Natural Language Engineering **4** (2): 97–114. Printed in the United Kingdom

© 1998 Cambridge University Press

A dependency-based method for evaluating broad-coverage parsers

DEKANG LIN

Department of Computer Science, University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2
e-mail: lindek@cs.umanitoba.ca

(*Received 3 March 1998*)

Abstract

With the emergence of broad-coverage parsers, quantitative evaluation of parsers becomes increasingly more important. We propose a dependency-based method for evaluating broad-coverage parsers that offers more meaningful performance measures than previous approaches. We also present a structural pattern-matching mechanism that can be used to eliminate inconsequential differences among different parse trees. Previous evaluation methods have only evaluated the overall performance of parsers. The dependency-based method can also evaluate parsers with respect to different kinds of grammatical relationships or different types of lexical categories. An algorithm for transforming constituency trees into dependency trees is presented, which makes the evaluation method applicable to both constituency grammars and dependency grammars.

1 Introduction

With the emergence of broad-coverage parsers (Lin 1993; Lin 1994; Magerman 1995; Collins 1996; Collins 1997), quantitative evaluation of parsers becomes increasingly more important. First, such an evaluation scheme is necessary to quantitatively measure the progress in the field of broad-coverage parsing, and to compare and evaluate different parsing techniques. Secondly, in the development of a broad-coverage parser, it is usually very difficult to predict the consequences of a change to the parser or the grammar. An attempt to extend the coverage that is motivated by a few examples may well cause the parser to badly overgenerate and/or lose coverage in other areas. Only quantitative evaluation based on a large set of sentences can really tell whether a modification is a step forward or not. Thirdly, efficiency and coverage are often conflicting goals for a parser. A meaningful trade-off can only be arrived at if both of them can be measured precisely. Finally, quantitative evaluation may provide crucial information for determining the suitability of a parser in a particular domain and/or for a particular task.

It is generally accepted that quantitative parser evaluation should be conducted by comparing the parser-generated parse trees (we call them *answers*) with manually constructed parse trees (we call them *keys*). However, how such comparison should

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

be performed is still subject to debate. Several proposals have been put forward (Black et al. 1991; Black, Lafferty and Roukos 1992; Magerman 1994), all of which are based on comparing phrase boundaries between answers and keys. We (Lin 1995) proposed a dependency-based evaluation scheme and explained the advantages of dependency-based method over previous parser evaluation methods. We also proposed an algorithm to translate constituency structures into dependency structures so that the evaluation scheme applies to both kinds of parse trees. In this paper, we further develop, extend and refine the proposal in Lin (1995).

In the remainder of the paper, we first review previous approaches to quantitative evaluation of broad-coverage parsers and identify their problems. We then introduce a dependency-based parser evaluation scheme in section 3. Section 4 discusses the transformation of constituency trees into dependency trees so that the evaluation method can also be applied to constituency trees. Sections 5 and 6 address the issues of word segmentation and selective evaluation. Finally, in section 7, we conclude that the dependency-based method presented in this paper offers a better approach to the evaluation of broad-coverage parsers than the phrase boundary methods.

2 Previous approaches

The main obstacle in parser evaluation is that different grammatical theories or different grammars within a particular theory may have different notions of what constitutes a correct parse. The best one can do is to compare a common divisor, i.e. information that is encoded in all types of parse trees. The common divisor used in previous parser evaluation methods is the set of phrase boundaries identified by parse trees.

Given a node in a parse tree, the sequence of words dominated by the node forms a phrase and the boundary of the phrase can be denoted by an integer interval $[i, j]$, where i is the index of the first word in the phrase and j is the index of the last word in the phrase. For example, the parse tree in (1) contains three phrase boundaries: [0,2], [1,1] and [1,2]:

(1) [They [[came] yesterday]]

Previous evaluation schemes can be classified as either *phrase-level* or *sentence-level* evaluation. Phrase-level evaluation computes the following measures:

Recall and precision: the phrase boundaries in the answer and the key are treated as two sets (A and K, respectively). Recall is defined as the percentage of phrase boundaries in the key that are also found in the answer ($\frac{|A \cap K|}{|K|}$). Precision is defined as the percentage of phrase boundaries in the answer that are also found in the key ($\frac{|A \cap K|}{|A|}$).

Number of crossing-brackets: a pair of phrase boundaries $[i, j]$ and $[i', j']$ are said to be crossing brackets if $i < i' \leq j < j'$ or $i' < i \leq j' < j$. Parsers can be evaluated by the average pairs of crossing brackets per sentence.

