Tree Edit Models for Recognizing Textual Entailments, Paraphrases, and Answers to Questions

Michael Heilman Noah A. Smith
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{mheilman,nasmith}@cs.cmu.edu

Abstract

We describe tree edit models for representing sequences of tree transformations involving complex reordering phenomena and demonstrate that they offer a simple, intuitive, and effective method for modeling pairs of semantically related sentences. To efficiently extract sequences of edits, we employ a tree kernel as a heuristic in a greedy search routine. We describe a logistic regression model that uses 33 syntactic features of edit sequences to classify the sentence pairs. The approach leads to competitive performance in recognizing textual entailment, paraphrase identification, and answer selection for question answering.

1 Introduction

Many NLP tasks involve modeling relations between pairs of sentences or short texts in the same language. Examples include recognizing textual entailment, paraphrase identification, and question answering. Generic approaches are, of course, desirable; we believe such approaches are also feasible because these tasks exhibit some similar semantic relationships between sentences.

A popular method for such tasks is Tree Edit Distance (TED), which models sentence pairs by finding a low or minimal cost sequence of editing operations to transform a tree representation of one sentence (e.g., a dependency or phrase structure parse tree) into a tree for the other. Unlike grammar-based models and shallow-feature discriminative approaches, TED provides an intuitive story for tree pairs where one tree is derived from the other by a sequence of simple transformations.

The available operations in standard TED are the following: insertion of a node, relabeling (i.e., renaming) of a node, and deletion (i.e., removal) of a node. While the restriction to these three operations permits efficient dynamic programming solutions for finding a minimum-cost edit sequence (Klein, 1989; Zhang and Shasha, 1989), certain interesting and prevalent phenomena involving reordering and movement cannot be elegantly captured. For example, consider the following sentence pair, which is a simplified version of a true entailment (i.e., the premise entails the hypothesis) in the development data for the RTE-3 task.

Premise: *Pierce built the home for his daughter off Rossville Blvd, as he lives nearby.*

Hypothesis: *Pierce lives near Rossville Blvd.*

In a plausible dependency tree representation of the premise, *live* and *Rossville Blvd* would be in separate subtrees under *built*. In the hypothesis tree, however, the corresponding nodes would be in a grandparent-child relationship as part of the same phrase, *lives near Rossville Blvd*. In general, one would expect that short transformation sequences to provide good evidence of true entailments. However, to account for the grandparent-child relationship in the hypothesis, TED would produce a fairly long sequence, relabeling *nearby* to be *near*, deleting the two nodes for *Rossville Blvd*, and then reinserting those nodes under *near*.

We describe a tree edit approach that allows for more effective modeling of such complex reordering phenomena. Our approach can find a shorter and more intuitive edit sequence, relabeling *nearby* to be *near*, and then moving the whole subtree *Rossville Blvd* to be a child of *near*, as shown in Figure 1.

A model should also be able to consider characteristics of the tree edit sequence other than its overall length (e.g., how many proper nouns were deleted). Using a classifier with a small number of syntactic

Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL, pages 1011–1019,
Los Angeles, California, June 2010.

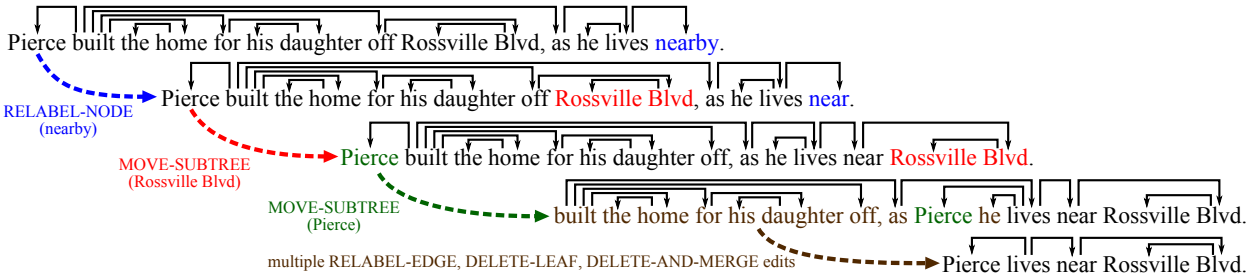

Figure 1: A tree edit sequence transforming a premise to an entailed hypothesis. Dependency types and parts of speech are omitted for clarity.

features, our approach allows us to learn—from labeled examples—how different types of edits should affect the model's decisions (e.g., about whether two sentences are paraphrases).

