Abstract
In this paper we describe an open source tool for automatic induction of transfer rules. Transfer rule induction is carried out on pairs of dependency structures and their node alignment to produce all rules consistent with the node alignment. We describe an efficient algorithm for rule induction and give a detailed description of how to use the tool.

1. Introduction

Statistical Machine Translation (SMT) using deep linguistic representations for transfer is a relatively new and growing research area (Bojar and Hajič, 2008, Graham, 2008, Bojar et al., 2007a, Bojar and Čmejrek, 2007b, Graham et al., 2007, Ding and Palmer, 2006, Riezler and Maxwell, 2006, Ding and Palmer, 2005, Ding and Palmer, 2004a, Ding and Palmer, 2004b, Čmejrek et al., 2003, Eisner, 2003, Ding and Palmer, 2003, Hajič et al., 2002, Alshawi et. al., 2000a, Alshawi et. al., 2000b, Alshawi et. al., 1998). Training requires highly efficient algorithms; the training data is hierarchical dependency graphs as opposed to surface form strings and is therefore more complex than training data used for other methods of SMT (Brown et al., 1993); large numbers of rules (that contain a lot of morphological information) are needed to achieve high coverage of unseen data and rich statistical information. We provide a tool that uses an efficient algorithm for rule induction and outputs the rules in efficient O(n) size data structures (Graham, 2008). The rule induction tool, the parser/generator engine XLE ¹ and transfer decoder ² constitute a complete Transfer-Based SMT system.

¹“XLE is available to a limited number of researchers ... For more information about obtaining a license, please contact thking at parc.com.”
²The transfer decoder is currently in development and will be released as an open source tool in the near future.
2. Transfer-Based SMT

Transfer-Based SMT is composed of three parts, i) parsing to linguistic structure, ii) transfer from SL linguistic structure to TL linguistic structure and iii) generation of TL sentence. Each step uses a statistical model to select the best output. For parsing, a disambiguation model is used to rank the parses and select the n-best output parses. Decoding (transfer) is then done on a parse structure (or n-best list of parse structures) via beam-search producing an n-best list of TL structures. For generation, the input is an n-best list of TL structures and all possible TL sentences are produced. The best TL sentence is then selected using an n-gram language model.

3. Training

The aim of rule induction is to acquire rules useful for transferring unseen SL structures by capturing correspondences between example training structure pairs of a parsed bilingual corpus. Figure 1(a) shows a rule that captures an isomorphic German-English correspondence, while Figure 1(b) captures a non-isomorphic correspondence. Lexicalized nodes contain a vector of feature value pairs storing the morphological information belonging to that node.

3.1. Transfer Rule Induction

The transfer rule induction algorithm takes as input i) a dependency structure pair and ii) a one-to-one set of alignments between nodes of the dependency structure pair. Any alignment method can be used to align the nodes. For example, we currently use Giza++ (Och et al., 1999) for node alignment by constructing a bilingual training corpus from the predicate lemmas of the dependency structure nodes of the parsed bitext. Word alignment can then be run, in both

\[\text{LHS contains a single lexicalized node with predicate value (voran)kommen with two arguments, a subject (X_0) and an adjunct (X_1) and the the RHS has the same structure but with progress as the root node.}\]

\[\text{The LHS has the German predicate halten as its root node with three arguments, a subject (the lexicalized node with predicate pro), an object (X_0) and an xcomplement-predicate (the lexicalized node with predicate fur and object X_1) and the RHS of the rule has be as its root node with two arguments, a subject (X_0) and an xcomplement (X_1).}\]
language directions, as in Phrase-Based SMT (Koehn et al., 2003), followed by symmetrisation of the word alignments. We currently use the intersection of the word alignments as this gives a reliable set of one-to-one alignments.

3.1.1. Consistent Transfer Rules

As in Phrase-Based SMT, where a word alignment for each example sentence pair is first established before phrases consistent with that word alignment are extracted (Och et al., 1999, Koehn et al., 2003), we induce transfer rules that are consistent with the node alignment. We define a consistent transfer rule using a simplification of the actual training dependency structures and temporarily consider them as tree structures by ignoring edges that cause cycles in the graph or edges that share an end node with another edge.⁵ Definition 1 applied to a (simplified) dependency structure pair yields a set of rules containing no variables by constraining rule induction using both the alignments between nodes and the position of the nodes within the two structures:

**Definition 1.**

Given a one-to-one set of alignments $A$ between nodes in dependency pair $(F, E)$, $(f, e)$ is a rule consisting of nodes $(N_f, N_e)$, rooted at $(r_f, r_e)$, with descendents $(D_f, D_e)$ of $r_f$ and $r_e$ in $F$ and $E$ respectively, if

\[
N_f = r_f \cup D_f \\
N_e = r_e \cup D_e \\
\forall f_i \in N_f : (f_i, e_j) \in A \Rightarrow e_j \in N_e \\
\forall e_j \in N_e : (f_i, e_j) \in A \Rightarrow f_i \in N_f \\
\exists e_j \in N_e : (r_f, e_j) \in A \\
\exists f_i \in N_f : (f_i, r_e) \in A
\]

