Rule Extraction for Tree-to-Tree Transducers by Cost Minimization

Pascual Martínez-Gómez[1]
pascual.mg@aist.go.jp

Yusuke Miyao[1,2,3]
yusuke@nii.ac.jp

[1]Artificial Intelligence Research Center, AIST
[2]National Institute of Informatics and JST, PRESTO
[3]The Graduate University for Advanced Studies (SOKENDAI)
Tokyo, Japan

Abstract

Tree transducers that model expressive linguistic phenomena often require word-alignments and a heuristic rule extractor to induce their grammars. However, when the corpus of tree/string pairs is small compared to the size of the vocabulary or the complexity of the grammar, word-alignments are unreliable. We propose a general rule extraction algorithm that uses cost functions over tree fragments, and formulate the extraction as a cost minimization problem. As a by-product, we are able to introduce back-off states at which some cost functions generate right-hand-sides of previously unseen left-hand-sides, thus creating transducer rules "on-the-fly". We test the generalization power of our induced tree transducers on a QA task over a large Knowledge Base, obtaining a reasonable syntactic accuracy and effectively overcoming the typical lack of rule coverage.

1 Introduction

Tree transducers are general and solid theoretical models that have been applied to a variety of NLP tasks, such as machine translation (Knight and Graehl, 2005), text summarization (Cohn and Lapata, 2009), question answering (Jones et al., 2012), paraphrasing and textual entailment (Wu, 2005). One strategy to obtain transducer rules is by exhaustive enumeration; however, this method is ineffective when there is a high structural language variability and we wish to have an expressive model. Another strategy is to heuristically extract rules from a corpus of tree/string pairs and word-alignments, as GHKM algorithm does (Galley et al., 2004); however, word-alignments are difficult to estimate when the corpus is small. This would be the case, for example, of machine translation for low-resourced languages where there is often small numbers of parallel sentences, or in Question Answering (QA) tasks where the number of Knowledge Base (KB) identifiers (concepts) is much larger than QA datasets.

Our main contribution is an algorithm that formulates the rule extraction as a cost minimization problem, where the search for the best rules is guided by an ensemble of cost functions over pairs of tree fragments. In GHKM, a tree fragment and a sequence of words are extracted together if they are minimal and their word alignments do not fall outside of their respective boundaries. However, given that alignment violations are not allowed, the quality of the extracted rules degrades as the rate of misaligned words increases. In our framework, we can mimic GHKM by assigning an infinite cost to pairs of tree fragments that violate such conditions on word alignments and by adding a cost regularizer on the size of the tree fragments. Smoother cost functions, however, would permit controlled misalignments, contributing to generalization. Given the generality of these cost functions, we believe that the applicability of tree transducers will be extended.

A by-product of introducing these cost functions is that some of them may act as rule back-offs, where transducer rules are built "on-the-fly" when the transducer is at a predefined back-off state but there is no rule whose left-hand-side (*lhs*) matches the input subtree. These back-off states can be seen as functions that are capable of generating right-hand-sides (*rhs*) for unseen input subtrees.

Our rule extraction algorithm and back-off scheme are general, in the sense that they can be applied to any tree transformation task. However, in this paper, we extrinsically evaluate the quality of the extracted rules in a QA task, where the objective is to transform syntactic trees of questions into constituent trees that represent Sparql queries on Freebase, a large Knowledge Base. Implementing all components of a QA system at a sufficient level is out of the scope of this paper; for that reason, in order to evaluate our contribution in isolation, we use the FREE917 corpus released by Cai and Yates (2013), for which an entity and predicate lexicon is available[1]. We show that a tree-to-tree transducer induced using our rule extraction and back-off scheme is accurate and generalizes well, which was not previously achieved with tree transducers in semantic parsing tasks such as QA over large KBs.

2 Related Work

Tree transducers were first proposed by Rounds (1970) and Thatcher (1970), and have been greatly developed recently (Knight and Graehl, 2005). Jones et al. (2012) used tree transducers to semantically parse narrow-domain questions into Prolog queries for GeoQuery (Wong and Mooney, 2006), a small database of 700 geographical facts. Rules were exhaustively enumerated, which was possible given the small size of the database and low variability of questions. Another strategy is that of Li et al. (2013), where they used a variant of GHKM to induce tree transducers that parse into λ-SCFG. Word-to-node alignments could be reliably estimated with the IBM models (Brown et al., 1993) given, again, the small vocabulary and database size of GeoQuery. In such small-scale tasks, our rule extraction and back-off scheme offers no obvious advantage. However, when doing QA over larger and more realistic KBs (and other tasks with similar characteristics), exhaustive enumeration of rules or reliable estimations of alignments are not possible, which prevents the application of tree transducers. Thus, it is on the latter type of tasks where we focus our contribution.

A similar problem has been considered in the tree mapping literature in the form of the *tree-to-tree edit distance*. In that formulation, three edit operations are defined, namely, deleting and inserting single nodes, and replacing the label of a node. These edit operations have a cost associated to them, and the task consists of finding the minimum edit cost and its corresponding edit script[2] that transforms a source into a target tree. The problem was first solved by Tai (1979), and later Zhang and Shasha (1989) proposed a simpler and faster dynamic programming algorithm that operates in polynomial time, and that has inspired multiple variations (Bille, 2005).

However, we need edit operations that involve tree fragments (e.g., noun phrases or parts of verb phrases), rather than single nodes, when searching for the best mappings. We address this problem by searching for non-isomorphic tree mappings, in line with Eisner (2003), except that our rule extraction algorithm is guided by an ensemble of cost functions over *pairs of tree fragments*. This algorithm is capable of extracting rules more robustly than GHKM by permitting misalignments in a controlled manner. Finding a tree mapping solves simultaneously the alignment and the rule extraction problem.

