Learning Non-Isomorphic Tree Mappings for Machine Translation

Jason Eisner, Computer Science Dept., Johns Hopkins Univ. <jason@cs.jhu.edu>

Abstract

Often one may wish to learn a tree-to-tree mapping, training it on unaligned pairs of trees, or on a mixture of trees and strings. Unlike previous statistical formalisms (limited to isomorphic trees), synchronous TSG allows local distortion of the tree topology. We reformulate it to permit dependency trees, and sketch EM/Viterbi algorithms for alignment, training, and decoding.

1 Introduction: Tree-To-Tree Mappings

Statistical machine translation systems are trained on pairs of sentences that are mutual translations. For example, (beaucoup d’enfants donnent un baiser`a Sam, kids kiss Sam quite often). This translation is somewhat free, as is common in naturally occurring data. The first sentence is literally Lots of’children give a kiss to Sam.

This short paper outlines “natural” formalisms and algorithms for training on pairs of trees. Our methods work on either dependency trees (as shown) or phrase-structure trees. Note that the depicted trees are not isomorphic.

Our main concern is to develop models that can align and learn from these tree pairs despite the “mismatches” in tree structure. Many “mismatches” are characteristic of a language pair: e.g., preposition insertion (of → ε), multword locutions (kiss ↔ give a kiss to: misinform ↔ wrongly inform), and head-swapping (float down ↔ descend by floating). Such systematic mismatches should be learned by the model, and used during translation.

It is even helpful to learn mismatches that merely tend to arise during free translation. Knowing that beaucoup d’ is often deleted will help in aligning the rest of the tree.

When would learned tree-to-tree mappings be useful? Obviously, in MT, when one has parsers for both the source and target language. Systems for “deep” analysis and generation might wish to learn mappings between deep and surface trees (Böhmová et al., 2001) or between syntax and semantics (Shieber and Schabes, 1990). Systems for summarization or paraphrase could also be trained on tree pairs (Knight and Marcu, 2000). Non-NLP applications might include comparing student-written programs to one another or to the correct solution.

Our methods can naturally extend to train on pairs of forests (including packed forests obtained by chart parsing). The correct tree is presumed to be an element of the forest. This makes it possible to train even when the correct parse is not fully known, or not known at all.

2 A Natural Proposal: Synchronous TSG

We make the quite natural proposal of using a synchronous tree substitution grammar (STSG). An STSG is a collection of (ordered) pairs of aligned elementary trees. These may be combined into a derived pair of trees. Both the elementary tree pairs and the operation to combine them will be formalized in later sections.

As an example, the tree pair shown in the introduction might have been derived by “vertically” assembling the 6 elementary tree pairs below. The symbol denotes a frontier node of an elementary tree, which must be replaced by the circled root of another elementary tree. If two frontier nodes are linked by a dashed line labeled with the state X, then they must be replaced by two roots that are also linked by a dashed line labeled with X.

The elementary trees represent idiomatic translation “chunks.” The frontier nodes represent unfilled roles in the chunks, and the states are effectively nonterminals that specify the type of filler that is required. Thus, donnent un baiser à (“give a kiss to”) corresponds to kiss, with the French subject matched to the English subject, and the French indirect object matched to the English direct object. The states could be more refined than those shown above: the state for the subject, for example, should probably be not NP but a pair (Npl, NPpl).

STSG is simply a version of synchronous tree-adjoining grammar or STAG (Shieber and Schabes, 1990) that lacks the adjunction operation. (It is also equivalent to top-down tree transducers.) What, then, is new here?

First, we know of no previous attempt to learn the “chunk-to-chunk” mappings. That is, we do not know at training time how the tree pair of section 1 was derived, or even what it was derived from. Our approach is to reconstruct all possible derivations, using dynamic programming to decompose the tree pair into aligned pairs of elementary trees in all possible ways. This produces a packed forest of derivations, some more probable than
others. We use an efficient inside-outside algorithm to do Expectation-Maximization, reestimating the model by training on all derivations in proportion to their probabilities. The runtime is quite low when the training trees are fully specified and elementary trees are bounded in size.\footnote{Goodman (2002) presents efficient TSG parsing with unbounded elementary trees. Unfortunately, that clever method does not permit arbitrary models of elementary tree probabilities, nor does it appear to generalize to our synchronous case. (It would need exponentially many nonterminals to keep track of an matching of unboundedly many frontier nodes.)}

Second, it is not a priori obvious that one can reasonably use STSG instead of the slower but more powerful STAG. TSG can be parsed as fast as CFG. But without an adjunction operation,\footnote{Or a sister-adjunction operation, for dependency trees.} one cannot break the training trees into linguistically minimal units. An elementary tree pair $A = (elle est finalement partie, finally she left)$ cannot be further decomposed into $B = (elle est partie, she left)$ and $C = (finalement, finally).$ This appears to miss a generalization. Our perspective is that the generalization should be picked up by the statistical model that defines the probability of elementary tree pairs. $p(A)$ can be defined using mainly the same parameters that define $p(B)$ and $p(C),$ with the result that $p(A) \approx p(B) \cdot p(C).$ The balance between the STSG and the statistical model is summarized in the last paragraph of this paper.

