Incremental Tree Substitution Grammar for Parsing and Sentence Prediction

Federico Sangati and **Frank Keller**
Institute for Language, Cognition, and Computation
School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB, UK
federico.sangati@gmail.com keller@inf.ed.ac.uk

Abstract

In this paper, we present the first incremental parser for Tree Substitution Grammar (TSG). A TSG allows arbitrarily large syntactic fragments to be combined into complete trees; we show how constraints (including lexicalization) can be imposed on the shape of the TSG fragments to enable incremental processing. We propose an efficient Earley-based algorithm for incremental TSG parsing and report an F-score competitive with other incremental parsers. In addition to whole-sentence F-score, we also evaluate the partial trees that the parser constructs for sentence prefixes; partial trees play an important role in incremental interpretation, language modeling, and psycholinguistics. Unlike existing parsers, our incremental TSG parser can generate partial trees that include predictions about the upcoming words in a sentence. We show that it outperforms an n-gram model in predicting more than one upcoming word.

1 Introduction

When humans listen to speech, the input becomes available gradually as the speech signal unfolds. Reading happens in a similarly gradual manner when the eyes scan a text. There is good evidence that the human language processor is adapted to this and works incrementally, i.e., computes an interpretation for an incoming sentence on a word-by-word basis (Tanenhaus et al., 1995; Altmann and Kamide, 1999). Also language processing systems often deal with speech as it is spoken, or text as it is being typed. A dialogue system should start interpreting a sentence while it is being spoken, and a question answering system should start retrieving answers before the user has finished typing the question.

Incremental processing is therefore essential both for realistic models of human language processing and for NLP applications that react to user input in real time. In response to this, a number of incremental parsers have been developed, which use context-free grammar (Roark, 2001; Schuler et al., 2010), dependency grammar (Chelba and Jelinek, 2000; Nivre, 2007; Huang and Sagae, 2010), or tree-adjoining grammar (Demberg et al., 2014). Typical applications of incremental parsers include speech recognition (Chelba and Jelinek, 2000; Roark, 2001; Xu et al., 2002), machine translation (Schwartz et al., 2011; Tan et al., 2011), reading time modeling (Demberg and Keller, 2008), or dialogue systems (Stoness et al., 2004). Another potential use of incremental parsers is sentence prediction, i.e., the task of predicting upcoming words in a sentence given a prefix. However, so far only n-gram models and classifiers have been used for this task (Fazly and Hirst, 2003; Eng and Eisner, 2004; Grabski and Scheffer, 2004; Bickel et al., 2005; Li and Hirst, 2005).

In this paper, we present an incremental parser for Tree Substitution Grammar (TSG). A TSG contains a set of arbitrarily large tree fragments, which can be combined into new syntax trees by means of a substitution operation. An extensive tradition of parsing with TSG (also referred to as data-oriented parsing) exists (Bod, 1995; Bod et al., 2003), but none of the existing TSG parsers are incremental. We show how constraints can be imposed on the shape of the TSG fragments to enable incremental processing. We propose an efficient Earley-based algorithm for incremental TSG parsing and report an F-score competitive with other incremental parsers.

Transactions of the Association for Computational Linguistics 1: May, 111–124, 2013.

TSG fragments can be arbitrarily large and can contain multiple lexical items. This property enables our incremental TSG parser to generate partial parse trees that include predictions about the upcoming words in a sentence. It can therefore be applied directly to the task of sentence prediction, simply by reading off the predicted items in a partial tree. We show that our parser outperforms an n-gram model in predicting more than one upcoming word.

The rest of the paper is structured as follows. In Section 2, we introduce the ITSG framework and relate it to the original TSG formalism. Section 3 describes the chart-parser algorithm, while Section 4 details the experimental setup and results. Sections 5 and 6 present related work and conclusions.

2 Incremental Tree Substitution Grammar

The current work is based on Tree Substitution Grammar (TSG, Schabes 1990; for a recent overview see Bod et al. 2003). A TSG is composed of (i) a set of arbitrarily large *fragments*, usually extracted from an annotated phrase-structure treebank, and (ii) the *substitution operation* by means of which fragments can be combined into complete syntactic analyses (derivations) of novel sentences.

Every fragment's node is either a lexical node (word), a substitution site (a non-lexical node in the yield of the structure),[1] or an internal node. An internal node must always keep the same daughter nodes as in the original tree. For an example of a binarized[2] tree and a fragment extracted from it see Figure 1.

A TSG derivation is constructed in a top-down generative process starting from a fragment in the grammar rooted in S (the unique non-lexical node all syntactic analysis are rooted in). A partial derivation is extended by subsequently introducing more fragments: if X is the left-most substitution site in the yield of the current partial derivation, a fragment rooted in X is chosen from the grammar and substituted into it. When there are no more substitution sites (all nodes in the yield are lexical items) the generative process terminates.

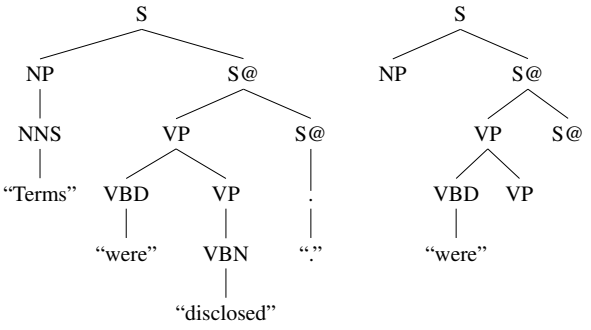


Figure 1: An example of a binarized[2] parse tree and a lexicalized fragment extracted from it.

