training this model is straightforward. Starting with the aligned parallel dependency tree corpus, we first annotate each source and target node with its head-relative position. Each target node with a parent provides a single training data point; at each node, we read off each of the features listed above, and gather all the features together into a single training set. We then employ a DT learner which attempts to predict the head-relative position of a target side node given all the other features.

To score a candidate translation tree at runtime, we follow a similar approach. First we annotate both source and target trees with head-relative positions. Then, for each target node with a parent, we consult the decision tree to find the appropriate leaf node corresponding to the features of that dependency tree node, and we estimate the probability of the constructed node by MLE over the distribution of positions at that DT leaf node.

Unfortunately this model is subject to problems with data sparsity. Dependency trees with large numbers of modifiers tend to have few training examples for the distant modifiers and hence may not have smooth distributions. It is also limited in its linguistic generalizations: while it may notice that an English adjective in position -1 becomes a French adjective in position +1, this gives the model no information about where an English adjective in position -2 should go.

3.2.2 Swap/Challenge model

To alleviate some of the previous problems, we considered an alternate approach to ordering that attempts to incorporate several linguistic intuitions. First, the prediction of whether a modifier moves to the opposite side of the head during translation is somewhat orthogonal from its final relative position. Secondly, source scope is very often preserved during translation.

Thus this alternate model acts in two distinct phases. First it attempts to predict whether each child is premodifier or a postmodifier given the same conditioning information as above; that is, we model whether it should "swap sides". We then have a set of pre-modifiers and post-modifiers; it remains to predict their relative order. We can model this as a series of decisions (or "challenges") regarding whether we should preserve source scope or promote some target node to be closer to the head. That is, the pre-modifier closest to the head must have won a challenge against all the other pre-modifiers, the next closest pre-modifier must have won a challenge against all remaining pre-modifiers, and likewise until no pre-modifiers is remaining. The post-modifiers are ordered in the same way. Note that each possible order of modifiers is uniquely defined by a series of swap and challenge operations.

Let's return to the example of *pour creer dfinition*. In this case, each word does not swap sides, and since there is only a single pre-modifier and a single post-modifier, the challenges are rather trivial. Here is the decomposition:

$$\begin{aligned}
&\mathbf{Pr}(\langle \text{pour}, \textit{créer}, \text{définition}\rangle \mid \ldots) = \\
&\quad \mathbf{Pr}(\textsc{swap}(\text{pour}) = \text{FALSE} \mid \ldots) \cdot \\
&\quad \mathbf{Pr}(\textsc{swap}(\text{définition}) = \text{FALSE} \mid \ldots) \cdot \\
&\quad \mathbf{Pr}(\text{CHALLENGE}(\text{pour}, \{\}) = \text{TRUE} \mid \ldots) \cdot \\
&\quad \mathbf{Pr}(\text{CHALLENGE}(\text{définition}, \{\}) = \text{TRUE} \mid \ldots)
\end{aligned}$$

Consider the alternate ordering where *créer* has two post-modifiers as a demonstration of the challenge model:

$$\begin{aligned}
&\mathbf{Pr}(\langle \textit{créer}, \text{définition}, \text{pour}\rangle \mid \ldots) = \\
&\quad \mathbf{Pr}(\textsc{swap}(\text{pour}) = \text{TRUE} \mid \ldots) \cdot \\
&\quad \mathbf{Pr}(\textsc{swap}(\text{définition}) = \text{FALSE} \mid \ldots) \cdot \\
&\quad \mathbf{Pr}(\text{CHALLENGE}(\text{définition}, \{\text{pour}\}) = \text{TRUE} \mid \ldots) \cdot \\
&\quad \mathbf{Pr}(\text{CHALLENGE}(\text{pour}, \{\}) = \text{TRUE} \mid \ldots)
\end{aligned}$$

In this ordering, *pour* is now a post-modifier, while its translation in the source sentence was a pre-modifier, hence it has been swapped. Similarly, *définition* has won a challenge against *pour*, as it was placed closer to the head.

This model was trained using decision trees, though we split each phase into a separate decision tree. For the swap phase, we visit each modifier in turn to gather the same set of features used for the head-relative order model; the target feature is a boolean value indicating whether the modifier swapped sides.

The challenge phase is modeled as a sequential selection process, based on the binary conditional ordering model introduced in the Amalgam project ([31]). It estimates a distribution over the boolean decision that a modifier should be placed at the current point given a set of features computed on the set of not-yet-unordered modifiers and, optionally, some subset of the already-ordered modifiers that are closest to the head. As in the head-relative model, we omit features computed on the already-ordered nodes, so the resultant model is not a Markov model. Training this phase is slightly more complicated since we must produce both positive and negative training examples. The positive examples are simply read off of correctly-ordered nodes. We produce negative examples by permuting the tree and reading off one negative training point for each incorrectly ordered node in that permutation.

To score a candidate with this swap/challenge order model, we begin by determining the unique set of SWAP and CHALLENGE operations required to produce this output. Then we score each operation by identifying the relevant leaf node in the decision tree and producing an MLE estimate of the probability over the distribution at that leaf.

However, in the end we found this swap/challenge model did not outperform the order model predicting head-relative positions. On initial inspection, this approach seems to place too much emphasis on scope-preserving translation. In practice, certain absolute positions often loosely correspond to semantic relations: the first premodifier of a verb tends to be the subject, and that subject may move in idiosyncratic ways. In the future we plan to investigate this and other model structures more fully, but for now we employ only the head-relative model in our evaluations.

3.3 Target language model

Given an ordered target language dependency tree, it is trivial to read off the surface string, which can be evaluated by a standard trigram target language model. In practice, this serves to incorporate agreement information and generally improve the fluency of translation. Following Goodman's careful analysis of different approaches for language modeling ([13]), we employed a trigram model using modified Kneser-Ney smoothing.

3.4 Agreement model

We have also experimented with a dependency-based bigram model that attempts to model the probability of a modifier given a head. That is, the bigrams are in the dependency tree, not in the surface string. The intention is to model agreement phenomena, for instance it should predict that $\mathbf{Pr}(la|table)$ is much greater than $\mathbf{Pr}(le|table)$. At first glance, this may seem redundant with the target language model, yet it is useful for two reasons. It is not limited to a trigram window: it produces identical results regardless of the number of intervening modifiers. Secondly, this model can be evaluated before any ordering has been attempted. The latter property is particularly useful in A* decoding, as we shall discuss later.

Training this model is a simple matter: from the target side dependency trees that were projected in the manner described in Section 2.4, we simply read off all head-modifier bigrams.

3.5 Miscellaneous other models

Having such a general framework at our fingertips allows us to incorporate a variety of additional information in the translation process. For instance, we have reason to believe that larger treelet translation pairs often provide better translations, since they capture more context and allow fewer possibilities for search and model error. Therefore we can add a feature function that counts the number of phrases used in any subtree, in the hopes that it will be assigned a negative weight.

