LAMP-TR-152 November 2009
CS-TR-4947
UMIACS-TR-2009-15

Extending Phrase-Based Decoding with a Dependency-Based Reordering Model

Tim Hunter and Philip Resnik

Computational Linguistics and Information Processing Laboratory
Language and Media Processing Laboratory
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742-3275
{*timh,resnik*}*@umd.edu*

Abstract

Phrase-based decoding is conceptually simple and straightforward to implement, at the cost of drastically oversimplified reordering models. Syntactically aware models make it possible to capture linguistically relevant relationships in order to improve word order, but they can be more complex to implement and optimise.

In this paper, we explore a new middle ground between phrase-based and syntactically informed statistical MT, in the form of a model that supplements conventional, non-hierarchical phrase-based techniques with linguistically informed reordering based on syntactic dependency trees. The key idea is to exploit linguistically-informed hierarchical structures only for those dependencies that cannot be captured within a single flat phrase. For very local dependencies we leverage the success of conventional phrase-based approaches, which provide a sequence of target-language words appropriately ordered and ready-made with the appropriate agreement morphology.

Working with dependency trees rather than constituency trees allows us to take advantage of the flexibility of phrase-based systems to treat non-constituent fragments as phrases. We do impose a requirement — that the fragment be a novel sort of "dependency constituent" — on what can be translated as a phrase, but this is much weaker than the requirement that phrases be traditional linguistic constituents, which has often proven too restrictive in MT systems.

This work has been supported in part by Department of Defense contract RD-02-5700 and the GALE program of the Defense Advanced Research Projects Agency, Contract No. HR0011-06-02-001. Any opinions, findings, conclusions or recommendations expressed are those of the authors and do not necessarily reflect the views of the sponsors.

1 Introduction

A significant step forward in machine translation was the move from word-based translation models, originating from the IBM approach (Brown et al., 1990, 1993), to phrase-based translation models (Och et al., 1999; Koehn et al., 2003). This allows a sequence of contiguous source-language words to be atomically replaced with a sequence of contiguous target-language words. These "phrases" need not be phrases in the sense of any pre-existing theory of natural language syntax. Indeed the freedom to translate fragments such as 'he said' as a unit, despite the fact that this is not normally treated as a phrase in syntactic theories, is often considered a strength of this approach.

Another natural modification of the IBM models is to arrange (the target-language sides of) the translation units in a tree structure rather than in a flat linear structure, in an attempt to more accurately model long-distance dependencies and cross-linguistic variation in word order. The choice of flat structures or tree structures is independent of the choice of words or phrases as translation units.

A more recent step forward in statistical machine translation, hierarchical phrase-based models, integrates these two variations on the IBM approach, using multi-word translation units arranged in a tree structure. In addition, however, these systems allow phrases to be nested within each other, and therefore use translation units which are themselves hierarchical, unlike the original phrase-based systems.

The alternative suggested here arranges multi-word translation units in hierarchical structures (specifically, dependency structures), but the translation units themselves have no hierarchical structure: they are exactly those used in the original phrase-based systems. In this respect it explores a kind of "middle ground" between existing approaches. As with all syntax-based approaches, we use structural relations to judge the goodness of an ordering of (the target-language sides of) translation units in a somewhat linguistically-informed way; in contrast, in the original phrase-based systems, the only well-defined question to ask about a particular ordering of phrases is to what extent it differs from a direct monotonic translation (the "default case", carried over from the speech recognition systems which inspired the IBM models).

It must be stated at the outset that the approach we describe failed to produce improvements over a conventional phrase-based MT baseline. Therefore the value in this paper is primarily as a case study in developing an idea from a set of motivating observations, through representational and algorithm development, to experimental evaluation and clear-eyed assessment of the results.

The rest of this paper is organised as follows. In Section 2 we review in more detail previous approaches to statistical machine translation and how they relate to the system we propose here. We present in detail our dependency-based reordering model in Section 3, and the accompanying decoding algorithm in Section 4. Finally we discuss results of experiments translating Czech to English and Arabic to English in Section 5 and provide some concluding thoughts in Section 6.

2 Previous Related Work

In this section we briefly review other approaches to integrating syntactic information with phrase-based translation.

In the original phrase-based translation systems (Och et al., 1999; Marcu and Wong, 2002; Koehn et al., 2003), we can identify two ways in which "reorderings" of sentence fragments, be they words or larger fragments, can occur. First, the two reordered pieces might be covered by a single phrase. For example, we might have extracted $\langle$*ballon rouge*, *red ball*$\rangle$ as a translation unit during training. In this case the reordering "comes for free". The second, more complicated case, is where the two reordering pieces are not covered by a single phrase. To generate correct translations of this sort, the decoder must decide to reverse the order of the two separate phrases. For example, if our extracted phrases include only $\langle$*rouge*, *red*$\rangle$ and $\langle$*ballon*, *ball*$\rangle$ and we wish to translate *ballon rouge*, the decoder must decide to prefer the order *red ball* over *ball red*. Standard phrase-based translation systems simply set a preference for "monotonic" orderings of phrases, and rely on other models (eg. language models) to override this preference when appropriate.

Another line of work, originally independent of the shift from word-based to phrase-based translation systems, sought to take advantage of cross-linguistic generalisations concerning variation in word order. In general this requires arranging the words of a sentence in some sort of *hierarchical* representation of the structure of sentences, in contrast to the phrase-based approaches described above. Yamada and Knight (2001), for example, parse the source sentence to be translated and apply a statistical model to choose suitable reorderings. For each node of the parse tree, they use a probability distribution over the $n!$ different orderings of the node's n children; word-to-word translation applies at each terminal node of the resulting reordered tree (with the additional possibility that nodes may be inserted). This "reorder-insert-translate" noisy channel model is used at decode-time. Similar strategies, where hierarchical structures of words are evaluated for linguistic "goodness" at decode-time, were adopted by Wu and Wong (1998), Gildea (2003) and Galley et al. (2004), among others. Another approach, developed by Collins et al. (2005) and Xia and McCord (2004), is to use hierarchical syntactic structures to reorder the source sentence as a preprocessing step, before feeding the reordered sentence to a standard phrase-based decoder. The intuition behind this is that if all the significant reorderings are taken care of in the preprocessing, then the phrase-based system's bias towards monotonic decoding will be relatively well-suited to translating the result.

These two advances — translating multiple words atomically, and linguistically evaluating hierarchical arrangements of words — were integrated in a number of systems that arranged multi-word translation units in hierarchical structures. Chiang (2005) generalises the training procedure to extract not only pairs of flat phrases like $\langle$*ballon rouge*, *red ball*$\rangle$, but also pairs of sequences that may contain "variables" (or, understood as part of a synchronous context-free grammar, non-terminals), such as $\langle$*ne X pas*, *not X*$\rangle$. These rules are not based on any parse trees, but rather are extracted from an aligned parallel corpus on the basis of co-occurrence; the resulting system is therefore, as Chiang notes,"formally syntax-based" but not "linguistically syntax-based". Quirk et al. (2005) adopt a similar but linguistically syntax-based approach using dependency trees, extracting treelet-pairs from a parallel corpus with dependency parses on one side during training; Shen et al. (2008) integrate this dependency-treelet idea with a dependency language model that conditions on linear order somewhat similarly to the reordering model we present below. In these systems, once a selection of translation units has been decided upon that covers the source sentence to be translated, there is no independent question of how these translation units should be ordered; the order of *not* and the translation of X, in the simple example above, is determined by the target side of the translation unit. This distinguishes them from both flat phrase-based approaches and from earlier syntax-based approaches, where the choice of translation units was independent of the order in which the selected fragments of the target language will be arranged.

The approach we describe in this paper explores a middle ground between the three kinds of systems described above. We combine multi-word translation units with hierarchical notions. However, hierarchy is only introduced *among translation units*. Unlike Chiang (2005) and Quirk et al. (2005), our translation units are not hierarchical; they are exactly the ones used in the original phrase-based systems (Koehn et al. (2003) etc.). Rather than the purely monotonic-biased reordering model, we use hierarchical notions to decide among orderings of these flat phrases. These decisions are made as part of the decoding process, as for Yamada and Knight (2001) and others, not as a preprocessing step as for Collins et al. (2005) and Xia and McCord (2004). Like Quirk et al. (2005), but unlike Chiang (2005), we adopt a linguistically syntax-based approach, and also use dependency structures rather than constituency structures; this choice permits some additional flexibility with respect to the sense in which the two languages' structures must "match", as we will discuss below. Recent work presented in Galley and Manning (2008) is very similar to ours in that it builds hierarchical structures of flat phrases, but in contrast is formally syntax-based and not linguistically syntax-based.

A system that occupies this middle ground inherits a number of technical simplicities from the original phrase-based systems. Since we use only the conventional flat phrases as translation units, our system can use "phrase tables" designed for use in earlier systems. We also maintain a left-to-right decoding algorithm, avoiding the need for more complex tree-based decoding procedures. Our approach differs from a standard phrase-based system only in having an additional model contribute scores to hypotheses. In short, we aim to maintain as much as possible of the benefit and simplicity that phrase-based translation brought, but replace the least appealing aspect of such systems — the bias towards monotonic translations — with a linguistically-informed reordering model.

The relationships between this new approach and various existing systems are illustrated in Figure 1 and Table 1.

3 Model

As in standard phrase-based systems, the probability of an English translation e given a foreign sentence f is computed via a log-linear model:

$$P(e|f) = P_\phi(f|e)^{\lambda_\phi} \times P_\ell(e)^{\lambda_\ell} \times P_d(e|f)^{\lambda_d} \times \omega^{|e|\lambda_\omega} \tag{1}$$

where P_ϕ is the translation probability model, P_ℓ is the language model, P_d is the distortion model, and ω is the word penalty. Only the distortion model P_d differs from that used in standard phrase-based systems.

In order to clearly present the model we use for assigning scores to orderings of phrases in Section 3.2, we first introduce a model assigning scores to orderings of *words* in Section 3.1, which provides the basic intuition to be exploited.

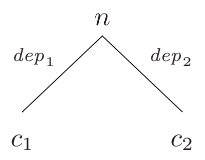
3.1 Background: Word-Reordering Model

Given a dependency-parsed corpus of (say) English, we can train a model of the probability (in English) of various linearisations of the nodes of any dependency tree. *Ignoring linearisations where dependency-subtrees are not contiguous*, choosing a linearisation amounts to choosing an ordering for each set $\{n, c_1, c_2, ..., c_k\}$, where the c_i are the child subtrees of node n. We can approximate the probability of such an ordering O by assuming that the probability of a child subtree being at a particular linear offset from the parent n is independent of the offset of all its sisters from the shared parent n:

$$P(O) = \prod_{i=1}^{k} P(\delta_i | dep_i) \tag{2}$$

where dep_i is the label on the dependency linking (the head of) child subtree c_i to its parent n, and δ_i is the offset between n and (the closest edge of) c_i in ordering O, measured as a number of words. The probabilities on the right hand side of (2) can be easily estimated via maximum likelihood estimates from frequencies in a sufficiently-large dependency-parsed English corpus.