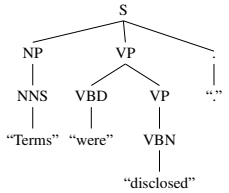
2.1 Incrementality

In this work we are interested in defining an *incremental* TSG (in short ITSG). The new generative process, while retaining the original mechanism for combining fragments (by means of the substitution operation), must ensure a way for deriving syntactic analyses of novel sentences in an incremental manner, i.e., one word at the time from left to right. More precisely, at each stage of the generative process, the partially derived structure must be connected (as in standard TSG) and have a prefix of the sentence at the beginning of its yield. A partial derivation is connected if it has tree shape, i.e., all the nodes are dominated by a common root node (which does not necessarily have to be the root node of the sentence). For instance, the right fragment in Figure 1 shows a possible way of starting a standard TSG derivation which does not satisfy the incrementality constraint: the partial derivation has a substitution site as the first element in its yield.

In order to achieve incrementality while maintaining connectedness, we impose one further constraint on the type of fragments which are allowed in an ITSG: each fragment should be *lexicalized*, i.e., contain at least one word (lexical anchor) at the first or the second position in its yield. Allowing more than one substitution site at the beginning of a fragment's yield would lead to a violation of the incrementality requirement (as will become clear in Section 2.2).

[1] For example nodes NP, VP, S@ are the substitution sites of the right fragment in Figure 1.

[2] The tree is right-binarized via artificial nodes with @ symbols, as explained in Section 4.1. The original tree is



The generative process starts with a fragment anchored in the first word of the sentence being generated. At each subsequent step, a lexicalized fragment is introduced (by means of the substitution operation) to extend the current partial derivation in such a way that the prefix of the yield of the partial structure is lengthened by one word (the lexical anchor of the fragment being introduced). The lexicalization constraint allows a fragment to have multiple lexical items, not necessarily adjacent to one another. This is useful to capture the general ability of TSG to produce in one single step an arbitrarily big syntactic construction ranging from phrasal verbs (e.g., ask someone out), to parallel constructions (e.g., either X or Y), and idiomatic expressions (e.g., took me to the cleaners). For an example of a fragment with multiple lexical anchors see the fragment in the middle of Figure 2.

2.2 Symbolic Grammar

An ITSG is a tuple $\langle \mathcal{N}, \mathcal{L}, \mathcal{F}, \subseteq, \supseteq, \oplus \rangle$, where $\mathcal{N}$ and $\mathcal{L}$ are the set of non-lexical and lexical nodes respectively, $\mathcal{F}$ is a collection of lexicalized fragments, $\subseteq$ and $\supseteq$ are two variants of the substitution operation (backward and forward) used to combine fragments into derivations, and $\oplus$ is the stop operation which terminates the generative process.

Fragments A fragment $f \in \mathcal{F}$ belongs to one of the three sets $\mathcal{F}_{init}, \mathcal{F}^X_{lex}, \mathcal{F}^Y_{sub}$:

- An *initial fragment* (f_{init}) has the lexical anchor in the first position of the yield, being the initial word of a sentence (the left-most lexical node of the parse tree from which it was extracted).
- A *lex-first fragment* (f^X_{lex}) has the lexical anchor (non sentence-initial) in the first position of the yield, and is rooted in X.[3]
- A *sub-first fragment* (f^Y_{sub}) has the lexical anchor in the second position of its yield, and a substitution site Y in the first.

Fringes We will use fringes (Demberg et al., 2014) as a compressed representation of fragments, in which the internal structure is replaced by a triangle ($\bigtriangleup$ or $\lhd$) and only the root and the yield are visible. It is possible in a grammar that multiple fragments map to the same fringe; we will refer to those as *ambiguous fringes*. We use both vertical ($\bigtriangleup$, e.g., in Figure 3 and 4) and horizontal ($\lhd$) fringe notation. The latter is used for describing the states in our chart-parsing algorithm in Section 3. For instance, the horizontal fringe representation of the right fragment in Figure 1 is S $\lhd$ NP "were" VP S@.

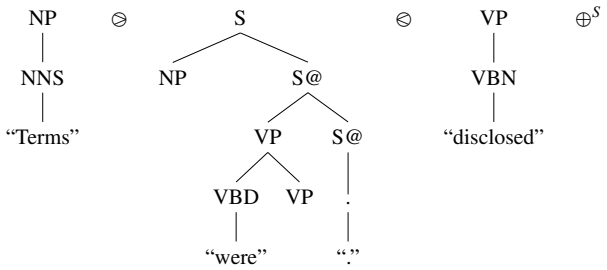


Figure 2: An example of an ITSG derivation yielding the tree on the left side of Figure 1. The second and third fragment are introduced by means of forward and backward substitution, respectively.

Incremental Derivation An incremental derivation is a sequence of lexicalized fragments $\langle f_1, f_2, \ldots, f_n \rangle$ which, combined together in the specified order, give rise to a complete parse tree (see Figure 2 for an example). The first fragment f_1 being introduced in the derivation must be an initial fragment, and its lexical anchor constitutes the one-word prefix of the sentence being generated. Subsequent fragments are introduced by means of the substitution operation, which has two variants: backward substitution ($\subseteq$), which is used to substitute lex-first fragments into the partial derivation generated so far, and forward substitution ($\supseteq$), which is used to substitute sub-first fragments into the partial derivation. After a number of fragments are introduced, a stop operation ($\oplus$) may terminate the generative process.

Operations The three ITSG operations take place under specific conditions within an incremental derivation, as illustrated in Figure 3 and explained hereafter. At a given stage of the generative process (after an initial fragment has been inserted), the connected partial structure may or may not have sub-

[3] A fragment can be both an initial and a lex-first fragment (e.g., if the lexical anchor is a proper noun). We will make use of two separate instances of the same fragment in the two sets.