The structure of this paper is as follows. §2 introduces our model and describes the edit operations that were implemented for our experiments. §3 details the search-based procedure for extracting edit sequences for pairs of sentences. §4 describes the classifier for sentence pairs based on features of their corresponding edit sequences. §5 describes and presents the results of experiments involving recognizing textual entailment (Giampiccolo et al., 2007), paraphrase identification (Dolan et al., 2004), and an answer selection task for question answering (Wang et al., 2007). §6 addresses related work, and §7 provides concluding remarks.

2 Extended Tree Edit Sequences

This section defines a tree edit sequence and describes the operations used in our experiments.

We begin with some conventions. We use dependency trees as the structure upon which the tree edits will operate. The child nodes for a given parent are represented in a head-outward fashion such that the left and right children are separate lists, with the left- and right-most elements as the last members of their respective lists, as in most generative dependency models (Eisner, 1996). Each node consists of a lemmatized word token as its main label (hereafter, lemma), a part of speech tag (POS), and a syntactic relation label for the edge to its parent. We assume the root node has a special dummy edge label `ROOT`.

Let T_c be a "current tree" that is being transformed and let T_t be a "target tree" into which T_c will ultimately be transformed. Let $T(i)$ be a node with an index i into the tree T, where the indices are arbitrary (e.g., they could be word positions).

2.1 Definition

We define a tree edit sequence to be a series of edit operations that transform a source tree (the initial T_c) into a target tree T_t.[1] While TED permits only insert, relabel, and delete operations, edit sequences may contain more complex operations, such as moving entire subtrees and re-ordering child nodes.

2.2 Implemented Operations

For our experiments, we used the types of edit operations listed in Table 1.[2] The first six operations are straightforward extensions of the insert, relabel and delete operations allowed in TED. The final three operations, `MOVE-SUBTREE`, `NEW-ROOT`, and `MOVE-SIBLING`, enable succinct edit sequences for complex transformations. For a given current tree, there may be many instantiations of each operation (e.g., `DELETE-LEAF` could be invoked to delete any of a number of leaf nodes). Note that any tree can be transformed into any other simply by deleting all nodes from the one tree and inserting all the nodes in the other. However, our set of tree edit operations permits more concise and intuitive edit sequences.

3 Searching for Tree Edit Sequences

To model sentence pairs effectively, we seek a short sequence of tree edits that transforms one tree into another. The space of possible edit sequences, as with TED and many other methods involving trees,

[1] Such a sequence is sometimes called a "script" for TED.

[2] We leave for future work the exploration of other operations (e.g., swapping parent and child nodes).

Operation	Arguments	Description
INSERT-CHILD	node index j, new lemma l, POS p, edge label e, side $s \in \{left, right\}$	Insert a node with lemma l, POS p, and edge label e as the last child (i.e., farthest from parent) on side s of $T(j)$.
INSERT-PARENT	non-root node index j, new lemma l, new POS p, edge label e, side $s \in \{left, right\}$	Create a node with lemma l, POS p, and edge label e. Make $T(j)$ a child of the new node on side s. Insert the new node as a child of the former parent of $T(j)$ in the same position.
DELETE-LEAF	leaf node index j	Remove the leaf node $T(j)$.
DELETE-&-MERGE	node index j (s.t. $T(j)$ has exactly 1 child)	Remove $T(j)$. Insert its child as a child of $T(j)$'s former parent in the same position.
RELABEL-NODE	node index j, new lemma l, new POS p	Set the lemma of $T(j)$ to be l and its POS to be p.
RELABEL-EDGE	node index j, new edge label e	Set the edge label of $T(j)$ to be e.
MOVE-SUBTREE	node index j, node index k (s.t. $T(k)$ is not a descendant of $T(j)$), side $s \in \{left, right\}$	Move $T(j)$ to be the last child on the s side of $T(k)$.
NEW-ROOT	non-root node index j, side $s \in \{left, right\}$	Make $T(j)$ the new root node of the tree. Insert the former root as the last child on the s side of $T(j)$.
MOVE-SIBLING	non-root node index j, side $s \in \{left, right\}$, position $r \in \{first, last\}$	Move $T(j)$ to be the r child on the s side of its parent.