There is a wide array of tree transducers with different expressive capabilities (Knight and Graehl, 2005). We consider *extended*[3] *root-to-frontier*[4] *linear*[5] transducers (Maletti et al., 2009), possibly with *deleting*[6] operations. In this paper, we syntactically parse the natural language question and transform it into a meaning representation, similarly to Ge and Mooney (2005). But instead of using Prolog formulae or λ-SCFG, we use constituent representations of $\lambda-$DCS expressions (Liang, 2013), which is a formal language convenient to represent Sparql queries where variables are eliminated by making existential quantifications implicit (see example in Figure 1).

Another challenge is to construct transducers with sufficient rule coverage, which would require billions of lexical rules that map question phrases to database entities or relations. Even if those rules were available, estimating their rule probabilities would be difficult given the small data sets of ques-

[1]The entity lexicon was released by the authors of FREE917, and the predicate lexicon is ours.

[2]Sequence of edit operations.

[3]*lhs* may have depth larger than 1.

[4]Top-down transformations.

[5]*lhs* variables appear at most once in the *rhs*.

[6]Some variables on the *lhs* may not appear in the *rhs*.

tions paired with their logical representations. We solve the problem by constructing lexical rules "on-the-fly" at the decoding stage, similarly to the candidate generation stage of entity linking systems (Ling et al., 2015). Rule weights are also predicted on-the-fly given rule features and model parameters similar to Cohn and Lapata (2009).

3 Background

Tree transducers apply to general tree transformation problems, but for illustrative purposes, we use the tree pair s and t in Figure 1 (from FREE917) as a running example. s is the syntactic constituent tree of the question "how many teams participate in the uefa", whereas t is a constituent tree of an executable meaning representation in the $\lambda-$DCS formalism:

$$\texttt{count}(\texttt{Team.League.Uefa})$$

Its corresponding lambda expression is

$$\texttt{count}(\lambda x.\exists a.\texttt{Team}(x,a) \wedge \texttt{League}(a, \texttt{Uefa}))$$

which can be converted into a Sparql KB query:

```
SELECT COUNT(?x) WHERE {
  ?a   Team     ?x    .
  ?a   League   Uefa  .}
```

Following the terminology of Graehl and Knight (2004), we define a tree-to-tree transducer as a 5-tuple $(\mathcal{Q}, \Sigma, \Delta, q_{\text{start}}, \mathcal{R})$ where $\mathcal{Q}$ is the set of states, Σ and Δ are the sets of symbols of the input and output languages, q_{start} is the initial state, and $\mathcal{R}$ is the set of rules. For convenience, define T_Σ as the set of trees with symbols in Σ, $T_\Sigma(\mathcal{A})$ the set of trees with symbols in $\Sigma \cup \mathcal{A}$ where symbols in $\mathcal{A}$ only appear in the leaves, $\mathcal{X}$ as the set of variables $\{x_1, \ldots, x_n\}$, and $\mathcal{A}.\mathcal{B}$ for the cross-product of two sets $\mathcal{A}$ and $\mathcal{B}$. A rule $r \in \mathcal{R}$ has the form $q.t_i \xrightarrow{s} t_o$, where $q \in \mathcal{Q}$ is a state, $t_i \in T_\Sigma(\mathcal{X})$ is the left-hand-side (*lhs*) tree pattern (or elementary tree), $t_o \in T_\Delta(\mathcal{Q}.\mathcal{X})$ the right-hand-side (*rhs*), and $s \in \mathbb{R}$ the rule score.

Tree-to-tree transducers apply a sequence of rules to transform a source s into a target t tree. A root-to-frontier transducer starts at the root of the source tree and searches $\mathcal{R}$ for a rule whose i) tree pattern t_i on the *lhs* matches the root of the source tree, and ii) the state q of the rule is the initial state of the transducer. An incipient target tree is created by copying the *rhs* of the rule. Then, the transducer recursively and independently visits the subtrees of the source tree at the *lhs* variable positions of the rule from their new states, and copies the results into the same variable on the target tree.