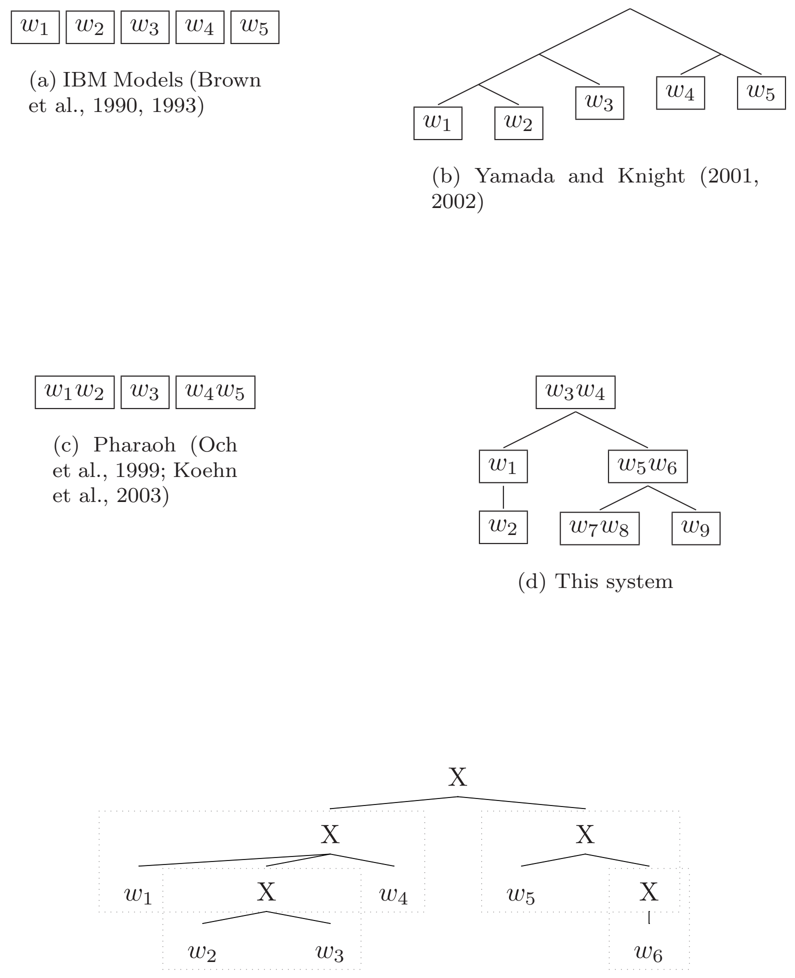
For example, given the following simple dependency tree for a three-word sentence, the probability of each of the six possible orderings of the three words is computed as shown:



$$\begin{aligned}
P(nc_1c_2) &= P(+1|dep_1)\,P(+2|dep_2)\\
P(nc_2c_1) &= P(+1|dep_2)\,P(+2|dep_1)\\
P(c_1nc_2) &= P(-1|dep_1)\,P(+1|dep_2)\\
P(c_1c_2n) &= P(-2|dep_1)\,P(-1|dep_2)\\
P(c_2nc_1) &= P(-1|dep_2)\,P(+1|dep_1)\\
P(c_2c_1n) &= P(-2|dep_2)\,P(-1|dep_1)
\end{aligned}$$

In the first case (nc_1c_2), c_1 occurs "one step to the right" of the parent n to which it is linked via a dependency labelled dep_1, hence the offset of +1 and the factor $P(+1|dep_1)$. Similarly, c_2 occurs "two steps to the right" and is linked to n via a dependency labelled dep_2, hence the factor $P(+2|dep_2)$.

By assuming in addition that the probability of a particular ordering of each node and its child subtrees is independent of the ordering of any other node in the tree and its child subtrees, we can find the linearisation of the words in a foreign dependency-parsed sentence which most closely matches "English word order", by choosing the best ordering O for each set containing a node and its child subtrees.



(a) IBM Models (Brown et al., 1990, 1993)

(b) Yamada and Knight (2001, 2002)

(c) Pharaoh (Och et al., 1999; Koehn et al., 2003)

(d) This system

(e) Hiero (Chiang, 2005)

Figure 1: Various strategies for arranging translation units. Boxes represent translation units.

Translation units	**Arranged linearly**	**Arranged hierarchically**
one word	Brown et al. (1990, 1993)	Yamada and Knight (2001, 2002)
flat multi-word	Och et al. (1999); Koehn et al. (2003)	this system
hierarchical multi-word		Chiang (2005)

Table 1: One way of segmenting the space of statistical machine translation systems

Given a table of translations for single foreign words, a naive translation method would be to take the most "English-like" ordering of the foreign words, and translate each foreign word individually. The next section integrates the motivation behind this naive approach with the advantages of phrase-based translation systems.

3.2 Phrase-Reordering Model

The reordering model P_d in (1) needs to assign a probability to a particular ordering of phrases, rather than words. To combine the advantages of the word-reordering model with the advantages of phrase-based translation systems, we try to have the phrase-translation model translate small subtrees completely, and order the translations of these subtrees according to the reordering model.

3.2.1 An Example

Suppose that we need to translate the foreign language sentence `MAN THE WOMAN TALL THE SAW` (call this sentence f) into English given the dependency tree in Figure 2.

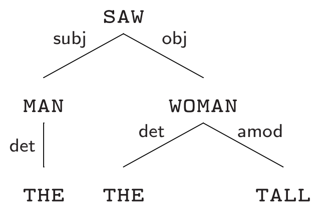


Figure 2: Dependency parse of the foreign sentence to be translated

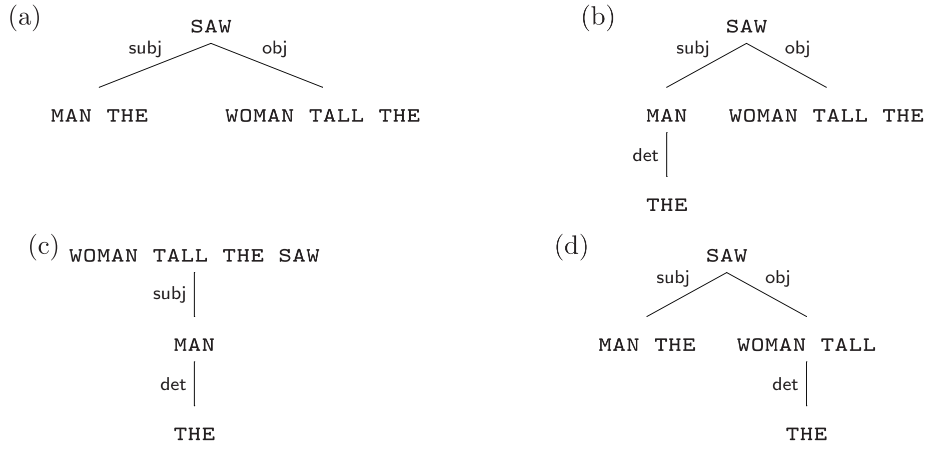


Figure 3: Four possible "compressions" of the dependency tree in Figure 2.

Before applying a reordering model to determine an English-like ordering of these one-word nodes, we can "compress" some portions of the dependency tree to produce flat, multi-word phrases. Four of the many possibilities for this dependency tree are shown in Figure 3. Details of the possible ways to compress dependency trees will be explained in Section 3.2.2.

Suppose we choose the compressed tree in Figure 3(b), and an English translation for each of the four foreign phrases in that tree, say *saw* for `SAW`, *the* for `THE`, *guy* for `MAN`, and *the tall lady* for `WOMAN TALL THE`. These are taken from exactly the same kind of phrase table as is used in standard phrase-based systems. We must now choose how to order these four English phrases. The distortion scores for some of the possibilities are:

$$P_d([\mathtt{saw}][\mathtt{the\ tall\ lady}][\mathtt{the}][\mathtt{guy}]|f) = P(+4|\mathsf{subj})\ P(+1|\mathsf{obj})\ P(-1|\mathsf{det}) \quad (3)$$
$$P_d([\mathtt{the\ tall\ lady}][\mathtt{saw}][\mathtt{the}][\mathtt{guy}]|f) = P(+1|\mathsf{subj})\ P(-1|\mathsf{obj})\ P(-1|\mathsf{det}) \quad (4)$$
$$P_d([\mathtt{the}][\mathtt{guy}][\mathtt{saw}][\mathtt{the\ tall\ lady}]|f) = P(-1|\mathsf{subj})\ P(+1|\mathsf{obj})\ P(-1|\mathsf{det}) \quad (5)$$

Since English subjects generally tend to be at a small negative (leftwards) offset from their governors (verbs), and English objects generally tend to be at a small positive (rightwards) offset from their governors (verbs), the third of these orderings will probably have the highest score. Note that in (3) we take the offset from the governor *saw* to the *closest word* of (the closest phrase of) the subj dependent, namely *the*, despite the fact that the head of the subj dependent is MAN.

3.2.2 The Allowable "Compressions" of a Tree

Choosing a "compression" of a dependency tree rooted at a node n with child subtrees $c_1, c_2, ..., c_k$ amounts to choosing a set $C \subseteq \{c_1, c_2, ..., c_k\}$. The chosen subtrees are the ones which are to be "squashed up" to combine with their parent node: the words contained in the subtrees $c_i \in C$, plus the word at node n, together form the phrase at the new root node n'. As children, n' has one compression of each of the $c_i \notin C$.[1]

The choice of the set C is restricted by the following constraints on the phrase at n' (that is, the words contained in each of the subtrees $c_i \in C$ together with the word at node n):

- the phrase must be contiguous in the original sentence
- the length of the phrase in words must not exceed a pre-specified threshold

The total range of possible compressions of the dependency tree in Figure 2, with a phrase-length threshold of four, is represented in Figure 4.

This tree can be read somewhat like an "and-or tree". The left-hand path at the first branch of the tree represents choosing to use [SAW] as one of the phrases to cover the input sentence with.[2] From this point, we must find a way to cover both the subj dependent *and* the obj dependent. In order to cover the subj dependent we must use either the phrases [MAN] (and in turn [THE]), *or* the phrase [MAN THE]; in order to cover the obj dependent we must use either [WOMAN] *or* [WOMAN TALL] *or* [WOMAN TALL THE] (with further consequences in the first two cases).

The two children of the node labelled $*$ represent the two alternate ways of compressing the subtree linked by the $*$ dependency to the ROOT node; that is, the two alternate ways of compressing the entire original dependency tree.

Note that there is no compression of this subtree where WOMAN and THE have been combined into one phrase with TALL as a child, because WOMAN and THE are not contiguous in the original source sentence.

To translate the foreign sentence we must choose a subset of the "phrase nodes" of this graph which covers the sentence exactly. These "phrase nodes" will form a dependency tree like those in Figure 3. Completing a translation therefore means choosing an ordering of these foreign phrases and choosing an English translation for each of these phrases from the phrase table.

3.2.3 Subtrees must be linearly contiguous

Given the set of n phrases we are going to use to cover the input sentence, not all of the $n!$ possible orderings of (the English translations of) these phrases are permissable. The reordering model imposes a further restriction: it only assigns a score to those linearisations where each subtree is linearly contiguous.

[1]This algorithm has the handy property that two phrases will be linked by a dependency in the compressed tree if and only if their two head words are linked by a dependency in the original, uncompressed tree.

[2]For the moment we talk as if this is the "first" choice made in constructing a covering of the input sentence, but of course the actual order used in decoding does not correspond to this simple top-down view. See Section 4 for details of the decoding algorithm.