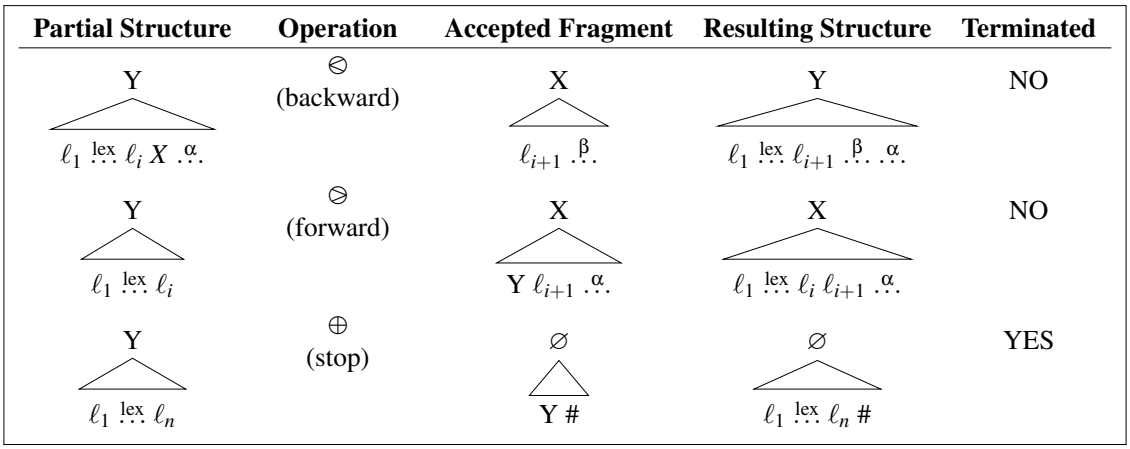


Figure 3: Schemata of the three ITSG operations. All tree structures (partial structure and fragments) are represented in a compact notation, which displays only the root nodes and the yields. The i-th words in the structure's yield is represented as ℓ_i, while α and β stands for any (possibly empty) sequence of words and substitution sites.

stitution sites present in its yield. In the first case, a backward substitution ($\ominus$) must take place in the following generative step: if X is the left-most substitution site, a new fragment of type f^X_{lex} is chosen from the grammar and substituted into X. If the partially derived structure has no substitution site (all the nodes in its yield are lexical nodes) and it is rooted in Y, two possible choices exist: either the generative process terminates by means of the stop operation ($\oplus^Y$), or the generative process continues. In the latter case a forward substitution ($\oslash$) is performed: a new f^Y_{sub} fragment is chosen from the grammar, and the partial structure is substituted into the left-most substitution site Y of the fragment.[4]

Multiple Derivations As in TSG, an ITSG may be able to generate the same parse tree in multiple ways: multiple incremental derivations yielding the same tree. Figure 4 shows one such example.

Generative Capacity It is useful to clarify the difference between ITSG and the more general TSG formalism in terms of generative capacity. Although both types of grammar make use of the substitution operation to combine fragments, an ITSG imposes more constraints on (i) the type of fragments which are allowed in the grammar (initial, lex-first, and sub-first fragments), and (ii) the generative process with which fragments are combined (incrementally left to right instead of top-down). If we compare a TSG and an ITSG on the same set of (ITSG-compatible) fragments, then there are cases in which the TSG can generate more tree structures than the ITSG.

In the following, we provide a more formal characterization of the strong and weak generative power

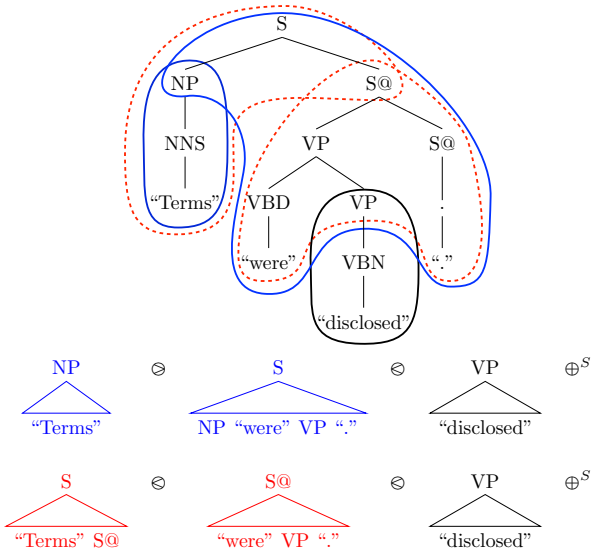


Figure 4: Above: an example of a set of fragments extracted from the tree in Figure 1. Below: two incremental derivations that generate it. Colors (and lines strokes) indicate which derivation fragments belong to.

[4] A stop operation can be viewed as a forward substitution when using an artificial sub-first fragment $\varnothing \lhd Y\#$ (stop fragment), where # is an artificial lexical node indicating the termination of the sentence. For simplicity, stop fragments are omitted in Figure 2 and 4 and Y is attached to the stop symbol ($\oplus^Y$).

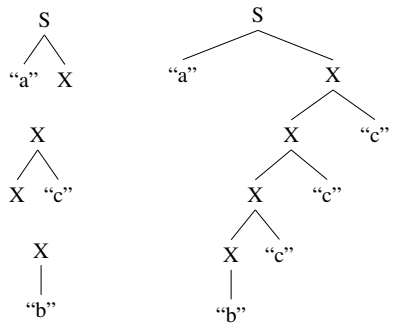


Figure 5: Left: an example of a CFG with left recursion. Right: one of the structures the CFG can generate.

of ITSG with respects to context-free grammar (CFG) and TSG. (However, a full investigation of this issue is beyond the scope of this paper.) We can limit our analysis to CFG, as TSG is strongly equivalent to CFG. The weak equivalence between ITSG and CFG is straightforward: for any CFG there is a way to produce a weakly equivalent grammar in Greibach Normal Form in which any production has a right side beginning with a lexical item (Aho and Ullman, 1972). The grammar that results from this transformation is an ITSG which uses only backward substitutions.

