Evaluating Dependency Parsing: Robust and Heuristics-Free Cross-Annotation Evaluation

Reut Tsarfaty
Uppsala University
Sweden

Joakim Nivre
Uppsala University
Sweden

Evelina Andersson
Uppsala University
Sweden

Abstract

Methods for evaluating dependency parsing using attachment scores are highly sensitive to representational variation between dependency treebanks, making cross-experimental evaluation opaque. This paper develops a robust procedure for cross-experimental evaluation, based on deterministic unification-based operations for harmonizing different representations and a refined notion of tree edit distance for evaluating parse hypotheses relative to multiple gold standards. We demonstrate that, for different conversions of the Penn Treebank into dependencies, performance trends that are observed for parsing results in isolation change or dissolve completely when parse hypotheses are normalized and brought into the same common ground.

1 Introduction

Data-driven dependency parsing has seen a considerable surge of interest in recent years. Dependency parsers have been tested on parsing sentences in English (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; McDonald et al., 2005) as well as many other languages (Nivre et al., 2007a). The evaluation metric traditionally associated with dependency parsing is based on scoring labeled or unlabeled attachment decisions, whereby each correctly identified pair of head-dependent words is counted towards the success of the parser (Buchholz and Marsi, 2006). As it turns out, however, such evaluation procedures are sensitive to the annotation choices in the data on which the parser was trained.

Different annotation schemes often make different assumptions with respect to how linguistic content is represented in a treebank (Rambow, 2010). The consequence of such annotation discrepancies is that when we compare parsing results across different experiments, even ones that use the same parser and the same set of sentences, the gap between results in different experiments may not reflect a true gap in performance, but rather a difference in the annotation decisions made in the respective treebanks.

Different methods have been proposed for making dependency parsing results comparable across experiments. These methods include picking a single gold standard for all experiments to which the parser output should be converted (Carroll et al., 1998; Cer et al., 2010), evaluating parsers by comparing their performance in an embedding task (Miyao et al., 2008; Buyko and Hahn, 2010), or neutralizing the arc direction in the native representation of dependency trees (Schwartz et al., 2011).

Each of these methods has its own drawbacks. Picking a single gold standard skews the results in favor of parsers which were trained on it. Transforming dependency trees to a set of pre-defined labeled dependencies, or into task-based features, requires the use of heuristic rules that run the risk of distorting correct information and introducing noise of their own. Neutralizing the direction of arcs is limited to unlabeled evaluation and local context, and thus may not cover all possible discrepancies.

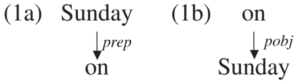
This paper proposes a new three-step protocol for cross-experiment parser evaluation, and in particular for comparing parsing results across data sets that adhere to different annotation schemes. In the first step all structures are brought into a single formal space of events that neutralizes representation peculiarities (for instance, arc directionality). The second step formally computes, for each sentence in the data, the common denominator of the different gold standards, containing all and only linguistic content that is shared between the different schemes. The last step computes the normalized distance from this common denominator to parse hypotheses, minus the cost of distances that reflect mere annotation idiosyncrasies. The procedure that implements this protocol is fully deterministic and heuristics-free.

We use the proposed procedure to compare dependency parsing results trained on Penn Treebank trees converted into dependency trees according to five different sets of linguistic assumptions. We show that when starting off with the same set of sentences and the same parser, training on different conversion schemes yields apparently significant performance gaps. When results across schemes are normalized and compared against the shared linguistic content, these performance gaps decrease or dissolve completely. This effect is robust across parsing algorithms. We conclude that it is imperative that cross-experiment parse evaluation be a well thought-through endeavor, and suggest ways to extend the protocol to additional evaluation scenarios.

2 The Challenge: Treebank Theories

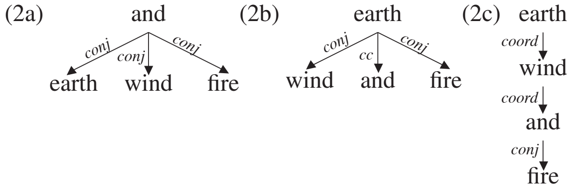
Dependency treebanks contain information about the grammatically meaningful elements in the utterance and the grammatical relations between them. Even if the formal representation in a dependency treebank is well-defined according to current standards (Kübler et al., 2009), there are different ways in which the trees can be used to express syntactic content (Rambow, 2010). Consider, for instance, algorithms for converting the phrase-structure trees in the Penn Treebank (Marcus et al., 1993) into dependency structures. Different conversion algorithms implicitly make different assumptions about how to represent linguistic content in the data. When multiple conversion algorithms are applied to the same data, we end up with different dependency trees for the same sentences (Johansson and Nugues, 2007; Choi and Palmer, 2010; de Marneffe et al., 2006). Some common cases of discrepancies are as follows.

Lexical vs. Functional Head Choice. In linguistics, there is a distinction between lexical heads and functional heads. A lexical head carries the semantic gist of a phrase while a functional one marks its relation to other parts of the sentence. The two kinds of heads may or may not coincide in a single word form (Zwicky, 1993). Common examples refer to prepositional phrases, such as the phrase “on Sunday”. This phrase has two possible analyses, one selects a lexical head (1a) and the other selects a functional one (1b), as depicted below.

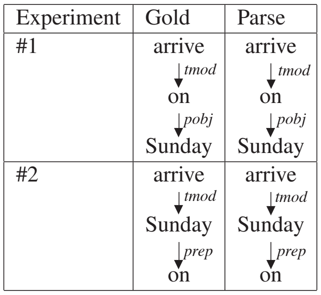


Similar choices are found in phrases which contain functional elements such as determiners, coordination markers, subordinating elements, and so on.