Table 1: Possible operations in our extended tree edit implementation. All are described as operations to tree T.

is exponentially large in the size of the trees. However, while dynamic programming solutions exist for TED (Klein, 1989; Zhang and Shasha, 1989), it is unlikely that such efficient algorithms are available for our problem because of the lack of locality restrictions on edit operations.[3]

3.1 Algorithm for Extracting Sequences

Rather than dynamic programming, we use greedy best-first search (Pearl, 1984) to efficiently find sensible (if not minimal) edit sequences. The distinguishing characteristic of greedy best-first search is that its function for evaluating search states is simply a heuristic function that estimates the remaining cost, rather than a heuristic function plus the cost so far (e.g., number of edits), as in other types of search.

Here, the initial search state is the source tree, the current state is T_c, and the goal state is T_t. The function for generating the successors for a given state returns returns trees for all possible specifications of operations on T_c (§2.2), subject to the minimal constraints to be described in §3.3. The enumeration order of the edits in the search procedure (i.e., the order in which states are explored) follows the order of their presentation in Table 1. In preliminary experiments, varying this order had no effect on the extracted transformations.

3.2 Tree Kernel Heuristic

In our greedy search approach, the evaluation function's value for a state depends only on the heuristic function's estimate of how different the current tree at that state is from the target tree. Using this function, at each step, the search routine chooses the next state (i.e., edit) so as to minimize the difference between the current and target trees.

We use a tree kernel to define the heuristic function. A kernel is a special kind of symmetric function from a pair of objects to a real number. It can be interpreted as the inner product of those objects represented in some real-valued feature space (Schölkopf and Smola, 2001). A tree kernel, as proposed by Collins and Duffy (2001), is a convolution kernel[4] whose input is a pair of trees and whose output is a positive number indicating the similarity of the sets of all their subtrees.

The dimensionality of the feature vector associated with a tree kernel is thus unbounded in general, and larger trees generally lead to larger kernel values. Direct use as a search heuristic would lead to the exploration of states for larger and larger trees, even ones larger than the target tree. Thus, as in

[3] Gildea (2003) proposes a dynamic programming algorithm for a related tree alignment problem, but it is still exponential in the maximum number of children for a node.

[4] Haussler (1999) provides a proof, which can be extended for our kernel, that tree kernels are valid kernel functions.

Equation 1, the search heuristic H "normalizes" the kernel K of the current tree T_c and target tree T_t to unit range by dividing by the geometric mean of the kernels comparing the individual trees to themselves.[5] Also, the normalized value is subtracted from 1 so as to make it a difference rather than a similarity. The search routine will thus reach the goal state when the heuristic reaches 0, indicating that the current and target trees are identical.

$$H(T_c) = 1 - \frac{K(T_c, T_t)}{\sqrt{K(T_c, T_c) \times K(T_t, T_t)}} \quad (1)$$

Kernels are most commonly used in the efficient construction of margin-based classifiers in the implied representation space (e.g., Zelenko et al., 2003). Here, however, the kernel helps to find a representation (i.e., an edit sequence) for subsequent modeling steps.

We are effectively mapping the source, current, and target trees to points on the surface of a high-dimensional unit sphere associated with the normalized kernel. In this geometric interpretation, the search heuristic in Equation 1 leads the search algorithm to explore reachable trees along the surface of this sphere, always choosing the one whose angle with the target tree is smallest, until the angle is 0. The path on the sphere corresponds to an edit sequence, from which we will derive edit features in §4 for classification.

Our kernel is based on the partial tree kernel (PTK) proposed by Moschitti (2006). It considers matches between ordered subsequences of children in addition to the full sequences of children as in Collins and Duffy (2001). This permits a very fine-grained measure of tree pair similarity. Importantly, if two nodes differ only by the presence or position of a single child, they will still lead to a large kernel function value. We also sum over the similarities between all pairs of nodes, similar to (Collins and Duffy, 2001).

Since the PTK considers non-contiguous subsequences, it is very computationally expensive. We therefore restrict our kernel to consider only contiguous subsequences, as in the contiguous tree kernel (CTK) (Zelenko et al., 2003).

[5]This normalized function is also guaranteed to be a kernel function (Schölkopf and Smola, 2001).