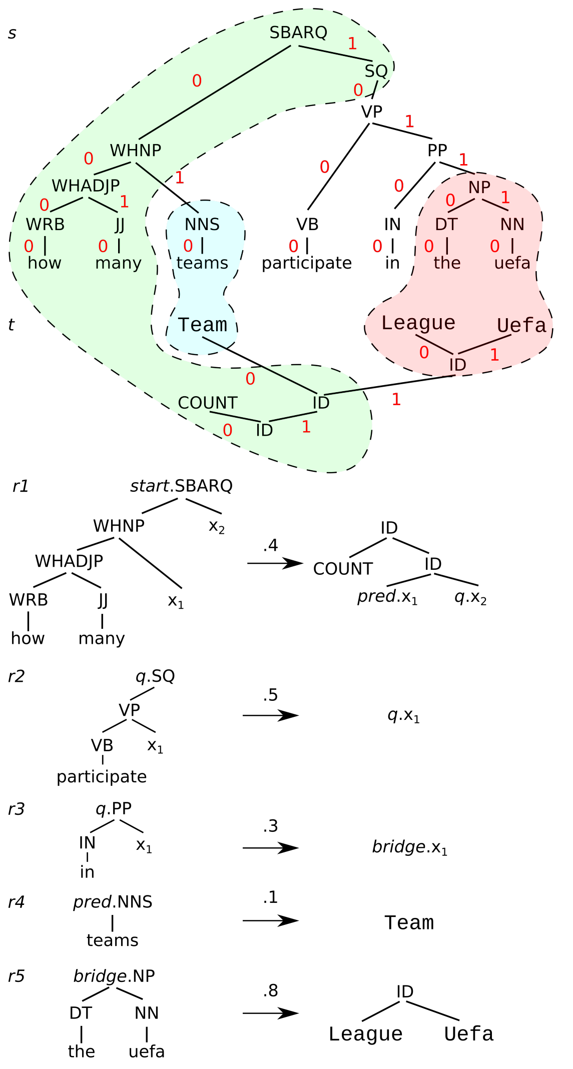


Figure 1: (s) Constituent tree of a question; (t) executable meaning representation; $r1$ - $r5$ are typical transducer rules extracted by our algorithm, where q is a generic state, *pred* and *bridge* are predicate and bridged entity back-off states.

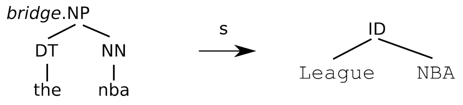
In Figure 1, the sequential application of rules $r1$ to $r5$ is a derivation that transforms the question s into the query t. For example, rule $r1$ consumes a tree fragment of s (e.g. "how", "many", "WRB", etc.) and produces a tree fragment with terminals ("COUNT", x_1, x_2) and non-terminals ("ID") with a specific structure. Rules $r2$ and $r3$ only consume but do not produce symbols (other than variables). The *rhs* of rules are target tree fragments that connect to each other at the frontier nodes (those with variables). Rules $r4$ and $r5$ are terminal rules, where $r4$ produces the predicate `Team` and rule $r5$ produces the entity `Uefa` and a disambiguating predicate `League` that has no lexical support on the source side, similarly to the role that *bridging predicates* play in Berant et al. (2013).

Given a corpus of source and target tree pairs, the learning stage aims to obtain rules such as $r1 - r5$ in Figure 1 and their associated probabilities or scores. We discuss our novel approach to rule extraction in Section 5. For the assignment of rule scores, we adopt the latent variable averaged structured perceptron, a discriminative procedure similar to Tsochantaridis et al. (2005) and Cohn and Lapata (2009). Here, we instantiate feature values $\mathbf{f}$ for every rule, and reward the weights $\mathbf{w}$ of rules that participate in a derivation (latent variable) that transforms a training source tree into a meaning representation that retrieves the correct answer.

At decoding stage, rule scores can be predicted as $s = \mathbf{w} \cdot \mathbf{f}$. However, we cannot expect to have extracted all necessary rules at the training stage given the small training data and large-scale KB. For that reason, we propose in Section 4 a novel rule back-off scheme to alleviate coverage problems.

4 Back-off rules

As an illustrative example, consider the question "how many teams participate in the nba", and the rules $r1$ to $r5$ in Figure 1. When the transducer attempts to transform the noun phrase (NP (DT the) (NN nba)), no rule's *lhs* matches it. However, since the transducer is at state *bridge* (as specified by the *rhs* of $r3$), it should be able to produce a list of bridged entities, among which the target subtree (ID `League` `NBA`) will be hopefully included. Thus, the following rule should be created for the occasion:



This mechanism produces rules "on-the-fly", allowing us to compensate low rule coverage by consuming and producing tree fragments that were not necessarily observed in the training data.

Back-off rules are produced when the transducer is at a back-off state $q_b \in \mathcal{Q}_b \subset \mathcal{Q}$, similarly as the back-off mechanisms in finite-state language models where we produce estimates (probabilities) of input structures (sequences) under less conditioning. In our scheme, a back-off state (or function) q_b produces estimates that are target structures $t_2 \in T_\Delta$ with score $s \in \mathbb{R}$, given some information of the source tree fragment $t_1 \in T_\Sigma$. That is, a function $q_b : T_\Sigma \rightarrow \{(T_\Delta, \mathbb{R}), \ldots\}$. In our QA application, we only use the leaves of the input subtree t_1 and use lexicons or entity/predicate linkers to retrieve KB entities, KB relations or a compound of a disambiguating relation and an entity from back-off states *ent*, *pred* and *bridge*, respectively. Other back-off functions would transliterate the leaves of the input tree in machine translation, or produce synonyms/hypernyms in a paraphrasing application.

We associate a score s to these newly created rules, which we learn to predict using the discriminative training procedure suggested by Tsochantaridis et al. (2005), as described in Section 3.

Back-off rules are then constructed on-demand as $q_b.t_1 \xrightarrow{s} t_2$, and the discrete set of rules $\mathcal{R}$ is augmented with them. It remains now to recognize those back-off states when inducing tree transducer grammars, which is covered in Section 5.1.

5 Rule Extraction

Given a pair of trees, our rule extraction algorithm finds a tree mapping that implicitly describes the rules that transform a source into a target tree. In the search of the best mapping, we need to explore the space of edit operations, which are substitutions of source by target tree fragments. We define cost functions for these edit operations, and formulate the tree mapping as a cost minimization problem. Whereas our tree mapping algorithm and back-off scheme are generic and can be used in any tree transformation task, cost functions depend on the application.

5.1 Cost functions

In general, cost functions are defined over edit operations, which are pairs of source and target tree fragments, cost : $T_\Sigma(\mathcal{X}) \times T_\Delta(\mathcal{Q}.\mathcal{X}) \rightarrow \mathbb{R}^{\geq 0}$, and they are equivalent to feature functions. Some cost functions are defined over all pairs of tree fragments.