Multi-Headed Constructions. Some phrases are considered to have multiple lexical heads, for instance, coordinated structures. Since dependency-based formalisms require us to represent all content as binary relations, there are different ways we could represent such constructions. Let us consider the coordination of nominals below. We can choose between a functional head (1a) and a lexical head (2b, 2c). We can further choose between a flat representation in which the first conjunct is a single head (2b), or a nested structure where each conjunct/marker is the head of the following element (2c). All three alternatives empirically exist. Example (2a) reflects the structures in the CoNLL 2007 shared task data (Nivre et al., 2007a). Johansson and Nugues (2007) use structures like (2b). Example (2c) reflects the analysis of Mel’čuk (1988).



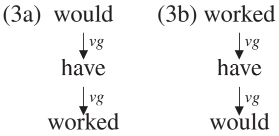
Periphrastic Marking. When a phrase includes periphrastic marking — such as the tense and modal marking in the phrase “would have worked” below — there are different ways to consider its division into phrases. One way to analyze this phrase would be to choose auxiliaries as heads, as in (3a). Another alternative would be to choose the final verb as the



Gold: Parse	#1	# 2
#1	1.0	0.0
#2	0.0	1.0

Figure 1: Calculating cross-experiment LAS results

main head, and let the auxiliaries create a verb chain with different levels of projection. Each annotation decision dictates a different direction of the arcs and imposes its own internal division into phrases.



In standard settings, an experiment that uses a data set which adheres to a certain annotation scheme reports results that are compared against the annotation standard that the parser was trained on. But if parsers were trained on different annotation standards, the empirical results are not comparable across experiments. Consider, for instance, the example in Figure 1. If parse1 and parse2 are compared against gold2 using labeled attachment scores (LAS), then parse1 results are lower than the results of parse2, even though both parsers produced linguistically correct and perfectly useful output.

Existing methods for making parsing results comparable across experiments include heuristics for converting outputs into dependency trees of a predefined standard (Briscoe et al., 2002; Cer et al., 2010) or evaluating the performance of a parser within an embedding task (Miyao et al., 2008; Buyko and Hahn, 2010). However, heuristic rules for cross-annotation conversion are typically hand written and error prone, and may not cover all possible discrepancies. Task-based evaluation may be sensitive to the particular implementation of the embedding task and the procedures that extract specific task-related features from the different parses. Beyond that, conversion heuristics and task-based procedures are currently developed almost exclusively for English. Other languages typically lack such resources.

A recent study by Schwartz et al. (2011) takes a different approach towards cross-annotation evaluation. They consider different directions of head-dependent relations (such as on→Sunday and Sunday→on) and different parent-child and grandparent-child relations in a chain (such as arrive→on and arrive→sunday in “arrive on sunday”) as equivalent. They then score arcs that fall within corresponding equivalence sets. Using these new scores Schwartz et al. (2011) neutralize certain annotation discrepancies that distort parse comparison. However, their treatment is limited to local context and does not treat structures larger than two sequential arcs. Additionally, since arcs in different directions are typically labeled differently, this method only applies for unlabeled dependencies.

What we need is a fully deterministic and formally precise procedure for comparing any set of labeled or unlabeled dependency trees, by consolidating the shared linguistic content of the complete dependency trees in different annotation schemes, and comparing parse hypotheses through sound metrics that can take into account multiple gold standards.

3 The Proposal: Cross-Annotation Evaluation in Three Simple Steps

We propose a new protocol for cross-experiment parse evaluation, consisting of three fundamental components: (i) abstracting away from annotation peculiarities, (ii) generalizing theory-specific structures into a single linguistically coherent gold standard that contains all and only consistent information from all sources, and (iii) defining a sound metric that takes into account the different gold standards that are being considered in the experiments.

In this section we first define *functional trees* as the common space of formal objects and define a deterministic conversion procedure from dependency trees to functional trees. Next we define a set of formal operations on functional trees that compute, for every pair of corresponding trees of the same yield, a single gold tree that resolves inconsistencies among gold standard alternatives and combines the information that they share. Finally, we define scores based on *tree edit distance*, refined to consider the distance from parses to the overall gold tree as well as the different annotation alternatives.

Preliminaries. Let $\mathcal{T}$ be a finite set of terminal symbols and let $\mathcal{L}$ be a set of grammatical relation labels. A dependency graph d is a directed graph which consists of nodes V_d and arcs $A_d \subseteq V_d \times V_d$. We assume that all nodes in V_d are labeled by terminal symbols via a function $label_V : V_d \rightarrow \mathcal{T}$. A well-formed dependency graph $d = (V_d, A_d)$ for a sentence $S = t_1, t_2, ..., t_n$ is any dependency graph that is a directed tree originating out of a node v_0 labeled $t_0 = ROOT$, and spans all terminals in the sentence, that is, for every $t_i \in S$ there exists $v_j \in V_d$ labeled $label_V(v_j) = t_i$. For simplicity we assume that every node v_j is indexed according to the position of the terminal label, i.e., that for each t_i labeling v_j, i always equals j. In a labeled dependency tree, arcs in A_d are labeled by elements of $\mathcal{L}$ via a function $label_A : A_d \rightarrow \mathcal{L}$ that encodes the grammatical relation between the terminals labeling the connected nodes. We define two auxiliary functions on nodes in dependency trees. The function $subtree : V_d \rightarrow \mathcal{P}(V_d)$ assigns to every node $v \in V_d$ the set of nodes accessible by it through the reflexive transitive closure of the arc relation A_d. The function $span : V_d \rightarrow \mathcal{P}(\mathcal{T})$ assigns to every node $v \in V_d$ a set of terminals such that $span(v) = \{t \in \mathcal{T} | t = label_V(u) \text{ and } u \in subtree(v)\}$.[1]