To define our kernel, we begin with a similarity function for pairs of nodes n_1 and n_2 that depends on their lemmas, POS tags, edge labels, and sides with respect to their parents:[6]

$$s(n_1, n_2) = \delta(l(n_1), l(n_2)) \times \sum_{f \in \{l,e,p,s\}} \delta(f(n_1), f(n_2)) \quad (2)$$

where δ returns 1 if its arguments are equivalent, 0 otherwise. l, e, p, and s are used here as functions to select the lemma, edge label, POS, and side of a node. Equation 2 encodes the linguistic intuition that the primary indicator of node similarity should be a lexical match between lemmas. If the lemmas match, then edge labels, POS, and the locations (sides) relative to their parents are also considered.

The kernel is defined recursively (starting from the roots), where n_i is a node in the set of nodes N_{T_i} in tree T_i:

$$K(T_1, T_2) = \sum_{n_1 \in \{N_{T_1}\}} \sum_{n_2 \in \{N_{T_2}\}} \Delta(n_1, n_2) \quad (3)$$

$$\Delta(n_1, n_2) = \mu \left(\lambda^2 s(n_1, n_2) + \sum_{J_1, J_2, |J_1| = |J_2|} \prod_{i=1}^{l(J_1)} \Delta(c_{n_1}[J_{1i}], c_{n_2}[J_{2i}]) \right) \quad (4)$$

$J_1 = \langle J_{11}, J_{12}, J_{13}, \ldots \rangle$ is an index sequence associated with any *contiguous* ordered sequence of children c_{n_1} of node n_1 (likewise for J_2). J_{1i} and J_{2i} point to the ith children in the two sequences. $|\cdot|$ returns the length of a sequence.

The kernel includes two decay factors: λ for the length of child subsequences, as in Zelenko et al. (2003) and Moschitti (2006); and μ for the height of the subtree, as in Collins and Duffy (2001) and Moschitti (2006). We set both to 0.25 in our experiments to encourage the search to consider edits leading to smaller matches (e.g., of individual parent-child dependencies) before larger ones.[7]

[6]The side of a node relative to its parent in a dependency tree is important: two parent nodes with the same children should not be considered exact matches if children are on different sides (e.g., *defeated the insurgents* and *the insurgents defeated*).

[7]From experiments with the paraphrase training set (§5.2), performance does not appear sensitive to the decay parameters. Settings of 0.1, 0.2, 0.3, and 0.4 led to 10-fold cross-validation

The main difference between our kernel and the CTK is that we sum over all pairs of subtrees (Equation 3). In contrast, the CTK only considers only one pair of subtrees. When the CTK is applied to relation extraction by Culotta and Sorensen (2004), each subtree is the smallest common subtree that includes the entities between which a relation may exist (e.g., the subtree for *Texas-based energy company Exxon Mobil* when extracting ORGANIZATION−LOCATION relations).

3.3 Constraints on the Search Space

For computational efficiency, we impose the following three constraints to simplify the search space. Note that the first two simply prune away obviously unhelpful search states.

1. For INSERT−CHILD, INSERT−PARENT, and RELABEL−NODE edits, the lemma and POS of the node to insert must occur in the target tree. Also, the pair consisting of the lemma for the node to insert and the lemma for its prospective parent must not appear more times in the resulting tree than in the target tree.
2. For MOVE−SUBTREE edits, the pair consisting of the lemma for the node to move and the lemma for its prospective parent must exist in the target tree.
3. For INSERT−CHILD and INSERT−PARENT edits, the edge labels attaching the newly inserted nodes to their parents are always the most frequent edge label for the given POS.[8] Further edits can modify these edge labels.

3.4 Search Error and Failure

The search does not always find optimal edit sequences, but most sequences seem reasonable upon inspection. However, for some cases, the search does not find a sequence in a reasonable number of iterations. We therefore set an upper limit of $maxIters = 200$ on the number of iterations.[9] In practice, this constraint is enforced a small fraction of the time (e.g., less than 0.1% of the time for the answer selection training data). If no goal state is found after $maxIters$ iterations, a special unknown sequence feature is recorded.

4 Classification of Sequences

Given a training set of labeled sentence pairs, after extracting edit sequences, we train a logistic regression (LR) classification model (Hastie et al., 2001) on the labels and features of the extracted sequences.[10] We optimize with a variant of Newton's method (le Cessie and van Houwelingen, 1997).