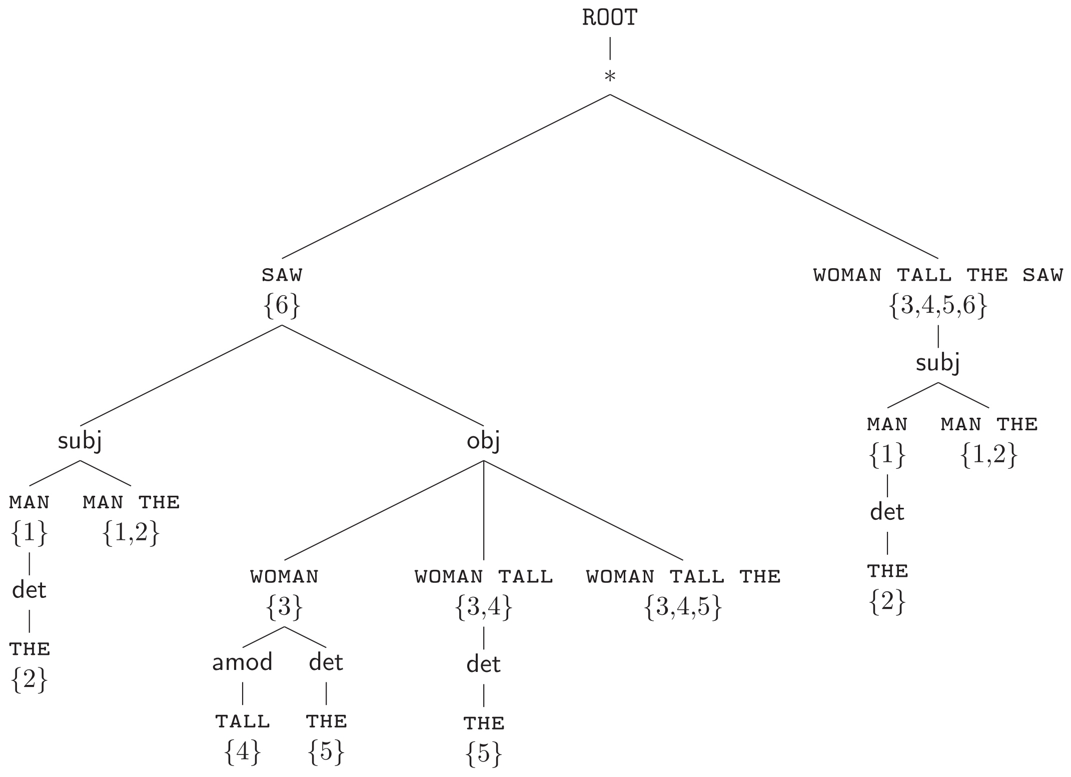
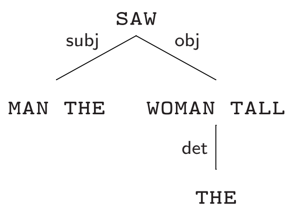


Figure 4: A representation of all the possible "compressions" of the dependency tree

For example, the selection of the phrases WOMAN TALL, THE, SAW and MAN THE from Figure 4 amounts to the selection of the compressed dependency tree in Figure 3(d), repeated here:



If these phrases were arranged linearly as, for example, [WOMAN TALL] [SAW] [THE] [MAN THE], the re-ordering model would not be able to assign a score to the obj dependency linking SAW and (the subtree headed by) WOMAN TALL, because there's no nearest edge of the subtree headed by WOMAN TALL; the subtree has been split.

3.2.4 Offsets from complex governors

In the examples in (3-5), the governor phrase of each dependency was always a single word. Those examples used compressions where dependent phrases were longer than a single word, in which case we took the closest edge of this phrase when calculating offsets. However, when a governor phrase is longer than a single word, it does not always make sense to determine offsets on the basis of the closest edge of the governor phrase.

For example, consider the following two orderings of phrases:

$$P_d([\texttt{saw the tall lady}][\texttt{the guy}]) = ?$$
$$P_d([\texttt{saw}][\texttt{the tall lady}][\texttt{the guy}]) = ?$$

We would like our model to assign the same score to the way we have linearised the subj dependency between SAW and MAN in each of these two cases, but if we only consider nearest edges, then the relevant offset for this dependency will be +1 in the first case but +4 in the second. The problem here is that in the first case, the governor phrase [*saw the tall lady*] itself contains (our translation of) the obj dependent of SAW, and the placement of this dependent is relevant to the linearisation of (our translation of) the subj dependent [*the guy*].

The solution is to calculate each dependency's offset on the basis of the word in the target-side governor phrase which is aligned to the governor word in the uncompressed source dependency tree. That sounds complicated but it's easy when you see it: we take the offset from the word *saw* to the phrase [*the guy*], because this is the word which is aligned to the source-language word which is the "real" governor of the subj dependency in the original uncompressed tree (Figure 2), namely SAW. We might say that SAW is the "head word" of the source-language phrase [WOMAN TALL THE SAW], of which *saw the tall lady* is our chosen translation. Note that *there is no sense in which the English word saw is the "head word" of the English phrase [saw the tall lady]* unless we take into account word alignments, because there is no uncompressed English dependency tree.

3.2.5 Fine Tuning

The extra details that the modifications described in this section add to the model are straightforward but tedious, so we abstract away from them in the rest of this paper for ease of exposition. Some sample calculations showing these details in all their glory are presented in Appendix C.

Normalisation of dependency labels As presented, this model assumes that there is a desirable target-language translation of the input sentence which shares precisely the same dependency tree (same structure, same labels). To allow for the possibility that a dependency labelled, say, subj in the input sentence should "correspond" to a dependency labelled, say, obj in the target sentence, we train a simple MLE-based model of the conversion from source-side dependency labels to target-side dependency labels. In fact, this may actually be required for the model to be able to assign scores to linearisations of dependencies because there may exist labels used in the source language dependency trees which do not appear in the target-language trees on which the linearisation model is trained.

To do this we look at the word-alignments generated in the course of phrase-table extraction, and parse trees on both sides of the training corpus. If two source-language words are linked by a dependency labelled subj, and two target-language words to which they are aligned are linked by a dependency labelled obj, we consider this to be one observation of a case where a subj dependency on the source side is realised as an obj dependency on the target side. For each source side dependency label we thus construct a probability distribution over target side dependency labels, according to which a source dependency label is "normalised" to a target dependency label before consulting our model of target side linearisation offsets.

Lexicalisation of the distortion model In order to allow for idiosyncratic linearisation properties of particular lexical items, we condition our distortion model probabilities not only on the label of the dependency being linearisation but also on the target-side lexical items appearing at each end of the dependency. If we have not observed enough instances of the (label, *gov_word*, *dep_word*) triple in the training data, we back off to less fine-grained conditions, the final option being to condition only on the (normalised) dependency label itself (as presented in the body of this paper).

3.3 Only a Subset of the Phrases Available to Standard Phrase-based Systems

This method allows only a subset of the foreign phrases which standard phrase-based systems allow. Such systems require that the words in a phrase be contiguous in the original sentence; our method requires this and more. For example, the phrase THE SAW in our example could be used by most phrase-based systems, but not by this system. It does not appear in the tree in Figure 4.

Two justifications for not considering this phrase are:

- We are better off translating words together in a phrase if there are dependencies between them, so that the English translations can be "ready made" with any agreement morphology which may result from the dependency. There is no dependency between `THE` and `SAW`.
- It is not clear what dependency would link the resulting phrase `THE SAW` to, for example, (the phrase containing) the word `WOMAN`; `THE` is its dependent by a dependency of type det, while `SAW` is its governor by a dependency of type obj. If orderings are to be scored based on the kind of model described above, the distortion score for a hypothesis translating `THE SAW` as a phrase would not be defined.

Note that a phrase such as *he said* is *not* excluded by this technique, because while these two words do not typically form a syntactic constituent, they are linked by a dependency. Attempts to improve on phrase-based systems' choices of phrases on the basis of *constituent* trees will penalise or reject such subject-verb combinations. Dependency trees are more forgiving, since the conversion from constituency trees to dependency trees abstracts away from hierarchy among elements that share a common governor. In other words, one can think of a dependency tree as representing an equivalence class of constituency trees. Fox (2002) has found that dependency structures are generally more faithfully maintained by human translations than constituency structures are.

4 Decoding

4.1 Search Space

The start state or initial hypothesis h_0 will be represented as follows:

$$h_0 : \boxed{\begin{array}{ll} \{\} & \\ 0 & \{\} \\ \epsilon & \end{array}}$$

The first line indicates the set of (indices of) foreign words covered so far (currently empty). The second line shows the number of dependencies which exist among the translated phrases (currently zero) and any "half-translated" dependencies (currently none). The third line shows the English translation so far. The translation is built up strictly left-to-right.

4.1.1 A Very Simple Example

To produce a new hypothesis from h_0, we must choose a phrase from Figure 4 and an English translation for that phrase. Suppose we choose the phrase `MAN THE` and the English translation *he*. Then the resulting hypothesis h_1 would cover words 1 and 2 from the input, which are linked by one dependency (the det dependency); and the subj dependency with governor word 6 and dependent word 1 would be "half-translated" (the significance of the half-translated dependencies will become clearer in the second example, below):

$$h_1 : \boxed{\begin{array}{ll} \{1,2\} & \\ 1 & \{((\mathsf{subj},6,1),1)\} \\ \mathit{he} & \end{array}}$$

The cost of this step through the hypothesis space is just the product of the phrase translation cost and the language model cost. The distortion model only plays a role when considering the ordering of phrases relative to other phrases, so it has nothing to do yet.

$$P(h_1) = P(h_0) \times P_\phi(\mathit{he}|\texttt{MAN THE})^{\lambda_\phi} \times P_\ell(\mathit{he}|h_0)^{\lambda_\ell} \tag{6}$$

From h_1 we might next choose the foreign phrase `WOMAN TALL THE SAW`, and the translation *`saw the tall woman`*. This phrase "contains" three dependencies (obj, det and amod), and also "completes" the subj dependency linking it to `MAN THE`, so the new hypothesis covers five dependencies (the entire sentence).

$$h_2 : \boxed{\begin{array}{ll} \{1,2,3,4,5,6\} & \\ 5 & \{\} \\ \multicolumn{2}{l}{\textit{he saw the tall woman}} \end{array}}$$

The cost of this step through the hypothesis space is the product of the phrase translation cost, the language model cost, and the cost of these two phrases being ordered as they are, given the subj dependency between them:

$$\begin{aligned} P(h_2) &= P(h_1) \times P_\phi(\textit{saw the tall woman}|\texttt{WOMAN TALL THE SAW})^{\lambda_\phi} \\ &\quad \times P_\ell(\textit{saw the tall woman}|h_1)^{\lambda_\ell} \times P_d(-1|\mathsf{subj})^{\lambda_d} \end{aligned} \tag{7}$$

Note that by selecting a particular set of phrases which cover the foreign sentence, we implicitly select a particular compression of the original dependency tree in the course of decoding. In this example, by selecting the phrases `MAN THE` and `WOMAN TALL THE SAW`, we have implicitly selected the following compression of the dependency tree as the basis for the translation:

$$\begin{tikzcd} \texttt{WOMAN TALL THE SAW} \arrow[d, dash, "\mathsf{subj}"'] \\ \texttt{MAN THE} \end{tikzcd}$$

4.1.2 A More Complex Example

Step 1 From the initial hypothesis h_0, suppose we choose to cover the phrase `[WOMAN TALL]` first:

$$h_1 : \boxed{\begin{array}{ll} \{3,4\} & \\ 1 & \{((\mathsf{det},3,2),1)\} \\ \multicolumn{2}{l}{\textit{tall lady}} \end{array}}$$

$$P(h_1) = P(h_0) \times P_\phi(\textit{tall lady}|\texttt{WOMAN TALL})^{\lambda_\phi} \times P_\ell(\textit{tall lady}|h_0)^{\lambda_\ell}$$

One dependency is "included" in the translated phrase (the one labelled amod), and one further dependency is now "half-translated": the det dependency from word 3 to word 2. The triple $(\mathsf{det},3,2)$ serves to uniquely identify this dependency.[3] The "translated half" of this dependency is "one word back" from the end of the English translation so far — *on the assumption that* the English word aligned with the "real" (foreign) head `WOMAN` is *`lady`* — hence $((\mathsf{det},3,2),1)$.