We also add a feature that counts the number of target words; this acts as a insertion/deletion bonus that can help balance a target language model biased toward shorter sentences or a channel model biased toward longer sentences.

Chapter 4

Decoding

The role of a decoder in a dependency tree-based SMT system is to efficiently find a high probability translation for a given source sentence. In our approach, the source is dependency parsed beforehand, and a sequence of operations can generate a large number of target dependency trees. The probability of a candidate translation is determined by our translation models (see Chapter 3).

The challenge of tree-based decoding is that the traditional left-to-right decoding approach of string-based systems is inapplicable. Additional challenges are posed by the need to handle incomplete subtrees, discontiguous and overlapping treelets, and a combinatorially explosive ordering search space.

In the following sections we describe two distinct approaches to decoding.

4.1 Exhaustive search and dynamic programming

A first approach to the decoding problem is strongly influenced by the Inversion Transduction Grammar approach ([38], [39]). We translate bottom up; first we find the best translations for a given subtree, then we reuse those lower-level subtree computations in finding the best translations of higher subtrees. Yet our approach extends this framework in a variety of ways. First, we employ treelet translation pairs instead of single word translations. Second, our dependency structure allows a more sophisticated ordering story. Instead of modeling rearrangements as either preserving source order or swapping source order, we allow the dependents of a node to be ordered in any arbitrary way, and use the order model described in Section 3.2 to estimate probabilities. We also allow larger order n-grams in our target language model, not just bigrams. Finally, we use a log-linear framework for model combination that allows any amount of other information to be modeled.

We will initially approach the decoding problem in the context of a bottom up, exhaustive search. For each input node we will define the set of all possible treelet translation pairs of the subtree rooted at each input node in the following manner.

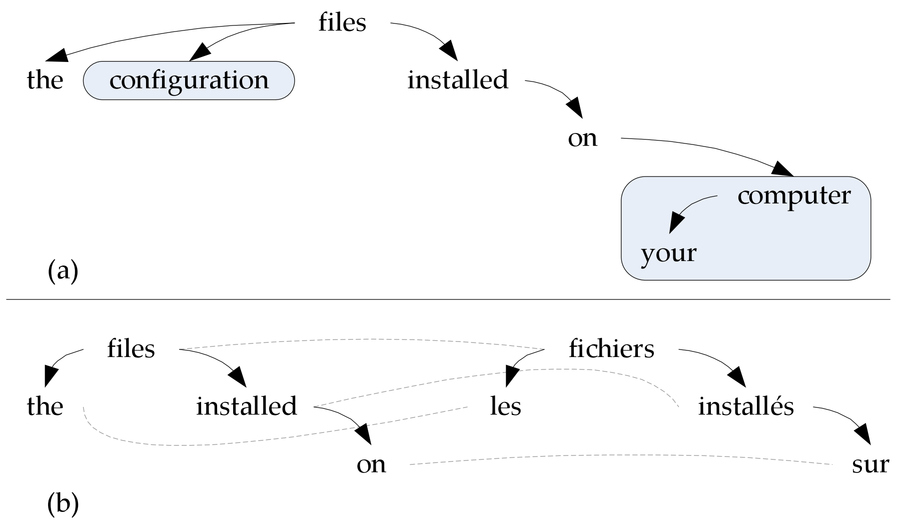

Figure 4.1: Example decoder setup. (a) shows an example input dependency tree. (b) shows an example treelet translation pair. The subtrees in (a) not covered by the treelet translation pair (a) are enclosed by boxes.

A treelet translation pair x is said to *match* the input dependency tree S iff there is some connected subgraph S' of S and an isomorphism f between S' and the $\sigma(x)$, source side of x. This isomorphism must preserve lexical items, parts-of-speech, dependency relations, and relative orderings. More formally, for all $s_i, s_j \in S'$:

1. There exists $s_k, s_l \in \sigma(x)$ such that $f(s_i) = s_k$ and $f(s_j) = s_l$.
2. $lex(s_i) = lex(s_k)$; $cat(s_i) = cat(s_l)$.
3. $h(s_i) = s_j$ iff $h(s_k) = s_l$.
4. $i < j$ iff $k < l$.

As noted before, a treelet has a unique root. We'll say a treelet translation pair x is *rooted* at a source node $s \in S$ iff x matches S and the root of $\sigma(x)$ maps to s under the matching isomorphism above.

In an initial pass, we scan through our treelet translation model to find all treelet translation pairs that match the input dependency tree. We place each matched pair on a list associated with the input node where the match is rooted. Next we move bottom up through the dependency tree, computing a list of possible target language trees that are translations of the source language subtree.

At each input node s, we consider each treelet translation pair in turn. The source treelet may cover only a portion of the input subtree rooted at s. As an example, consider Figure 4.1, which demonstrates an input dependency tree and an example treelet translation pair. We find the roots of all the input subtrees that are not covered by the treelet — in this case, *configuation* and *computer* — and look in their candidate lists (constructed from a previous bottom-up subtree translation call) for all possible translations. We define a *candidate set* to be a partially ordered set of lexical translations for a given treelet created by assigning a candidate to each source node left uncovered by the treelet. Then, for each covering treelet and each candidate set for that treelet, we consider each possible interleaving of the candidates with one another and with the children whose relative order is specified by the treelet. Pseudo-code for the exhaustive decoding algorithm is show in Algorithm 1.

The exhaustive search described above can be converted into a dynamic programming search rather easily. Note that in general, a higher level translation depends on a minimum of information from the translation subtree acquired from a lower level: the order model may ask for information about the root node, the target language model scores for the first two words in the subtree may be affected, and the last two words in the subtree may affect the translation of the next two words after this subtree. Therefore, we only need keep the best scoring translation candidate for a given subtree for each (head, first two words, last two words) tuple. Unfortunately this is provides minimal savings beyond exhaustive decoding in practice. In the worst case, the number of treelet translation pairs that must be stored in the input list for any particular node is proportional to the number of head lemmas, multiplied by the number of first two lemmas, multiplied by the number of last two items, which leads to a worst case list size of $O(V^5)$, where V is the vocabulary size. Thus the dynamic programming approach does not allow for great savings, mainly because a trigram target language model forces the algorithm to consider external context.

4.2 Optimizations to exhaustive search

The above decoder will find the optimal translation according to the models given sufficient time. Unfortunately, this turns out to be an awfully long time in practice. The following sections introduce a variety of optimizations and simplifications to allow real-time translation output.

4.2.1 n-best list

Instead of keeping the full list of translation candidates for a given input node, an easy win is to only keep some top-scoring subset of the candidates. While the decoder is no longer guaranteed to find the optimal translation at this point, the resulting drop in quality is minimal, even with a modest n-best list size of 12 candidates.