Left-recursion seems to be the main obstacle for strong equivalence between ITSG and CFG. As an example, the left side of Figure 5 shows a CFG that contains a left-recursive rule. The types of structures this grammar can generate (such as the one given on the right side of the same figure) are characterized by an arbitrarily long chain of rules that can intervene before the second word of the string, "b", is generated. Given the incrementality constraints, there is no ITSG that can generate the same set of structures that this CFG can generate. However, it may be possible to circumvent this problem by applying the left-corner transform (Rosenkrantz and Lewis, 1970; Aho and Ullman, 1972) to generate an equivalent CFG without left-recursive rules.

2.3 Probabilistic Grammar

In the generative process presented above there are a number of choices which are left open, i.e., which fragment is being introduced at a specific stage of a derivation, and when the generative process terminates. A symbolic ITSG can be equipped with a probabilistic component which deals with these choices. A proper probability model for ITSG needs to define three probability distributions over the three types of fragments in the grammar, such that:

$$\sum_{f_{init} \in \mathscr{F}_{init}} P(f_{init}) = 1 \tag{1}$$

$$\sum_{f^X_{lex} \in \mathscr{F}^X_{lex}} P(f^X_{lex}) = 1 \;\; (\forall X \in \mathscr{N}) \tag{2}$$

$$P(\oplus^Y) + \sum_{f^Y_{sub} \in \mathscr{F}^Y_{sub}} P(f^Y_{sub}) = 1 \;\; (\forall Y \in \mathscr{N}) \tag{3}$$

The probability that an ITSG generates a specific derivation d is obtained by multiplying the probabilities of the fragments taking part in the derivation:

$$P(d) = \prod_{f \in d} P(f) \tag{4}$$

Since the grammar may generate a tree t via multiple derivations $D(t) = d_1, d_2, \ldots, d_m$, the probability of the parse tree is the sum of the probabilities of the ITSG derivations in $D(t)$:

$$P(t) = \sum_{d \in D(t)} P(d) = \sum_{d \in D(t)} \prod_{f \in d} P(f) \tag{5}$$

3 Probabilistic ITSG Parser

We introduce a probabilistic chart-parsing algorithm to efficiently compute all possible incremental derivations that an ITSG can generate given an input sentence (presented one word at the time). The parsing algorithm is an adaptation of the Earley algorithm (Earley, 1970) and its probabilistic instantiation (Stolcke, 1995).

3.1 Parsing Chart

A TSG incremental derivation is represented in the chart as a sequence of chart states, i.e., a *path*.

For a given fringe in an incremental derivation, there will be one or more states in the chart, depending on the length of the fringe's yield. This is because we need to keep track of the extent to which the yield of each fringe has been consumed within a derivation as the sentence is processed incrementally.[5] At the given stage of the derivation, the states offer a compact representation over the partial structures generated so far.

[5] A fringe (state) may occur in multiple derivations (paths): for instance in Figure 4 the two derivations will correspond to two separate paths that will converge to the same fringe (state).

Start(ℓ_0)

$$\frac{X \lhd \ell_0 \nu}{0 : {}_0X \lhd \bullet \ell_0 \nu \ [\alpha, \gamma, \beta]}$$

$$\alpha = \gamma \quad = \quad P(X \lhd \ell_0 \nu) \qquad \beta \quad = \quad \beta(1 : {}_0X \lhd \ell_0 \bullet \nu)$$

Backward Substitution(ℓ_i)

$$\frac{i : {}_kX \lhd \lambda \bullet Y \mu \ [\alpha, \gamma, \beta] \qquad Y \lhd \ell_i \nu}{i : {}_iY \lhd \bullet \ell_i \nu \ [\alpha', \gamma', \beta']}$$

$$\alpha' \quad \stackrel{+}{=} \quad \alpha \cdot P(Y \lhd \ell_i \nu) \qquad \gamma' \quad = \quad P(Y \lhd \ell_i \nu)$$

Forward Substitution(ℓ_i)

$$\frac{i : {}_0Y \lhd \nu \bullet \ [\alpha, \gamma, \beta] \qquad X \lhd Y \ell_i \mu}{i : {}_0X \lhd Y \bullet \ell_i \mu \ [\alpha', \gamma', \beta']}$$

$$\begin{array}{lllll} \alpha' & \stackrel{+}{=} & \alpha \cdot P(X \lhd Y \ell_i \mu) & \beta & \stackrel{+}{=} \quad \beta' \cdot P(X \lhd Y \ell_i \mu) \\ \gamma' & \stackrel{+}{=} & \gamma \cdot P(X \lhd Y \ell_i \mu) & & \end{array}$$

Completion

$$\frac{i : {}_jY \lhd \ell_j \nu \bullet \ [\alpha, \gamma, \beta] \qquad j : {}_kX \lhd \lambda \bullet Y \mu \ [\alpha', \gamma', \beta']}{i : {}_kX \lhd \lambda Y \bullet \mu \ [\alpha'', \gamma'', \beta'']}$$

$$\begin{array}{llllll} \alpha'' & \stackrel{+}{=} & \alpha' \cdot \gamma & \beta & \stackrel{+}{=} & \beta'' \cdot \gamma' \\ \gamma'' & \stackrel{+}{=} & \gamma' \cdot \gamma & \beta' & \stackrel{+}{=} & \beta'' \cdot \gamma'' \end{array}$$

Scan(ℓ_i)

$$\frac{i : {}_kX \lhd \lambda \bullet \ell_i \mu \ [\alpha, \gamma, \beta]}{i+1 : {}_kX \lhd \lambda \ell_i \bullet \mu \ [\alpha', \gamma', \beta']}$$

$$\begin{array}{llllll} \alpha' & = & \alpha & \beta & = & \beta' \\ \gamma' & = & \gamma & & & \end{array}$$

Stop(#)

$$\frac{n : {}_0Y \lhd \nu \bullet \ [\alpha = \gamma, \beta] \qquad \emptyset \lhd Y \#}{n : {}_0\emptyset \lhd Y \bullet \# \ [\alpha', \gamma', \beta']}$$

$$\alpha' \quad = \quad \gamma' = \alpha \cdot P(\oplus^Y) \qquad \begin{array}{lll} \beta' & = & 1 \\ \beta & = & P(\oplus^Y) \end{array}$$

Figure 6: Chart operations with forward (α), inner (γ), and outer (β) probabilities.