For this QA application, these are:

$$\textit{csize}(t_1, t_2) = |\text{nodes}(t_1)|^2 + |\text{nodes}(t_2)|^2$$

which acts as a tree size regularizer, returning a cost quadratic to the size of the tree fragments, thus encouraging small rules. The cost function *ccount* assigns zero cost if (i) "how" and "many" appear in t_1, and (ii) "COUNT" appears in t_2. If only either (i) or (ii), the cost is a positive constant. Similarly, other operators (max, min, argmax, etc.) could be recognized, but this dataset did not require them.

Other cost functions only apply to some pairs of tree fragments. These are the back-off functions described in Section 4, but instead of returning scores for every target tree fragment, they return a cost, e.g. *cent* : $T_\Sigma \times T_\Delta \to \mathbb{R}^{\geq 0}$. An ensemble will produce up to three different costs for every pair of tree fragments, depending on what back-off functions were triggered. In the case of the entity cost function:

$$\begin{aligned}\gamma_{\textit{ent}}(t_1, t_2) = {} & \lambda_1 \cdot \textit{csize}(t_1, t_2) \\ & + \lambda_2 \cdot \textit{ccount}(t_1, t_2) \\ & + \lambda_3 \cdot \textit{cent}(t_1, t_2)\end{aligned} \tag{1}$$

where $\lambda_i \in \mathbb{R}^{\geq 0}$ are scaling factors. In the search of the lowest-cost mapping, the labels of the cost functions that are derived from the back-off functions (e.g. $\gamma_{\textit{ent}}$, $\gamma_{\textit{pred}}$) are memorized for the pairs (t_1, t_2) for which they were defined and for which they outputted a cost. These labels are then used as back-off rule state names when constructing rules.

5.2 Tree Mapping: Optimization Problem

Intuitively, the cost of mapping a source node n_s to a target node n_t is equal to the cost of transforming a tree fragment $T_\Sigma(\mathcal{X})$ rooted at node n_s into a tree fragment $T_\Delta(\mathcal{Q}.\mathcal{X})$ rooted at node n_t, plus the sum of costs of mapping the frontier nodes rooted at the variables. In order to formalize our tree mapping, we need a more precise definition of a tree fragment where the locations of variables $\mathcal{X}$ are specified by paths. The notation to specify subtrees is taken from (Graehl and Knight, 2004), and we introduce the $\perp$ operator for convenience.

A path p is a tuple, equivalent to a Gorn address, that uniquely identifies the node of a tree by specifying the sequence of child indices to the node from the root. In the tree s of Figure 1, the path to the VP node is $(1, 0)$, whereas in t, the path to `League` is $(1, 1, 0)$. The path $p = ()$ refers to the root of a tree. We denote by $s \downarrow p$ the subtree of tree s that is rooted at path p and that has no variables. In Figure 1, the left-hand-side (*lhs*) of $r5$ is the subtree $s \downarrow (1, 0, 1, 1)$. In order to introduce variables, we generalize the notion of subtree into a tree pattern $s \downarrow p \perp \{p_1, \ldots, p_n\}$, where n variables replace subtrees $s \downarrow p_i$ at subpaths $p_i \in \{p_1, \ldots, p_n\}$. For example, the *lhs* of $r1$ can be represented with the tree pattern $s \downarrow () \perp \{(0, 1), (1)\}$, and $r2$ with $s \downarrow (1) \perp \{(1, 0, 1)\}$. Note that the order of subpaths $\{p_1, \ldots, p_n\}$ matters. A tree pattern with no subpaths $s \downarrow p \perp \{\}$ is simply a subtree $s \downarrow p$, such as the *lhs* of rules $r4$ and $r5$; a tree pattern with only one subpath equal to its path $s \downarrow p \perp \{p\}$ is a single variable, such as the *rhs* of rules $r2$ and $r3$. Note that in $s \downarrow p \perp \{p_1, \ldots, p_n\}$, all paths p_i to variables are prefixed by p, and that no variables are descendants of any other variable in the same tree pattern. In other words, $\mathbf{p} = \{p_1, \ldots, p_n\}$ are *disjoint subpaths* given p, where $\mathbf{p}$ denotes a list of paths.

We can now formalize the tree mapping algorithm as an optimization problem. Let $\gamma\left(s \downarrow p_s \perp \mathbf{p}, t \downarrow p_t \perp \mathbf{p}'\right)$ be the cost to transform a source into a target tree pattern, as defined in Equation 1. To transform $s \downarrow p_s$ into $t \downarrow p_t$, we need to find the best combination of source subtrees rooted at $\{p_1, \ldots, p_n\}$ that can be transformed at minimum cost to the best combination of target subtrees at $\{p'_1, \ldots, p'_n\}$. The transformation cost of a certain tree pattern $s \downarrow p_s \perp \{p_1, \ldots, p_n\}$ into $t \downarrow p_t \perp \{p'_1, \ldots, p'_n\}$ is equal to the cost of transforming the source tree pattern into the target tree pattern, plus the minimum cost to transform $s \downarrow p_i$ into $t \downarrow p'_i$, for $i \geq 1$. That is:

$$\begin{aligned}& \mathrm{C}\left(s \downarrow p_s, t \downarrow p_t\right) = \\ & \min_{\mathbf{p}, \mathbf{p}'} \{ \gamma\left(s \downarrow p_s \perp \mathbf{p}, t \downarrow p_t \perp \mathbf{p}'\right) + \\ & \qquad \sum_{i=1}^{|\mathbf{p}|} \mathrm{C}\left(s \downarrow p_i, t \downarrow p'_i\right) \}\end{aligned} \tag{2}$$

subject to $|\mathbf{p}| = |\mathbf{p}'|$, that is, source and target tree patterns having the same number of variables. Then, the cost of transforming the source into the target tree would be given by the expression

$\mathrm{C}\,(s \downarrow (), t \downarrow ())$. Since we are only interested in the pairs of source and target tree patterns that lead to the minimum cost, we keep track of subpaths $\mathbf{p}$ and $\mathbf{p}'$ of tree pattern pairs that minimize the cost.