Third, our version of the STSG formalism is more flexible than previous versions. We carefully address the case of empty trees, which are needed to handle free-tree pairs. For expository reasons (and to fill a gap in the literature), we would like to estimate a model from unaligned data. (Positively,\footnote{However, the binary-branching SCFGs used by Wu (1997) and Alshawi et al. (2000) are strictly less powerful than STSG.} Alshawi et al. (2000) are strictly less powerful than STSG.)

Previous work in statistical synchronous grammars has been limited to forms of synchronous context-free grammar (Wu, 1997; Alshawi et al., 2000; Yamada and Knight, 2001). This means that a sentence and its translation must have isomorphic syntax trees, although they may have different numbers of surface words if null words $\epsilon$ are allowed in one or both languages. This rigidity does not fully describe real data.

The one exception is the synchronous DOP approach of (Poutsma, 2000), which obtains an STSG by decomposing aligned training trees in all possible ways (and using "naive" count-based probability estimates). However, we would like to estimate a model from unaligned data.

\section{Past Work}

Most statistical MT derives from IBM-style models (Brown et al., 1993), which ignore syntax and allow arbitrary word-to-word translation. Hence they are able to align any sentence pair, however mismatched. However, they have a tendency to translate long sentences into word salad. Their alignment and translation accuracy improves when they are forced to translate shallow phrases as contiguous, potentially idiomatic units (Och et al., 1999).

Several researchers have tried putting "more syntax" into translation models: like us, they use statistical versions of synchronous grammars, which generate source and target sentences in parallel and so describe their correspondence.\footnote{The joint probability model can be formulated, if desired, as a language model times a channel model.} This approach offers four features absent from IBM-style models: (1) a recursive phrase-based translation, (2) a syntax-based language model, (3) the ability to condition a word's translation on the translation of syntactically related words, and (4) polynomial-time optimal alignment and decoding (Knight, 1999).

4 A Probabilistic TSG Formalism

For expository reasons (and to fill a gap in the literature), we first formally present non-synchronous TSG. Let $Q$ be a set of states. Let $L$ be a set of labels that may decorate nodes or edges. Node labels might be words or nonterminals. Edge labels might include grammatical roles such as Subject. In many trees, each node’s children have an order, recorded in labels on the node’s outgoing edges.

An elementary tree is a a tuple $\langle V, V^i, E, \ell, q, s \rangle$ where $V$ is a set of nodes; $V^i \subseteq V$ is the set of internal nodes, and we write $V^f = V \setminus V^i$ for the set of frontier nodes; $E \subseteq V^i \times V$ is a set of directed edges (thus all frontier nodes are leaves). The graph $(V, E)$ must be connected and acyclic, and there must be exactly one node $r \in V$ (the root) that has no incoming edges. The function $\ell : (V^i \cup E) \rightarrow L$ labels each internal node or edge; $q \in Q$ is the root state, and $s : V^f \rightarrow Q$ assigns a frontier state to each frontier node (perhaps including $r$).
A TSG is a set of elementary trees. The generation process builds up a derived tree \( T \) that has the same form as an elementary tree, and for which \( V^f = \emptyset \). Initially, \( T \) is chosen to be any elementary tree whose root state \( T.q = \text{Start} \). As long as \( T \) has any frontier nodes, \( T.V^f \), the process expands each frontier node \( d \in T.V^f \) by substituting at \( d \) an elementary tree \( t \) whose root state, \( t.q \), equals \( d \)'s frontier state, \( T.s(d) \). This operation replaces \( T \) with \( \langle T.V \cup t.V - \{ d \}, T.V^f \cup t.V^f, T.E^f \cup t.E, T.t \cup t.t, T.q, T.s \cup t.s - \{ d, t.q \} \rangle \). Note that a function is regarded here as a set of (input, output) pairs. \( T.E^f \) is a version of \( T.E \) in which \( d \) has been replaced by \( t.r \).