Algorithm 1 Exhaustive decoding algorithm

```
function GetNBestTranslations($s$ : input node)
  $L \leftarrow \emptyset$; a sorted list of translations
  for all treelet translation pairs $x$ rooted at $s$ do
    $R \leftarrow$ roots of input subtrees not covered by $x$
    for all $r \in R$ do
      let $x[r] \leftarrow$ GetNBestTranslations($r$)
    end for
    for all sets of translations $Q$, one from $x[r]$ for each $r \in R$ do
      for all ordered attachments $Q'$ of $Q$ into the target side of $x$ do
        $T \leftarrow$ target subtree interleaving $Q'$ with target side of $x$
        score $T$ according to models
        add $T$ to $L$ in order
      end for
    end for
  end for
  return the n-best list $L$
end function
```

4.2.2 Duplicate translation checking

In order to limit the number of ordering operations conducted by the decoder, one can check to see if a given word-set has been previously ordered by the decoder before the ordering process begins. We have currently implemented this using a hash table that is indexed on unordered trees. Two trees are considered to be equal in this scheme if they have the same tree structure and lexical choices after sorting each parent's children into a canonical order. Checking for equality under these circumstances is not trivial in terms of complexity, nor is determining the hash function value. The bottle-neck for both processes is sorting the tree's nodes, which involves a recursive set-less-than function, which ends the recursion by comparing the lexical choices at leaf nodes.

This implementation, which adheres to tree structure, may be slow, but it does capture information that would not be present in a strict bag-of-words representation. We will know exactly when two trees are identical in terms of the ordering operations our decoder is capable of performing, and we will never prune a tree capable of forming a novel ordering of words that we have not yet seen. It seems likely that any method that attempts to use a true bag-of-words, or to collapse the tree to its string form before sorting would lose this information, and potentially miss valid translation possibilities.

4.2.3 Early dropping

Another means of restricting the search space is to drop candidates early in the decoding process, ideally before we explore the large and expensive ordering space. For instance, one safe way of pruning candidates early is to look at the

Algorithm 2 Greedy ordering algorithm

```
$O_{best}$ : empty ordering with 0 probability.
for all possible count of premodifiers and postmodifiers do
  for all premodifer position from right to left do
    for all unordered node do
      Evaluate this unordered node in this position
    end for
    Place the higest scoring unordered node in this position
    Remove that node from the unordered pool
  end for
  for all postmodifier position from left to right do
    for all unordered node do
      Evaluate this unordered node in this position
    end for
    Place the higest scoring unordered node in this position
    Remove that node from the unordered pool
  end for
  if this ordering has a higher probability than $O_{best}$ then
    $O_{best}$ := this ordering
  end if
end for
Return $O_{best}$
```

channel model scores of the treelet translation pair along with the completed scores of the child subtrees to be attached when creating the final translation. If the sum of these scores is lower than the lowest score currently on the n-best list, then the final candidate is bound to fall off the n-best list, since adding the order model probability will only drop the overall score. A nice benefit of this approach is that it does not affect the optimality of the algorithm.

4.2.4 Greedy ordering

The ordering stage is the decoder's most expensive step: its complexity grows with the factorial of the number of nodes to be ordered. It is also the inner-most loop; it is called for each possible combination of translation choices. Therefore, anything we can do to speed up ordering will result in significant decoder speed-ups.

One way to speed up ordering is to eliminate it. Given a fixed pre and post-modifier count, our order model is capable of evaluating a single ordering decision independently from other ordering decisions. We have implemented a version of the decoder that takes advantage of this to severely limit the number of ordering possibilities for a given translation set. Then during decoding, we can replace the step that considers all possible orderings with the approach outlined in Algorithm 2.

For any target tree node t, we let $n_m(t)$ be the number of child modifiers whose order is specified by the treelet translation pairs and $n_u(t)$ be the number of children whose order is not specified. Then the number of ordered candidates for a given treelet combination is on the order of

$$O\left(\prod_t \left(\binom{n_m(t)+n_u(t)}{n_u(t)} n_u(t)!\right)\right) = \left(\prod_t \left(\frac{(n_m(t)+n_u(t))!}{n_u(t)!}\right)\right)$$

In contrast, the greedy ordering outlined above only explores a small subset of those, on the order of:

$$O\left(\prod_t \left((n_m(t)+n_u(t))n_u(t)^2\right)\right)$$

While significantly faster, this ordering decision process is suboptimal in a number of ways. First of all, generating translation candidates based solely on the order model cuts the target language model out of the decoding process. The TLM is relegated to contributing to the grade for complete translations generated by the decoder, but it may never see the translations that it would assign a high likelihood to. Furthermore, if we were content to ignore the TLM, this is still not guaranteed to be the optimal ordering decision based on the decision tree alone. There is a hidden dependency in that at each greedy decision point, we remove one placeholder and one place-filler from the list. It is possible in taking the best option for an early place-holder, we will eliminate the ability to adequately fill another place-holder.

4.2.5 Variable-sized n-best lists

All of modifications listed above deal with trying to reduce the number of orderings the decoder attempts. However, ordering is only one source of complexity. The number of translations using a given treelet translation pair is exponential in the number of subtrees of the input not covered by that treelet translation pair. The decoder would see a significant speed-up by reducing the number of translation combinations that need to be attempted.

One possible solution to this problem is variable-sized *n*-best lists. This means that a recursive call to translate an uncovered subtree will limit the size of the returned *n*-best list according to the number of uncovered subtrees in the current treelet. The idea is that if a treelet covers very little of the tree, and will require several recursive calls, then those calls should return smaller *n*-best lists. This way, the treelet translation pairs that are most vulnerable to exponential blow-up will artificially lower the value of the exponent. Furthermore, it has the intuitive appeal of allowing the engine to thoroughly explore those treelet translation pairs that are likely to result in good translations, that is: those treelet translation pairs that already translate a lot of the tree for us. Meanwhile, it is less thorough when it needs to do most of the work from scratch, in which case it only explores the translations that are likely to produce good results.

The current implementation determines n at any point by dividing a seed value by the number of uncovered nodes that need to be resolved using recursive calls. We then round up, so no treelet translation pairs will ever get an effective n of less than 1. We have found that 12 makes a good seed value, as it results in an intuitive progression that falls off quickly, but still differentiates between interesting cases:

Number of uncovered subtrees	Effective n-best size
1	12
2	6
3	4
4	3
6	2
12	1

Table 4.1: Effective n-best list sizes

The use of variable-sized n-best lists has made it possible for the exhaustive decoder to outperform the greedy decoder. Before, the exhaustive decoder would rarely return, even for small values of n.