5.3 Algorithm

5.3.1 Overview

This problem can be solved for small depths of tree patterns and a small number of variables by storing intermediate results in the computation of Eq. 2. However, an exact implementation needs to enumerate all pairs of source and target disjoint subpaths ($\mathbf{p}$ and $\mathbf{p}'$), which has a computational complexity that grows combinatorially with $|\mathbf{p}|$ (variable permutations), and exponentially with the number of descendant nodes of p_s and p_t (powerset of variables).

Instead, we use a beam search algorithm (see Algorithm 1)[7] that constructs source and target disjoint paths ($\mathbf{p}$ and $\mathbf{p}'$) hierarchically (function GENERATEDISJOINT) in a bottom-up order, for any given path pair (p_s, p_t). First, n-best solutions (pairs of disjoint paths) are computed for children; then those partial solutions are combined into their parent using the cross-product. Solutions (with their associated cost) for every pair of paths (p_s, p_t) are stored in a weighted hypergraph, from which we can extract n-best derivations (sequences of rules). In the pseudocode, we use a helper function, $\mathrm{paths}(s \downarrow p_s)$, which denotes the list of subtree paths in bottom-up order: from the leaves up to p_s (including the latter).

5.3.2 Detailed Description

For a certain path pair (p_s, p_t), there are three cases. The first case (line 34-35) considers a pair of empty disjoint subpaths $(\mathbf{p}, \mathbf{p}') = (\{\}, \{\})$, where the cost c of transforming $s \downarrow p_s \perp \{\}$ into $t \downarrow p_t \perp \{\}$ is evaluated and the empty disjoint subpaths are added to the priority queue P, indexed with p_s. Such indexing is useful to retrieve the n-best pairs of disjoint subpaths accumulated at every tree node.

The second case (line 28 to 31) evaluates the cost of transforming single-variable tree patterns: $s \downarrow p_s \perp \{p_c\}$ into $t \downarrow p_t \perp \{p'_c\}$. In this case, variables substitute entire subtrees rooted at paths p_c and p'_c on the source and target tree patterns, respectively. Note that p_c ranges over all node addresses that are descendant of p_s (including p_s), and similarly for p'_c. The pairs of disjoint subpaths $(\mathbf{p}, \mathbf{p}') = (\{p_c\}, \{p'_c\})$ are added into the priority queue, indexed by p_c.

The third case (line 16 to 26) performs the combination of subpaths hierarchically from children to parents, and incrementally across children. For every path $p_c \in \mathrm{paths}(s \downarrow p_s)$, it visits its imme-

Algorithm 1 Extraction of optimal sequence of rules to transform a source s into a target tree t.

```
Input: Trees $s$ and $t$, and ensemble of cost functions $\gamma$.
Output: Sequence of optimal rules for $s \Rightarrow^* t$.
 1: let $H = (V, E)$ be a hypergraph of solutions with
    $V \leftarrow \{\}$ vertices and $E \leftarrow \{\}$ hyperedges.
 2: for $(p_s, p_t) \in \mathrm{paths}(s) \times \mathrm{paths}(t)$ do
 3:   add vertex $v = (p_s, p_t)$ to $V$
 4:   $PP \leftarrow$ GENERATEDISJOINT($s \downarrow p_s, t \downarrow p_t, \gamma$)
 5:   for $(\mathbf{p}, \mathbf{p}') \in PP$ do
 6:     ▷ Get cost of tree pattern pair.
 7:     $c \leftarrow \gamma\,(s \downarrow p_s \perp \mathbf{p}, t \downarrow p_t \perp \mathbf{p}')$
 8:     add edge $(p_s, p_t) \overset{c}{\rightarrow} (\mathbf{p}, \mathbf{p}')$ to $E$
 9:   end for
10: end for
11: return HYPERGRAPHSEARCH($H$)

12: function GENERATEDISJOINT($s \downarrow p_s, t \downarrow p_t, \gamma$)
13:   $P \leftarrow \{\}$ a priority queue of partial disjoint paths.
14:   for every $p_c \in \mathrm{paths}(s \downarrow p_s)$ do
15:     ▷ Costs when variables combined from children.
16:     for every $p_{ic}$ immediate child of $p_c$ (if any) do
17:       ▷ Retrieve $n$-best subpaths $\mathbf{p}$ and $\mathbf{p}'$ from $p_{ic}$.
18:       $C \leftarrow \arg\min^n_{(\mathbf{p},\mathbf{p}')}\{c \mid (p_{ic}, \mathbf{p}, \mathbf{p}', c) \in P\}$
19:       ▷ Combine subpaths with those accumulated
20:       ▷ from previous siblings and stored at path $p_c$.
21:       $A \leftarrow \arg\min^n_{(\mathbf{p},\mathbf{p}')}\{c \mid (p_c, \mathbf{p}, \mathbf{p}', c) \in P\}$
22:       for $(\mathbf{p}, \mathbf{p}') \in (C \cup (C.A))$ do
23:         $c \leftarrow \gamma\,(s \downarrow p_s \perp \mathbf{p}, t \downarrow p_t \perp \mathbf{p}')$
24:         add $(p_c, \mathbf{p}, \mathbf{p}', c)$ to priority queue $P$
25:       end for
26:     end for
27:     ▷ Cost of tree patterns with one variable.
28:     for every $p'_c \in \mathrm{paths}(t \downarrow p_t)$ do
29:       $c \leftarrow \gamma\,(s \downarrow p_s \perp \{p_c\}, t \downarrow p_t \perp \{p'_c\})$
30:       add $(p_c, \{p_c\}, \{p'_c\}, c)$ to priority queue $P$
31:     end for
32:   end for
33:   ▷ Cost of tree patterns with no variables.
34:   $c \leftarrow \gamma\,(s \downarrow p_s \perp \{\}, t \downarrow p_t \perp \{\})$
35:   add $(p_s, \{\}, \{\}, c)$ to priority queue $P$
36:   return $\arg\min^n_{(\mathbf{p},\mathbf{p}')}\{c \mid (p_s, \mathbf{p}, \mathbf{p}', c) \in P\}$
37: end function
```