A probabilistic TSG also includes a function \( p(t \mid q) \), which, for each state \( q \), gives a conditional probability distribution over the elementary trees \( t \) with root state \( q \). The generation process uses this distribution to randomly choose which tree \( t \) to substitute at a frontier node of \( T \) having state \( q \). The initial value of \( T \) is chosen from \( p(t \mid \text{Start}) \). Thus, the probability of a given derivation is a product of \( p(t \mid q) \) terms, one per chosen elementary tree.

There is a natural analogy between (probabilistic) TSGs and (probabilistic) CFGs. An elementary tree \( t \) with root state \( q \) and frontier states \( q_1 \ldots q_k \) (for \( k \geq 0 \)) is analogous to a CFG rule \( q \rightarrow q_1 \ldots q_k \). (By including \( t \) as a terminal symbol in this rule, we ensure that distinct elementary trees \( t \) with the same states correspond to distinct rules.) Indeed, an equivalent definition of the generation process first generates a derivation tree from this derivation CFG, and then combines its terminal nodes \( t \) (which are elementary trees) into the derived tree \( T \).

5 Tree Parsing Algorithms for TSG

Given a grammar \( G \) and a derived tree \( T \), we may be interested in constructing the forest of \( T \)'s possible derivation trees (as defined above). We call this tree parsing, as it finds ways of decomposing \( T \) into elementary trees.

Given a node \( c \in T.v \), we would like to find all the potential elementary subtrees \( t \) of \( T \) whose root \( t.r \) could have contributed \( c \) during the derivation of \( T \). Such an elementary tree is said to fit \( c \), in the sense that it is isomorphic to some subgraph of \( T \) rooted at \( c \).

The following procedure finds an elementary tree \( t \) that fits \( c \). Freely choose a connected subgraph \( U \) of \( T \) such that \( U \) is rooted at \( c \) (or is empty). Let \( t.V^i \) be the vertex set of \( U \). Let \( t.E \) be the set of outgoing edges from nodes in \( t.V^i \) to their children, that is, \( t.E = T.E \cap (t.V^i \times T.V) \). Let \( t.t \) be the restriction of \( T.t \) to \( t.V^i \cup t.E \), that is, \( t.t = T.t \cap ((t.V^i \cup t.E) \times L) \). Let \( t.V \) be the set of nodes mentioned in \( t.E \), or put \( t.V = \{ c \} \) if \( t.V^i = t.E = \emptyset \). Finally, choose \( t.q \) freely from \( Q \), and choose \( s : t.V^f \rightarrow Q \) to associate states with the frontier nodes of \( t \); the free choice is because the nodes of the derived tree \( T \) do not specify the states used during the derivation.

How many elementary trees can we find that fit \( c \)? Let us impose an upper bound \( k \) on \( \|t.V^i\| \) and hence on \( |U| \). Then in an \( m \)-ary tree \( T \), the above procedure considers at most \( \frac{m^n - 1}{m - 1} \) connected subgraphs \( U \) of order \( \leq k \) rooted at \( c \). For dependency grammars, limiting to \( m \leq 6 \) and \( k = 3 \) is quite reasonable, leaving at most 43 subgraphs \( U \) rooted at each node \( c \), of which the biggest contain only \( c \), a child \( c' \) of \( c \), and a child or sibling of \( c' \). These will constitute the internal nodes of \( t \), and their remaining children will be \( t \)'s frontier nodes.

However, for each of these 43 subgraphs, we must jointly hypothesize states for all frontier nodes and the root node. For \( |Q| > 1 \), there are exponentially many ways to do this. To avoid having exponentially many hypotheses, one may restrict the form of possible elementary trees so that the possible states of each node of \( t \) can be determined somehow from the labels on the corresponding nodes in \( T \). As a simple but useful example, a node labeled \( NP \) might be required to have state \( NP \). Rich labels on the derived tree essentially provide supervision as to what the states must have been during the derivation.

The tree parsing algorithm resembles bottom-up chart parsing under the derivation CFG. But the input is a tree rather than a string, and the chart is indexed by nodes of the input tree rather than spans of the input string:

1. for each node \( c \) of \( T \), in bottom-up order
2. for each \( q \in Q \), let \( \beta_r(q) = 0 \)
3. for each elementary tree \( t \) that fits \( c \), increment \( \beta_r(t.q) \) by \( p(t \mid q) \prod_{d \in t.V^f} \beta_d(t.s(d)) \)

The \( \beta \) values are inside probabilities. After running the algorithm, if \( r \) is the root of \( T \), then \( \beta_r(\text{Start}) \) is the probability that the grammar generates \( T \).

\( p(t \mid q) \) in line 4 may be found by hash lookup if the grammar is stored explicitly, or else by some probabilistic model that analyzes the structure, labels, and states of the elementary tree to compute its probability.