Step 1: Functional Representation Our first goal is to define a representation format that keeps all functional relationships that are represented in the dependency trees intact, but remains neutral with respect to the directionality of the head-dependent relations. To do so we define *functional trees* — linearly-ordered labeled trees which, instead of head-to-head binary relations, represent the complete functional structure of a sentence. Assuming the same sets of terminal symbols $\mathcal{T}$ and grammatical relation labels $\mathcal{L}$, and assuming extended sets of nodes V and arcs $A \subseteq V \times V$, a functional tree $\pi = (V, A)$ is a directed tree originating from a single root $v_0 \in V$ where all non-terminal nodes in π are labeled with grammatical relation labels that signify the grammatical function of the chunk they dominate inside the tree via $label_{NT} : V \rightarrow \mathcal{L}$. All terminal nodes in π are labeled with terminal symbols via a $label_T : V \rightarrow \mathcal{T}$ function. The function $span : V \rightarrow \mathcal{P}(V)$ now picks out the set of terminal labels of the terminal nodes accessible by a node $v \in V$ via A. We obtain functional trees from dependency trees using the following procedure:

- Initialize the set of nodes and arcs in the tree.

  $$V := V_d$$
  $$A := A_d$$

- Label each node $v \in V$ with the label of its incoming arc.

  $$label_{NT}(v) = label_A(u, v)$$

- In case $|span(v)| > 1$ add a new node u as a daughter designating the lexical head, labeled with the wildcard symbol *:

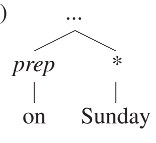
  $$V := V \cup \{u\}$$
  $$A := A \cup \{(v, u)\}$$
  $$label_{NT}(u) = *$$

- For each node v such that $|span(v)| = 1$, add a new node u as a daughter, labeled with its own terminal:

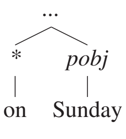
  ```
  $V := V \cup \{u\}$
  $A := A \cup \{(v, u)\}$
  if $(label_{NT}(v) \neq *)$
      $label_T(u) := label_V(v)$
  else
      $label_T(u) := label_V(parent(v))$
  ```

That is to say, we label all nodes with spans greater than 1 with the grammatical function of their head, and for each node we add a new daughter u designating the head word, labeled with its grammatical function. Wildcard labels are compatible with any, more specific, grammatical function of the word inside the phrase. This gives us a constituency-like representation of dependency trees labeled with functional information, which retains the linguistic assumptions reflected in the dependency trees. When applying this procedure, examples (1)–(3) get transformed into (4)–(6) respectively.

(4a)

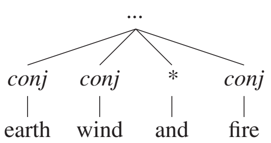


(4b)

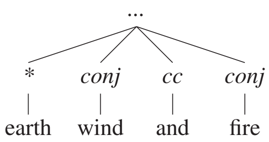


[1] If a dependency tree d is projective, than for all $v \in V_d$ the terminals in $span(v)$ form a contiguous segment of S. The current discussion assumes that all trees are projective. We comment on non-projective dependencies in Section 4.

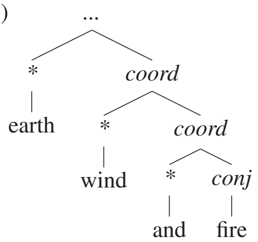
(5a)



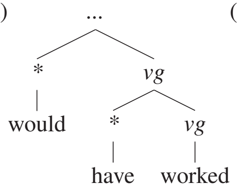
(5b)



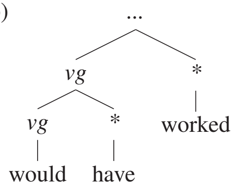
(5c)



(6a)



(6b)



Considering the functional trees resulting from our procedure, it is easy to see that for tree pairs (4a)–(4b) and (5a)–(5b) the respective functional trees are identical modulo wildcards, while tree pairs (5b)–(5c) and (6a)–(6b) end up with different tree structures that realize different assumptions concerning the internal structure of the tree. In order to compare, combine or detect inconsistencies in the information inherent in different functional trees, we define a set of formal operations that are inspired by familiar notions from unification-based formalisms (Shieber (1986) and references therein).

Step 2: Formal Operations on Trees The intuition behind the formal operations we define is simple. A completely flat tree over a span is the most general structural description that can be given to it. The more nodes dominate a span, the more linguistic assumptions are made with respect to its structure. If an arc structure in one tree merely elaborates an existing flat span in another tree, the theories underlying the schemes are compatible, and their information can be combined. Otherwise, there exists a conflict in the linguistic assumptions, and we need to relax some of the assumptions, i.e., remove functional nodes, in order to obtain a coherent structure that contains the information on which they agree.

Let π_1, π_2 be functional trees over the same yield $t_1, .., t_n$. Let the function $span(v)$ pick out the terminals labeling terminal nodes that are accessible via a node $v \in V$ in the functional tree through the relation A. We define first the tree subsumption relation for comparing the amount of information inherent in the arc-structure of two trees.[2]

> **T-Subsumption**, denoted $\sqsubseteq_t$, is a relation between trees which indicates that a tree π_1 is consistent with and more general than tree π_2. Formally: $\pi_1 \sqsubseteq_t \pi_2$ iff for every node $n \in \pi_1$ there exists a node $m \in \pi_2$ such that $span(n) = span(m)$ and $label(n) = label(m)$.

Looking at the functional trees of (4a)–(4b) we see that their unlabeled skeletons mutually subsume each other. In their labeled versions, however, each tree contains labeling information that is lacking in the other. In the functional trees (5b)–(5c) a flat structure over a span in (5b) is more elaborated in (5c). In order to combine information in trees with compatible arc structures, we define tree unification.