[7] https://github.com/pasmargo/t2t-qa

diate children p_{ic} one by one, and retrieves into C the n-best disjoint subpaths (line 18) that have already been obtained during previous iterations for p_{ic}. Then, we retrieve into A the n-best disjoint subpaths indexed at p_c, which is a list of the best subpaths that were combined from previous immediate children (the list is empty if this is the first immediate child that we visit). The cross-product of disjoint subpaths in C and A, that is $C.A$, is then evaluated and the best combinations are stored in the priority queue indexed at p_c.

As an example of a cross-product between two lists C and A of pairs of disjoint paths, let $C = \{(\mathbf{p_1}, \mathbf{p_1}'), (\mathbf{p_2}, \mathbf{p_2}')\}$ and $A = \{(\mathbf{p_3}, \mathbf{p_3}')\}$. Then the cross-product $C.A$ would be:

$$C.A = \{(\mathbf{p}_1 \cdot \mathbf{p}_3, \mathbf{p}_1' \cdot \mathbf{p}_3'), (\mathbf{p}_2 \cdot \mathbf{p}_3, \mathbf{p}_2' \cdot \mathbf{p}_3')\}$$

where $\mathbf{p}_1 \cdot \mathbf{p}_3 = \{(0,1),(0,2),(0,3),(0,4)\}$ if $\mathbf{p}_1 = \{(0,1),(0,2)\}$ and $\mathbf{p}_3 = \{(0,3),(0,4)\}$. At this stage, subpaths $\mathbf{p}_i$ or $\mathbf{p}_i'$ that are not disjoint are discarded, together with disjoint paths that produce tree patterns with depth larger than a certain user-defined threshold, or whose number of subpaths is larger than the number of variables allowed.

In line 24, the disjoint subpaths of C (in addition to their cross-product $C.A$) are also evaluated and added to the priority queue indexed by p_c, to propagate upwards in the hierarchy of solutions the decision of not combining disjoint subpaths.

Finally, GENERATEDISJOINT returns the n-best pairs of disjoint subpaths of minimum cost ($\mathbf{p}$, $\mathbf{p}'$) that accumulated in the priority queue P for path p_s.

5.3.3 Other Considerations

The n-best source and target pairs of disjoint subpaths are stored at every pair of source and target paths (p_s, p_t) (lines 2-10), forming a hypergraph, as in Figure 2. Then, with a hypergraph search (Huang and Chiang, 2007) we can retrieve at least n-best sequences of rules (derivations) that transform the source into the target tree (line 11).

To maintain diversity of partial disjoint subpaths, we divide P into a matrix of buckets with as many rows and columns as the number of non-variable terminals of the source and target tree patterns, trading memory for more effective search (Koehn, 2015). This operation is implicit in lines 24, 30 and 35.

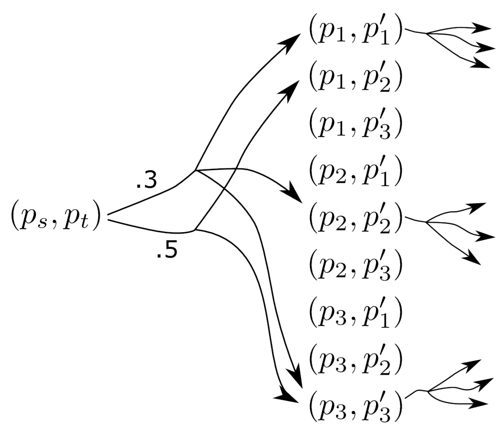


Figure 2: Hypergraph with 2-best pairs of disjoint subpaths for (p_s, p_t). Vertices are pairs of source and target paths. Hyperedges are pairs of tree patterns. The hyperedge with cost .3 denotes the pair $s \downarrow p_s \perp \{p_1, p_2, p_3\}$ and $t \downarrow p_t \perp \{p_1', p_2', p_3'\}$. The one with cost .5, $s \downarrow p_s \perp \{p_1, p_3\}$ and $t \downarrow p_t \perp \{p_2', p_3'\}$.

6 Experiments

6.1 Experiment Settings

Data The training data is a corpus of questions annotated with their logical forms that can be executed on Freebase to obtain a precise answer. For an unseen set of questions, the task is to obtain automatically their logical forms and retrieve the correct answer. Our objective is to evaluate the generalization capabilities of a transducer induced using our rule extraction on an unseen open-domain test set. We parsed questions from FREE917 into source constituent trees using the Stanford caseless model (Klein and Manning, 2003). Target constituent (meaning) representations were obtained by a simple heuristic conversion from the $\lambda-$DCS expressions released by Berant et al. (2013). We evaluate on the same training and testing split as in Berant et al. (2013). Tree pairs (2.9%) for which the gold executable meaning representation did not retrieve valid results were filtered out.