One can mechanically transform this algorithm to compute outside probabilities, the Viterbi parse, the parse forest, and other quantities (Goodman, 1999). One can also apply agenda-based parsing strategies.

For a fixed grammar, the runtime and space are only \( O(n) \) for a tree of \( n \) nodes. The grammar constant is the number of possible fits to a node \( c \) of a fixed tree. As noted above, there usually not many of these (unless the states are uncertain) and they are simple to enumerate.

As discussed above, an inside-outside algorithm may be used to compute the expected number of times each elementary tree \( t \) appeared in the derivation of \( T \). That is the E step of the EM algorithm. In the M step, these expected counts (collected over a corpus of trees) are used to reestimate the parameters \( \hat{\theta} \) of \( p(t \mid q) \). One alternates E and M steps till \( p(\text{corpus} \mid \hat{\theta}) \cdot p(\hat{\theta}) \) converges to a local maximum. The prior \( p(\hat{\theta}) \) can discourage overfitting.

We gloss over the standard difficulty that the derivation CFG may contain a unary rule cycle. For us, such a cycle is a problem only when it arises solely from single-node trees.
6 Extending to Synchronous TSG

We are now prepared to discuss the synchronous case. A synchronous TSG consists of a set of elementary tree pairs. An elementary tree pair \( t \) is a tuple \((t_1, t_2, q, m, s)\). Here \( t_1 \) and \( t_2 \) are elementary trees without state labels; we write \( t_j = (V_j, V_j', E_j, t_j) \), \( q \in Q \) is the root state as before. \( m \subseteq V_j' \times V_j' \) is a matching between \( t_1 \)'s and \( t_2 \)'s frontier nodes. \(^6\)

Let \( \bar{m} \) denote \( m \cup \{(d_1, \text{null}) : d_1 \text{ is unmatched in } m\} \cup \{(\text{null}, d_2) : d_2 \text{ is unmatched in } m\} \). Finally, \( s : \bar{m} \rightarrow Q \) assigns a state to each frontier node pair or unpaired frontier node.

In the figure of section 2, donnet un baiser à has 2 frontier nodes and kiss has 3, yielding 13 possible matchings. Note that at least one English node must remain unmatched; it still generates a full subtree, aligned with null.

As before, a derived tree pair \( T \) has the same form as an elementary tree pair. The generation process is similar to before. As long as \( T, \bar{m} \neq \emptyset \), the process expands some node pair \((d_1, d_2) \in T, \bar{m}\). It chooses an elementary tree pair \( t \) such that \( t.q = T.s(d_1, d_2) \). Then for each \( j = 1, 2 \), it substitutes \( t_j \) at \( d_j \) if non-null. (If \( d_j \) is null, then \( t.q \) must guarantee that \( t_j \) is the special null tree.)

In the probabilistic case, we have a distribution \( p(t \mid q) \) just as before, but this time \( t \) is an elementary tree pair.

Several natural algorithms are now available to us:

- **Training.** Given an unaligned tree pair \((T_1, T_2)\), we can again find the forest of all possible derivations, with expected inside-outside counts of the elementary tree pairs. This allows EM training of the \( p(t \mid q) \) model.

The algorithm is almost as before. The outer loop iterates bottom-up over nodes \( c_1 \) of \( T_1 \); an inner loop iterates bottom-up over \( c_2 \) of \( T_2 \). Inside probabilities (for example) now have the form \( \beta_{c_1, c_2}(q) \). Although this brings the complexity up to \( O(n^2) \), the real complication is that there can be many fits to \((c_1, c_2)\). There are still not too many elementary trees \( t_1 \) and \( t_2 \) rooted at \( c_1 \) and \( c_2 \); but each \((t_1, t_2)\) pair may be used in many elementary tree pairs \( t \), since there are exponentially many matchings of their frontier nodes. Fortunately, most pairs of frontier nodes have low \( \beta \) values that indicate that their subtrees cannot be aligned well; pairing such nodes in a matching would result in poor global probability. This observation can be used to prune the space of matchings greatly.

- **1-best Alignment** (if desired). This is just like training, except that we use the Viterbi algorithm to find the single best derivation of the input tree pair. This derivation can be regarded as the optimal syntactic alignment. \(^7\)

---

\(^6\)A matching between \( A \) and \( B \) is a 1-to-1 correspondence between a subset of \( A \) and a subset of \( B \).

\(^7\)As free-translation post-processing, one could try to match pairs of stray subtrees that could have aligned well, according to the chart, but were forced to align with null for global reasons.