4.2.6 Limiting treelet translation pairs

In practice, channel model scores and treelet size are powerful predictors of high quality translation. Pruning away low scoring treelet translation pairs before the search starts allows the decoder to spend more time inspecting combinations and orderings of high quality treelet translation pairs. We have attempted a variety of pruning heuristics, such as:

- Only keep those treelet translation pairs with an MLE probability above some threshold. A cutoff of 0.01 has proven effective.

- Given a set of treelet translation pairs where the source treelets are identical, only keep those pairs whose MLE probability is within some ratio of the best pair. For instance, we only keep treelet translation pairs whose MLE probability is no less than $1/20^{\text{th}}$ of the best MLE probability.

- After finding all treelet translation pairs that matched a given input dependency tree, keep only the top n treelet translation pairs rooted at each input node, as ranked first by size, then by MLE channel model score, then by Model 1 score. The threshold $n = 3$ works well in practice.

Although almost none of the above optimizations is guaranteed to preserve optimality, in practice the decoder, using these optimizations, is able to translate approximately one sentence per second, with no degradation in quality.

One drawback of the dynamic programming approach, however, is that this decoder makes no provision for handling overlapping treelet translation pairs, which previous experience with the MSR-MT([20]) system indicated to be an important contributor to translation quality.[1] This led us to explore an alternate decoding strategy.

4.3 A* decoding

Inspired by impressive recent results with standard search frameworks ([29]), our next decoder framework was an A* approach. The high-level organization of this decoder is significantly different: instead of building candidates bottom-up according to the input tree, we build candidates as collections of overlapping treelet translation pairs. The treelet translation pairs in each candidate must be compatible with each other, in that the overlapping portions must be consistent in their translations and implied ordering. The search is guided by actual model scores for the covered portions of the input and admissible (optimistic) estimates for the uncovered portions. The order and target models do not allow for actual scores until the treelet translation pairs cover a complete subtree. In actual practice our search does not attempt ordering until the entire input tree is covered. Until that point the ordering and target model rely on optimistic estimates. To compensate for this, we use an agreement model, not employed by the DP-style decoder. This is a tree-based bigram model, which measures agreement between a pair of parent and child nodes in the target language tree, and is trained off the projected target language dependency trees obtained in the training phase. This model is based on [2]. The remainder of this section covers the search process in more detail.

The A* search can be thought of as a traversal of a binary tree of choices, where the branches on each node represents the choice to use or not use a particular treelet translation pair.

We begin by assembling all treelet translation pairs into a single, global list L, which is then sorted by one of several desirability criteria. The most promising sort criteria are ⟨treelet size, channel model score⟩ and ⟨A* score⟩.

We then initialize a queue of incomplete and complete candidates. Each candidate is composed of the following members:

- The position of the next treelet translation to attempt in the global list of translations L.
- The set of treelet translations already selected in this candidate.
- A representation of the input nodes covered by this candidate.

[1]We did attempt to simply produce all possible combinations of overlapping treelet translation pairs in a pre-pass, but the combinatorial explosion proved to be completely infeasible in practice.

- The actual and optimistic estimates portions of the scores for this candidate according to each model.

We initialize the incomplete queue with the empty candidate: its position is set to the first treelet pair in the list, its channel model estimates include estimates for all input nodes, and the other estimates are initialized as appropriate.

At each step, we extract the highest scoring candidate from the incomplete queue. We consider the next treelet translation in the global list as indicated by the next-treelet pair pointer in each candidate. If there are no more treelet pairs, we drop the candidate.

If the next treelet translation is not compatible (described in more detail below) with the treelet translations already chosen for this candidate, we skip it and move to the next one. Similarly if the treelet is compatible but adds no new information (i.e. it does not cover any new input nodes) to the already chosen treelets, we also skip it and move on.

If the treelet translation is compatible and covers new input nodes, then we clone the current candidate, and we add the treelet translation to the candidate's previously chosen set of treelets. We then add in actual model scores for the newly covered input nodes, and subtract out the optimistic estimates for those nodes. The new score is subjected to several threshold tests (see below) and the candidate is discarded if it fails any of them. If it passes the tests, but does not yet cover the entire input, we enqueue it in the incomplete queue.

If the new candidate now covers the entire input, we merge the overlapping treelets, and pick the ordering with the highest model score (see details below). The model score in this case includes no estimates, but instead uses the actual scores for all models, including the order and target models. The completed candidate is then enqueued in the completed queue.

Meanwhile the unmodified copy of the candidate represents the "not chosen" branch in the search space. We update its next-treelet pair pointer. We also update its estimated scores to reflect the choices not taken. This has the effect of tightening up the estimate with each not-taken choice, with no loss of admissibility, and tighter estimates result in better pruning. The updated score is subjected to threshold tests and the candidate is discarded if it fails any of them, otherwise it is inserted back into the incomplete queue based on its updated score.

Thus at each stage, we add a maximum of two new candidates, corresponding to the 2^n search space of all possible subsets of the available treelets.

The process repeats until the incomplete queue is empty, or we have expended a specified amount of effort on the search. In the latter case, the result is not guaranteed to be optimal, so we have a number of heuristics to ensure that it is in practice as close to optimal as possible.

4.3.1 Admissible heuristics

Channel models

Computing optimistic estimates for the channel model turns out to be rather simple. To find the estimate for any given input node, we find the maximum estimate assigned by any treelet translation τ covering that input node. Let $p(\tau)$ be the channel model probability associated with τ and $\text{size}(\tau)$ be the number of input nodes that this treelet pair covers; then the optimistic estimate from τ will be simply the probability mass of τ divided evenly among the input nodes. More formally, we have:

$$\text{optest}(i) = \max_{\tau \text{ covering } i} \left\{ p(\tau)^{1/\text{size}(\tau)} \right\}$$

The justification for this is rather straightforward. Given any treelet translation σ, we know that its probability is less than or equal to the product of the optimistic estimates of the nodes that it covered. If this were not the case, then at least one of the estimates must be less than $p(\sigma)^{1/\text{size}(\sigma)}$: contradiction. This argument works equally well for both the Model 1 and the maximum likelihood channel model.

We actually compute this score based on every suffix of the list of translations L, which gives us at every point in the A* search an estimate based only on the treelet translations we have left to consider. This allows for progressive tightening of estimates with no loss of admissibility.

Furthermore, if a node cannot be covered by the remaining treelet translations, its estimate is set to negative infinity. This has the effect that a candidate is pruned as soon as it rejects the last available choice for translating any input node.

Agreement model

For each input node we compute the agreement score between all possible translations of that word (given the list of treelet translations) and all possible translations of its parent. The highest such score is used as the optimistic estimate.

This estimate is also progressively tightened, as described in the previous section.