For example, suppose, (1) is the key and (2) is the answer:

(2) [[They [came]] [yesterday]]

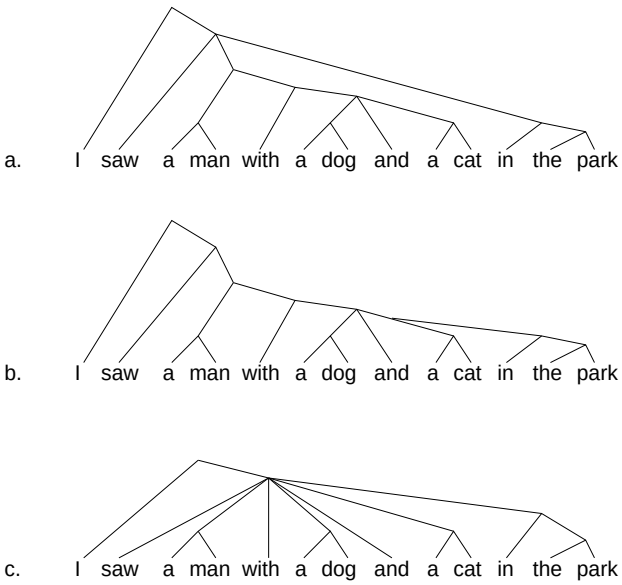
https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

The phrase boundaries in (2) are: [0,2], [1,1], [0,1] and [2,2]. Thus, the scores for (2) are: precision = $\frac{2}{4}$ = 50%, recall = $\frac{2}{3}$ = 66.7% and there is one pair of crossing brackets: [0,1] in the key (1) and [1,2] in the answer (2). These scores have to be considered together to be meaningful. For example, treating the sentence "they came yesterday" as a flat list of words [they came yesterday] would achieve 0-crossing-brackets and 100% precision. However, the recall is quite low ($\frac{1}{3}$=33.3%).

In sentence-level evaluation, a sentence is considered to be correctly parsed if certain criteria have been met. In Black *et al.* (1992), the correctness criterion is that the number of crossing-brackets is 0. In Magerman (1994), a sentence is correctly parsed if both precision and recall are 100%. The scores assigned to parsers are their percentages of correctly parsed sentences in the test corpus. For example, since (2) contains a pair of crossing-brackets when compared to (1), it is counted as a wrongly parsed sentence. If this is the only sentence in the evaluation corpus, the error rate would be 100%.

A problem with the phrase-level scores is that a single error may be counted multiple times. For example, suppose (3a) is the key and (3b), (3c) are answers returned by two parsers:

(3)



The only difference between (3a) and (3b) is that the prepositional phrase [in [the park]] is the sister of 'saw' in (3a), but the sister of 'a cat' in (3b). However, because of this single attachment error, there are three pairs of crossing brackets between (3a) and (3b):

- [a dog and a cat] vs. [a cat in the park]
- [with a dog and a cat] vs. [a dog and a cat in the park]
- [a man with a dog and a cat] vs. [with a dog and a cat in the park]

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

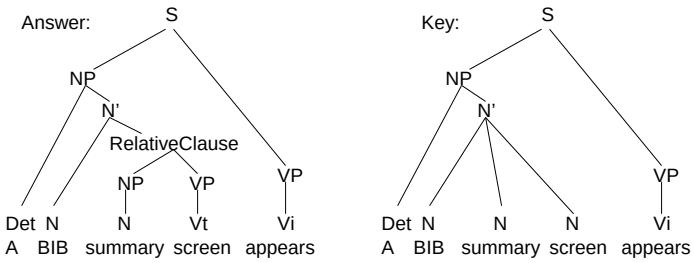


Fig. 1. Two parse trees of "A BIB summary screen appears".

The recall of (3b) is $\frac{6}{10} = 60\%$ and the precision is $\frac{7}{11} = 63.6\%$. In contrast, (3c) is a very shallow parse of the sentence. However, when compared with (3a), it has no crossing brackets, a perfect precision (100%) and a better recall ($\frac{7}{10} = 70\%$) than (3b). Intuitively, the structure (3b) has a lot more in common with (3a) than (3c) does. Yet, (3b) scored much worse than (3c) according to both the number of crossing-brackets and recall/precision of phrase boundaries.

In sentence-level evaluation, the parse tree for a whole sentence is consider to be an error if it contains a pair of crossing brackets or a missing/spurious phrase. Although this solves the problem of error propagation, it suffers from the opposite problem: an answer with a single mistake is treated the same as an answer riddled with errors. Since how much a parse tree deviates from the key greatly influences the chance of the sentence being interpreted correctly, the evaluation scheme should take the degree of deviation into account.

Another problem with phrase boundary based evaluation is that it measures the performance of parsers as if the purpose of the parsers is to retrieve the phrase boundaries from sentences. However, the purpose of parsing is usually to facilitate semantic interpretation. No semantic interpretation algorithm makes direct use of phrase boundaries. Therefore, it is hard to predict how useful a parse is even if it marks all the phrase boundaries correctly.

Consider the two parse trees 'Answer' and 'Key' in Figure 1. The answer treats the word 'screen' as a verb and 'summary screen' as a relative clause modifying the noun 'BIB'. The answer is obviously a very poor parse of the sentence. It is unlikely that the sentence can be interpreted correctly according to this analysis. However, according to the phrase-boundary scores proposed in Black *et al.* (1991), the answer has no crossing brackets, 100% recall and 75% precision, scores which usually indicate high quality parses.

A third problem with phrase-boundary based evaluation is that it only addresses the overall performance. However, we may need to answer more detailed questions such as:

- What is the performance of a parser with regard to predicate-argument relationships?
- What is the percentage of correctly attached prepositional phrases?