Step 2 Recall from Section 3.2.3 that the selected phrases can not be arranged in any order at all; each subtree must be linearly contiguous. This imposes a non-trivial restriction on the range of phrases we can choose from when considering ways to extend a given decoding hypothesis. It is not sufficient to choose any phrase which does not overlap with the current covering. In this example, having first translated `[WOMAN TALL]`, we are now forced to translate `[THE]` next, although this is certainly not the only phrase which does not overlap with `[WOMAN TALL]`.

A (somewhat) intuitive way to think about this is: with respect to the *original, uncompressed dependency tree*, we require that the next phrase translated contains only words from the "minimal incomplete

[3]The dependent index would actually suffice to uniquely identify each dependency, but it is handy to have the label and governor index too.

constituent": the smallest constituent which (a) includes all words from the most recently translated phrase, and (b) is not yet completely covered.

Given the *tree of all possible compressions* in Figure 4, the algorithm for finding the phrases we may cover next is as follows: beginning at the node(s) labelled with the most recently translated phrase, walk up the tree until you reach a node N which dominates a phrase whose words are not completely covered. Any phrase node dominated by N (which does not overlap with the covering so far) is a valid choice.

Returning to the example, if the phrase [WOMAN TALL] is chosen first, then the minimal incomplete constituent in the uncompressed dependency tree is that containing words {3,4,5}. From the node labelled [WOMAN TALL] in Figure 4, we do not need to walk up the tree at all; this node already dominates a node (namely [THE]) whose words are not completely covered, so any node dominated by [WOMAN TALL] is valid. The only choice is [THE] itself, so we translate this next.

$$h_2 : \boxed{\begin{array}{ll} \{3,4,5\} & \\ 2 & \{((\mathsf{obj},6,3),1)\} \\ \textit{\texttt{tall lady the}} & \end{array}}$$

$$P(h_2) = P(h_1) \times P_\phi(\textit{\texttt{the}}|\text{THE})^{\lambda_\phi} \times P_\ell(\textit{\texttt{the}}|h_1)^{\lambda_\ell} \times P_d(+1|\mathsf{det})^{\lambda_d}$$

The offset of +1 for the completed det dependency is simply read off the $((\mathsf{det},3,2),1)$ entry in h_1. We can think of the offsets recorded in these half-translated dependency entries as "what we would use as an offset if the other half was translated next". Completing this dependency means we have now "covered" a total of two dependencies.

In making this transition we have also completed the subtree that is the dependent of the $(\mathsf{obj},6,3)$ dependency. This dependency therefore becomes "half-translated", and since its right edge is "one word back" from the end of the English translation so far, we store $((\mathsf{obj},6,3),1)$. We only keep the absolute value of these offsets in the hypotheses, since the sign to be used when the dependency is completed is determined by whether it was completed by translating the dependent (in which case the sign should be positive) or the governor (negative).

Step 3 To determine the valid choices for the next phrase to cover, we have two nodes in the tree in Figure 4 to consider, because there are two nodes labelled with the phrase [THE].[4] Beginning from the right such node, we walk up to [WOMAN TALL] and then to [SAW] before finding that the current node dominates phrases whose words are not completely covered. Beginning from the left such node, we walk up to [WOMAN] first and then likewise continue on to [SAW]: we do not stop at [WOMAN] because while this node does dominate phrases that have not been *used* in the translation so far (namely [TALL] and [WOMAN]), it does not dominate any phrases *whose words* are not completely covered. So we are free to choose any phrase under the [SAW] node to translate next (subject, of course, to the overall requirement that the next phrase to translate does not overlap with the foreign words already covered). Specifically, we have four choices: [SAW], [MAN], [THE] or [MAN THE].[5]

[4]In the implementation, these are not two separate nodes but rather one node with two parents; the structure is therefore not actually a tree. (To do otherwise would be extremely inefficient once we deal with longer sentences: firstly in terms of space, because many identical subtrees would be created, and secondly in terms of time, because being able to hook up to previously-generated nodes like this effectively allows some memoisation when constructing the tree in Figure 4.) What is described in the text as two walks up the tree originating from different nodes is actually an "upwards depth-first search": turn Figure 4 upside down, collapse the two [THE] nodes, and do a normal depth-first search from there.

[5]Had the two searches ended at distinct nodes, we would have been free to choose any phrase under either of these two nodes: suppose, for example, that [THE] is the first phrase translated, in which case the two searches would end at [WOMAN] and [WOMAN TALL], and we would have been free to choose either of these two phrases or [TALL] to cover next.

We might choose to translate the phrase [THE] (word 2) next, in which case our hypothesis becomes:

$$h_3 : \boxed{\begin{array}{ll} \{2,3,4,5\} & \\ 2 & \{((\mathsf{obj},6,3),2),((\mathsf{det},2,1),1)\} \\ \multicolumn{2}{l}{\texttt{tall lady the the}} \end{array}}$$

$$P(h_3) = P(h_2) \times P_\phi(\texttt{the}|\texttt{THE})^{\lambda_\phi} \times P_\ell(\texttt{the}|h_2)^{\lambda_\ell}$$

The phrase [THE] does not "contain" any dependencies, nor do we complete any half-translated dependencies by covering it, so the number of dependencies covered remains 2. The half-translated obj dependency which was previously "one step back" is now "two steps back", since we have added a word in between the obj dependent and its as-yet-untranslated governor. We also have a new half-translated dependency: we have translated the dependency of the $(\mathsf{det}, 2, 1)$ dependency, which is one step back from the end of the English translater so far.

Step 4 Our search up the tree from [THE] (word 2) stops at [MAN]; this is the only possible candidate for translation at this point. We extend our hypothesis:

$$h_4 : \boxed{\begin{array}{ll} \{1,2,3,4,5\} & \\ 3 & \{((\mathsf{obj},6,3),3),((\mathsf{subj},6,1),1)\} \\ \multicolumn{2}{l}{\texttt{tall lady the the guy}} \end{array}}$$

$$P(h_4) = P(h_3) \times P_\phi(\texttt{guy}|\texttt{MAN})^{\lambda_\phi} \times P_\ell(\texttt{guy}|h_3)^{\lambda_\ell} \times P_d(-1|\mathsf{det})^{\lambda_d}$$

This transition completes the det dependency which was half-translated, so we have one more dependency "covered" (and three in total). Note again that the -1 offset for the completed det dependency is just read off the $((\mathsf{det}, 2, 1), 1)$ entry in h_3; we use -1 rather than $+1$ because it is the governor which we are translating at this step rather than the dependent. We have also added another word to the English translation which lengthens the offset between the already-translated obj dependent and its still-to-come governor, so the 2 that was previously associated with this dependency is incremented to 3. Finally, we have completed the subtree that is the dependent of the $(\mathsf{subj}, 6, 1)$ dependency, so this is added to the set of half-translated dependencies.

Step 5 Finally, we reach a goal state by translating [SAW]:

$$h_5 : \boxed{\begin{array}{ll} \{1,2,3,4,5,6\} & \\ 5 & \{\} \\ \multicolumn{2}{l}{\texttt{tall lady the the guy saw}} \end{array}}$$

$$P(h_5) = P(h_4) \times P_\phi(\texttt{saw}|\texttt{SAW})^{\lambda_\phi} \times P_\ell(\texttt{saw}|h_4)^{\lambda_\ell} \times P_d(-3|\mathsf{obj})^{\lambda_d} \times P_d(-1|\mathsf{subj})^{\lambda_d}$$

This completes both the subj and obj dependencies. Again, the offsets we use to determine distortion scores are simply read from the half-translated dependencies in h_4.

Note on half-translated dependencies and hypothesis recombination It may not be obvious why we bother tracking the set of half-translated dependencies in the form shown above, rather than just looking back at structure of the English translation so far when we need to determine "how far back" the other end of a newly-completed dependency occurs. The reason is that it allows hypothesis recombination. The set of half-translated dependencies as presented above contains all and only the information required to determine all future distortion model scores. If two hypotheses have the same half-translated dependency set (as well as the same set of words covered), then they can be combined.

4.1.3 Easily extended to multiple parse trees

The model can be trivially (though not necessarily efficiently) adapted to consider not just a single one-best dependency parse tree, but an n-best list of parse trees with associated probabilities. We simply begin with n distinct initial hypotheses, each with a starting score equal to the probability of the corresponding tree, and extend each hypothesis on the basis of (only) that parse tree. This essentially amounts to decoding with each of the n parse trees "in parallel", though hypotheses based on distinct trees compete with each other when pruning occurs.

4.2 Future cost estimation

For the purposes of future cost estimation we assume that any dependency which has not yet been linearised (whether it is "half-translated" or completely untouched as yet) will be linearised with the best possible offset which is still possible.

In the case of completely untouched dependencies (dependencies of which neither the governor nor the dependent have yet been linearised), all offsets are possible so we take the "unrestricted maximum" score. A half-translated dependency, however, will have a restricted range of offsets that are still possible for it. For example, consider hypothesis h_4 in Section 4.1.2. The eventual offset for the linearisation of the obj dependency from this point can not be any greater than -3, and similarly the eventual offset for the linearisation of the subj dependency can not be any greater than -1. These are the offsets we would use if the other half of these dependencies were translated straight away (as in fact it was in h_5), and the only way they can change is if we add more words to the English translation before finally translating these other halves. The future cost estimation for the distortion model for h_4 is therefore

$$\left(\max_{n \leq -3} P_d(n|\mathsf{obj})\right)^{\lambda_d} \left(\max_{n \leq -1} P_d(n|\mathsf{subj})\right)^{\lambda_d}$$

4.3 Pruning

Standard phrase-based systems avoid a bias against "more complete" hypothesis by arranging hypotheses in stacks, where each stack contains all the hypotheses which have covered a certain number of foreign words. In this new system however, two hypotheses with the same number of foreign words covered might not be comparable, because they have completed a different number of dependencies (and thus will include a different number of reordering probabilities). A stack must therefore be characterised by a number of foreign words covered and a number of dependencies covered. This is the reason for storing the number of completed dependencies in each hypothesis.

Also, in the same way that standard phrase-based systems make hypotheses which have covered the same number of words compete, irrespective of how many phrases they have used to do it, we make hypotheses which have covered the same number of dependencies compete, no matter how many phrases they have used to do it. (This provides a slight bias towards longer phrases.) This is the reason for adding any dependencies which are "contained in" the phrase being translated (eg. the det dependency when we translate [MAN THE]), and not just the dependencies *between* translated phrases.