> **T-Unification**, denoted $\sqcup_t$, is the operation that returns the most general tree structure π_3 that is subsumed by both π_1, π_2 if such exists, and fails otherwise. Formally: $\pi_1 \sqcup_t \pi_2 = \pi_3$ iff $\pi_1 \sqsubseteq_t \pi_3$ and $\pi_2 \sqsubseteq_t \pi_3$, and for all π_4 such that $\pi_1 \sqsubseteq_t \pi_4$ and $\pi_2 \sqsubseteq_t \pi_4$ it holds that $\pi_3 \sqsubseteq_t \pi_4$.

Tree unification collects the information from two trees into a single result if they are consistent, and detects an inconsistency otherwise. In case of an inconsistency, as is the case in the functional trees (6a) and (6b), we cannot unify the structures due to a conflict concerning the internal division of an expression into phrases. However, we still want to generalize these two trees into one tree that contains all and only the information that they share. For that we define the tree generalization operation.

> **T-Generalization**, denoted $\sqcap_t$, is the operation that returns the most specific tree that is more general than both trees. Formally, $\pi_1 \sqcap_t \pi_2 = \pi_3$ iff $\pi_3 \sqsubseteq_t \pi_1$ and $\pi_3 \sqsubseteq_t \pi_2$, and for every π_4 such that $\pi_4 \sqsubseteq_t \pi_1$ and $\pi_4 \sqsubseteq_t \pi_2$ it holds that $\pi_4 \sqsubseteq_t \pi_3$.

[2] Note that the wildcard symbol * is equal to any other symbol. In case the node labels consist of complex feature structures made of attribute-value lists, we replace $label(n) = label(m)$ in the subsumption definition with $label(n) \sqsubseteq label(m)$ in the sense of (Shieber, 1986).

Unlike unification, generalization can never fail. For every pair of trees there exists a tree that is more general than both: in the extreme case, pick the completely flat structure over the yield, which is more general than any other structure. For (6a)–(6b), for instance, we get that (6a)$\sqcap_t$(6b) is a flat tree over pre-terminals where "would" and "have" are labeled with *'vg'* and "worked" is the head, labeled with *'*'*.

The generalization of two functional trees provides us with one structure that reflects the common and consistent content of the two trees. These structures thus provide us with a formally well-defined gold standard for cross-treebank evaluation.

Step 3: Measuring Distances. Our functional trees superficially look like constituency-based trees, so a simple proposal would be to use Parseval measures (Black et al., 1991) for comparing the parsed trees against the new generalized gold trees. Parseval scores, however, have two significant drawbacks. First, they are known to be too restrictive with respect to some errors and too permissive with respect to others (Carroll et al., 1998; Kübler and Telljohann, 2002; Roark, 2002; Rehbein and van Genabith, 2007). Secondly, F_1 scores would still penalize structures that are correct with respect to the original gold, but are not there in the generalized structure. Here we propose to adopt measures that are based on tree edit distance (TED) instead. TED-based measures are, in fact, an extension of attachment scores for dependency trees. Consider, for instance, the following operations on dependency arcs.

reattach-arc remove arc $(u, v) \in A_d$ and add an arc $A_d \cup \{(w, v)\}$.

relabel-arc relabel arc $l_1(u, v)$ as $l_2(u, v)$

Assuming that each operation is assigned a cost, the attachment score of comparing two dependency trees is simply the cost of all edit operations that are required to turn a parse tree into its gold standard, normalized with respect to the overall size of the dependency tree and subtracted from a unity.[3] Here we apply the idea of defining scores by TED costs normalized relative to the size of the tree and substracted from a unity, and extend it from fixed-size dependency trees to ordered trees of arbitrary size.

Our formalization follows closely the formulation of the T-Dice measure of Emms (2008), building on his thorough investigation of the formal and empirical differences between TED-based measures and Parseval. We first define for any ordered and labeled tree π the following operations.

relabel-node change the label of node v in π

delete-node delete a non-root node v in π with parent u, making the children of v the children of u, inserted in the place of v as a subsequence in the left-to-right order of the children of u.

insert-node insert a node v as a child of u in π making it the parent of a consecutive subsequence of the children of u.

An edit script $ES(\pi_1, \pi_2) = \{e_0, e_1 .... e_k\}$ between π_1 and π_2 is a set of edit operations required for turning π_1 into π_2. Now, assume that we are given a cost function defined for each edit operation. The cost of $ES(\pi_1, \pi_2)$ is the sum of the costs of the operations in the script. An optimal edit script is an edit script between π_1 and π_2 of minimum cost.

$$ES^*(\pi_1, \pi_2) = \text{argmin}_{ES(\pi_1,\pi_2)} \sum_{e \in ES(\pi_1,\pi_2)} cost(e)$$

The tree edit distance problem is defined to be the problem of finding the optimal edit script and computing the corresponding distance (Bille, 2005).

A simple way to calculate the error δ of a parse would be to define it as the edit distance between the parse hypothesis π_1 and the gold standard π_2.

$$\delta(\pi_1, \pi_2) = cost(ES^*(\pi_1, \pi_2))$$

However, in such cases the parser may still get penalized for recovering nodes that are lacking in the generalization. To solve this, we refine the distance between a parse tree and the generalized gold tree to discard edit operations on nodes that are there in the native gold tree but are eliminated through generalization. We compute the intersection of the edit script turning the parse tree into the generalize gold with the edit script turning the native gold tree into the generalized gold, and discard its cost. That is, if parse1 and parse2 are compared against gold1 and gold2 respectively, and if we set gold3 to be the result of gold1$\sqcap_t$gold2, then δ_{new} is defined as:

[3] The size of a dependency tree, either parse or gold, is always fixed by the number of terminals.