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

- What would the performance be if prepositional attachment errors were ignored?
- How well does a parser handle conjunctions?
- How well does a parser deal with commas and quotation marks?

To answer these questions, one must be able to selectively evaluate different aspects of a parser. None of the previous evaluation schemes allow that.

Answers to these questions may help us determine the fitness of a parser for a particular domain or task. Suppose one parser performs better on predicate-argument relations than another, but has much weaker performance on prepositional phrase attachments and consequently weaker overall performance; the former may still be more suitable than the latter for machine translation between two languages where the attachment ambiguities are carried over from the source language to the target language.

The detailed evaluation results may also help the parser/grammar developers to diagnose problems within parsers/grammars. For example, by focusing on the differences between answers and keys on relative clauses, it is much easier to discover what kinds of relative clauses were not adequately covered.

3 Dependency-based evaluation

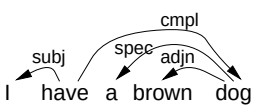
In this section, we present a dependency-based parser evaluation scheme. Instead of comparing phrase boundaries, we compare the dependency relationships in answers and keys.

3.1 Dependency trees

A dependency relationship (Hays 1964; Hudson 1984; Mel'čuk 1987) is an asymmetric binary relationship between a word called a *head* (or governor, parent), and another word called a *modifier* (or dependent, daughter). Dependency grammars represent sentence structures as a set of dependency relationships. Normally, the dependency relationships form a tree that connects all the words in a sentence. A word in the sentence may have several modifiers, but each word may modify at most one word. The root of the dependency tree does not modify any word. It is also called the head of the sentence.

For example, (4) is a dependency structure of a sentence. The head of the sentence is 'have'. There are four pairs of dependency relationships, depicted by four arcs from heads to modifiers:

(4)



https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

We use a list of tuples to represent a dependency tree. Each tuple represents a node in the dependency tree and has the following format:

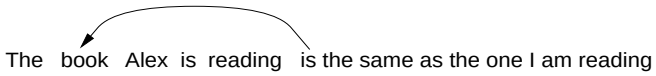
```
(word category [head] [relationship])
```

where `word` is the word represented by the node and `category` is the lexical category of `word`. The fields `head` and `relationship` are optional. The `head`-field specifies which word is modified by `word`. The `relationship`-field is a label assigned to the dependency relationship, for example, `subj` (subject), `adjn` (adjunct), `cmpl` (complement), `spec` (specifier), etc.

The `head`-field consists of a position indicator followed by a word. The position indicator takes one of the following values: {`>`, `<`, `>>`, `<<`, `>>>`, ...}. If the `head`-field is '$\underbrace{\texttt{>...>}}_{n}$ `w`', the head of `word` is the nth occurrence of `w` to the right of the modifier. The meaning of '$\underbrace{\texttt{<...<}}_{n}$ `w`' is similarly defined in the opposite direction.

For example, in the following sentence:

(5)



the word 'book' modifies the second 'is' to its right. Its `head`-field in the dependency tree should be: '`>> is`'.

The dependency tree in (4) is represented by the following tuples:

(6)

Modifier	Category	Head	Type
I	N	> have	subj
have	V		
a	Det	> dog	spec
brown	Adj	> dog	adjn
dog	N	< have	comp

In Grefensteette (1996) and Ting and Shiuan (1996), words in a sentence are assigned numerical indexes. Dependency relationships between words are specified with their indexes. Compared with their notation, the format we used is easier for humans to manually construct or correct.

In Carroll and Charniak (1992), dependency grammars are treated as highly restricted forms of context-free grammars, in which all the rules must be of the form

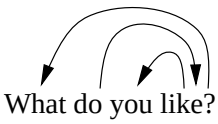
$$\overline{X} \longrightarrow \alpha X \beta$$

where X is a lexical category, α and β are (possibly empty) sequences of non-lexical categories. Every non-lexical category $\overline{X}$ must uniquely correspond to exactly one lexical category X. Not all dependency structures can be represented by dependency grammars in Carroll and Charniak (1992), though. For example, the dependency tree in (7a) can be written in our format as (7b). It can also be written in the formats used in Grefensteette (1996) and (Ting and Shiuan 1996). However, it cannot be represented by Carroll and Charniak (1992).

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

(7)

a.



b.

```
what  ~ N     < like  whn
do    ~ Aux
you   ~ N     < like  subj
like  ~ V_N   > do    pred
```

3.2 Evaluation measures and procedure

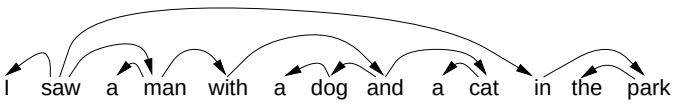
Once the answer and the key are both represented as dependency trees, we can compare them on a word-by-word basis and calculate two scores:

recall: the percentage of dependency relationships in the key that are also found in the answer.

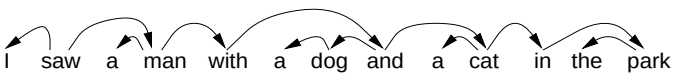
precision: the percentage of dependency relationships in the answer that are also found in the key.