5 Experiments and Results

5.1 Corpora used for the experiments

Czech We conducted Czech-English translation experiments, utilising the Prague Czech-English Dependency Treebank (PCEDT).[6] We trained on the aligned sentence-pairs in the training section of the PCEDT's PTB component plus the aligned pairs in the Reader's Digest component (65,110 sentences). We used the

[6]For more information see Čmejrek et al. (2004) and http://ufal.mff.cuni.cz/pcedt/. We used the provided parse trees from the Collins parser.

development and test sections (256 sentences each) of the PTB component. Development corpora were used to tune model weights via minimum error rate training (Och, 2003).

Arabic For Arabic, we trained on the eTIRR corpus (LDC2004E72), Arabic news (LDC2004T17) Ummah (LDC2004T18). Our development set was a randomly selected 250-line sample from the NIST MT06 dataset, and we tested on NIST MT05. Arabic and English were parsed using the Stanford parser (v. 1.6).

	Training	Dev	Test
Czech	65,110	256	256
Arabic	137,144	250	1056

Table 2: Size (in sentences) of the various corpora

5.2 Results of the experiments

Results for the experiments in Czech and Arabic are shown in Table 3. In short, the complete system described in this paper is the one on line (vi) of this table, which unfortunately has *decreased* BLEU scores compared to the baseline of a standard phrase-based decoder on line (i).

	Available phrases	**Distortion models**			**BLEU**		**Notes**
		Pharaoh	Dep.	Lex. Dep.	Czech	Arabic	
(i)	Pharaoh phrases	✓			33.28	45.28	Standard Pharaoh
(ii)	Random phrases	✓			20.53	29.75	Randomly crippled Pharaoh
(iii)	Licensed phrases	✓			32.54	28.02	Systematically crippled Pharaoh
(iv)			✓		24.13	24.50	Attempts at improvement
(v)		✓	✓		31.91	31.82	
(vi)		✓		✓	31.82	34.04	

Table 3: Results on Czech and Arabic for a number of variants of the system described in this paper

Table 3 summarises results. The complete system described in this paper (line vi) underperforms the baseline (line i). Lines (ii) through (v) summarise additional experimentation seeking insight into the system's (lack of) performance. We varied (a) the spans of the input sentence which we allow ourselves to cover by a single phrase ("Available spans"), and (b) the model(s) used to assign a score to a particular ordering of phrases ("Distortion models").

With respect to the possible spans, "Pharaoh spans" refers to the entire set of spans that would be considered by a standard phrase-based decoder, namely, all chunks of the input sentence up to a certain size (specifically, seven words). "Licensed spans" refers to the subset of spans licensed by the input sentence's dependency parse, as described in section 3.2. Holding the distortion model constant, comparing lines (i) and (iii) shows that restricting the model to only licensed spans resulted in only a small cost, despite a 54% average reduction in the size of the set of available spans. This suggests that dependency-based licensing for phrases may be on the right track. Line (ii) reinforces this observation: sampling a *random subset* of the spans used in line (i), equal in size to the set from line (iii), drops performance hugely compared to the principled dependency-based licensing.[7] Some further analysis reveals a striking reason for this: although only 46% of the spans available in line (i) are available in line (iii), over 99% of the *phrases* available in line (i) are still available in line (iii). In other words, the spans of the input sentence which violate our dependency restriction were almost never candidates for translation as a phrase in line (i) anyway, since no entry in the phrase table matched them. This indicates that the phrase-extraction process was already conforming to the intuitions behind our phrase-licensing scheme, perhaps as a result of the morphological richness of Czech. In other languages where the dependency structure is not as clearly encoded in the surface string, the phrase-extraction process may leave more of this work to be done at decoding time.

Of the three distortion models: "Pharaoh" refers to the standard monotonic-biased distortion model inherited from speech recognition, "Dep." refers to the dependency-based model described in section 3.2 *without* lexicalisation (i.e. offsets are conditioned solely on dependency label), and "Lex. Dep." refers to the dependency-based model *with* lexicalisation (i.e. offsets are conditioned on dependency label and the dependent word).[8] Lines (iv), (v), and (vi) summarise results using these distortion model variants. The overall pattern is that the conventional distortion model outperforms the dependency-based variant.

We conducted additional experimentation using Arabic-English as the language pair, but unfortunately the results also failed to provide evidence supporting the proposed distortion modeling approach. Indeed, all attempts to introduce dependency-based distortion led to large performance drops over the conventional phrase-based baseline, which we connect to the observation that phrase-based models for this language pair do perfectly well in comparison to hierarchical phrase-based models,[9] and the fact that various properties of Arabic (e.g. long run-on sentences, ubiquitous use of NP NP-modification idafa constructions) may make Arabic more difficult to parse correctly than Czech.[10]

6 Conclusion

In this paper, we described an attempt to employ a richer, linguistically informed reordering model while otherwise remaining strictly within the confines of phrase-based translation. The hope was to introduce an approach that could take advantage of monolingual syntactic dependencies, while still retaining a conventional statistical MT framework involving contiguous word strings as translation units, with the attendant advantages of conceptual simplicity, ease of implementation, availability of existing tools for training and tuning, and the ability to integrate seamlessly with other log-linear modeling features.

From a technical standpoint, we found it interesting to explore the idea that portions of a source syntactic dependency trees can be "compressed" into the translation units licensed by standard flat-phrase extraction. Despite the negative results of the distortion model we have introduced, the fact that restricting the search space to only translations using dependency-licensed phrases had such a small effect suggests that there is something fundamentally sound about the intuition that linearisation of flat phrases can be thought of in terms of cross-phrase head-modifier relationships.

Appendices

A Implementation details

In this section we describe how the system described above has been integrated into the Pyro decoder framework.[11] In this very modular framework, significant modifications to the standard phrase-based system often amount to simply providing a new implementation of the abstract class `Model`. Unfortunately this was not quite possible in this case: besides adding a new implementation of `Model`, a few small but fundamental changes to the interface between the decoding algorithm itself and the models were required.

[7]Lines (ii) and (iii) are at one additional disadvantage compared with line (i): they use phrases of maximum length 4 whereas the baseline uses 7. Line (iii) uses the lower limit since permitting phrases of length greater than 4 produces an unnecessary explosion in the number of possible compressions of the dependency tree; line (ii) uses the same limit to permit a clear comparison. Less than 0.01% of the phrases usable on our test set by the baseline system were of length longer than 4.

[8]Note that the dependency-based models are only compatible with the "Licensed phrases" option, since they cannot assign scores to orderings of arbitrary phrases.

[9]Chris Dyer, personal communication.

[10]Chris Manning, personal communication.

[11]Pyro is a clone of Pharaoh written in Python by David Chiang and maintained by Adam Lopez. The design of the version which served as the starting point for this implementation bears a strong resemblance to the design of Hiero version 2.

We describe these fundamental changes to the framework first, before turning to the implementation of the new `DependencyDistortionModel` within this modified framework.

The decoder implementation described here, training scripts and sample data are available at `http://www.ling.umd.edu/~timh/decoder/decoder-distrib.tgz`.

A.1 Modifications to the decoder/model interface

These changes are also described by rather detailed comments in the relevant pieces of code. They add some very generic flexibility to the existing framework which could well prove useful for other unrelated extensions of Pyro (and possibly even Hiero). This section may not be useful (or even meaningful) if the reader is not familiar with the existing Pyro/Hiero framework; in this case Section A.2 itself should hopefully be understandable anyway.

A.1.1 Finer-grained binning criteria

In Pyro, any two hypotheses with an equal number of foreign words covered are "pruning competitors" (they go in the same "stack"/`Bin`). Recall from Section 4.3, however, that we require a more fine-grained classification of hypotheses into stacks, because hypotheses should not be compared for pruning purposes unless they also have the same number of dependencies covered.

The original Pyro `Chart` object kept a one-dimensional array of `Bin` objects called `bins`, the ith element of which was a `Bin` containing all hypotheses with exactly i words covered. This is insufficient for our purposes. The minimal modification to get things working would be to change `bins` to a two-dimensional array, the (i, j) element of which was a `Bin` containing all hypotheses with exactly i words covered and exactly j dependencies covered. But to avoid such strong coupling between the `Chart` class and our particular `Model` implementation, a more general solution was implemented.

We added a function to the `Model` interface called `additional_binning_criteria`. (Though I now wish I had called it `additional_equivalence_criteria`.) Any model which requires any finer-grained binning of hypotheses than standard Pyro should provide an implementation of this method. When determining which `Bin` to place a hypothesis in, the decoder passes the hypothesis to each of the models' `additional_binning_criteria` functions.[12] The tuple containing the results returned by each `Model` (known as the "pruning class") acts as an additional "key" in determining which `Bin` the hypothesis should go in. In other words, two hypotheses will be pruning competitors if and only if (i) they have the same number of words covered, and (ii) every model's `additional_binning_criteria` returns the same result for both hypotheses. By having our new model return the number of dependencies covered by the hypothesis in question we achieve the desired "two-dimensional" arrangement of `Bin` objects.

The default implementation of `additional_binning_criteria` simply returns `None`, so any `Model` for which the original Pyro equivalence criteria are sufficient can ignore this function altogether.

Relevant places in the code For detailed comments on and/or implementation of the above ideas; in particular, compare with the corresponding places in Pyro, if you have access to that.

- the initialisation of `self.bins` in the `__init__` function of the `Chart` class in `decoder.py`
- the `get_pruning_class`, `get_bin` and `add` functions of the `Chart` class in `decoder.py`
- the `for`-loop in the `translate` function of the `Decoder` class in `decoder.py`
- the (default) `additional_binning_criteria` function of the abstract `Model` class in `model.py`
- the `additional_binning_criteria` function of the `DependencyDistortionModel` class in `dependency_model.py`

[12] Actually, each model only receives its own "part" of the hypothesis to examine, i.e. its `state`. If this sounds rather cryptic to the reader now, it is a trivial point which will become clear very quickly after a short look at any Pyro/Hiero code.

A.1.2 Multiple initial states

In Pyro there is exactly one initial hypothesis that the decoding process begins with. However, recall from Section 4.1.3 that our model permits a number of distinct initial states, one for each of the parse trees for the input sentence. Pyro constructs its initial hypothesis by taking the `initial` member variable from each of the models present and essentially combining them into a tuple.

To overcome this restriction of allowing each model only one initial state, we added a function `get_initial_states` to the `Model` interface. A model should return a list of initial states from this function. The set of integrated initial states is then the Cartesian product of the sets of initial states returned by each model. Once all these initial states are added to the `Chart` at the beginning of the decoding process, nothing else special needs to be done.

Relevant places in the code For detailed comments on and/or implementation of the above ideas; in particular, compare with the corresponding places in Pyro, if you have access to that.

- the `add_axioms` function of the `Chart` class in `decoder.py`
- the (default) `get_initial_states` function of the abstract `Model` class in `model.py`
- the `get_initial_states` function of the `DependencyDistortionModel` class in `dependency_model.py`

A.2 The `DependencyDistortionModel` class

A.2.1 The `input` function

This is one of the hooks provided by the existing Pyro framework. It is called for each model when translation of a source sentence begins, allowing the model to carry out any relevant initialisation steps.