**Definition 2.**

For any rule $(f, e)$ in dependency pair $(F, E)$ rooted at $(r_f, r_e)$ consisting of nodes $N_f$ and $N_e$, where $(s, t)$ is also a rule in $(F, E)$ rooted at $(r_s, r_t)$ consisting of nodes $N_s$ and $N_t$ where $r_s \neq r_f$, $r_t \neq r_e$, if $r_s \in N_f$ and $r_t \in N_e$ then there is a rule $(a, b)$ rooted at $(r_f, r_e)$ with nodes $r_s$ and $r_t$ replaced by variable $x_k$, where $k$ is an index unique to the transfer rule, consisting of nodes:

\[
N_a : N_f \setminus N_s \cup x_k \\
N_b : N_e \setminus N_t \cup x_k
\]

To help visualize what is considered a consistent transfer rule, Figure 2(b) shows the example dependency structure of Figure 2(a) divided into parts by a number of boxes with corresponding parts of the dependency structure pair labelled with the numbers 1-6. Each consistent transfer rule can be realized by assigning a true or false value to each pair of boxes, so that

⁵For example, if the subject of node A is also the subject of node B, one of these these edges is ignored temporarily. This is done by labelling the nodes using an increasing index in depth first order (only labelling each node once). Edges with an end node label less than their start node are ignored.
Figure 2. Consistent Transfer Rules

boxes assigned true are included in the rule and boxes assigned false are left out. Combinations of true and false values are constrained and this can be visualized by only allowing adjoining boxes in Figure 2(b) to be labelled true for any one rule. Figures 2(c), 2(d) and 2(e) show consistent rules with truth value combinations. According to Definition 1, two nodes may form the root of a transfer rule if they both have the same non-empty set of aligned descendents (rule root nodes are in bold in Figure 2(b)). This ensures that the entire subtree rooted at the SL root of a transfer rule corresponds to the entire subtree rooted at the TL root. In addition, the root nodes of a rule must each be an aligned node, i.e. each must be aligned to some node in the other side of the rule. This ensures that unaligned nodes do not form the root of a transfer rule and are, therefore, only included in rules that also contain their head node, giving them a meaningful context. For example, the rule in Figure 2(e) is allowed by Definition 1, but a rule with this LHS and a RHS rooted at of is disallowed. The node of does not have a lexical correspondent in the German structure, but instead its head diversity, the dependency label adjunct.
and node *of* together correspond to *Vielfalt* and the dependency label *adjunct-gen* (genative adjunct). Definition 2 describes how rules with variables are produced by replacing both sides of a nested rule, i.e. a rule nested inside a larger rule, with a variable $X_n$. For example, in Figure 2, rule (d) was produced from rule (c) by replacing the nested rule (e) with variable $X_1$. Introducing variables to transfer rules in this way potentially produces a very large number of rules. In the worst case, when we have isomorphic structures with all nodes aligned, the number of rules is exponential. However, in reality, dependency structures of real world parallel corpora are very rarely entirely isomorphic with a very high number of aligned nodes. If the number of rules induced does indeed become unmanageable, however, rule induction can be constrained further by putting a limit on the size of rules.⁶

### 3.2. Transfer Rule Induction Algorithm

The transfer rule induction algorithm works by encoding all consistent rules of the SL and TL dependency structure pair in a single structure. The most complex part of the algorithm decides which node pairs within the dependency structure pair form *rule roots*. Once the *rule roots* of the dependency structure pair have been decided, the entire set of SL and TL dependency triples are then simply recorded with each dependency triple slightly modified by labelling it with a *context variable* ($A_i$) (Maxwell and Kaplan, 1991) and by replacing the original node labels with variables ($X_i$). Labelling the dependency triples with *context variables* allows us to encode all rules consistent with the node alignment in a single structure. This method of encoding allows $O(2^n)$ rules to be encoded in an $O(n)$ size structure and is described in detail in Graham and van Genabith (2008). The algorithm for choosing the *rule root* nodes of the dependency structure pair is given in Figure 3.⁷ The complexity of the algorithm is $O(\alpha^2 \log \alpha)$ in the worst case, where $\alpha$ is the number of aligned node pairs.

### 4. Using the Rule Induction Tool

The rule induction tool requires a Prolog engine to run. The system has been developed and tested with SICStus Prolog. Included in the download package are 4000 German-English sample dependency structure pairs from a portion of the Europarl Corpus (Koehn et al., 2005).⁸ The sample dependency structures are divided into sets, each containing 1000 sentences.

---

⁶We currently do not limit the size of rules as we train on a restricted sentence length of 5 to 15 words. This results in an average number of rules induced per sentence pair of 32.47, with average sentence length 9.89 for German and 10.48 for English.

⁷The program itself is written in Prolog, and uses some of Prolog’s built-in features that are not available in other programming languages. To keep the pseudocode as implementation independent as possible, when we use a Prolog specific function or control structure we describe it in pseudocode using an equivalent function or control structure available in most programming languages. For example, Prolog has a built in indexing of terms, that uses the first argument of the term as a key to achieve an $O(\log n)$ return time when searching for that term in memory. We use this in our Prolog implementation but where we do so we described it in pseudocode as using a hash table.