(8)

a. Key:



b. Answer:



c.

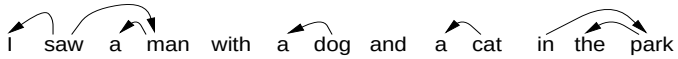
Key: (8a)				Answer: (8b)				correct
word	cate	head	rel	word	cate	head	rel	
I	N	> saw	subj	I	N	> saw	subj	yes
saw	V			saw	V			
a	Det	> man	spec	a	Det	> man	spec	yes
man	N	< saw	cmpl	man	N	< saw	cmpl	yes
with	P	< man	adjn	with	P	< man	adjn	yes
a	Det	> dog	spec	a	Det	> dog	spec	yes
dog	N	> and	conj	dog	N	> and	conj	yes
and	Conj	< with	cmpl	and	Conj	< with	cmpl	yes
a	Det	> cat	spec	a	Det	> cat	spec	yes
cat	N	< and	conj	cat	N	< and	conj	yes
in	P	< saw	adjn	in	P	< cat	adjn	no
the	Det	> park	spec	the	Det	> park	spec	yes
park	N	< in	cmpl	park	N	< in	cmpl	yes

For example, there are 12 dependency relationships in key (8a), and 12 dependency relationships in answer (8a). When they are compared in (8c), 11 dependency relationships are in both the key and the answer. Therefore, the recall and precision

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

(9)

a. Answer:



b.

Key: (8a)				Answer: (9b)				correct
word	cate	head	rel	word	cate	head	rel	
I	N	> saw	subj	I	N	> saw	subj	yes
saw	V			saw	V			
a	Det	> man	spec	a	Det	> man	spec	yes
man	N	< saw	cmpl	man	N	< saw	cmpl	yes
with	P	< man	adjn	with	P			
a	Det	> dog	spec	a	Det	> dog	spec	yes
dog	N	> and	conj	dog	N			
and	Conj	< with	cmpl	and	Conj			
a	Det	> cat	spec	a	Det	> cat	spec	yes
cat	N	< and	conj	cat	N			
in	P	< saw	adjn	in	P			
the	Det	> park	spec	the	Det	> park	spec	yes
park	N	< in	cmpl	park	N	< in	cmpl	yes

of (8b) are both $11/12 = 92\%$. In contrast, the shallow dependency tree (9a) contains only 7 dependency relationships, all of them are also in the key. Therefore, the precision of (9a) is 100% and its recall is $7/12 = 58\%$. Such a high precision and low recall scores correctly reflects the nature of shallow parses.

The dependency-based evaluation algorithm is shown in (10). It ignores the labels assigned to the dependency relationships. This is analogous to ignoring constituency labels in phrase boundary based evaluation. If both the answer and the key use the same set of dependency labels or a mapping between equivalent labels can be established, labels can be incorporated into the evaluation algorithm by requiring not only that a word modifies the same word in the answer as in the key, but also that the dependency relationships have the same or equivalent labels. In section 6, we will discuss the use of labels in selective evaluation.

(10)
```
Score evaluate(DepTree key, DepTree ans)
{
    Score score;
    int keyCount = 0, ansCount = 0, correct = 0;
    for each word in key {
        DepNode keyHead = head(key, word);
        DepNode ansHead = head(ans, word);
        if (ansHead is not NIL)
            ansCount = ansCount+1;
        if (keyHead is not NIL)
            keyCount = keyCount+1;
        if (neither ansHead nor keyHead is NIL and
            keyHead and ansHead are the same word)
            correct = correct + 1;
    }
```

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

```
    score.recall = correct/keyCount;
    score.precision = correct/ansCount;
    return score;
}
```

When both the answer and the key are full parses, both of them have $N - 1$ dependencies, where N is the number of words in the sentence. That means that the precision and recall value will be the same. Therefore, in our earlier proposal (Lin 1995), the ratio between the number of correct dependencies and the number of words was adopted as the evaluation metric. However, a number of situations exist in which the number of dependency relationships in the answer may not be the same as in the key:

- Due to differences in tokenization, the answer and key may be segmented into different numbers of words (cf. section 5).
- When we selectively evaluate different aspects of a parser, only a subset of the dependencies participates in the evaluation (cf. section 6).
- Although it is safe to assume that the key is always a fully connected dependency tree, the answer may consist of several fragmentary dependency trees, as in (9a).

The precision and recall measures are more flexible under these circumstances.

Compared with the number of crossing-brackets and the recall/precision of phrase boundaries, the recall/precision measures of dependency relationships are intuitively more meaningful. Since the phrase boundaries themselves are not used directly in semantic interpretation, it is hard to predict how missing or spurious phrase boundaries affect semantic interpretation. On the other hand, since semantic dependencies are embedded in syntactic dependencies, the semantic interpretation process should be more sensitive to the percentage of correct syntactic dependencies than the percentage of correct phrase boundaries.