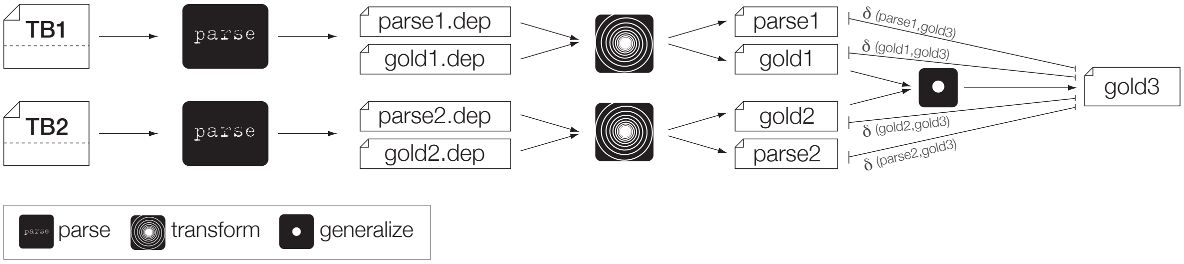


Figure 2: The evaluation pipeline. Different versions of the treebank go into different experiments, resulting in different parse and gold files. All trees are transformed into functional trees. All gold files enter generalization to yield a new gold. The different δ arcs represent the different tree distances used for calculating the TED-based scores.

$$\begin{aligned}&\delta_{new}(\text{parse1, gold1,gold3}) = \\ &\delta(\text{parse1,gold3}) \\ &-cost(ES^*(\text{parse1,gold3})\cap ES^*(\text{gold1,gold3}))\end{aligned}$$

Now, if gold1 and gold3 are identical, then $ES^*(\text{gold1,gold3})=\emptyset$ and we fall back on the simple tree edit distance score $\delta_{new}(\text{parse1,gold1,gold3})=\delta(\text{parse1, gold3})$. When parse1 and gold1 are identical, i.e., the parser produced perfect output with respect to its own scheme, then $\delta_{new}(\text{parse1,gold1,gold3})=\delta_{new}(\text{gold1,gold1,gold3}) =\delta(\text{gold1,gold3}) - cost(ES^*(\text{gold1,gold3}))=0$, and the parser does not get penalized for recovering a correct structure in gold1 that is lacking in gold3.

In order to turn distances into accuracy measures we have to normalize distances relative to the maximal number of operations that is conceivable. In the worst case, we would have to remove all the internal nodes in the parse tree and add all the internal nodes of the generalized gold, so our normalization factor ι is defined as follows, where $|\pi|$ is the size[4] of π.

$$\iota(\text{parse1,gold3}) = |\text{parse1}| + |\text{gold3}|$$

We now define the *score* of parse1 as follows:[5]

$$1 - \frac{\delta_{new}(\text{parse1,gold1,gold3})}{\iota(\text{parse1,gold3})}$$

Figure 2 summarizes the steps in the evaluation procedure we defined so far. We start off with two versions of the treebank, TB1 and TB2, which are parsed separately and provide their own gold standards and parse hypotheses in a labeled dependencies format. All dependency trees are then converted into functional trees, and we compute the generalization of each pair of gold trees for each sentence in the data. This provides the generalized gold standard for all experiments, here marked as gold3.[6] We finally compute the distances $\delta_{new}(\text{parse1,gold1,gold3})$ and $\delta_{new}(\text{parse2,gold2,gold3})$ using the different tree edit distances that are now available, and we repeat the procedure for each sentence in the test set.

To normalize the scores for an entire test set of size n we can take the arithmetic mean of the scores.

$$\frac{\sum_{i=1}^{|\text{test-set}|} score(\text{parse1}_i,\text{gold1}_i,\text{gold3}_i)}{|\text{test-set}|}$$

Alternatively we can globally average of all edit distance costs, normalized by the maximally possible edits on parse trees turned into generalized trees.

$$1 - \frac{\sum_{i=1}^{|\text{test-set}|} \delta_{new}(\text{parse1}_i,\text{gold1}_i,\text{gold3}_i)}{\sum_{i=1}^{|\text{test-set}|} \iota(\text{parse1}_i,\text{gold3}_i)}$$

The latter score, global averaging over the entire test set, is the metric we use in our evaluation procedure.

4 Experiments

We demonstrate the application of our procedure to comparing dependency parsing results on different versions of the Penn Treebank (Marcus et al., 1993).

The Data We use data from the PTB, converted into dependency structures using the LTH software, a general purpose tool for constituency-to-dependency conversion (Johansson and Nugues, 2007). We use LTH to implement the five different annotation standards detailed in Table 3.

[4] Following common practice, we equate size $|\pi|$ with the number of nodes in π, discarding the terminals and root node.

[5] If the trees have only root and leaves, $\iota = 0$, $score := 1$.

[6] Generalization is an associative and commutative operation, so it can be extended for n experiments in any order.