⁸The sample sentences were parsed with XLE parse engine (Kaplan et al., 2002) to Lexical Functional Grammar (Kaplan and Bresnan, 1982, Bresnan, 2001, Dalrymple, 2001) f-structures using German and English grammars (Rie-
Algorithm RuleRoots( List sl_nodes, List tl_nodes,
   HashTable <sl_node_id,alignment_id> sl_alignments,
   HashTable <tl_node_id,alignment_id> tl_alignments):

# For each aligned SL node create a list
# containing the alignment ids of its aligned descendents
# Put lists in a Hash Table
sl_aligned_descs = new HashTable<list_of_aligned_descs,sl_node_id>
foreach s ∈ S
   if exists sl_alignments.get(s.node_id) then
      list = new empty list
      foreach d ∈ descendents(s)
         if exists sl_alignments.get( d.node_id) then
            a_id = sl_alignment.get( d.node_id)
            list.add( a_id)
      sl_aligned_descs.put( list, s)

# Likewise for TL nodes
tl_aligned_descs = new HashTable<list_of_aligned_descs,tl_node_id>
foreach t ∈ T
   if exists tl_alignments.get(t.node_id) then
      list = new empty list
      foreach d ∈ descendents(t)
         if exists tl_alignments.get( d.node_id) then
            a_id = tl_alignment.get( d.node_id)
            list.add( a_id)
      tl_aligned_descs.put( list, t)

# Find node pairs with matching sets of aligned descendents
roots = new empty List
foreach key in keys( sl_aligned_desc)
   if exists tl_aligned_descs.get( key)
      # A pair has been found
      i = sl_aligned_descs.get( key)
      j = tl_aligned_descs.get( key)
      roots.add( i, j)
return roots

Figure 3. Algorithm to choose the rule roots in the SL and TL dependency structures
1. Download the package from  
   http://www.computing.dcu.ie/~ygraham/software.html
2. Unzip ria.tar.gz and extract the files. This should produce a folder called ria.
3. Create an environment variable called RIA and set it to the location of the tool, for example:  
   RIA=/home/jsmith/ria; export RIA;
4. Add the location of the bin directory to your PATH environment variable, for example:  
   PATH=$PATH:$RIA/bin; export PATH;
5. Test the tool by typing the following command to induce all rules from set 0 of the sample  
   training sentences: ria 0
A file containing the rules of set 0 should now be written in the following directory: $RIA/output/rules.

4.1. Output Format

   The rules induced from each training dependency pair are stored in a file containing the  
   rules (in a single compact size structure), and a file with some additional information about  
   the rules. For example, the following two files store rules for sentence 123 of set 0:
   - $RIA/output/rules/sents_0000/R123.pl
   - $RIA/output/rules/sents_0000/I123.pl
To retrieve the rules, both of these files should be loaded by SICStus before calling the following  
   predicate:
   - transfer_rule(+S, +T, +Fs_id, +Root_id, -LHS, -RHS, +Options).
We include an example program ⁹ that retrieves all rules for a specified sentence pair and  
   records them:
   - write_rules 123
The enumerated rules for sentence id 123 should now be written in:  
   - $RIA/output/example/R123

4.2. Running the Tool on New Training Data

   Start by converting the dependency structures into the same Prolog format of the sample  
   structures.¹⁰ The training structures should be divided into sets of 1000 and put in directories  
   in the following locations:
   - $RIA/data/sl_train
   - $RIA/data/tl_train
   - $RIA/data/alignments

---

⁹The rules should be retrieved by loading the rule and information files and then calling findall on transfer_rules/7,  
as is done in the example program, as it is not necessary to enumerate all rules on disk to use them.

¹⁰The sample structures are in the output format of the XLE parse engine (for further details see  
For example, SL and TL dependency structures for sentence id 134067 should be put in two separate files;
- $RIA/data/sl_train/sents_0134/S067.pl and
- $RIA/data/tl_train/sents_0134/S067.pl,
and node alignments in a file called
- $RIA/data/alignments/sents_0134/A067.pl.
These files can then be archived and zipped. Name them as follows:
- $RIA/data/sl_train/sents_0134.tar.gz
- $RIA/data/tl_train/sents_0134.tar.gz
- $RIA/data/alignments/sents_0134.tar.gz
Rule induction can then be run for this set of structures using the command:
- ria 134

5. Conclusion

We presented an open-source tool for automatic transfer rule induction from parsed bilingual corpora. We described an efficient algorithm that induces transfer rules from dependency structure pairs encoding rules in an efficient O(n) size data structure (Graham, 2008). We hope that this tool is used to produce interesting research.

Acknowledgments

Many thanks to John Maxwell, Joachim Wagner, Marcus Furlong, Mary Hearne, Philipp Koehn and our reviewers.

Bibliography


Yuan Ding and Martha Palmer. 2006. Better Learning and Decoding for Syntax Based SMT Using PSDIG. In *Proceedings of AMTA 2006*