As another example, the constituency trees in Figure 1 are converted to the following dependency trees:

Key				Answer				corr
word	cate	head	rel	word	cate	head	rel	
a	Det	> screen	det	a	Det	> bib	det	no
bib	N	> summary	nn	bib	N	> appeared	subj	no
summary	N	> screen	nn	summary	N	> screen	subj	no
screen	V	> appeared	subj	screen	V	< summary	rel	no
appeared	V			appeared	V			

While the phrase boundary method yielded 100% recall, 75% precision for the answer in Figure 1, the scores obtained by the dependency-based method are 0% recall and 0% precision, which are appropriate for a very bad parse.

4 Converting constituency trees into dependency trees

Since the evaluation procedure (10) takes dependency trees as inputs, whereas most broad-coverage parsers and treebanks (Marcus, Santorini and Marcinkiewicz 1993;

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

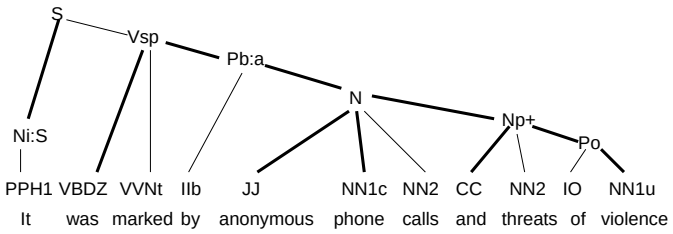


Fig. 2. A constituency tree

Sampson 1995) use constituency grammars, a crucial issue that must be resolved is how to apply the method to constituency grammars.

Conversion from dependency tree to constituency tree goes back as early as Gaifman (1965). In this section, we present an algorithm for transforming the constituency trees into dependency trees. If one or both of the key and the answer are constituency trees, they can be transformed into dependency trees, which are then evaluated.

The transformation from constituency to dependency tree is based on the following assumption:

(11) **Single Head Assumption:** Each constituent in a constituency tree has exactly one head word (also known as the lexical head).

Certain structures seem to require multiple heads. A typical example is conjunction. When a conjunction is the object of a verb, every conjunct must be compatible with the verb. Since the selectional constraints between verbs and their objects are normally stated in terms of the features of the verbs and the features of the heads of the objects, the head of a conjunction need to include the heads of all the conjuncts. However, in parser evaluation, we are only concerned with whether the conjunctions have the same set of conjuncts in the answer as in the key. It does not matter whether the connective (e.g. 'and', 'or') is the head, or the head of first (or last) conjunct is the head, as long as it allows us to identify the set of conjuncts.

Given a non-leaf node X in a constituency tree, its **head child** is defined to be a child node of X that is or dominates the head word of X. For example, in Figure 2, if 'marked' is the head word of S, Vsp is the head child of S. The Single Head Assumption entails that every non-leaf node has exactly one head child.

The links in a constituency tree can therefore be classified into two categories:

head-to-head links: A head-to-head link is one that is between a node and its head child;

head-to-modifier links: all other links are head-to-modifier links.

For example, in Figure 2, the thick links are head-to-modifier links, the thin ones are head-to-head links.

A constituency tree can be converted into a dependency tree by collapsing all the nodes on a chain of head-to-head links into a single node and labeling it with the label of the leaf node. Every head-to-modifier link corresponds to a dependency link.

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

Since the classification of head-to-head vs. head-to-modifier links requires the recognition of head child of each non-leaf node in the constituency tree, the crucial question in the transformation from constituency tree into dependency tree is how to determine the head child of a node.

In Magerman (1994) the head child of a node is determined by a decision table. For each type of node, the decision table contains an ordered list of potential candidates for its head child. The labels in the SUSANNE corpus includes markers to indicate the roles played by the constituents as well as indexes to specify the coreference relationships between empty categories and their antecedents. This gives rise to a large number of possible category labels, making Megerman's method inconvenient.

We also used a decision table to determine the head child of a node. However, candidates for head children are specified with regular expressions. Each entry in the decision table determines the head child of one type of nodes in constituency trees. An entry in the decision table is a triple:

```
(parentCond position childCond)
```

where `parentCond` and `childCond` are predicates on nodes in constituency trees and `position` is either `first` or `last`.

Examples of decision table entries are shown in (12). They were part of a decision table that we used to convert the SUSANNE corpus (Sampson 1995) into dependency format.[1]

(12)

parentCond	position	childCond
`(reg-match [NM])`	`last`	`(and (reg-match [NM] P[PN])` `(not (reg-match .*[+-]))`
`(reg-match [VS])`	`last`	`(reg-match V[^+-]*$)`
`(reg-match P[a-z])`	`first`	`(reg-match I CSN)`

The condition `(reg-match pat1 pat2 ...)` is satisfied by strings whose prefix match the regular expression `pat1` or `pat2`, ... The pattern `[NM]` matches any string that begins with the letter `N` or `M`. The `childCond` of in the first entry states that the head child must have a label that begins with letters `N`, `M`, `PN` or `PP` and the label must not contain `+` or `-` which are indicators of subordinate conjuncts in SUSANNE corpus. The head child of a node X is determined in two steps:

- find the first entry in the decision table such that X satisfies `parentCond` of the entry;
- find the first or the last child node (depending on the value of the `position` field of the entry) of X that satisfies `childCond` of the entry.