Target model

To derive an optimistic estimate for the target model, we first gather all the target words provided by the set of treelet translations that matched the input. We then compare all possible n-grams of those available target language lemmas against the target language model to find the context that maximizes the probability of every target token. This provides a relatively loose target model estimate.

Since we do not order (and hence cannot score the target model) until we have entirely covered the input this estimate is not updated incrementally, but instead replaced with an actual target model score when ordering is completed.

Other models

For all other models we are currently using the most naïve optimistic estimate, the maximum possible value regardless of any contextual information. For probabilistic models, we assign a probability of 1; for the non-probabilistic models, we assign the greatest possible value or smallest possible value, according to the sign of the respective lambda.

4.3.2 Compatibility

A highly restrictive sense of compatibility might consider two treelet pairs incompatible if they overlap, i.e. cover the same input node. This leads to a decoder very similar in expressive power to the above dynamic programming decoder: no overlapping treelet pairs are allowed.

However, a more complicated notion of compatibility may be beneficial to translation quality. Consider the input sentence: “Click the selected button.” Now assume that we have translations for “click button” and “selected button” that agree on the translation of “button”. It seems counterintuitive and perhaps detrimental to force a choice between these two translations instead of allowing their translational preferences to mutually reinforce. Thus we extend the notion of compatibility to allow these sorts of alignments.

Note, however that treelet translations may provide one-to-one, one-to-many, many-to-one or many-to-many translations and/or insertion or deletion of words. Our treelets, therefore, retain node-level alignment information from training time. Each treelet translation is divided into a disjoint set of paired source and target minimal translation units (MTUs), where each MTU corresponds to the minimum unit of alignment. Unaligned nodes are rolled up into their parent MTUs, so that we model word insertion and deletion as one-to-many and many-to-one translation.

Note also that since treelets are ordered, each treelet node implies a partial ordering among its children.

Therefore, we consider two treelet translations to be compatible if, on the portions of the input where they overlap, they agree on both the boundaries and the content of the MTUs. Furthermore, for each target node in the overlap there must exist a total ordering of child nodes that respects the partial ordering implied by each treelet translation.

4.3.3 Ordering

Given a set of treelet translations that cover the entire input, we proceed in a bottom-up manner, stitching (potentially overlapping) treelet pairs together.

At each node we compute the best ordering by considering all possible orderings that respect the constraints imposed by the individual treelet translations. Each ordering is scored using the ordering and target language models. We save the *n*-best orderings on each node. At the parent node, we score all possible orderings of the children, considering all combinations of the child *n*-best lists for each ordering.

Since ordering is the most expensive step in the search, we impose several restrictions. If a node has more than a specified number of siblings (N1), the size of its *n*-best list is reduced to 1, to limit combinatorial explosion on its parent. If ordering a candidate takes more than a specified amount of time, the *n*-best lists are reduced to size 1 at every node.

More drastically, if a node has more than a specified number of children (N2, N2 $<$ N1), we skip exhaustive ordering entirely. Instead we evaluate the source order, reverse source order and the greedy order. The greedy order is computed by greedily picking the most probable modifier for each position in turn.

4.3.4 Optimizations

We use the same limits on treelet translations as used by the DP-style decoder. In addition we employ the following A*-specific optimizations.

Round robin

As has become common practice, we speed the search process by considering in each iteration not just the highest scoring incomplete candidate, but one such candidate for each candidate size (measured in terms of input nodes covered). This is because overly optimistic scores associated with smaller treelet pairs (where a greater portion of the score is an optimistic estimate rather than actual) can result in the decoder spending too much time in the shallower parts of the search space. ([17])

We extend this by also considering in each iteration, for each input node η, at least one candidate that has already covered η and at least one candidate that has not already covered η. This is because the divergence between actual scores and optimistic estimates can vary significantly from one input node to another. Without this heuristic, the decoder can devote too much time to exploring every choice related to one portion of the input and run out of time before it has searched an adequate number of choices relating to a different part of the input.

These two optimizations have no effect if the search runs to completion.

Thresholds

We discard any incomplete candidate whose estimated score is lower than the actual score of the current best completed candidate. This is an admissible heuristic. To speed up its application, we pre-populate the completed queue with the “greedy candidate”, i.e. the candidate obtained by taking the “yes” branch at every decision point in the search space until the input is covered. Given a good ordering of treelet translations, the greedy candidate turns out to be the winner surprisingly often, and hence it provides a good tight A* threshold, with no loss of admissibility.

The second threshold we apply does not meet the admissibility criterion, but has very little negative effect in practice. We compare the portion of an incomplete candidate score for which we have tight estimates (i.e. two channel models and the agreement models), with the corresponding portion of the score for the best completed candidate. If an incomplete candidate's score is lower than a specified ratio of the best candidate's score, it is pruned. ([17])

Chapter 5

Optimizing model weights

Even with only two models, one must carefully choose the weights (lambdas) assigned to each in a log-linear framework to effectively balance the contribution of a target language model and a channel model. Given that we have six or more models, it becomes even more important to balance their contributions.

Our general approach is to first choose a simple setting for each of the lambdas, produce an n-best list of candidates for a large body of sentences with known translations, and then attempt to select lambdas that maximize some objective function or discriminative ability. Our initial lambdas are simple: for each probability model, assign an initial lambda of 1, and for the non-probabilistic models (phrase count, word count, etc), assign an initial lambda of 0. We then translate, preserving up to 1,000 best candidates.

We apply two different automated evaluation metrics (or oracle scores) for lambda evaluation: word-error rate (WER), and BLEU. Our final development evaluation metric was BLEU, so we were unsurprised to find that maximum BLEU training was able to produce the best results.

5.1 Perceptron

Our first attempt at selecting lambdas was to use Perceptron-based discriminative reranking [33], due to its simple algorithmic structure and fast training speed. Two reranking methods are suggested in this paper: the first based on splitting the candidate set into broad brush subsets, and the second based on a more exhaustive pairwise comparison. The objective function used in these experiments was Word Error Rate (WER). The next two sections are a review of the approaches described by Shen. The remainder of this section is devoted to our modifications and experiences with these approaches.

5.1.1 Splitting

The splitting algorithm divides each n-best list into three sections according to their oracle scores: a set of good translations, a set of bad translations, and a set of neutral translations. After sorting the n-best lists by oracle score, the sets are defined by the externally selected variables r and k, where r is the number of top translations considered to be good, and k is the number of bottom translations considered to be bad. If at any point the weighted linear combination of available features indicates that a translation from the bad set is better than a translation from the good set, then it is treated as an error, and the perceptron weights are adjusted according to the difference of their feature values. Let $\mathbf{x}_g$ be the feature vector of the good example and $\mathbf{x}_b$ be the feature vector of the bad example. The perceptron algorithm will adjust its weights $\mathbf{w}$ according to:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \left(\mathbf{x}_g - \mathbf{x}_b\right)$$

It stores all errors seen for a particular n-best list, and updates the weights only after processing that list. It then uses the updated weight-set on the next n-best list. The process repeats until either the algorithm reaches some maximum number of iterations, or until the current weight set perfectly discriminates between the good and bad sets of every n-best list in the training data.