Train		**Default**	**Old LTH**	**CoNLL07**
Gold				
Default	UAS	**0.9142**	0.6077	0.7772
	LAS	**0.8820**	0.4801	0.6454
	U-TED	**0.9488**	0.8926	0.9237
	L-TED	**0.9241**	0.7811	0.8441
Old LTH	UAS	0.6053	**0.8955**	0.6508
	LAS	0.4816	**0.8644**	0.5771
	U-TED	0.8931	**0.9564**	0.9092
	L-TED	0.7811	**0.9317**	0.8197
CoNLL07	UAS	0.7734	0.6474	**0.8917**
	LAS	0.6479	0.5722	**0.8736**
	U-TED	0.9260	0.9097	**0.9474**
	L-TED	0.8480	0.8204	**0.9233**
Default-OldLTH	U-TED	0.9500	**0.9543**	
	L-TED	0.9278	**0.9324**	
Default-CoNLL07	U-TED	0.9444†		**0.9453**†
	L-TED	**0.9266**†		0.9260†
oldLTH-CoNLL07	U-TED		**0.9519**	0.9490
	L-TED		**0.9323**	0.9283
default-oldLTH-CoNLL	U-TED	0.9464†	**0.9515**	0.9471†
	L-TED	0.9281†	**0.9336**	0.9280†

Train		**CoNLL07**	**Functional**	**Lexical**
Gold				
CoNLL07	UAS	**0.8917**	0.8054	0.6986
	LAS	**0.8736**	0.7895	0.6831
	U-TED	**0.9474**	0.9357	0.9237
	L-TED	**0.9233**	0.8960	0.8606
Functional	UAS	0.8040	**0.8970**	0.6110
	LAS	0.7873	**0.8793**	0.5977
	U-TED	0.9347	**0.9466**	0.9107
	L-TED	0.8948	**0.9239**	0.8316
Lexical	UAS	0.7013	0.6138	**0.8823**
	LAS	0.6875	0.6022	**0.8635**
	U-TED	0.9252	0.9132	**0.9500**
	L-TED	0.8623	0.8345	**0.9266**
CoNLL07-Functional	U-TED	0.9473†	0.9473†	
	L-TED	0.9233	**0.9247**	
CoNLL07-Lexical	U-TED	0.9490†		**0.9500**†
	L-TED	0.9253†		**0.9266**†
Functional-Lexical	U-TED		0.9489†	**0.9501**†
	L-TED		0.9266†	**0.9267**†
CoNLL07-Functional-Lexical	U-TED	0.9489†	0.9489†	**0.9501**†
	L-TED	0.9254†	0.9266†	**0.9267**†

Table 1: Cross-experiment dependency parsing evaluation for MaltParser trained on multiple schemes. We report standard LAS scores and TEDEVAL global average metrics. Boldface results outperform the rest of the results reported in the same row. The † sign marks pairwise results where the difference is not statistically significant.

Train		**Default**	**Old LTH**	**CoNLL07**
Gold				
Default	UAS	**0.9173**	0.6085	0.7709
	LAS	**0.8833**	0.4780	0.6414
	U-TED	**0.9513**	0.8903	0.9236
	L-TED	**0.9249**	0.7727	0.8424
Old LTH	UAS	0.6078	**0.8952**	0.6415
	LAS	0.4809	**0.8471**	0.5669
	U-TED	0.8960	**0.9550**	0.9096
	L-TED	0.7823	**0.9224**	0.8170
CoNLL07	UAS	0.7767	0.6517	**0.8991**
	LAS	0.6504	0.5725	**0.8709**
	U-TED	0.9289	0.9087	**0.9479**
	L-TED	0.8502	0.8159	**0.9208**
Default-oldLTH	U-TED	**0.9533**	0.9515	
	L-TED	**0.9289**	0.9224	
Default-CoNLL	U-TED	**0.9474**†		0.9460†
	L-TED	**0.9281**		0.9238
OldLTH-CoNLL	U-TED		0.9479	**0.9493**
	L-TED		0.9234	**0.9258**
Default-OldLTH-CoNLL	U-TED	**0.9492**†	0.9461	0.9480†
	L-TED	**0.9298**	0.9241†	0.9258†

Train		**CoNLL07**	**Functional**	**Lexical**
Gold				
CoNLL07	UAS	**0.8991**	0.8077	0.7018
	LAS	**0.8709**	0.7902	0.6804
	U-TED	**0.9479**	0.9373	0.9221
	L-TED	**0.9208**	0.8955	0.8505
Functional	UAS	0.8083	**0.8978**	0.6150
	LAS	0.7895	**0.8782**	0.5975
	U-TED	0.9356	**0.9476**	0.9092
	L-TED	0.8929	**0.9226**	0.8218
Lexical	UAS	0.6997	0.6161	**0.8826**
	LAS	0.6835	0.6034	**0.8491**
	U-TED	0.9259	0.9152	**0.9483**
	L-TED	0.8593	0.8340	**0.9160**
CoNLL-Functional	U-TED	0.9479†	**0.9487**†	
	L-TED	0.9209	**0.9237**	
CoNLL-Lexical	U-TED	**0.9497**		0.9483
	L-TED	**0.9228**		0.9161
Functional-Lexical	U-TED		**0.9504**	0.9483
	L-TED		**0.9258**	0.9161
CoNLL-Functional-Lexical	U-TED	0.9498	**0.9504**†	0.9483†
	L-TED	0.9229	**0.9258**	0.9161

Table 2: Cross-experiment dependency parsing evaluation for the MST parser trained on multiple schemes. We report standard LAS scores and TEDEVAL global average metrics. Boldface results outperform the rest of the results reported in the same row. The † sign marks pairwise results where the difference is not statistically significant.

ID	Description
Default	The LTH conversion default settings
OldLTH	The conversion used in Johansson and Nugues (2007)
CoNLL07	The conversion used in the CoNLL shared task (Nivre et al., 2007a)
Lexical	Same as **CoNLL**, but selecting only lexical heads when a choice exists
Functional	Same as **CoNLL**, but selecting only functional heads when a choice exists

Table 3: LTH conversion schemes used in the experiments. The LTH conversion settings in terms of the complete feature-value pairs associated with the LTH parameters in different schemes are detailed in the supplementary material.

The Default, OldLTH and CoNLL schemes mainly differ in their coordination structure, and the Functional and Lexical schemes differ in their selection of a functional and a lexical head, respectively. All schemes use the same inventory of labels.[7] The LTH parameter settings for the different schemes are elaborated in the supplementary material.