According to the decision table (12), the head children of the non-leaf nodes in Figure 2 are as follows:

(13)

Node:	S	Ni:S	Vsp	Pb:a	N	Np+	Po
Head Child:	Vsp	PPH1	VVNt	IIb	NN2 calls	NN2 threats	IO

[1] The converted SUSANNE corpus is available at ftp://ftp.cs.umanitoba.ca/pub/lindek/

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

The head child of N in Figure 2 is 'NN2 calls', because:

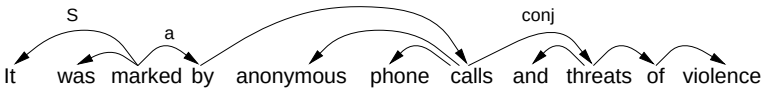
- The label N matches the regular expression '`[NM]`', therefore, the first entry in (12) is used to make the decision.
- The node N has four children: JJ, NN1c, NN2, and Np+, among which NN1c and NN2 satisfies the `childCond`. Since the `position` field in the entry is `last`, NN2 is the head child.

The constituency-to-dependency conversion algorithm is as follows:

(14)
```
DepNode makeDeps(Node root)
{
   if (root is a leaf node)
      return depNode(root);
   Node headChild = findHeadChild(root);
   DepNode lexHead = makeDeps(headChild);
   for each child of root other than headChild {
      DepNode lexHeadOfChild = makeDeps(child);
      String label = getLabel(root, headChild, child);
      addDepRel(label, lexHead, lexHeadOfChild);
   }
   return lexHead;
}
```

The dependency tree converted from the constituency tree in Figure 2 is as follows:

(15)



The functions used in `makeDeps` are explained as follows:

findHeadChild(`node`) returns the head child of `node`.

addDepRel(`label`, `head`, `modifier`) creates a dependency relationship between `head` and `modifier`.

depNode(`node`) creates a dependency tree node corresponding to a leaf node in the constituency tree.

getLabel(`parent, headChild, child`) takes as arguments three nodes in the constituency tree, and returns the label assigned to a dependency link that corresponds to the head-to-modifier link between `parent` and `child`. The node `headChild` is the head child of `parent`.

The labels are determined by a decision table which contains entries in the following format:

```
(label parentCond headChildCond childCond)
```

where `parentCond`, `headChildCond` and `childCond` are predicates on nodes in constituency trees. If `parent` satisfies `parentCond`, `child` satisfies `childCond` and `headChild` satisfies `headChildCond`, the `label` is assigned to the dependency relationship between the head of `parent` and the head of `child`.

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

For example, (16) shows example entries in the decision table for determining the labels of dependency links that we used to convert SUSANNE corpus to dependency format (the predicate `(t)` is satisfied by all nodes):

(16)

label	parentCond	headChildCond	childCond
`conj`	`(t)`	`(t)`	`(reg-match .*[+-])`
`S`	`(t)`	`(t)`	`(reg-match .*:S)`
`a`	`(t)`	`(t)`	`(reg-match .*:a)`

For example, the dependency link from `calls` to `threats` corresponds to the head-to-modifier link from N to Np+ in Figure 2. The head child of N is 'NN2 calls'. Since N, NN2, and Np+ satisfies the three conditions in the first entry of (16), respectively, the dependency link from `calls` to `threats` is labeled `conj`.

In the SUSANNE corpus, some of the roles played by constituents are encoded in the labels of the constituents. For example, if a constituent label contains ':S', the constituent is a surface subject, such as subjects of passive verbs, expletive 'it', etc. Similarly, if ':a' is part of a constituent label (e.g. Pb:a in Figure 2), the constituent is the 'agent of passive' p. 362. This allows the dependency label to be determined solely by the label assigned to the child node of a head-to-modifier link.

5 Dealing with differences in word segmentation

The evaluation algorithm (10) compares the answer and key word by word. This implicitly assumes that the answer and the key segment the input sentence into words in the same way. Unfortunately, this is not always true. Although what constitutes a word is clear in most cases, at least for the English language, there are also many situations where the notion of a word is not well defined, especially with regard to punctuation marks. For example, words like `don't` may be treated as a single word, or as two words: "`do + n't`" or "`don + 't`." Phrases such as "the Houston, Texas.-based Jiffy-Lube" are quite likely to be segmented differently.

The rules for word segmentation may be motivated by the grammatical analysis adopted by parsers or treebanks. Since a parser evaluation scheme should be theory neutral, it should accommodate different types of word segmentation, instead of favoring any one over the others.