5.1.2 Ordinal regression

Ordinal regression works in very much the same way as splitting, except instead of adjusting when an item in the "good" set is confused with an item in the "bad" set, it adjusts for all pair-wise errors that meet some distance criteria. The algorithm currently registers a pair-wise error when the weight set ranks $\mathbf{x}_b$ above $\mathbf{x}_g$, and the true oracle rank $o(\mathbf{x}_g)$ is twenty ranks above $o(\mathbf{x}_b)$, and $2o(\mathbf{x}_g)$ is still a better rank than $o(\mathbf{x}_b)$ (to avoid having the algorithm spend too much effort in correcting errors at the bottom of the list).([32])

The second innovation in ordinal regression is to adjust weights (and count errors) differently depending on the severity of the error. A function g ensures that an error confusing the candidates near the top in oracle ranking is more important than confusing candidates near the bottom of the list, while confusing candidates that are far apart is more important than confusing candidates that are close together. In effect, the weight adjustment described above in the Splitting section is weighted according to g:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + g(\mathbf{x}_g, \mathbf{x}_g)\left(\mathbf{x}_g - \mathbf{x}_b\right)$$

where

$$g(x, y) = \frac{1}{o(x)} - \frac{1}{o(y)}$$

5.1.3 Modifications

Due to problems finding error criteria that make our data linearly separable, several modifications were made to the perceptron algorithm to handle the cases where it does not converge to a solution with no training errors. The first solution to this problem was to introduce a learning rate, which is not mentioned [33], but presumably is necessary in most implementations. [22] prefaces all perceptron convergence guarantees with, "using a low enough learning rate," in addition to the linear separability assumption. The update term is multiplied by this learning rate before each weight is updated.

In the cases where linear separability is not possible (or perhaps not desirable), then a low learning rate can be used to allow the algorithm to approximate some form of gradient descent minimizing square error [22]. In this case, the algorithm may not necessarily converge; instead, the best error rate seen so far is tracked, and at the end of a fixed number of iterations, the weights corresponding to this error rate are returned.

The rerank algorithm used here is neither a standard perceptron nor an averaged perceptron[1]. The weight vector returned to the user is the average of all weight vectors used across the various n-best lists in the last iteration. This makes it more stable than a standard perceptron, but perhaps not as stable as a true averaged perceptron or a voting perceptron.

5.1.4 Choosing r and k

In the case of the splitting algorithm, a large number of strategies were attempted to try to find good values for r and k. This section will describe one method briefly. r is set to the number of candidates in the n-best list that have an oracle score greater than the top-scoring candidate under the initial weighting scheme. It effectively targets these candidates as the ones whose scores need to be fixed. In order to balance the number of positive and negative examples (personal communication, Libin Shen), we set k to be the same size as r. Afterward, a check corrects for overlap (no one is both a good and bad translation), and then rebalances for set size. Note that we may select a different r and k for each sentence.

5.1.5 Experience in practice

It is very important that the data be linearly separable. All the work done by Shen et al to introduce splitting, or g() functions or distance criteria has only one purpose: to define the error criteria so that the data set becomes linearly separable. In addition, one must use large n-best list sizes to achieve separability, though we found little improvement in results when using more than 1,000 translations.

We were only able to achieve linear separability with the splitting approach only by (1) ignoring candidate lists with fewer than 500 entries and (2) considering a very few good and bad translations (e.g. $r = k = 9$). In this case, the

[1]At each stage in perceptron training, we have a weight for each of the features. Normally the weights established by a perceptron are the weights of the final iteration. One simple means of preventing overfitting is to instead return the average of the weights over all timesteps; we refer to this as an *averaged perceptron*.

algorithm behaves very well, marching toward zero errors, and reacting predictably when features are uniformly negated. An alternate approach, which ignores linear separability and simply relies on a low training rate to slowly inch toward good weights, is also rather effective.

Ordinal regression is intuitively more appealing than splitting, however it is much slower. Splitting is capable of ignoring most of the pair-wise comparisons in the data set. It need only walk through all of k for each item in r. However, regression has a more complex distance criteria, which makes it have complexity of $O(n^2)$ in the number of candidates. For this reason, we did not explore it as completely as splitting, though the limited experimentation that we did pursue showed no great advantage in performance.

5.1.6 Remaining issues

Despite all our experimentation with parameter settings, we found little improvement in final BLEU score with the lambdas suggested by these methods. We suspect three main causes. First, all the methods are focused on finding a linear separator for some subsets of candidates, yet this is not necessarily our desired task. Instead, we wish to maximize the objective function score of the single best translation. A second major cause may be that we evaluate in terms of BLEU score, but the method is trying to discriminate based on WER. We did not try the perceptron algorithm with an ordering based on BLEU scores.

Finally, the methods described above focus very specifically on n-best list rank order, without regard for the actual objective function values. In most n-best lists, we observe only a small number of objective function values, so for a 1000-best list, there may be 50 or more translations with the same WER score. In the splitting approach, we choose a fixed r and k at the beginning regardless of the objective function values: such a cutoff often will not include all translations with the same score, and which translations it selects is rather arbitrary. Similarly, ordinal regression assumes that the n-best list is totally ordered by the objective function, when it fact it may only be partially ordered, and tries to recreate an arbitrary total ordering of that partial order. In addition, certain sentences may have nearly uniform translation quality, where others vary widely between high quality and poor translations; using the same cutoff for all sentence pairs fails to capture this distinction. While the weighting function g described above helps alleviate some of these problems in the ordinal regression approach, a training method more focused on the objective function holds greater promise in our opinion.

5.2 Maximum BLEU training

Recent work by Franz Och ([24]) describes a method of optimizing any objective function directly–whether it is BLEU, WER, or some unknown new evaluation score. As above, this method requires a translation system that can provide an n-best list for any input sentence, a set of continuously valued model scores for each translation, and some objective function. The remainder of this section is a review of Och's method. The next section describes some improvements we made to increase the stability of the algorithm.

At a high level, the Maximum BLEU algorithm operates much like any multi-dimensional function optimization approach: A direction is selected and the objective function is maximized along that direction using a line search. This is repeated until no effect is produced.