The tree edit models use a set of 33 features of edit sequences to classify sentence pairs. We used the training data for the paraphrase task (§5.2) to develop this set. All features are integer-valued, and most are counts of different types of edits. Five are counts of the nodes in the source tree that were not edited directly by any operations (though their ancestors or descendants may have been). Table 2 describes the features in detail.

5 Experiments

Experiments were conducted to evaluate tree edit models for three tasks: recognizing textual entailment (Giampiccolo et al., 2007), paraphrase identification (Dolan et al., 2004), and an answer selection task (Wang et al., 2007) for question answering (Voorhees, 2004). The feature set and first tree edit model were developed for paraphrase, and then applied to the other tasks with very few modifications (all explained below) and no further tuning.[11]

5.1 Recognizing Textual Entailment

A tree edit model was trained for recognizing textual entailment (RTE). Here, an instance consists of

accuracy values that were not significantly different from each other. However, we did observe that increased search failure (§3.4) resulted from settings above 0.5.

[8] Edge label frequencies for each POS were computed from the training data for the MST parser (McDonald et al., 2005).

[9] $maxIters = 400$ for the textual entailment experiments to account for multi-sentence premises. For all tasks, extracting sequences took about 5 seconds on average per sentence pair with 1 GB of RAM on a 3.0 GHz machine.

[10] In cross-validation experiments with the training data, we found that unregularized LR outperformed SVMs (Vapnik, 1995) and ℓ_2-regularized LR, perhaps due to the small number of features in our models.

[11] All datasets were POS-tagged using Ratnaparkhi's (1996) tagger and parsed for dependencies using the MST Parser (McDonald et al., 2005). Features were computed from POS and edge label information in the dependency parses. The WordNet API (Miller et al., 1990) was used for lemmatization only. An appendix with further experimental details is available at http://www.ark.cs.cmu.edu/mheilman/tree-edit-appendix/.

Feature	Description
totalEdits	# of edits in the sequence.
*X*Edits	#s of X edits (where X is one of the nine edit types in Table 1).
relabelSamePOS, relabelSameLemma, relablePronoun, relabelProper, relabelNum	#s of RELABEL-NODE edits that: preserve POS, preserve lemmas, convert between nouns and pronouns, change proper nouns, change numeric values by more than 5% (to allow rounding), respectively.
insertVorN, insertProper	#s of INSERT-CHILD or INSERT-PARENT edits that: insert nouns or verbs, insert proper nouns, respectively.
removeVorN, removeProper, removeSubj, removeObj, removeVC, removeRoot	#s of REMOVE-LEAF or REMOVE-&-MERGE edits that: remove nouns or verbs, remove proper nouns, remove nodes with subject edge labels, remove nodes with object edge labels, remove nodes with verb complement edge labels, remove nodes with root edge labels (which may occur after NEW-ROOT edits), respectively.
relabelEdgeSubj, relabeledgeObj, relabelEdgeVC, relabelEdgeRoot	#s of RELABEL-EDGE edits that: change to or from subject edge labels, change to or from object edge labels, change to or from verb complement edge labels, change to or from root edge labels, respectively.
uneditedNodes, uneditedNum, uneditedVerbs, uneditedNouns, uneditedProper	#s of unedited nodes: in total, that are numeric values, that are verbs, that are nouns, that are proper nouns, respectively.
unknownSeq	1 if no edit sequence was found and 0 otherwise (§3.4).

Table 2: Tree edit sequence classification features.

a "premise," which is a sentence or paragraph about a particular topic or event, and a "hypothesis," which is a single, usually short, sentence that may or may not follow from the premise. The task is to decide whether or not the hypothesis is entailed by the premise (Giampiccolo et al., 2007).

Tree edit sequences were extracted in one direction, from premise to hypothesis.[12] Since premises may consist of multiple sentences, we attach sentences as children of dummy root nodes, for both the premise and hypothesis. The model was trained on the development set (i.e., training data) for RTE-3 along with all the data from the RTE-1 and RTE-2 tasks. It was then evaluated on the RTE-3 test set. We report precision and recall for true entailments, and overall accuracy (i.e., percentage correct).