Each state is composed of a fringe and some additional information which keeps track of where the fringe is located within a path. A chart state can be generally represented as

$$i : {}_kX \lhd \lambda \bullet \mu \tag{6}$$

where $X \lhd \lambda\mu$ is the state's fringe, Greek letters are (possibly empty) sequences of words and substitution sites, and $\bullet$ is a placeholder indicating to which extent the fragment's yield has been consumed: all the elements in the yield preceding the dot have been already accepted. Finally, i and k are indices of words in the input sentence:

- i signifies that the current state is introduced after the first i words in the sentence have been scanned. All states in the chart will be grouped according to this index, and will constitute *state-set i*.
- k indicates that the fringe associated with the current state was first introduced in the chart after the first k words in the input sentence had been scanned. The index k is therefore called the *start index*.

For instance when generating the first incremental derivation in Figure 4, the parser will pass through state $1 : {}_1S \lhd$ NP $\bullet$ "were" VP "." indicating that the second fringe is introduced right after the parser has scanned the first word in the sentence and before having scanned the second word.

3.2 Parsing Algorithm

We will first introduce the symbolic part of the parsing algorithm, and then discuss its probabilistic component in Section 3.3. In line with the generative process illustrated in Section 2.2, the parser operates on the chart states in order to keep track of all possible ITSG derivations as new words are fed in. It *starts* by reading the first word ℓ_0 and introducing new states to state-set 0 in the chart, those mapping to initial fragments in the grammar with ℓ_0 as lexical anchor. At a given stage, after i words have been scanned, the parser reads the next word (ℓ_i) and introduces new states in state-sets i and $i+1$ by applying specific operations on states present in the chart, and fringes in the grammar.

Parser Operations The parser begins with the *start* operation just described, and continues with a cycle of four operations for every word in the input sentence ℓ_i (for $i \geq 0$). The order of the four operations is the following: completion, backward substitution ($\circledcirc$), forward substitution ($\circledcirc$), and scan. When there are no more words in input, the parser terminates with a *stop* operation. We will now describe the parser operations (see Figure 6 for their formal definition), ignoring the probabilities for now.

Start(ℓ_0): For every *initial fringe* in the grammar anchored in ℓ_0, the parser inserts a (scan) state for that fringe in the state-set 0.

Backward Substitution(ℓ_i) applies to *acceptor states*, i.e., those with a substitution site following the dot, say X. For each acceptor state in state-set i, and any lex-first fringe in the grammar rooted in X and anchored in ℓ_i, the parser inserts a (scan) state for that fringe in state-set i.

Forward Substitution(ℓ_i) applies to *donor states*, i.e., those that have no elements following the dot and with start index 0. For each donor state in state-set i, rooted in Y, and any sub-first fringe in the grammar with Y as the left-most element in its yield, the parser inserts a (scan) state for that fringe in state-set i, with the dot placed after Y.

Completion applies to complete states, i.e., those with no elements following the dot and with start index $j > 0$. For every complete state in state-set i, rooted in Y, with starting index j, and every acceptor state in set j with Y following the dot, the parser inserts a copy of the acceptor state in state-set i, and advances the dot.

Scan(ℓ_i) applies to *scan states*, i.e., those with a word after the dot. For every scan state in state-set i having ℓ_i after the dot, the parser inserts a copy of that state in state-set $(i+1)$, and advances the dot.

Stop(#) is a special type of forward substitution and applies to donor states, but only when the input word is the terminal symbol #. For every donor state in state-set n (the length of the sentence), if the root of the fringe's state is Y, the parser introduces a *stop state* whose fringe is a stop fringe with Y as the left most substitution site.

Comparison with the Earley Algorithm It is useful to clarify the differences between the proposed ITSG parsing algorithm and the original Earley algorithm. Primarily, the ITSG parser is based on a left-right processing order, whereas the Earley algorithm uses a top-down generative process. Moreover, our parser presupposes a restricted inventory of fragments in the grammar (the ones allowed by an ITSG) as opposed to the general CFG rules allowed by the Earley algorithm. In particular, the *Backward Substitution* operation is more limited than the corresponding *Prediction* step of the Earley algorithm: only lex-first fragments can be introduced using Backward Substitution, and therefore left recursion (allowed by the Earley algorithm) is not possible here.[6] This restriction is compensated for by the existence of the *Forward Substitution* operation, which has no analog in the Earley algorithm.[7] The worst case complexity of Earley algorithm is dominated by the *Completion* operation which is identical to that in our parser, and therefore the original total time complexity applies, i.e., $O(l^3)$ for an input sentence of length l, and $O(n^3)$ in terms of the number of non-lexical nodes n in the grammar.

Derivations Incremental (partial) derivations are represented in the chart as (partial) paths along states. Each state can lead to one or more successors, and come from one or more antecedents. Scan is the only operation which introduces, for every scan state, a new single successor state (which can be of any of the four types) in the following state-set. Complete states may lead to several states within the current state-set, which may belong to any of the four types. An acceptor state may lead to a number of scan states via backward substitution (depending on the number of lex-first fringes that can combine with it). Similarly, a donor state may lead to a number of scan states via forward substitution.

After i words have been scanned, we can retrieve (partial) paths from the chart. This is done in a backward direction starting from scan states in state-set i all the way back to the initial states. This is possible since all the operations are reversible, i.e., given a state it is possible to retrieve its antecedent state(s).

As an example, consider the ITSG grammar consisting of the fragments in Figure 7 and the two derivations of the same parse tree in the same figure; Figure 7 represents the parsing chart of the same grammar, containing the two corresponding paths.