Och's novel contribution is in the implementation of the line search. First we restrict ourselves to the n-best list of one translation. Let **w** be our current set of weights and **d** be the direction along which we wish to line search, then the final score assigned to any translation **x** at point λ is $f_{\mathbf{x}}(\lambda) = \mathbf{x} \cdot (\mathbf{w} + \lambda \mathbf{d})$, a linear function of λ. Thus we know that the highest score $\hat{f}(\lambda)$ is piecewise linear, which allows for an efficient computation and representation of the highest-scoring translation at value of λ:

$$\hat{\mathbf{x}}(\lambda) = \underset{\mathbf{x}}{\operatorname{argmax}} \{ f_{\mathbf{x}}(\lambda) \}$$

This in turn leads to a compact representation of the objective function at any value of λ. It is a simple matter to extend this to a whole corpus by combining the piecewise linear representations of each sentence. Finding the maximal BLEU score along that line only requires walking the exhaustive representation and returning the maximal scoring value.

Despite some claims to the contrary ([24]), we found that the algorithm often converged on different maxima given different starting points. We therefore attempt to run the algorithm ten times each from a different, random starting point, and pick the weights corresponding to the maximal objective function value.

5.2.1 Stability improvements

One potential cause of instability is that the algorithm seeks the maximum value without regard for the range of parameters under which this value is maximized. Thus a narrow, unstable peak with the highest value might be selected in favor of a broad, more stable peak with a slightly lesser value. To counteract this phenomenon, we propose maximizing a moving average of the objective function as opposed to maximizing the objective function directly. More formally, assume we are given the objective function $\hat{\mathbf{x}}(\lambda)$ and a breadth of the moving average b. Then we can compute a moving average function as follows:

$$avg_{\hat{\mathbf{x}},b}(\lambda) = \int_{\lambda-b}^{\lambda+b} \hat{\mathbf{x}}(\lambda')\, d\lambda'$$

Again, the piecewise linear form of $\hat{\mathbf{x}}$ allows for an easy computation of this moving average.

Chapter 6

Experiments

We evaluated the translation quality of the system using the BLEU metric ([30]). Both the DP-inspired and the A* decoder were evaluated; the latter was tested under several settings to explore the speed vs. quality trade-offs.

We compared against several systems to demonstrate the competitiveness of this approach:

- A monotone phrasal SMT system. In this system word reordering is restricted to the local reordering captured within phrase translation pairs. The BLEU score difference between a monotone decoder and a decoder allowing reorderings appears to be on the order of five percentage points ([37]).
- A hybrid MT system developed here at Microsoft Research ([20]). The English-French version of this system employs a deep parser in both the source and target language and a machine-learned sentence realization system in French ([31]).

6.1 Data

We started with a parallel English-French corpus of Microsoft technical data: this includes product documentation, support articles, and application strings, amongst others. From these 1,566,265 sentence pairs, we selected a small subset for quicker training and test. We only kept sentence pairs where both sides were between 40 and 160 characters long and contained no XML or HTML tags[1], leaving a set of 570,562 sentence pairs. From this set, we held out 2,000 sentences for development testing, 2,000 sentences for testing, and 250 sentences for lambda training. We experimented with various training set sizes, ranging from 1,000 to 300,000.

[1]The use of tags in our corpus is rather idiosyncratic. In some sentences they denote technical terms or phrases; in others they denote purely presentation information. Therefore we decided to omit all tagged sentences to simplify the situation and make our evaluation more standard.

6.2 Training

Tree-to-string system

We parsed the source (English) side of the corpus using the handcrafted NLP-WIN parser developed at Microsoft Research. This parser is able to produce many varieties of analysis, ranging from surface (morphological analysis and part-of-speech tagging) to deep (predicate argument structure with many types of syntactic normalizations). For the purposes of these experiments we only used a dependency tree output identifying part-of-speech tags, but without morphological analysis.

For word alignment, we used the GIZA++ package [26]. We followed a rather standard training regimen of 5 iterations of Model 1, followed by 5 iterations of the HMM Model, followed by 5 iterations of Model 4; this was applied in both directions.

We then projected the dependency trees and used the aligned dependency tree pairs to extract treelet translation pairs, train the order model, and train the agreement model. The target language model was trained using only the French side of the corpus; additional sentences may improve the performance of such a target language model.

Finally we trained lambdas via Maximum BLEU on 250 held out sentences, each with a single reference translation.

Monotone phrasal SMT baseline

The same GIZA++ alignments used in the tree-to-string system were used to train the monotone phrasal SMT system. As is common practice, we used a heuristic combination of the monolingual alignments that attempts to minimize AER[2]. From this combined alignment, we extracted phrasal translation pairs. The same target language model used above was employed here. At decoding time, only two models were used: a MLE channel model, and the target language model. Each was given equal weight.

MSR-MT

The existing machine translation system MSR-MT used its own word alignment approach as described in [20] on the same training data.

6.3 Results

We present BLEU scores on a 2,000 sentence test set using a single reference translation for each sentence. Table 6.1 compares results from a variety of different approaches. The monotone decoder produces results that are significantly worse than the other approaches. Although the stated limitations of this

[2]The heuristic combination for the phrasal SMT system followed [28] rather than the method described in Chapter 2

implementation — such as the lack of different translation models and its limited reordering capabilities — account for some amount of that shortfall, the difference is still quite large. In addition, the tree-to-string system slightly outperforms our existing MSR-MT system on the same training data. The MSR-MT system has proven effective in translating Microsoft technical documentation, so a win against that system provides validation that the tree-to-string translations will be useful to end users.

Decoding approach	BLEU score
Monotone, max len 4	29.00 +/- 1.15
MSR-MT	37.92 +/- 1.36
T2S A*-fast	39.05 +/- 1.14
T2S A*-slow	39.02 +/- 1.11
T2S DP-fast	38.96 +/- 1.17

Table 6.1: Comparison of various approaches on 100k training data.

To determine the impact of adding additional data, we also experimented using different dataset sizes. Note that all results are presented for the DP-inspired decoder using high-speed settings.

Training set size	BLEU score
1,000	15.72 +/- 0.99
3,000	24.66 +/- 1.07
10,000	31.05 +/- 1.34
30,000	34.45 +/- 1.53
100,000	38.96 +/- 1.17
300,000	40.64 +/- 1.60

Table 6.2: Learning curve.

Chapter 7

Conclusions and future work

We have described a novel approach to syntactically-informed statistical machine translation. Key components include the use of a parsed dependency tree representation of the source language in every aspect of the translation system, the application of the best modeling and decoding advances of the string-based phrasal SMT community to these dependency tree representations, and a particular emphasis on a novel tree-based ordering model. The introduction of two efficient decoding approaches enable quick experimentation in this area and open the door to real-world applications.

Initial experimentation suggests this approach holds promise. However, this effort only scratches the surface of possible lines of experimentation; we see quite a few areas for improvement and enhancement.