Baselines We compared to two baselines. The first one is SEMPRE (Berant et al., 2013), a state-of-the-art semantic parser that uses a target language grammar to over-generate trees, and a log-linear model to estimate the parameters that guide the decoder towards trees that generate correct answers. For FREE917, SEMPRE uses a manually-created entity lexicon released by (Cai and Yates, 2013), but an automatically generated predicate lexicon. Instead, our system and the second baseline use manually created entity *and* predicate lexicons, where the latter was created by selecting all words from every question that relate to the target predicate. For example, for the question "what olympics has egypt participated in", we created an entry that maps the discontinuous phrase "olympics participated in" to the predicate `OlympicsParticipatedIn`.

The second baseline is a tree-to-tree transducer whose rules are extracted using a straightforward adaptation of the GHKM algorithm (Galley et al., 2004) for pairs of trees. Word-to-concept alignments are extracted using three different strategies: i) **ghkm-g** uses the IBM models (Brown et al., 1993) as implemented in GIZA++ (Och and Ney, 2003), ii) **ghkm-m** maps KB concepts (target leaves) to as many source words as present in the entity/predicate lexicons, and iii) **ghkm-c** maps KB concepts as in ii) but only retaining the longest contiguous sequence of source words (or right-most sequence if there is a tie). Bridging predicates are assumed when a KB concept does not align (according to the lexicon) to any source word. Finally, rule state names are set according to the mechanism described in Section 5.

Our *ent*, *pred* and *bridge* cost/back-off functions assign a low cost (or high score) to source and target tree patterns with no variables whose leaves appear in either the entity or the predicate lexicons. Scaling factors λ_i (see Eq. 1) were subjectively tuned on 20 training examples. When used as back-off functions, they generate as many *rhs* as entities or predicates can be retrieved from the lexicons by at least one of the words in the source tree pattern. Bridging predicates are dispreferred by adding an extra constant cost. At back-off, this score function generates a variable predicate, acting as a wildcard in Sparql.

Our system t2t For the rule extraction, we use a beam size of 10, and we output 100 derivations for every tree pair. We do not impose any limit in the depth of *lhs* or *rhs*, or in the number of variables. To increase the coverage of our rules, we produce deleting tree transducers by replacing fully lexicalized branches that are directly attached to the root of a *lhs* with a deleting variable.

For the parameter estimation, we used 3 iterations of the latent variable averaged structured perceptron, where the number of iterations was selected on 20% of held-out training data. To assess the equality between the gold and the decoded tree, we compare their denotations. The features for the discriminative training were the *lhs* and *rhs* roots, the number of variables, deleting variables and leaves, the presence of entities or predicates in the *rhs*, the rule state and children states, and several measures of character overlaps between the leaves of the source and information associated to leaves in target tree patterns.

For decoding, we used standard techniques (Graehl and Knight, 2004) to constrain and prune weighted regular tree grammars given a tree transducer and a source tree, and used the cube-growing algorithm to generate $10,000$ derivations, converted them to Sparql queries, and retained those that were valid (either syntactically correct or that retrieved any result). We compute the accuracy of the system as the percentage of questions for which the 1-best output tree retrieves the correct answer, and the coverage as the percentage for which the correct answer is within the $10,000$ best derivations. The average rule extraction time per tree pair when using beam size 1 was 0.46 seconds (median 0.35, maximum 2.94 seconds). When using beam size 10, the average was 4.7 seconds (median 2.02, maximum 104.4 seconds), which gives us a glimpse of how the beam size influences the computational complexity for the typical tree size of FREE917 questions.

6.2 Results

Results are in Table 1. Note that although we compare our results to those obtained with SEMPRE (Berant et al., 2013), the systems cannot really be compared since Berant et al. (2013) did not have access to a manually created lexicon of predicates. When comparing the average number of entity and predicate rules that the back-off functions generate, we see that the number of predicate rules is much larger, implying a higher ambiguity. Despite this, our base system still produces promising results in terms of accuracy and coverage.

We also carried out several ablation experiments to investigate what are the characteristics of the system that contribute the most to the accuracy: In **no nbest**, we only extract one sequence of rules that transform a source into a target tree. In **no del**, we do not introduce deleting variables. In **beam 1**, we use beam size 1 in rule extraction. In **no size**, no tree size regularizer cost function is used. And in

Systems	Acc.	Cov.	# Preds.	# Ents.	# Rules
SEMPRE	.62	–	–	–	–
ghkm-c	.49	.80	155	14	384
ghkm-m	.48	.77	147	14	399
ghkm-g	.08	.57	102	5	135
t2t	.64	.78	187	19	437
t2t-e	.69	.85	191	20	430
no del	.64	.78	187	19	437
no size	.59	.78	195	19	483
no nbest	.58	.70	93	5	128
beam 1	.53	.65	84	5	112
no back	.00	.01	0	0	175
train-600	.62	.78	187	19	429
train-500	.61	.77	184	19	413
train-400	.62	.75	178	19	390
train-300	.59	.75	177	18	363
train-200	.52	.74	169	17	317
train-100	.52	.67	138	14	317

Table 1: Accuracy and coverage results; average number of predicate rules, entity rules and all rules per input tree.

no back, no rule back-offs are used. As we see, removing the back-off rule capabilities is critical in this setting and makes the QA task unfeasible. We also studied the impact of the size of the training data in the generalization of our system, by training the system in $\{100, 200, \ldots, 600\}$ examples. We found that the accuracy saturates at only 400 training instances, which might be advantageous in tasks where training resources are scarce. Finally, in order to estimate the upper bound in the coverage and accuracy of our approach on FREE917, we also run our pipeline **t2t-e** with a refined version of Cai and Yates (2013)'s entity lexicon, where 65 missing entities are added (7.8% of the total). We can observe a significant increase in the accuracy and coverage of the system, suggesting that the bottleneck may lie in the entity/predicate linking procedures.