To achieve this, the evaluation algorithm (10) is revised to take the intersection of word boundaries in the answer and the key. The intersected boundaries constitute a revised segmentation of words. Each re-segmented word is a sequence of one or more original words in the answer or the key. Figure 3 shows three dependency structures. The short vertical bars represent the word boundaries in each of the parses. The

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

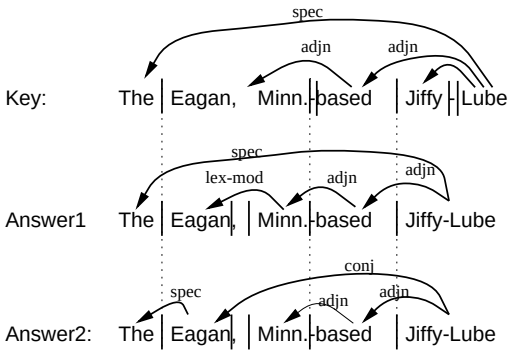


Fig. 3. Re-segmentation of words.

doted lines represent the intersection of the word boundaries. Therefore, when we compare Key with Answer1 or Answer2, the re-segmented sentence consists of 4 words: The Eagan, Mann. -based Jiffy-Lube.

The revised evaluation algorithm that accommodates differences in word segmentation is as follows:

(17)
```
Score evaluate(DepTree key, DepTree answer)
{
    Score score;
    int keyCount = 0, ansCount = 0, correct = 0;

    resegmentedSentence = resegment(key, answer);

    for each segment in resegmentedSentence {
        BagOfSegments keyHeads = externalHeads(segment.key);
        BagOfSegments ansHeads = externalHeads(segment.ans);

        keyCount = keyCount + sizeof(keyHeads);
        ansCount = ansCount + sizeof(ansHeads);
        correct = correct + sizeof(intersect(keyHeads, ansHeads));
    }
    score.recall = correct/keyCount;
    score.precision = correct/ansCount;
    return score;
}
```

The function 'resegment' computes the intersection of the word boundaries and returns the re-segmented sentence. The function 'sizeof' computes the number of elements in a bag. The function 'externalHeads' takes a re-segmented word as an argument. It first computes the bag-union of the heads assigned to the original words in the segment, then removes the heads that are in the same segment as the modifiers and then maps the heads to the re-segmented word to which they belong.

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

The external heads of the four segments in Figure 3 are shown in (18).

(18)

Revised Segments	External Heads		
	Key	Answer1	Answer2
The Eagan, Minn. -based Jiffy-Lube	Jiffy-Lube -based Jiffy-Lube	Jiffy-Lube -based Jiffy-Lube	Eagan, Minn. -based, Jiffy-Lube Jiffy-Lube

As a result, the recall and precision of Answer1 are both 100%, even though it contains a dependency link from 'Minn.' to 'Eagan', which is not found in Key. The scores for Answer2 are: recall $= \frac{2}{3} = 66.7\%$, and precision $= \frac{2}{4} = 50\%$.

Since we excluded dependencies within a single re-segmented word, a caveat is in order. If a parser groups all the words in a sentence as a single segment whenever it failed to parse the sentence, it effectively forces the evaluation program to consider only the sentences it succeeded to parse. One way to prevent this from happening is to impose additional constraints on word segmentation, e.g. "white spaces must be word boundaries".

6 Selective evaluation

In the evaluation algorithm (17), every word (or re-segmented word) is scored. If we only score a subset of the words, we will be able to evaluate different aspects of parsers.

Selective evaluation is achieved by adding two new parameters to the evaluation function (17): `keySelector` and `ansSelector`, which are predicates on nodes in dependency trees. The two predicates are passed to the function 'externalHeads' so that 'externalHeads(`segment, selector`)' only returns the external heads of nodes that satisfy `selector`. When `selector` is NIL, all nodes are consider to satisfy the selector.

The revised evaluation function that allows selective evaluation is shown in (19) in which the underlined portions are the differences between (17) and (19). When both selectors are NIL, the evaluation algorithm (19) behaves in exactly the same way as (17). When one of them is NIL and the other one is not, (19) only scores the words for which the non-NIL selector returns a non-empty set of external heads.

(19)
```
Score evaluate(
  DepTree key, DepTree answer, Pred keySelector, Pred ansSelector
  )
{
   Score score;
   int keyCount = 0, ansCount = 0, correct = 0;

   resegmentedSentence = resegment(key, answer);

   for each segment in resegmentedSentence {
    BagOfSegments keyHeads = externalHeads(segment.key, keySelector);
```

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

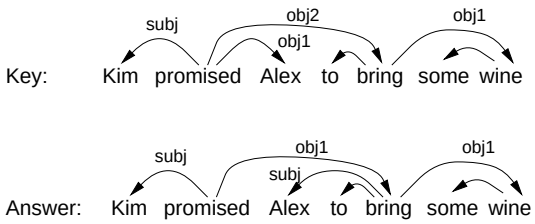
```
        BagOfSegments ansHeads = externalHeads(segment.ans, ansSelector);

        if ((not (keySelector is NIL and empty(ansHeads))) and
            (not (ansSelector is NIL and empty(keyHeads)))) {
            keyCount = keyCount + sizeof(keyHeads);
            ansCount = ansCount + sizeof(ansHeads);
            correct = correct + sizeof(intersect(keyHeads, ansHeads));
        }
    }
    score.recall = correct/keyCount;
    score.precision = correct/ansCount;
    return score;
}
```

Different types of selectors can be used to evaluate different aspects of parser outputs. Assuming that the dependency relationships are labeled, we can use the labels to select a subset of the dependency relationships, such as subject-verb relationship or object-verb relationship.