System	Acc. %	Prec. %	Rec. %
Harmeling, 2007	59.5	-	-
de Marneffe et al., 2006	60.5	61.8	60.2
M&M, 2007 (NL)	59.4	**70.1**	36.1
M&M, 2007 (Hybrid)	**64.3**	65.5	63.9
Tree Edit Model	62.8	61.9	**71.2**

Table 3: Results for recognizing textual entailments. Precision and recall values are for the true entailment class. Results for de Marneffe et al. (2006) were reported by MacCartney and Manning (2008). Harmeling (2007) only reported accuracy.

We compare to four systems that use syntactic dependencies and lexical semantic information.[13] De Marneffe et al. (2006) described an RTE system that finds word alignments and then classifies sentence pairs based on those alignments. MacCartney and Manning (2008) used an inference procedure based on Natural Logic, leading to a relatively high-precision, low-recall system. MacCartney and Manning (2008) also tested a hybrid of the natural logic system and the complementary system of de Marneffe et al. (2006) to improve coverage. Harmeling (2007) took an approach similar to ours involving classification based on transformation sequences, but with less general operations and a more complex, heuristic procedure for finding sequences.

Table 3 presents RTE results, showing that the tree edit model performs competitively. While it does not outperform state-of-the-art RTE systems, the tree edit model is simpler and less tailored to this task than many other RTE systems based on similar linguistic information.

[12] It is counter-intuitive to model *adding* information through extensive insertions, for both entailment and answer selection.

[13] The top-performing RTE systems often involve significant manual engineering for the RTE task. Also, many employ techniques that make them not very comparable to our approach (e.g., theorem proving). We also note that Kouylekov and Magnini (2005) report 55% accuracy for RTE-2 using TED. See Giampiccolo et al. (2007) for more RTE-3 results.

System	Acc. %	Prec. %	Rec. %
Wan et al., 2006	75.6	77	90
D&S, 2009 (QG)	73.9	74.9	**91.3**
D&S, 2009 (PoE)	**76.1**	**79.6**	86.0
Tree Edit Model	73.2	75.7	87.8

Table 4: Paraphrase identification results, with precision and recall measures for true (positive) paraphrases. Wan et al. (2006) report precision and recall values with only two significant digits.

System	MAP	MRR
Punyakanok et al., 2004	0.3814	0.4462
+WN	0.4189	0.4939
Cui et al., 2005	0.4350	0.5569
+WN	0.4271	0.5259
Wang et al., 2007	0.4828	0.5571
+WN	0.6029	0.6852
Tree Edit Model	**0.6091**	**0.6917**

Table 5: Results for the task of answer selection for question answering. +WN denotes use of WordNet features.

5.2 Paraphrase Identification

A tree edit model was trained and tested for paraphrase identification using the the Microsoft Research Paraphrase Corpus (Dolan et al., 2004). The task is to identify whether two sentences convey essentially the same meaning.

The standard training set was used to train the tree edit classification model to distinguish between true and false paraphrases. Since there is no predefined direction for paraphrase pairs, we extracted two sequences for each pair (one in each direction) and summed the feature values. The model was evaluated with the standard test set.

We report accuracy, positive class precision (i.e., percentage of predicted positive paraphrases that had positive gold-standard labels), and positive class recall (i.e., percentage of positive gold-standard labels that were predicted to be positive paraphrases).

We compare to two of the best performance approaches to paraphrase. One approach, by Wan et al. (2006), uses an SVM classifier with features based on syntactic dependencies, TED, unigram overlap, and BLEU scores (Papineni et al., 2002). The other system, by Das and Smith (2009), is based on a quasi-synchronous grammar (QG; Smith and Eisner, 2006), a probabilistic model that allows loose alignments between trees but prefers tree isomorphism. In addition to syntactic dependencies, the QG model utilizes entity labels from BBN Identifinder (Bikel et al., 1999) and lexical semantics knowledge from WordNet. Das and Smith (2009) also use a product of experts (PoE) (Hinton, 1999) to combine the QG model with lexical overlap features.

Table 4 shows the test set results for all of the systems. While the tree edit model did not outperform the other systems, it produced competitive results. Moreover, the tree edit model does not make use of BLEU scores (Wan et al., 2006), entity labeling components, lexical semantics knowledge sources such as WordNet (beyond lemmatization), or system combination techniques (Das and Smith, 2009).