3.3 Probabilistic Parser

In the probabilistic version of the parser, each fringe in the grammar has a given probability, such that Equations (1)–(3) are satisfied.[8] In the probabilistic chart, every state $i : {}_kX \lhd \lambda \bullet \mu$ is decorated with three

[6] This further simplifies the probabilistic version of our parser, as there is no need to resort to the probabilistic reflexive, transitive left-corner relation described by Stolcke (1995).

[7] This operation would violate Earley's top-down constraint; donor states are in fact the terminal states in Earley algorithm.

[8] The probability of an ambiguous fringe is the marginal probability of the fragments mapping to it.

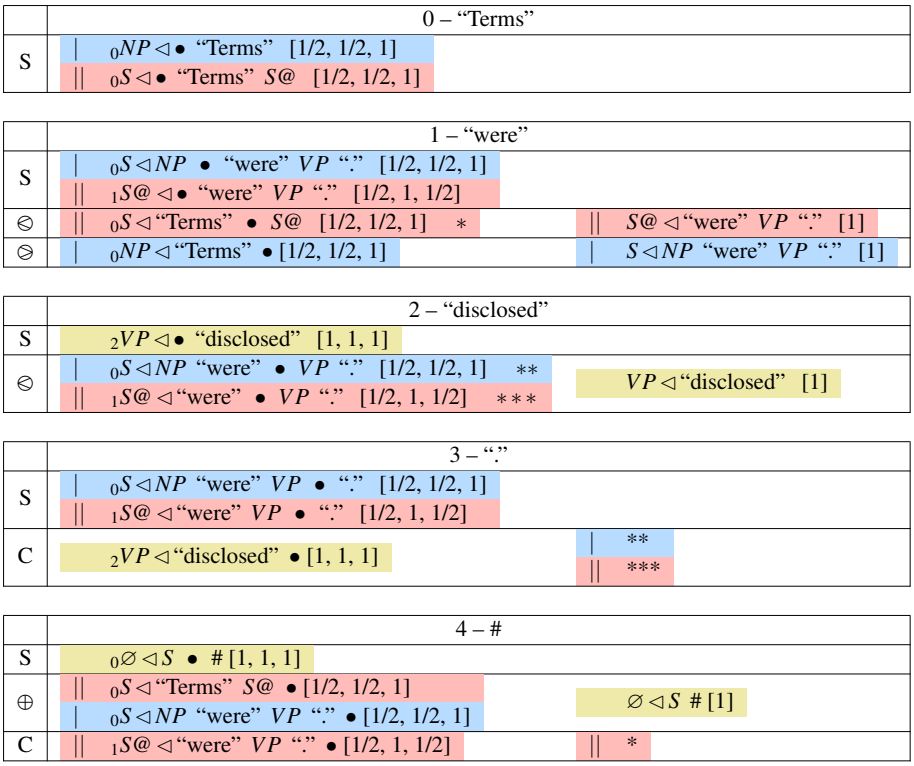


Figure 7: The parsing chart of the two derivations in Figure 4. Blue states or fringes (also marked with |) are the ones in the first derivation, red (||) in the second, and yellow (no marks) are the ones in common. Each state-set is represented as a separate block in the chart, headed by the state-set index and the next word. Each row maps to a chart operation (specified in the first column, with S and C standing for 'scan' and 'complete' respectively) and follows the same notation of figure 6. Symbols $*$ are used as state placeholders.

probabilities $[\alpha, \gamma, \beta]$ as shown in the chart example in Figure 7.

- The *forward probability* α is the marginal probability of all the paths starting with an initial state, scanning all initial words in the sentence until ℓ_{i-1} included, and passing through the current state.
- The *inner probability* γ is the marginal probability of all the paths passing through the state $k : {}_kX \triangleleft \bullet\lambda\mu$, scanning words $\ell_k, \ldots, \ell_{i-1}$ and passing through the current state.
- The *outer probability* β is the marginal probability of all the paths starting with an initial state, scanning all initial words in the sentence until ℓ_{k-1} included, passing through the current state, and reaching a stop state.

Forward (α) and inner (γ) probabilities are propagated while filling the chart incrementally, whereas outer probabilities (β) are back-propagated from the stop states, for which $\beta = 1$ (see Figure 6). These probabilities are used for computing prefix and sentence probabilities, and for obtaining the most probable partial derivation (MPD) of a prefix, the MPD of a sentence, its minimum risk parse (MRP), and to approximate its most probable parse (MPP).

Prefix probabilities are obtained by summing over the forward probabilities of all scan states in state-set i having ℓ_i after the dot:[9]

$$P(\ell_0, \ldots, \ell_i) = \sum_{\substack{\text{state } s \\ i:{}_kX \triangleleft \lambda \bullet \ell_i \mu}} \alpha(s) \tag{7}$$

3.4 Most Probable Derivation (MPD)

The Most Probable (partial) Derivation (MPD) can be obtained from the chart by backtracking the Viterbi path. Viterbi forward and inner probabilities

[9] *Sentence probability* is obtained by marginalizing the forward probabilities of the stop states in the last state-set n.

(α^*, γ^*) are propagated as standard forward and inner probabilities except that summation is replaced by maximization, and the probability of an ambiguous fringe is the maximum probability among all the fragments mapping into it (instead of the marginal one). The Viterbi partial path for the prefix $\ell_0, \ldots, \ell_i$ can then be retrieved by backtracking from the scan state in state-set i with max α^*: for each state, the most probable preceding state is retrieved, i.e., the state among its antecedents with maximum α^*. The Viterbi complete path of a sentence can be obtained by backtracking the Viterbi path from the stop state with max α^*. Given a Viterbi path, it is possible to obtain the corresponding MPD. This is done by retrieving the associated sequence of fragments[10] and connecting them.