The Setup We use two different parsers: (i) MaltParser (Nivre et al., 2007b) with the arc eager algorithm as optimized for English in (Nivre et al., 2010) and (ii) MSTParser with the second-order projective model of McDonald and Pereira (2006). Both parsers were trained on the different instances of sections 2-21 of the PTB obeying the different annotation schemes in Table 3. Each trained model was used to parse section 23. All non-projective dependencies in the training and gold sets were projectivized prior to training and parsing using the algorithm of Nivre and Nilsson (2005). A more principled treatment of non-projective dependency trees is an important topic for future research. We evaluated the parses using labeled and unlabeled attachment scores, and using our TEDEVAL software package.

Evaluation Our TEDEVAL software package implements the pipeline described in Section 3. We convert all parse and gold trees into functional trees using the algorithm defined in Section 3, and for each pair of parsing experiments we calculate a shared gold standard using generalization determined through a chart-based greedy algorithm.[8] Our scoring procedure uses the TED algorithm defined by Zhang and Shasha (1989).[9] The unlabeled score is obtained by assigning $cost(e) = 0$ for every e relabeling operation. To calculate pairwise statistical significance we use a shuffling test with 10,000 iterations (Cohen, 1995). A sample of all files in the evaluation pipeline for a subset of 10 PTB sentences is available in the supplementary materials.[10]

Results Table 1 reports the results for the inter- and cross-experiment evaluation of parses produced by MaltParser. The left hand side of the table presents the parsing results for a set of experiments in which we compare parsing results trained on the Default, OldLTH and CoNLL07 schemes. In a second set of experiments we compare the CoNLL07, Lexical and Functional schemes. Table 2 reports the evaluation of the parses produced by MSTParser for the same experimental setup. Our goal here is not to compare the parsers, but to verify that the effects of switching from LAS to TEDEVAL are robust across parsing algorithms.

In each of the tables, the top three groups of four rows compare results of parsed dependency trees trained on a particular scheme against gold trees of the same and the other schemes. The next three groups of two rows report the results for comparing pairwise sets of experiments against a generalized gold using our proposed procedure. In the last group of two rows we compare all parsing results against a single gold obtained through a three-way generalization.

As expected, every parser appears to perform at its best when evaluated against the scheme it was trained on. This is the case for both LAS and TEDEVAL measures and the performance gaps are statistically significant. When moving to pairwise evaluation against a single generalized gold, for instance, when comparing CoNLL07 to the Default settings, there is still a gap in performance, e.g., between OldLTH and CoNLL07, and between OldLTH and Default. This gap is however a lot smaller and is not always statistically significant. In fact, when evaluating the effect of linguistically disparate annotation variations such as Lexical and Functional on the performance of MaltParser, Table 1 shows that when using TEDEVAL and a generalized gold the performance gaps are small and statistically insignificant.

Moreover, observed performance trends when evaluating individual experiments on their original training scheme may change when compared against a generalized gold. The Default scheme, for MaltParser, appears better than OldLTH when both are evaluated against their training schemes. But looking at the pairwise-evaluated experiments, it is the other way round (the difference is smaller, but statistically significant). In evaluating against a three-way

[7] In case the labels are not taken from the same inventory, e.g., subjects in one scheme are marked as SUB and in the other marked as SBJ, it is possible define a a set of zero-cost operation types — in such case, to the operation **relabel**(SUB,SBJ) — in order not to penalize string label discrepancies.

[8] Our algorithm has space and runtime complexity of $O(n^2)$.

[9] Available via `http://web.science.mq.edu.au/~swan/howtos/treedistance/`

[10] The TEDEVAL software package is available via `http://stp.lingfil.uu.se/~tsarfaty/unipar`

generalization, all the results obtained for different training schemes are on a par with one another, with minor gaps in performance, rarely statistically significant. This suggests that apparent performance trends between experiments when evaluating with respect to the training schemes may be misleading.

These observations are robust across parsing algorithms. In each of the tables, results obtained against the training schemes show significant differences whereas applying our cross-experimental procedure shows small to no gaps in performance across different schemes. Annotation variants which seem to have crucial effects have a relatively small influence when parsed structures are brought into the same formal and theoretical common ground for comparison. Of course, it may be the case that one parser is better trained on one scheme while the other utilizes better another scheme, but objective performance gaps can only be observed when they are compared against shared linguistic content.

5 Discussion and Extensions

This paper addresses the problem of cross-experiment evaluation. As it turns out, this problem arises in NLP in different shapes and forms; when evaluating a parser against different annotation schemes, when evaluating parsing performance across parsers and different formalisms, and when comparing parser performance across languages. We consider our contribution successful if after reading it the reader develops a healthy suspicion to blunt comparison of numbers across experiments, or better yet, across different papers. Cross-experiment comparison should be a careful and well thought-through endeavor, in which we retain as much information as we can from the parsed structures, avoid lossy conversions, and focus on an object of evaluation which is agreed upon by all variants.

Our proposal introduces one way of doing so in a streamlined, efficient and formally worked out way. While individual components may be further refined or improved, the proposed setup and implementation can be straightforwardly applied to cross-parser and cross-framework evaluation. In the future we plan to use this procedure for comparing constituency and dependency parsers. A conversion from constituency-based trees into functional trees is straightforward to define: simply replace the node labels with the grammatical function of their dominating arc – and the rest of the pipeline follows.