5.3 Answer Selection for Question Answering

A tree edit model was trained for answer selection in question answering (QA). In this task, an instance consists of a short factual question (e.g., *Who wrote the 'Tale of Genji'?*) and a candidate answer sentence retrieved by the information retrieval component of a question answering system. For a positive instance, the text will correctly answer the question—though perhaps indirectly. It may also contain various extraneous information (e.g., *Kano script made possible the development of a secular Japanese literature, beginning with such Late Heian classics as Lady Murasaki's "Tales of Genji."*). For a given set of questions, the task here is to *rank* candidate answers (Wang et al., 2007).

The experimental setup is the same as in Wang et al. (2007). We trained the tree edit model on the manually judged positive and negative QA pairs from previous QA tracks at the Text REtrieval Conference (TREC-8 through TREC-12). The goal of the task is to rank answer candidates rather than classify them; therefore, after training a logistic regression classifier, we rank the answer candidates for a given question by their posterior probabilities of correctness according to the model.

We tested our model with QA pairs from TREC-13. We report Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR), which are information retrieval measures for ranked lists.

Tree edit sequences were extracted only in one direction, from answer to question. We compare our tree edit model to three other systems as they are reported by Wang et al. (2007). Wang et al. use a QG model, incorporating information from dependency

trees, entity labels from BBN Identifinder (Bikel et al., 1999), and lexical semantics knowledge from WordNet (Miller et al., 1990). Cui et al. (2005) developed an information theoretic measure based on dependency trees. Punyakanok et al. (2004) used a generalization of TED to model the QA pairs. For their experiments, Wang et al. also extended both of the latter models to utilize WordNet.

Table 5 displays answer selection results, including test set results for the baseline systems with and without lexical semantic information from WordNet. The tree edit model, which does not use lexical semantics knowledge, produced the best result reported to date. The results for the tree edit model are statistically significantly different (sign test, $p < 0.01$) from the results for all except the Wang et al. (2007) system with WordNet ($p > 0.05$).

5.4 Discussion

The parameter settings learned for the features in Table 2 were broadly similar for the three tasks. For example, operations involving changes to subjects and proper nouns tended to be associated with non-paraphrases, false entailments, and incorrect answers. We did not observe any interesting differences in the parameter values.

While the tree edit models perform competitively in multiple tasks by capturing relevant syntactic phenomena, it is clear that syntax alone cannot solve these semantic tasks. Fortunately, this approach is amenable to extensions, facilitated by the separation of the representation extraction and classification steps. Richer edits could be included; lexical semantics could be integrated into the classifier or the search heuristic; or edit sequences might be found for other types of trees, such as semantic parses.

6 Related Work

TED is a widely studied technique with many applications (Klein, 1989; Zhang and Shasha, 1989; Punyakanok et al., 2004; Schilder and McInnes, 2006). See Bille (2005) for a review. Chawathe and Garcia-Molina (1997) describe a tree edit algorithm for detecting changes in structured documents that incorporates edits for moving subtrees and reordering children. However, they make assumptions unsuitable for natural language, such as the absence of recursive syntactic rewrite rules. Bernard et al. (2008) use EM to learn the costs for simple insert, relabel, and delete edits, but they only discuss experiments for digit recognition and a task using artificial data.

Much research has focused on modeling word reordering phenomena and syntactic alignments (e.g., Gildea, 2003; Smith and Eisner, 2006; *inter alia*), and such methods have been applied successfully to semantic tasks (de Marneffe et al., 2006; Wang et al., 2007; Das and Smith, 2009). While we not describe connections to such approaches in detail due to space limitations, we note that theoretical connections are possible between transformations and alignments (Chawathe and Garcia-Molina, 1997).

Tree kernels have been applied to a variety of natural language tasks (Collins and Duffy, 2001; Zelenko et al., 2003; Culotta and Sorensen, 2004). Of particular interest, Zanzotto and Moschitti (2006) describe a kernel for RTE that takes tree pairs, rather than single trees, as input. To our knowledge, our use of a tree kernel as a search heuristic is novel.

7 Conclusion

We described tree edit models that generalize TED by allowing operations that better account for complex reordering phenomena and by learning from data how different edits should affect the models decisions about output variables of interest (e.g., the correctness of answers). They offer an intuitive and effective method for modeling sentence pairs. They led to competitive performance for three tasks: paraphrase identification, recognizing textual entailment, and answer selection for question answering.