3.5 Most Probable Parse (MPP)

According to Equation (5), if we want to compute the MPP we need to retrieve all possible derivations of the current sentence, sum up the probabilities of those generating the same tree, and returning the tree with max marginal probability. Unfortunately the number of possible derivations grows exponentially with the length of the sentence, and computing the exact MPP is NP-hard (Sima'an, 1996). In our implementation, we approximate the MPP by performing this marginalization over the Viterbi-best derivations obtained from all stop states in the chart.

3.6 Minimum Risk Parse (MRP)

MPD and MPP aim at obtaining the structure of a sentence which is more likely *as a whole* under the current probabilistic model. Alternatively, we may want to focus on the single components of a tree structures, e.g., CFG rules covering a certain span of the sentence, and search for the structure which has the highest number of correct constituents, as proposed by Goodman (1996). Such structure is more likely to obtain higher results according to standard parsing evaluations, as the objective being maximized is closely related to the metric used for evaluation (recall/precision on the number of correct labeled constituents).

[10]For each scan state in the path, we obtain the fragment in the grammar that maps into the state's fringe. For ambiguous fringes the most probable fragment that maps into it is selected.

In order to obtain the minimum risk parse (MRP) we utilize both inner (γ) and outer (β) probabilities. The product of these two probabilities equals the marginal probability of all paths generating the entire current sentence and passing through the current state. We can therefore compute the probability of a fringe $f = X \lhd \lambda \bullet \mu$ covering a specific span $[s,t]$ of the sentence:

$$P(f,[s,t]) = \gamma(t : {}_s f\bullet) \cdot \beta(t : {}_s f\bullet) \tag{8}$$

We can then compute the probability of each fragment spanning $[s,t]$,[11] and the probability $P(r,[s,t])$ of a CFG-rule r spanning $[s,t]$.[12] Finally the MRP is computed as

$$MRP = \arg\max_T \prod_{r\in T} P(r,[s,t]) \tag{9}$$

4 Experiments

For training and evaluating the ITSG parser, we employ the Penn WSJ Treebank (Marcus et al., 1993). We use sections 2–21 for training, section 22 and 24 for development and section 23 for testing.

4.1 Grammar Extraction

Following standard practice, we start with some preprocessing of the treebank. After removing traces and functional tags, we apply right binarization on the training treebank (Klein and Manning, 2003), with no horizontal and vertical conditioning. This means that when a node X has more than two children, new artificial constituents labeled $X@$ are created in a right recursive fashion (see Figure 1).[13] We then replace words appearing less than five times in the training data by one of 50 unknown word categories based on the presence of lexical features as described in Petrov (2009).

Fragment Extraction In order to equip the grammar with a representative set of lexicalized fragments, we use the extraction algorithm of Sangati

[11]For an ambiguous fringe, the spanning probability of each fragment mapping into it is the fraction of the fringe's spanning probability with respect to the marginal fringe probability.

[12]Marginalizing the probabilities of all fragments having r spanning $[s,t]$.

[13]This shallow binarization (H0V1) was used based on gold coverage of the unsmoothed grammar (extracted from the training set) on trees in section 22: H0V1 binarization results on a coverage of 88.0% of the trees, compared to 79.2% for H1V1.

et al. (2010) which finds maximal fragments recurring twice or more in the training treebank. To ensure better coverage, we additionally extract one-word fragments from each training parse tree: for each lexical node ℓ in the parse tree we percolate up till the root node, and for every encountered internal node $X_0, X_1, \ldots, X_i$ we extract the lexicalized fragment whose spine is $X_i - X_{i-1} - \ldots - X_0 - \ell$, and where all the remaining children of the internal nodes are substitution sites (see for instance the right fragment in Figure 1). Finally, we remove all fragments which do not comply with the restrictions presented in Section 2.1.[14]

For each extracted fragment we keep track of its frequency, i.e., the number of times it occurs in the training corpus. Each fragment's probability is then derived according to its relative frequency in the corresponding set of fragments $(f_{init}, f^X_{lex}, f^Y_{sub})$, so that equations(1)–(3) are satisfied. The final grammar consists of 2.2M fragments mapping to 2.0M fringes.

Smoothing Two types of smoothing are performed over the grammar's fragments: *Open class smoothing* adds simple CFG rewriting rules to the grammar for open-class[15] $\langle \text{PoS}, \text{word} \rangle$ pairs not encountered in the training corpus, with frequency 10^{-6}. *Initial fragments smoothing* adds each lex-first fragment f to the initial fragment set with frequency $10^{-2} \cdot freq(f)$.[16]

All ITSG experiments we report used exhaustive search (no beam was used to prune the search space).

4.2 Evaluation

In addition to standard *full-sentence parsing results*, we propose a novel way of evaluating our ITSG on *partial trees*, i.e., those that the parser constructs for sentence prefixes. More precisely, for each prefix of the input sentence (length two words or longer) we compute the parsing accuracy on the minimal structure spanning that prefix. The minimal structure is obtained from the subtree rooted in the minimum common ancestor of the prefix nodes, after pruning those nodes not yielding any word in the prefix.

As observed in the example derivations of Figure 4, our ITSG generates partial trees for a given prefix which may include predictions about unseen parts of the sentence. We propose three new measures for evaluating sentence prediction:[17]

Word prediction PRD(m): For every prefix of each test sentence, if the model predicts $m' \geq m$ words, the prediction is correct if the first m predicted words are *identical* to the m words following the prefix in the original sentence.

Word presence PRS(m): For every prefix of each test sentence, if the model predicts $m' \geq m$ words, the prediction is correct if the first m predicted words are *present*, in the same order, in the words following the prefix in the original sentence (i.e., the predicted word sequence is a subsequence of the sequence of words following the prefix).[18]

Longest common subsequence LCS: For every prefix of each test sentence, it computes the longest common subsequence between the sequence of predicted words (possibly none) and the words following the prefix in the original sentence.

Recall and precision can be computed in the usual way for these three measures. Recall is the total number (over all prefixes) of correctly predicted words (as defined by PRD(m), PRS(m), or LCS) over the total number of words expected to be predicted (according to m), while precision is the number of correctly predicted words over the number of words predicted by the model.