A pre-condition for cross-framework evaluation is that all representations encode the same set of grammatical relations by, e.g., annotating arcs in dependency trees or decorating nodes in constituency trees. For some treebanks this is already the case (Nivre and Megyesi, 2007; Skut et al., 1997; Hinrichs et al., 2004) while for others this is still lacking. Recent studies (Briscoe et al., 2002; de Marneffe et al., 2006) suggest that evaluation through a single set of grammatical relations as the common denominator is a linguistically sound and practically useful way to go. To guarantee extensions for cross-framework evaluation it would be fruitful to make sure that resources use the same set of grammatical relation labels across different formal representation types. Moreover, we further aim to inquire whether we can find a single set of grammatical relation labels that can be used across treebanks for multiple languages. This would then pave the way for the development of cross-language evaluation procedures.

6 Conclusion

We propose an end-to-end procedure for comparing dependency parsing results across experiments based on three steps: (i) converting dependency trees to functional trees, (ii) generalizing functional trees to harmonize information from different sources, and (iii) using distance-based metrics that take the different sources into account. When applied to parsing results of different dependency schemes, dramatic gaps observed when comparing parsing results obtained in isolation decrease or dissolve completely when using our proposed pipeline.

Acknowledgments We thank the developers of the LTH and TED software who made their code available for our use. We thank Richard Johansson for providing us with the LTH parameter settings of existing dependency schemes. We thank Ari Rappoport, Omri Abend, Roy Schwartz and members of the NLP lab at the Hebrew University of Jerusalem for stimulating discussion. We finally thank three anonymous reviewers for useful comments on an earlier draft. The research reported in the paper was partially funded by the Swedish Research Council.

References

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

Ezra Black, Steven P. Abney, D. Flickenger, Claudia Gdaniec, Ralph Grishman, P. Harrison, Donald Hindle, Robert Ingria, Frederick Jelinek, Judith L. Klavans, Mark Liberman, Mitchell P. Marcus, Salim Roukos, Beatrice Santorini, and Tomek Strzalkowski. 1991. Procedure for quantitatively comparing the syntactic coverage of English grammars. In E. Black, editor, *Proceedings of the workshop on Speech and Natural Language*, HLT, pages 306–311. Association for Computational Linguistics.

Ted Briscoe, John Carroll, Jonathan Graham, and Ann Copestake. 2002. Relational evaluation schemes. In *Proceedings of LREC Workshop"Beyond Parseval – Towards improved evaluation measures for parsing systems"*.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of CoNLL-X*, pages 149–164.

Ekaterina Buyko and Udo Hahn. 2010. Evaluating the impact of alternative dependency graph encodings on solving event extraction tasks. In *Proceedings of EMNLP*, pages 982–992.

John Carroll, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of LREC*, pages 447–454.

Daniel Cer, Marie-Catherine de Marneffe, Daniel Jurafsky, and Christopher D. Manning. 2010. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In *Proceedings of LREC*.

Jinho D. Choi and Martha Palmer. 2010. Robust constituent-to-dependency conversion for English. In *Proceedings of TLT*.

Paul Cohen. 1995. *Empirical Methods for Artificial Intelligence*. The MIT Press.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, pages 449–454.

Martin Emms. 2008. Tree-distance and some other variants of evalb. In *Proceedings of LREC*.

Erhard Hinrichs, Sandra Kübler, Karin Naumann, Heike Telljohan, and Julia Trushkina. 2004. Recent development in linguistic annotations of the TüBa-D/Z Treebank. In *Proceedings of TLT*.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for English. In *Proceedings of NODALIDA*.

Sandra Kübler and Heike Telljohann. 2002. Towards a dependency-oriented evaluation for partial parsing. In *Proceedings of LREC Workshop"Beyond Parseval – Towards improved evaluation measures for parsing systems"*.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Number 2 in Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, pages 81–88.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98.

Igor Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

Yusuke Miyao, Rune Sætre, Kenji Sagae, Takuya Matsuzaki, and Jun'ichi Tsujii. 2008. Task-oriented evaluation of syntactic parsers and their representations. In *Proceedings of ACL*, pages 46–54.

Joakim Nivre and Beata Megyesi. 2007. Bootstrapping a Swedish Treebank using cross-corpus harmonization and annotation projection. In *Proceedings of TLT*.

Joakim Nivre and Jens Nilsson. 2005. Pseudo projective dependency parsing. In *Proceeding of ACL*, pages 99–106.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of COLING*, pages 64–70.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007a. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre, Jens Nilsson, Johan Hall, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007b. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(1):1–41.

Joakim Nivre, Laura Rimell, Ryan McDonald, and Carlos Gómez-Rodríguez. 2010. Evaluation of dependency parsers on unbounded dependencies. pages 813–821.

Owen Rambow. 2010. The Simple Truth about Dependency and Phrase Structure Representations: An Opinion Piece. In *Proceedings of HLT-ACL*, pages 337–340.

Ines Rehbein and Josef van Genabith. 2007. Why is it so difficult to compare treebanks? Tiger and TüBa-D/Z revisited. In *Proceedings of TLT*, pages 115–126.

Brian Roark. 2002. Evaluating parser accuracy using edit distance. In *Proceedings of LREC Workshop"Beyond Parseval – Towards improved evaluation measures for parsing systems"*.

Roy Schwartz, Omri Abend, Roi Reichart, and Ari Rappoport. 2011. Neutralizing linguistically problematic annotations in unsupervised dependency parsing evaluation. In *Proceedings of ACL*, pages 663–672.

Stuart M. Shieber. 1986. *An Introduction to Unification-Based Grammars*. Center for the Study of Language and Information.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word-order languages. In *Proceedings of the fifth conference on Applied natural language processing*, pages 88–95.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceeding of IWPT*, pages 195–206.

Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. In *SIAM Journal of Computing*, volume 18, pages 1245–1262.

Arnold M. Zwicky. 1993. Heads, bases, and functors. In G.G. Corbett, N. Fraser, and S. McGlashan, editors, *Heads in Grammatical Theory*, pages 292–315. Cambridge University Press.