Acknowledgments

We acknowledge partial support from the Institute of Education Sciences, U.S. Department of Education, through Grant R305B040063 to Carnegie Mellon University; and the National Science Foundation through a Graduate Research Fellowship for the first author and grant IIS-0915187 to the second author. We thank Mengqiu Wang and Dipanjan Das for their help with the data, André Martins for his geometric interpretation of our search procedure, and the anonymous reviewers for their comments.

References

M. Bernard, L. Boyer, A. Habrard, and M. Sebban. 2008. Learning probabilistic models of tree edit distance.

Pattern Recognition.

D. M. Bikel, R. Schwartz, and R. M. Weischedel. 1999. An algorithm that learns what's in a name. *Machine Learning*, 34.

P. Bille. 2005. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337.

S. Chawathe and H. Garcia-Molina. 1997. Meaningful change detection in structured data. In *Proc. of ACM SIGMOD*.

M. Collins and N. Duffy. 2001. Convolution kernels for natural language. In *Proc. of NIPS*.

H. Cui, R. Sun, K. Li, M. Kan, , and T. Chua. 2005. Question answering passage retrieval using dependency relations. In *Proc. of ACM-SIGIR*.

A. Culotta and J. Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proc. of ACL*.

D. Das and N. A. Smith. 2009. Paraphrase identification as probabilistic quasi-synchronous recognition. In *Proc. of ACL-IJCNLP*.

M. de Marneffe, B. MacCartney, T. Grenager, D. Cer, A. Rafferty, and C. D. Manning. 2006. Learning to distinguish valid textual entailments. In *Proc. of the Second PASCAL Challenges Workshop*.

B. Dolan, C. Quirk, and C. Brockett. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proc. of COLING*.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of COLING*.

D. Giampiccolo, B. Magnini, I. Dagan, and B. Dolan, editors. 2007. *The third pascal recognizing textual entailment challenge*.

D. Gildea. 2003. Loosely tree-based alignment for machine translation. In *Proc. of ACL*.

S. Harmeling. 2007. An extensible probabilistic transformation-based approach to the third Recognizing Textual Entailment challenge. In *Proc. of ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*.

T. Hastie, R. Tibshirani, and J. Friedman. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

D. Haussler. 1999. Convolution kernels on discrete structures. Technical Report ucs-crl-99-10, University of California Santa Cruz.

G. E. Hinton. 1999. Product of experts. In *Proc. of ICANN*.

P. N. Klein. 1989. Computing the edit-distance between unrooted ordered trees. In *Proc. of European Symposium on Algorithms*.

M. Kouylekov and B. Magnini. 2005. Recognizing textual entailment with tree edit distance algorithms. In *Proc. of the PASCAL RTE Challenge*.

S. le Cessie and J. C. van Houwelingen. 1997. Ridge estimators in logistic regression. *Applied Statistics*, 41.

B. MacCartney and C. D. Manning. 2008. Modeling semantic containment and exclusion in natural language inference. In *Proc. of COLING*.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of HLT-EMNLP*.

G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. 1990. WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4).

A. Moschitti. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proc. of ECML*.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*.

J. Pearl. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley.

V. Punyakanok, D. Roth, and W. Yih. 2004. Mapping dependencies trees: An application to question answering. In *Proc. of the 8th International Symposium on Artificial Intelligence and Mathematics*.

A. Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Proc. of EMNLP*.

F. Schilder and B. T. McInnes. 2006. TLR at DUC 2006: approximate tree similarity and a new evaluation regime. In *Proc. of DUC*.

B. Schölkopf and A. J. Smola. 2001. *Learning with Kernels*. MIT Press.

D. A. Smith and J. Eisner. 2006. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proc. of HLT-NAACL Workshop on Statistical Machine Translation*.

V. N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer.

E. M. Voorhees. 2004. Overview of TREC 2004. In *Proc. of TREC*.

S. Wan, M. Dras, R. Dale, and C. Paris. 2006. Using dependency-based features to take the "para-farce" out of paraphrase. In *Proc. of the Australasian Language Technology Workshop*.

M. Wang, N. A. Smith, and T. Mitamura. 2007. What is the Jeopardy model? A quasi-synchronous grammar for QA. In *Proc. of EMNLP-CoNLL*.

F. M. Zanzotto and A. Moschitti. 2006. Automatic learning of textual entailments with cross-pair similarities. In *Proc. of COLING/ACL*.

D. Zelenko, C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *J. of Machine Learning Research*, 3.

K. Zhang and D. Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18.