For example, for the pair of dependency trees in (20), the recall and precision of object-verb relationships are 33.3% and 50% respectively when the distinction between `obj1` and `obj2` is taken into account. Similarly, for the subject-verb relationships the recall is 100%, but the precision is 50%.

(20)



The dependency relationships can also be selectively evaluated according to the properties of the modifiers in the dependency relationship. For example, by selecting all dependency relationships in which the modifier is a preposition, we can evaluate how well a parser attaches prepositional phrases.

In phrase boundary evaluation, it is much more difficult to selectively evaluate different aspect of a parser. For example, the differences between the answer and the key in (21) are: the answer has two spurious phrases: [the box] and [in the box]; the key has a missing phrase: [in the box on the table]. It is not clear how one could infer from these three missing and spurious phrases that there is one prepositional attachment.

(21)

Answer:	[sleep [in [the box]] [on [the table]]]
Key:	[sleep [in [the box [on [the table]]]]]

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

7 Conclusion

We have presented a dependency-based method for evaluating broad-coverage parsers. We also presented an algorithm that transforms constituency trees into dependency trees so that the evaluation method is applicable to both dependency and constituency grammars.

The dependency-based method offers several advantages over previous methods that relied on the comparison of phrase boundaries. Since the semantic dependencies are normally embedded in syntactic dependencies, the recall and precision of syntactic dependencies are much better indicators of how useful the parser outputs are going to be in semantic interpretation. By evaluating a selected subset of dependencies, we can measure parser performance with respect to different types of grammatical relationships, or different types of lexical properties. The results of detailed evaluation may help better assess the suitability of a parser in a particular domain or task. They can also facilitate the diagnosis of problems during the development of parsers and/or grammars.

Acknowledgements

The author is a member of the Institute for Robotics and Intelligent Systems (IRIS) and wishes to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada, the Natural Sciences and Engineering Research Council (NSERC), and the participation of PRECARN Associates Inc. This research has also been partially supported by NSERC Research Grant OGP121338.

References

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B. and Strzalkowski, T. (1991) A procedure for quantitatively comparing the syntactic coverage of English grammars. *Proceedings of Speech and Natural Language Workshop*, pp. 306–311. DARPA.

Black, E., Lafferty, J. and Roukos, S. (1992) Development and evaluation of a broad-coverage probabilistic grammar of English-language computer manuals. *Proceedings of ACL-92*, pp. 185–192. Newark, DE.

Carroll, G. and Charniak, E. (1992) Learning probability dependency grammars from labelled text. *AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, pp. 25–32. Menlo Park, CA. AAAI Press.

Collins, M. J. (1996) A new statistical parser based on bigram lexical dependencies. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pp. 184–191. Santa Cruz, CA.

Collins, M. J. (1997) Three generative, lexicalized models for statistical parsing. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pp. 16–23. Madrid, Spain.

Gaifman, C. (1965) Dependency systems and phrase structure systems. *Infor. & Control* **8**:304–337.

Grefensteette, G. (1996) Using the SEXTANT low-level parser to analyse a software manual corpus. In *Industrial Parsing of Software Manuals*, Sutcliffe, R. F.E., Koch, H.-D. and McElligott, A. (eds.), pp. 139–158. Rodopi.

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press

Hays, D. (1964) Dependency theory: a formalism and some observations. *Language*]**40**: 511–525.

Hudson, R. (1984) *Word Grammar*. Blackwell.

Lin, D. (1993) Principle-based parsing without overgeneration. *Proceedings of ACL–93*, pp. 112–120. Columbus, OH.

Lin, D. (1994) Principar – an efficient, broad-coverage, principle-based parser. *Proceedings of COLING–94*, pp. 482–488. Kyoto, Japan.

Lin, D. (1995) A dependency-based method for evaluating broad-coverage parsers. *Proceedings of IJCAI-95*, pp. 1420–1425. Montreal, Canada.

Magerman, D. (1995) Statistical decision-tree model for parsing. *Proceedings of ACL-96*, pp. 276–283. Cambridge, MA.

Magerman, D. M. (1994) Natural Language Parsing as Statistical Pattern Recognition. *PhD thesis*, Stanford University.

Marcus, M. P., Santorini, B. and Marcinkiewicz, M. A. (1993) Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* **19**(2): 313–330.

Mel'čuk, I. A. (1987) *Dependency syntax: theory and practice*. State University of New York Press.

Sampson, G. R. (1995) *English for the Computer*. Oxford University Press.

Sutcliffe, R. F.E., Koch, H.-D. and McElligott, A. (eds.) (1996) *Industrial Parsing of Software Manuals*. Rodopi.

Ting, C. H. A. and Shiuan, P. (1996) Using a dependency structure parser without any grammar formalism to analyse a software manual corpus. In *Industrial Parsing of Software Manuals*, Sutcliffe, R. F.E., Koch, H.-D. and McElligott, A. (eds.), pp. 159–178. Rodopi.

https://doi.org/10.1017/S1351324998001958 Published online by Cambridge University Press