7 Future Work

One step further in the generalization of the rule extraction is to remove the necessity of explicitly providing cost functions such as word-to-word hard-alignments or costs between tree fragments. This would allow us to remove the bias introduced by engineered cost functions and to obtain rules that are globally optimal. In this setup, the parameters of the cost functions are to be learned with the objective to maximize the likelihood on the training data or a downstream application performance. However, since rules (or tree mappings) would become hidden variables, this generalized rule extraction may require faster methods to enumerate plausible rules. Another extension would be to make the rule extraction more robust against parsing errors, using pairs of forests instead of pairs of trees, similarly as in Liu et al. (2009).

Regarding the QA application, there are two natural extensions that we want to address, namely to develop general and automatic entity and predicate linking mechanisms for large knowledge bases, and to test our approach in datasets that require higher levels of compositionality such as the QALD challenges (Unger et al., 2015) or those datasets produced by Wang et al. (2015).

8 Conclusion

We proposed to induce tree to tree transducers using a rule extraction algorithm that uses cost functions over pairs of tree fragments (instead of word-alignments), which increases the applicability of these models. Some cost functions may act as rule back-offs, generating new *rhs* given unseen *lhs*, thus producing transducer rules "on-the-fly". The scores of these rules are obtained on demand using a discriminative training procedure that estimates weights for rule features. This strategy was useful to compensate the lack of rule coverage when inducing tree transducers from small tree corpora.

As a proof-of-concept, we tested the tree transducer induced with our algorithm on a QA task over a large KB, a domain in which tree transducers have not been effective before. In this task, lexicon mappings were naturally introduced as cost functions and rule back-offs, without loss of generality. Despite using a manually created lexicon of predicates, we showed a high accuracy and coverage of non-final rules, which are promising results.

Acknowledgments

This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO), and is also supported by JSPS KAKENHI Grant Number 16K16111. We thank Yoshimasa Tsuruoka, Yo Ehara and the anonymous reviewers for their helpful comments.

References

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA, October. Association for Computational Linguistics.

Philip Bille. 2005. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1–3):217 – 239.

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

Qingqing Cai and Alexander Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 423–433, Sofia, Bulgaria, August. Association for Computational Linguistics.

Trevor Anthony Cohn and Mirella Lapata. 2009. Sentence compression as tree transduction. *Journal of Artificial Intelligence Research*, 34:637–674.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 2*, ACL '03, pages 205–208, Stroudsburg, PA, USA. Association for Computational Linguistics.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule. In *HLT-NAACL'2004: Main Proceedings*, pages 273–280.

Ruifang Ge and Raymond J. Mooney. 2005. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, CONLL '05, pages 9–16, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jonathan Graehl and Kevin Knight. 2004. Training tree transducers. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 105–112, Boston, Massachusetts, USA, May 2 - May 7. Association for Computational Linguistics.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151, Prague, Czech Republic, June. Association for Computational Linguistics.

Bevan Keeley Jones, Mark Johnson, and Sharon Goldwater. 2012. Semantic parsing with bayesian tree transducers. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 488–496, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July. Association for Computational Linguistics.

Kevin Knight and Jonathan Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 3406 of *Lecture Notes in Computer Science*, pages 1–24. Springer Berlin Heidelberg.

Philipp Koehn. 2015. *Moses Manual*.

Peng Li, Yang Liu, and Maosong Sun. 2013. An extended GHKM algorithm for inducing Lambda-SCFG. pages 605–611.

Percy Liang. 2013. Lambda dependency-based compositional semantics. *CoRR*, abs/1309.4408.

Xiao Ling, Sameer Singh, and Daniel Weld. 2015. Design challenges for entity linking. *Transactions of the Association for Computational Linguistics*, 3:315–328.

Yang Liu, Yajuan Lü, and Qun Liu. 2009. Improving tree-to-tree translation with packed forests. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 558–566, Suntec, Singapore, August. Association for Computational Linguistics.

Andreas Maletti, Jonathan Graehl, Mark Hopkins, and Kevin Knight. 2009. The power of extended top-down tree transducers. *SIAM Journal on Computing*, 39(2):410–430.

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

William C. Rounds. 1970. Mappings and grammars on trees. *Mathematical systems theory*, 4(3):257–287.

Kuo-Chung Tai. 1979. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, July.

James W. Thatcher. 1970. Generalized sequential machine maps. *Journal of Computer and System Sciences*, 4(4):339 – 367.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. 2005. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, volume 6, pages 1453–1484.

Christina Unger, Corina Forascu, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. 2015. Question Answering over Linked Data (QALD-5). In Linda Cappellato, Nicola Ferro, Gareth Jones, and Eric San Juan, editors, *Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum*, volume 1391. Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China, July. Association for Computational Linguistics.

Yuk Wah Wong and Raymond J. Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 439–446, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dekai Wu. 2005. Recognizing paraphrases and textual entailment using inversion transduction grammars. In *Proceedings of the ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*, EMSEE '05, pages 25–30, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262.