In our case, this is where we record the parse tree(s) associated with the input sentence. The Pyro framework allows XML markup to appear in the input, and passes any such annotations to this function in the `meta` argument. This argument is a list of four-tuples of the form `(element, attributes, start, end)`; each such four-tuple represents one element of the XML markup, where `element` is the name of this element, `attributes` is a dictionary (mapping strings to strings) containing the attributes of this element, and `start` and `end` are indices representing the span of the input sentence which this element covers. To illustrate the XML markup used to annotate input sentence with dependency trees, the dependency tree in Figure 2 would be represented as follows:

```
<sent numparses="1">
<word id="1" govid="6" dep="subj">MAN</word>
<word id="2" govid="1" dep="det">THE</word>
<word id="3" govid="6" dep="obj">WOMAN</word>
<word id="4" govid="3" dep="amod">TALL</word>
<word id="5" govid="3" dep="det">THE</word>
<word id="6" govid="0" dep="*">SAW</word>
</sent>
```

Actual input files must be formatted with exactly one sentence per line. For each word, we simply record the ID of its governor (there is at most one) and the label of the dependency linking it to that governor. The order in which the words appear is, in fact, irrelevant, because all we care about is the unordered dependency tree; the IDs used to identify governors need not bear any relation to linear order. The root word is linked to the imaginary node with index 0 by a dependency labelled $*$ (as in Figure 4).

This markup is used to create instances of the `DepTree` class (located in the `deptree` file/module) representing the appropriate dependency structure. We also call the `squash_tree` function (located in the `deptree_squashed` file/module) on each such structure, which constructs an instance of the `TreeNode` class (also located in `deptree_squashed`) representing the range of possible compressions of the input tree, analogous to the tree shown in Figure 4. The `self.trees` variable maintains both (i) the uncompressed dependency tree, and (ii) the tree representing the range of possible compressions, in the form of a dictionary mapping the latter to the former.

To represent multiple dependency parses of a single sentence, it suffices to set the `numparses` attribute to the appropriate number greater than one, and include the separate annotations for each parse tree in the `govid` and `deplabel` attributes separated by colons. For example, to represent both the tree in Figure 2 and that where the `subj` and `obj` dependencies have been reversed, we would provide input as follows:

```
<sent numparses="2">
<word id="1" govid="6:6" dep="subj:obj">MAN</word>
<word id="2" govid="1:1" dep="det:det">THE</word>
<word id="3" govid="6:6" dep="obj:subj">WOMAN</word>
<word id="4" govid="3:3" dep="amod:amod">TALL</word>
<word id="5" govid="3:3" dep="det:det">THE</word>
<word id="6" govid="0:0" dep="*:*">SAW</word>
</sent>
```

None of the corpora I have actually used have more than one dependency parse per sentence, but if more than one tree is provided they are all read in and added to the `self.trees` dictionary. The effect of having more than one parse tree at the moment is just to create multiple initial hypotheses and decode "in parallel", as described in section 4.1.3, but if a more intelligent way of using multiple parse trees is devised in the future, no changes to this function should be necessary.

A.2.2 The `get_initial_states` function

This function simply provides one initial hypothesis for each parse tree provided for the current sentence. See sections A.1.2, A.2.1 and 4.1.3.

A.2.3 The `additional_binning_criteria` function

This function returns the total number of dependencies covered by a hypothesis, since the stack that a hypothesis goes into for pruning purposes depends on this as well as the standard criterion of number of words covered. See section A.1.1.

A.2.4 The `transition` function

This function is another one of the hooks provided by the original Pyro framework. It contains the real substance of the model: given a particular extension of a hypothesis (a "transition"), this function needs to determine its score/cost. This is complex.

Aside from some simple checks and initialisation, you will notice that the function consists essentially of one large `for` loop and one large `while` loop. The `for` loop deals with any dependencies of which the hypothesis extension covers the governor, and the `while` loop deals with any dependencies of which the hypothesis extension covers the dependent. I explain each in turn.

The `for` loop iterates through each of the foreign *words* that are dependents of the *head word* of the phrase that is being used to extend our hypothesis. Each such word will fall into exactly one of these three cases:

- If the dependent word has been covered by an earlier phrase, then the extension currently being evaluated will complete this dependency. We extract the relevant information to condition on (the dependency label, and the target language words used to translate the dependent word, for the lexicalised version of the distortion model) from the dictionary of half-translated dependencies. (This dictionary, `half_covered_deps`, is available as part of the `state` that the model maintains.) The offset is also retrieved, but may need to be adjusted to properly take into account the position of the target-language word aligned with the source-language head word, as described in section 3.2.4; this is the purpose of the calculation of `head_idx_e`. Finally, we call `get_distortion_score` to determine to actual cost to be accrued for this dependency. If there are multiple possible `(parent_word,child_word)` pairs to be considered as conditions for the lexicalised version of the model (due to many-to-one word alignments), we take the best option; see the list comprehension with the `min` function applied to it.

- If the dependent word has not been covered by an earlier phrase, and is *not* contained in the phrase we are currently translating, then this dependency needs to be recorded as a half-translated dependency: as a result of this transition we will have covered its governor but not its dependent. An entry is added to `half_covered_deps`, recording the target-language words that are being used to translate the governor, and any initial offset that needs to be counted due to the (estimated) head of the target-language phrase not being at the right edge (again, see `head_idx_e` and section 3.2.4).
- If neither of the above are true, then this dependent word is covered elsewhere in the phrase currently being added. We do not need to do anything in this case. We do not actually store the total number of covered dependencies with each hypothesis (although the notation for representing hypotheses in section 4 is probably misleading in this respect); it is computed dynamically when needed in `additional_binning_criteria`.

The `while` loop walks up the unsquashed dependency tree, beginning at the head word of the phrase being added, to the root. We do this in order to consider each subtree that contains the head word of the phrase being added, and the nodes we go through on this walk are precisely the nodes of these subtrees. Each such subtree may be relevant in one of two ways:

- If the subtree was *completely uncovered before* this transition, then the phrase currently being translated is forming the *left* edge of this subtree; therefore if the governor that has (the root of) this subtree as a dependent has already been translated, then the current extension is completing a half-translated dependency. In this case we extract the information to condition on from the dictionary `half_covered_deps` and call `get_distortion_score`, again taking the best possible option if we have more than one lexicalised condition to choose from.
- If the subtree becomes *completely covered as a result of* this transition, then the phrase currently being translated is forming the *right* edge of this subtree; therefore if the governor that has (the root of) this subtree as a dependent has not yet been translated, then this dependency becomes half-translated. This was the case in Step 2 of the example in section 4 when we added the $(\texttt{obj}, 6, 3)$ dependency to the set of half-translated dependencies, and in Step 4 when we added the $(\texttt{subj}, 6, 1)$ dependency. We add an entry with the appropriate conditioning information to the dictionary.

Having considered all “downward dependencies” in the `for` loop and all “upward dependencies” in the `while` loop, we have accumulated the total cost/score of the transition in the `score` variable. We also have any new half-translated dependencies in the `new_half_covered_deps` variable, but these need to be combined with any pre-existing half-translated dependencies (that were not completed by this transition), updated with their offsets incremented by the number of target-language words in the added phrase.

A.2.5 The `estimate_future` function

This is also a hook function in the Pyro framework. It computes the model’s estimated future cost for a particular hypothesis, as described in section 4.2. It first loops through all dependencies of the tree, accumulating the future cost estimations of any “completely uncovered” dependencies; and then loops through each of the half-translated dependencies encoded in the current hypothesis, accumulating a future cost estimate for each one.

A.2.6 The generate_spans function

The Pyro framework requires that *exactly one* of the models used implements this hook function. Given a particular hypothesis, it must compute the set of spans of the source sentence which can be covered by the next transition (represented by a list of pairs of indices). In the case of vanilla Pyro for example, this is where the distortion limit comes into play. In this more complex system it implements the requirement described in section 3.2.3, using "walking up the tree" algorithm discussed in section 4.1.2.

A.2.7 The read function

This function, intended to be called in the config file immediately after the `DependencyDistortionModel` object is created, reads the (possibly lexicalised; see section 3.2.5) dependency linearisation model from a file. From each line we read a (condition, offset, count) triple, indicating how many times dependencies of the kind represented by the condition (perhaps just a dependency label, perhaps a dependency label combined with lexical items if we are using a lexicalised version) occurred with the given offset. The format is as shown in appendix C. For example, the line

```
    esubj|saw|man   -2      10
```

indicates that in the training corpus there were 10 instances of dependencies labelled `esubj` with *`saw`* as their governor and *`man`* as their dependent, being linearised with an offset of -2.

If any there are any "holes" in the range of offsets observed for a particular kind of dependency, we linearly interpolate to estimate the count the should be associated with these offsets. For example, if we have 10 observations of a `esubj|saw|man` dependency occurring at offset -2 (as above), and 4 observations of these dependencies occurring at offset -4, but no observations of these dependencies occurring at offset -3 (i.e. no such line appears in the file), we pretend that there were in fact $\frac{10+4}{2} = 7$ observations of such dependencies occurring at offset -3. This relies on the assumption that these "holes" in the range of observed offsets are anomalies, and that the "true" distribution of offsets for a dependency is unimodal.

A.2.8 The read_normalisation_model function

This is also intended to be called in the config file immediately after the `DependencyDistortionModel` object is created. It reads the dependency-label normalisation model (described in section 3.2.5) from a file. Each line contains a source dependency label, a target dependency label and a count, representing the number of times those two labels "coincided". The format of this file is as shown in appendix C. A simple MLE model for normalising source-language dependency labels to target-language dependency labels is computed.

A.2.9 The get_normalisation_distribution function

This function, given a source-language dependency label, simply returns a probability distribution (implemented as a dictionary) over target-language dependency labels computed by the `read_normalisation_model` function.

A.2.10 The best_possible_score function

This function, given a condition (i.e. dependency label, possibly along with lexical items) and optional upper and lower bounds, returns the best score the model could assign to the dependency under the assumption that the offset must be within the bounds given. It calls `get_distortion_score` to do this. It is used by the `estimate_future` function.

A.2.11 The get_distortion_score function

This function computes the score for a dependency of the given source-language label (and possibly lexical items) being linearised with the given offset. For each possible normalisation of the source-language dependency label into a target-language dependency label, it performs a lookup into the dependency-linearisation model constructed by the `read` function, and assumes the most favourable target-language dependency label possible. In other words, given a source dependency label *fdep* and an offset δ, this function returns

$$\max_i P(edep_i|fdep)P(\delta|edep_i)$$

where the first probability is obtained via `get_normalisation_distribution` and the second is obtained via a lookup into the model constructed by the `read` function.

B Procedures for running experiments

Dependencies among different stages of the experimenting process are managed using a Makefile. Everything is based in a root directory which I'll call `$ROOTDIR`. As I have set things up, `$ROOTDIR` is `/fs/clip-mt_3par/timh/decoder`.

B.1 Overview

The idea of using a Makefile is that you should be able to issue commands indicating what you want produced, and since the Makefile understands what depends on what, it will build everything that's required. For example, the Makefile specifies that running MERT requires that a phrase table exist, and that doing the final decoding run requires that a file containing tuned weights exist. Furthermore, it provides the commands for building these prerequisites, in case they turn out to not exist yet. So if you issue a command asking for the final decoder output, and no tuned weights exist, the Makefile will issue the command used to create this weights file (namely, the command to run MERT) first.