Currently we only consider the top parse of an input sentence. One means of considering alternate possibilities is to build a packed forest of dependency trees and use this in decoding translations of each input sentence. In the DP-inspired decoder, we can then translate starting at each possible root node in the forest, and, in the end, pick the best translation at any of the possible roots.

Our models are trained using only the Viterbi word alignment from a surface string alignment component. Another potential improvement would be to use several of the top ranked alignments instead of the single best, or even to incorporate parse information (as in [5]) into the word alignment process.

A novel contribution of our system is a linguistically motivated approach toward ordering based on the source dependency tree. While this paper explores two promising models, we believe that this merely scratches the surface. Within this paradigm, different model structures (whether discriminative or generative), alternate machine learning methods, and novel representations of the target features all have the potential for significant improvements.

Finally and perhaps most importantly, we plan to stage more exhaustive comparisons against other systems. The performance of the monotone phrasal SMT decoder suggests that we may be competitive with state-of-the-art SMT systems, but a more careful evaluation must be performed.

Bibliography

[1] Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. Learning dependency translation models as collections of finite-state head transducers. *Computational Linguistics*, 26(1):45–60, 2000.

[2] Anthony Aue, Arul Menezes, Robert C. Moore, Chris Quirk, and Eric Ringger. Statistical machine translation using labeled semantic dependency graphs. In *Proceedings of the Tenth International Conference on Theoretical and Methodological Issues in Machine Translation,*, 2004.

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

[4] Eugene Charniak, Kevin Knight, and Kenji Yamada. Syntax-based language models for statistical machine translation. In *Proceedings of the MT Summit*, 2003.

[5] Colin Cherry and Dekang Lin. A probability model to improve word alignment. In *Proceedings of the ACL*, 2003.

[6] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[7] Yuan Ding and Martha Palmer. Automatic learning of parallel dependency treelet pairs. In *Proceedings of the First International Joint Conference on Natural Language Processing*, 2004.

[8] Yuan Ding and Martha Palmer. Synchronous dependency insertion grammars: A grammar formalism for syntax based statistical MT. In *COLING 2004: Workshop on Recent Advances in Dependency Grammars*, 2004.

[9] Jason Eisner. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the ACL*, 2003.

[10] Heidi J. Fox. Phrasal cohesion and statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Proceedings*, 2002.

[11] Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. Fast decoding and optimal decoding for machine translation. In *Proceedings of the ACL*, 2001.

[12] Daniel Gildea. Loosely tree-based alignment for machine translation. In *Proceedings of the ACL*, 2003.

[13] Joshua Goodman. A bit of progress in language modeling, extended version. Technical Report MSR-TR-2001-72, Microsoft Research, 2001.

[14] Jonathan Graehl and Kevin Knight. Training tree transducers. In *Proceedings of the Joint HLT/NAACL Conference*, 2004.

[15] Kevin Knight. Decoding complexity in word-replacement translation models. *Computational Linguistics: Squibs and Discussion*, 25(4), 1999.

[16] Philipp Koehn. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In *Proceedings of the 6th Conference of the Association for Machine Translation in the Americas (AMTA)*, pages 115–124, Washington, USA, September 2004.

[17] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the Joint HLT/NAACL Conference*, 2003.

[18] Dekang Lin. A path-based transfer model for machine translation. In *Proceedings of COLING*, 2004.

[19] I. Dan Melamed and Wei Wang. Statistical machine translation by parsing. Technical Report 04-024, Proteus Project, 2004.

[20] Arul Menezes and Stephen D. Richardson. A best-first alignment algorithm for automatic extraction of transfer mappings from bilingual corpora. *Proceedings of the Workshop on Data-driven Machine Translation*, 39, 2001.

[21] Rada Mihalcea and Ted Pedersen. An evaluation exercise for word alignment. In Rada Mihalcea and Ted Pedersen, editors, *HLT-NAACL 2003 Workshop: Building and Using Parallel Texts: Data Driven Machine Translation and Beyond*, pages 1–10, Edmonton, Alberta, Canada, May 31 2003. Association for Computational Linguistics.

[22] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

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

[24] Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the ACL*, 2003.

[25] Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. A smorgasbord of features for statistical machine translation. In *Proceedings of the Joint HLT/NAACL Conference*, 2004.

[26] Franz Josef Och and Hermann Ney. Improved statistical alignment models. In *Proceedings of the ACL*, pages 440–447, Hongkong, China, October 2000.

[27] Franz Josef Och and Hermann Ney. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the ACL*, 2002.

[28] Franz Josef Och, Christoph Tillmann, and Hermann Ney. Improved alignment models for statistical translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Proceedings*, 1999.

[29] Franz Josef Och, Nicola Ueffing, and Hermann Ney. An efficient A* search algorithm for statistical machine translation. In *ACL 2001: Data-Driven Machine Translation Workshop*, pages 55–62, Toulouse, France, July 2001.

[30] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the ACL*, pages 311–318, Philadelpha, Pennsylvania, 2002.

[31] Eric Ringger, Michael Gamon, Robert C. Moore, David Rojas, Martine Smets, and Simon Corston-Oliver. Linguistically informed statistical models of constituent structure for ordering in sentence realization. In *Proceedings of COLING*, pages 673–679, 2004.

[32] Libin Shen. Personal communication, July 2004.

[33] Libin Shen, Anoop Sarkar, and Franz Josef Och. Discriminative reranking for machine translation. In *HLT/NAACL 2004*, Boston, USA, May 2–7 2004.

[34] Gregor Thurmair. Comparing rule-based and statistical MT output. In *The Amazing Utility of Parallel and Comparable Corpora Workshop, LREC*, 2004.

[35] Christoph Tillmann, Stephan Vogel, Hermann Ney, and Alex Zubiaga. A DP-based search using monotone alignments in statistical translation. In *ACL*, 1997.

[36] Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *Proceedings of the ACL*, 1996.

[37] Stephan Vogel, Ying Zhang, Fei Huang, Alicia Tribble, Ashish Venugopal, Bing Zhao, and Alex Waibel. The CMU statistical machine translation system. In *Proceedings of the MT Summit*, 2003.

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

[39] Dekai Wu and Hongsing Wong. Machine translation with a stochastic grammatical channel. In *Proceedings of the ACL*, 1998.

[40] Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proceedings of the ACL*, 2001.

[41] Richard Zens and Hermann Ney. A comparative study on reordering constraints in statistical machine translation. In *Proceedings of the ACL*, 2003.

[42] Richard Zens and Hermann Ney. Improvements in phrase-based statistical machine translation. In *Proceedings of the Joint HLT/NAACL Conference*, pages 257–264, Boston, USA, May 2004.

[43] Richard Zens, Hermann Ney, Taro Watanabe, and Eiichiro Sumita. Reordering constraints for phrase-based statistical machine translation. In *Proceedings of COLING*, 2004.