The operation of the Makefile is parametrised by the values of certain variables. The most salient are the variable `$FR`, which indicates the source language being translated, and the variable `$SYS`, which indicates the decoder version to use. There are also variables indicating the particular corpora to be used, described in more detail below. The values of these variables, in combination with the thing you want created (eg. tuned weights, a phrase table, final output), determine the entire path and filename of what `make` should build. For example, if you are using a particular decoder version called `r01` on Chinese, with a training corpus called `small`, a dev set called `mt02` and a test set called `mt03`, you might issue the following command:

```
make SYS=r01 FR=zh TRAINING_CORPUS=small DEV_CORPUS=mt02 TEST_CORPUS=mt03 test
```

This is understood by `make` as a request to construct the file `$ROOTDIR/run-zh-r01/mt03.mert-mt02.txt`, which we can read as "the output of decoding `mt03`, using weights tuned on `mt02` and decoder `r01`". From pattern-based rules in the Makefile, `make` realises that this file depends on (among other things) these two other files:

```
$ROOTDIR/training-zh-small/phrasetable-mt03.gz
$ROOTDIR/run-zh-r01/weights-mt02-all.txt
```

which we can read as "the phrase table trained on the Chinese `small` training corpus, filtered for the `mt03` test set" and "the weights file resulting from running MERT with decoder version `r01` on Chinese `mt02`".[13] The prerequisites required to construct these files, if they do not already exist, are similarly specified in this parametrised way. By tracking parametrised dependencies like this, `make` can run the entire process from end to end if necessary, or as much as is required if some of the pieces have already been constructed.

With respect to any particular combination of parameters (source language, decoder version, corpora, etc.), the files the Makefile deals with will be separated into four directories:

`$ROOTDIR/corpora-$FR`
: This directory contains the raw corpora for the source language `$FR`, including both sides of training corpora and all dev and test corpora.

[13]The `-all` suffix added to the dev set indicates that all of the dev set was used in tuning. If a subsample had been used, the size of this sample would have been appended instead. This flexibility is probably obsolete now.

`$ROOTDIR/wc-$SYS`
: This directory contains the version of the decoder designated by `$SYS`.

`$ROOTDIR/training-${FR}-${TRAINING_CORPUS}`
: This directory contains everything related to the phrase model resulting from the training corpus `$TRAINING_CORPUS`.

`$ROOTDIR/run-${FR}-${SYS}`
: This directory contains everything depending on the combination of the particular decoder version and the particular language. For example, it contains the weights files resulting from MERT tuning (further distinguished within the directory on the basis of the dev set used), appropriate configuration files for the decoder, and test set output and evaluation output.

B.2 Corpora

Suppose you want to run an experiment on a new language, whose abbreviation (eg. `zh`, `ar`) we'll call `$FR`. Similarly the target language (probably English) is `$EN`. Choose a name for your training, dev and test corpora; for example `small` or `large` for the training set, and `mt02`/`mt03`/etc. might make sense as dev and test names. The name must be unique among corpora for this particular language, but you can have an `mt02` for Chinese and an `mt02` for Arabic, for example. Let these three chosen names be `$TRAINING_CORPUS`, `$DEV_CORPUS` and `$TEST_CORPUS`. Then `$ROOTDIR/Makefile` can do everything necessary once you provide the following files in the appropriate location:

`$ROOTDIR/corpora-$FR/${TRAINING_CORPUS}.{$FR,$EN}`
: The source and target sides of the training bitext, in Cyro format (see Section A.2.1).

`$ROOTDIR/corpora-$FR/${DEV_CORPUS}_$FR.cyro`
: The parsed source side of the development corpus, in Cyro format (see Section A.2.1).

`$ROOTDIR/corpora-$FR/${DEV_CORPUS}_$EN.txt.*`
: The reference translations for the development corpus, in plain text format.

`$ROOTDIR/corpora-$FR/${TEST_CORPUS}_$FR.cyro`
: The parsed source side of the test corpus, in Cyro format (see Section A.2.1).

`$ROOTDIR/corpora-$FR/${TEST_CORPUS}_$EN.txt.*`
: The reference translations for the test corpus, in plain text format.

You may want to include additional language-specific sections of the Makefile for putting these files in place. This is what I have done for the Czech and Arabic corpora I have used so far; there are conditional sections of the Makefile which basically generated the files mentioned above in `corpora-$FR` from corpora "as I found them" in which are in `data-$FR`. The details of how this happens will of course depend on whether your corpora are already parsed, how you want to parse them if not, any other preprocessing which needs to take place, etc. In any case, until you put the files listed above in place, "the ball is in your court"; once you've done that, the rest of this section describes how to run experiments.

B.3 Decoder system

Besides the corpora you also need to provide a specific decoder to use for translation (and accompanying settings). Suppose the particular version you are testing is called `$SYS`. Then your decoding software should be placed in `$ROOTDIR/wc-$SYS`. (The `wc-` is for "working copy"; I used Subversion revision numbers for `$SYS`, so my decoding software lived in directories with names like `$ROOTDIR/wc-r01`.) In this directory must exist an executable file `phrase_table.py` which filters a phrase table (as in original Pyro), and an executable file `cyro.py` which is the decoder itself.

To run a particular decoder `$SYS` on a particular language `$FR`, you will need a directory `$ROOTDIR/run-${FR}-${SYS}` (eg. `run-cz-r01`). In this directory you need to provide a file `config-template.py` which is a template file which is *almost identical* to standard Pyro/Hiero/Cyro ini files. The idea is that the `config-template.py` in `run-cz-r01` will differ from that in, say, `run-zh-r01` because it will refer to a different language model, and it will differ from that in, say, `run-cz-r02` because the `r02` decoder ini file might have some specific options which don't make sense for `r01`. But given a language (`$FR`) and a decoder version (`$SYS`), we assume that all configuration settings are uniquely determined.[14]

The only exception to this (hence the "almost identical" above) is that the phrase table may differ depending on the set being decoded. This variation is systematic though, and the Makefile knows where the filtered phrase tables are since it built them, so you can effectively forget about this variation. Just provide one `config-template.py` file which contains the string `@@@PHRASE_TABLE@@@` in any position where you would like the appropriate phrase table filename to be inserted. A bit of a hack but so be it. The Makefile will replace it to generate the actual config files given to the decoder.

Everything which is generated in the `$ROOTDIR/run-${FR}-${SYS}` directory (eg. files containing tuned MERT weights, files containing decoder output), is further uniquely identified by the corpora used. So if you have two different development sets, say `mt02` and `mt03`, the tuned weights will be saved in files named `weights-mt02.txt` and `weights-mt03.txt` respectively. Therefore the Makefile realises that the following two commands are asking to produce different things:

```
make DEV_CORPUS=mt02 tuning                   # builds weights-mt02.txt
make DEV_CORPUS=mt03 tuning                   # builds weights-mt03.txt
```

and neither will ever clobber the other. Similarly, if you also have two test sets, say, `mt05` and `mt06`, there will be four distinct files for storing decoder output inside the `$ROOTDIR/run-${FR}-${SYS}` directory, each with a name of the form `${TEST_CORPUS}.mert-${DEV_CORPUS}.txt` and the Makefile realises that the following four commands are all asking for different things:

```
make DEV_CORPUS=mt02 TEST_CORPUS=mt05 tuning  # builds mt05.mert-mt02.txt
make DEV_CORPUS=mt02 TEST_CORPUS=mt06 tuning  # builds mt06.mert-mt02.txt
make DEV_CORPUS=mt03 TEST_CORPUS=mt05 tuning  # builds mt05.mert-mt03.txt
make DEV_CORPUS=mt03 TEST_CORPUS=mt06 tuning  # builds mt06.mert-mt03.txt
```

Unfortunately the files within this directory are not distinguished on the basis of the training corpus used. (The training corpus's name does not appear in the filenames above.)[15]

B.4 Makefile targets for logical steps of experimentation

The following targets provide convenient shorthands for common "stopping points" along the pipeline.

`corpora`
: Ensures that the files listed in Section B.2 are in place. If they are not, and you have provided a custom section of the Makefile for building them, they will be built; if they are not in place, and you have not provided a custom section of the Makefile, `make` will just die complaining that it doesn't know how to make them.

`training`
: Builds the phrase table using the Moses trainer. Depends on the source and target sides of the training bitext being in place in `$ROOTDIR/corpora-$FR`. The resulting phrase model goes into `$ROOTDIR/training-${FR}-${TRAINING_CORPUS}`. To be safe, make sure this directory exists before you make this target.

[14] If this isn't true on the understanding that `$SYS` values are Subversion revision numbers, then simply define a finer-grained notion of `$SYS` values.

[15] I didn't realise I was making this mistake because I only ever had one training corpus per language. If necessary you could work around this by considering `${FR}` as an identifier for a pairing of a language with a training corpus. Alternatively it would probably not be difficult to modify the Makefile to correct this.

`pre-tuning`
: Builds everything required to get MERT tuning started, but doesn't actually run MERT. Basically it ensures that a filtered phrase table exists for the development set, and that appropriate configuration settings have been provided for the decoder.

`tuning`
: Runs MERT to get a tuned weights file. The result goes into `$ROOTDIR/run-${FR}-${SYS}`.

`pre-test`
: Builds everything required to do the test run, but doesn't actually run it. Basically it ensures that a filtered phrase table exists for the test set, and that appropriate configuration settings have been provided for the decoder.

`test`
: Decodes the test set. The result goes into `$ROOTDIR/run-${FR}-${SYS}`.

`eval`
: Evaluates the output of decoding the test set (generates a BLEU score), and builds an HTML file providing a visualisation of this output (with input sentences, references, glosses, etc.)

So, in principle, you can just run `make eval` with all the appropriate parameters set, and the Makefile will run everything required to get you a BLEU score, building everything necessary along the way. And it will build nothing unnecessary; for example, if this happens to be the first time you've run with a particular test set, so you don't have a phrase table filtered for it, `make` will realise this and build one, and the next time you run with that test set, it won't.

While in principle `make eval` is all you need, I found I usually wanted to keep a closer eye on individual steps so I used the targets listed above, but these still let me forget about dependencies among steps to a large degree.

C Sample calculations illustrating the lexicalised version of the distortion model and the label-normalisation model

Given this toy English/target linearisation model:

```
esubj|saw|man   -2      10
esubj|saw|man   -1      140
esubj|saw|man   1       20
esubj|saw|man   2       10
esubj|saw|dog   -2      10
esubj|saw|dog   -1      20
esubj|saw|dog   1       140
esubj|saw|dog   2       10
edet|man|the    -2      10
edet|man|the    -1      10
edet|man|the    1       20
edet|man|the    2       10
edet|man|a      -1      10
edet|man|a      1       100
```

and this toy source-to-target dependency label normalisation model:

```
fsubj   esubj   9
fsubj   edet    1
fdet    edet    9
fdet    esubj   1
```

here are some sample calculations, assuming a value of 100 for `MIN_OBSERVATIONS_LEXICALISATION`.

Values for $p(n|\mathsf{fsubj}, \mathit{saw}, \mathit{man})$

$$\begin{aligned} p(n|\mathsf{fsubj}, \mathit{saw}, \mathit{man}) &= p(\mathsf{esubj}|\mathsf{fsubj})p(n|\mathsf{esubj}, \mathit{saw}, \mathit{man}) + p(\mathsf{edet}|\mathsf{fsubj})p(n|\mathsf{edet}, \mathit{saw}, \mathit{man}) \\ &= \tfrac{9}{10} \times p(n|\mathsf{esubj}, \mathit{saw}, \mathit{man}) + \tfrac{1}{10} \times p(n|\mathsf{edet}, \mathit{saw}, \mathit{man}) \end{aligned}$$

We have over 100 (specifically, 180) observations for the `esubj|saw|man` condition, so $p(n|\mathsf{esubj}, \mathit{saw}, \mathit{man})$ is a simple MLE from those counts. But we don't have any observations for `edet|saw|man`, so we need to back off. We first look to `edet|*|man` and then to `edet|saw|*`[16], neither of which have been seen either, so we back off all the way to `edet|*|*`, ie. include all observations of any edet dependency, for $p(n|\mathsf{edet}, \mathit{saw}, \mathit{man})$. (There happen to be over 100 of these, but that's of no significance; we would use it anyway.)

$$p(n|\mathsf{fsubj}, \mathit{saw}, \mathit{man}) = \tfrac{9}{10} \times p(n|\mathsf{esubj}, \mathit{saw}, \mathit{man}) + \tfrac{1}{10} \times p(n|\mathsf{edet}, *, *)$$

$$\begin{aligned} p(-2|\mathsf{fsubj}, \mathit{saw}, \mathit{man}) &= \tfrac{9}{10} \times \tfrac{10}{180} + \tfrac{1}{10} \times \tfrac{10}{160} = 0.056 \\ p(-1|\mathsf{fsubj}, \mathit{saw}, \mathit{man}) &= \tfrac{9}{10} \times \tfrac{140}{180} + \tfrac{1}{10} \times \tfrac{20}{160} = 0.713 \\ p(+1|\mathsf{fsubj}, \mathit{saw}, \mathit{man}) &= \tfrac{9}{10} \times \tfrac{20}{180} + \tfrac{1}{10} \times \tfrac{120}{160} = 0.175 \\ p(+2|\mathsf{fsubj}, \mathit{saw}, \mathit{man}) &= \tfrac{9}{10} \times \tfrac{10}{180} + \tfrac{1}{10} \times \tfrac{10}{160} = 0.056 \end{aligned}$$

Values for $p(n|\mathsf{fsubj}, \mathit{man}, \mathit{the})$

$$\begin{aligned} p(n|\mathsf{fsubj}, \mathit{man}, \mathit{the}) &= p(\mathsf{esubj}|\mathsf{fsubj})p(n|\mathsf{esubj}, \mathit{man}, \mathit{the}) + p(\mathsf{edet}|\mathsf{fsubj})p(n|\mathsf{edet}, \mathit{man}, \mathit{the}) \\ &= \tfrac{9}{10} \times p(n|\mathsf{esubj}, \mathit{man}, \mathit{the}) + \tfrac{1}{10} \times p(n|\mathsf{edet}, \mathit{man}, \mathit{the}) \end{aligned}$$

We have no observations for `esubj|man|the`, or either of the semi-backed-off versions, so we back off all the way to using all 360 `esubj|*|*` observations for $p(n|\mathsf{esubj}, \mathit{man}, \mathit{the})$. We only have 50 observations for the `edet|man|the` condition, and only (the same) 50 for the next try, the `edet|*|the` condition; but we have 160 for the `edet|man|*` condition. By coincidence, these are all observations in the `edet|*|*` condition.

$$p(n|\mathsf{fsubj}, \mathit{man}, \mathit{the}) = \tfrac{9}{10} \times p(n|\mathsf{esubj}, *, *) + \tfrac{1}{10} \times p(n|\mathsf{edet}, \mathit{man}, *)$$

$$\begin{aligned} p(-2|\mathsf{fsubj}, \mathit{man}, \mathit{the}) &= \tfrac{9}{10} \times \tfrac{20}{360} + \tfrac{1}{10} \times \tfrac{10}{160} = 0.056 \\ p(-1|\mathsf{fsubj}, \mathit{man}, \mathit{the}) &= \tfrac{9}{10} \times \tfrac{160}{360} + \tfrac{1}{10} \times \tfrac{20}{160} = 0.413 \\ p(+1|\mathsf{fsubj}, \mathit{man}, \mathit{the}) &= \tfrac{9}{10} \times \tfrac{160}{360} + \tfrac{1}{10} \times \tfrac{120}{160} = 0.475 \\ p(+2|\mathsf{fsubj}, \mathit{man}, \mathit{the}) &= \tfrac{9}{10} \times \tfrac{20}{360} + \tfrac{1}{10} \times \tfrac{10}{160} = 0.056 \end{aligned}$$

Values for $p(n|\mathsf{fdet}, *, \mathit{man})$

A probability like this might be needed (not only when we've back off from a more specific condition, but also) when we know the dependent word but not the governor word, at the point in decoding when this reordering score needs to be included.

$$\begin{aligned} p(n|\mathsf{fdet}, *, \mathit{man}) &= p(\mathsf{edet}|\mathsf{fdet})p(n|\mathsf{edet}, *, \mathit{man}) + p(\mathsf{esubj}|\mathsf{fdet})p(n|\mathsf{esubj}, *, \mathit{man}) \\ &= \tfrac{9}{10} \times p(n|\mathsf{edet}, *, \mathit{man}) + \tfrac{1}{10} \times p(n|\mathsf{esubj}, *, \mathit{man}) \end{aligned}$$

[16]We try to keep the dependent before trying to keep the governor. The logic here is that idiosyncratic properties of a dependent seem more likely to affect a dependency's linearisation than idiosyncratic properties of a governor. In fact it's not really obvious to me at all that keeping the governor without the dependent is really useful at all; maybe we should skip straight to just the dependency label.

We don't have any observations with 'man' as the dependent of an edet dependency, so for $p(n|\mathsf{edet},*,\mathit{man})$ we use all 160 edet dependencies. We have 180 such observations with an esubj dependency.

$$p(n|\mathsf{fdet},*,\mathit{man}) = \frac{9}{10} \times p(n|\mathsf{edet},*,*) + \frac{1}{10} \times p(n|\mathsf{esubj},*,\mathit{man})$$

$$\begin{aligned} p(-2|\mathsf{fdet},*,\mathit{man}) &= \tfrac{9}{10} \times \tfrac{10}{160} + \tfrac{1}{10} \times \tfrac{10}{180} = 0.062 \\ p(-1|\mathsf{fdet},*,\mathit{man}) &= \tfrac{9}{10} \times \tfrac{20}{160} + \tfrac{1}{10} \times \tfrac{140}{180} = 0.190 \\ p(+1|\mathsf{fdet},*,\mathit{man}) &= \tfrac{9}{10} \times \tfrac{120}{160} + \tfrac{1}{10} \times \tfrac{20}{180} = 0.686 \\ p(+2|\mathsf{fdet},*,\mathit{man}) &= \tfrac{9}{10} \times \tfrac{10}{160} + \tfrac{1}{10} \times \tfrac{10}{180} = 0.062 \end{aligned}$$

Values for $p(n|\mathsf{fdet},\mathit{hamster},\mathit{a})$

$$\begin{aligned} p(n|\mathsf{fdet},\mathit{hamster},\mathit{a}) &= p(\mathsf{edet}|\mathsf{fdet})p(n|\mathsf{edet},\mathit{hamster},\mathit{a}) + p(\mathsf{esubj}|\mathsf{fdet})p(n|\mathsf{esubj},\mathit{hamster},\mathit{a}) \\ &= \tfrac{9}{10} \times p(n|\mathsf{edet},\mathit{hamster},\mathit{a}) + \tfrac{1}{10} \times p(n|\mathsf{esubj},\mathit{hamster},\mathit{a}) \end{aligned}$$

We don't have any 'hamster' observations at all, but we have 110 observations in the `edet|*|a` condition, so we use these for $p(n|\mathsf{edet},\mathit{hamster},\mathit{a})$. We don't have any esubj observations which match on either lexical item, so we use all 360 `esubj|*|*` observations for $p(n|\mathsf{esubj},\mathit{hamster},\mathit{a})$.

$$p(n|\mathsf{fdet},\mathit{hamster},\mathit{a}) = \frac{9}{10} \times p(n|\mathsf{edet},*,\mathit{a}) + \frac{1}{10} \times p(n|\mathsf{esubj},*,*)$$

$$\begin{aligned} p(-1|\mathsf{fdet},\mathit{hamster},\mathit{a}) &= \tfrac{9}{10} \times \tfrac{10}{110} + \tfrac{1}{10} \times \tfrac{160}{360} = 0.126 \\ p(+1|\mathsf{fdet},\mathit{hamster},\mathit{a}) &= \tfrac{9}{10} \times \tfrac{100}{110} + \tfrac{1}{10} \times \tfrac{160}{360} = 0.862 \end{aligned}$$

References

Brown, P. F., Cocke, J., Pietra, S. A. D., Pietra, V. J. D., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2):263–311.

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

Čmejrek, M., Cuřín, J., and Havelka, J. (2004). Prague Czech-English Dependency Treebank: Any hopes for a common annotation scheme? In *Proceedings of HLT/NAACL 2004 Workshop: Frontiers in Corpus Annotation*, pages 47–54.

Collins, M., Koehn, P., and Kučerová, I. (2005). Clause restructuring for statistical machine translation. In *Proceedings of ACL 2005*, pages 531–540.

Fox, H. J. (2002). Phrasal cohesion and statistical machine translation. In *Proceedings of EMNLP 2002*, pages 304–311.

Galley, M., Hopkins, M., Knight, K., and Marcu, D. (2004). What's in a translation rule? In *Proceedings of HLT-NAACL 2004*, pages 273–280.

Galley, M. and Manning, C. D. (2008). A simple and effective hierarchical phrase reordering model. In *Proceedings of EMNLP 2008*, pages 848–856.

Gildea, D. (2003). Loosely tree-based alignment for machine translation. In *Proceedings of ACL 2003*, pages 80–87.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase based translation. In *Proceedings of HLT-NAACL*, pages 127–133.

Lopez, A. (2008). Statistical machine translation. *ACM Computing Surveys*, 40(3).

Marcu, D. and Wong, W. (2002). A phrase-based, joint probability model for statistical machine translation. In *Proceedings of EMNLP 2002*, pages 133–139.

Och, F. (2003). Minimum error rate training for statistical machine translation. In *Proceedings of ACL 2003*, pages 160–167.

Och, F. J., Tillman, C., and Ney, H. (1999). Improved alignment models for statistical machine translation. In *Proceedings of the Joint Conference of Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28.

Quirk, C., Menezes, A., and Cherry, C. (2005). Dependency tree translation: Syntactically informed phrasal SMT. In *Proceedings of ACL 2005*, pages 271–279.

Shen, L., Xu, J., and Weischedel, R. (2008). A new string-to-dependency machine translation algorithm with a target dependency language model. In *Proceedings of ACL 2008*, pages 577–585.

Wu, D. and Wong, H. (1998). Machine translation with a stochastic grammatical channel. In *Proceedings of ACL-COLING*, pages 1408–1415.

Xia, F. and McCord, M. (2004). Improving a statistical mt system with automatically learned rewrite patterns. In *Proceedings of COLING 2004*, pages 508–514.

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In *Proceedings of ACL 2001*, pages 523–530.

Yamada, K. and Knight, K. (2002). A decoder for syntax-based statistical MT. In *Proceedings of ACL 2002*, pages 303–310.