We compare the ITSG parser with the incremental parsers of Schuler et al. (2010) and Demberg et al. (2014) for full-sentence parsing, with the Roark (2001) parser[19] for full-sentence and partial pars-

[14] The fragment with no lexical items, and those with more than one substitution site at the beginning of the yield.

[15] A PoS belongs to the open class if it rewrites to at least 50 different words in the training corpus. A word belongs to the open class if it has been seen only with open-class PoS tags.

[16] The parameters were tuned on section 24 of the WSJ.

[17] We also evaluated our ITSG model using perplexity; the results obtained were substantially worse than those obtained using Roark's parsers.

[18] Note that neither PRD(m) nor PRS(m) correspond to word error rate (WER). PRD requires the predicted word sequence to be identical to the original sequence, while PRS only requires the predicted words to be present in the original. In contrast, WER measures the minimum number of substitutions, insertions, and deletions needed to transform the predicted sequence into the original sequence.

[19] Apart from reporting the results in Roark (2001), we also run the latest version of Roark's parser, used in Roark et al. (2009), which has higher results compared to the original work.

	R	P	F1
Demberg et al. (2014)	79.4	79.4	79.4
Schuler et al. (2010)	83.4	83.7	83.5
Roark (2001)	86.6	86.5	86.5
Roark et al. (2009)	**87.7**	**87.5**	**87.6**
ITSG (MPD)	81.5	83.5	82.5
ITSG (MPP)	81.6	83.6	82.6
ITSG (MRP)	82.6	85.8	84.1
ITSG Smoothing (MPD)	83.0	83.5	83.2
ITSG Smoothing (MPP)	83.2	83.6	83.4
ITSG Smoothing (MRP)	**83.9**	**85.6**	**84.8**

Table 1: Full-sentence parsing results for sentences in the test set of length up to 40 words.

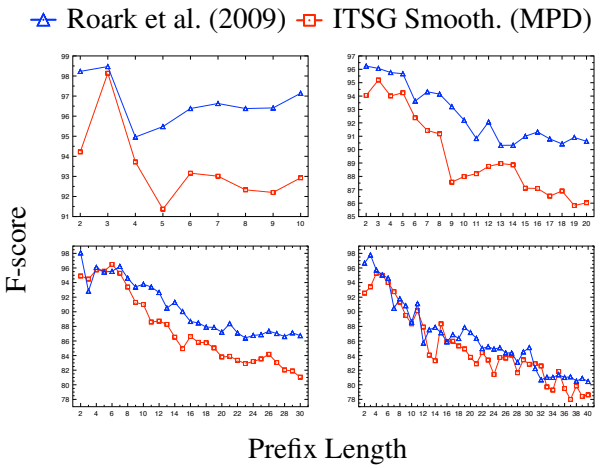


Figure 8: Partial parsing results for sentences of length 10, 20, 30, and 40 (from upper left to lower right).

ing, and with a language model built using SRILM (Stolcke, 2002) for sentence prediction. We used a standard 3-gram model trained on the sentences of the training set using the default setting and smoothing (Kneser-Ney) provided by the SRILM package. (Higher n-gram model do not seem appropriate, given the small size of the training corpus.) For every prefix in the test set we compute the most probable continuation predicted by the n-gram model.[20]

4.3 Results

Table 1 reports full-sentence parsing results for our parser and three comparable incremental parsers from the literature. While Roark (2001) obtains the best results, the ITSG parser without smoothing performs on a par with Schuler et al. (2010), and outperforms Demberg et al. (2014).[21] Adding smoothing results in a gain of 1.2 points F-score over the Schuler parser. When we compare the different parsing objectives of the ITSG parser, MRP is the best one, followed by MPP and MPD.

Incremental Parsing The graphs in Figure 8 compare the ITSG and Roark's parser on the incremental parsing evaluation, when parsing sentences of length 10, 20, 30 and 40. The performance of both models declines as the length of the prefix increases, with Roark's parser outperforming the ITSG parser on average, although the ITSG parser seems more competitive when parsing prefixes for longer (and therefore more difficult) sentences.

Sentence Prediction Table 2 compares the sentence prediction results of the ITSG and the language model (SRILM). The latter is outperforming the former when predicting the next word of a prefix, i.e. PRD(1), whereas ITSG is better than the language model at predicting a single future word, i.e. PRS(1). When more than one (consecutive) word is considered, the SRILM model exhibits a slightly better recall while ITSG achieves a large gain in precision. This illustrates the complementary nature of the two models: while the language model is better at predicting the next word, the ITSG predicts future words (rarely adjacent to the prefix) with high confidence (89.4% LCS precision). However, it makes predictions for only a small number of words (5.9% LCS recall). Examples of sentence predictions can be found in Table 3.

5 Related Work

To the best of our knowledge, there are no other incremental TSG parsers in the literature. The parser of Demberg et al. (2014) is closely related, but uses tree-adjoining grammar, which includes both substitution and adjunction. That parser makes predictions, but only for upcoming structure, not for upcoming words, and thus cannot be used directly for sentence prediction. The incremental parser of Roark (2001) uses a top-down algorithm and works

[20] We used a modified version of a script by Nathaniel Smith available at `https://github.com/njsmith/pysrilm`.

[21] Note that the scores reported by Demberg et al. (2014) are for TAG structures, not for the original Penn Treebank trees.

	ITSG			SRILM		
	Correct	R	P	Correct	R	P
PRD(1)	4,637	8.7	12.5	11,430	**21.5**	**21.6**
PRD(2)	864	1.7	**13.9**	2,686	**5.3**	5.7
PRD(3)	414	0.9	**20.9**	911	**1.9**	2.1
PRD(4)	236	0.5	**23.4**	387	**0.8**	1.0
PRS(1)	34,831	**65.4**	**93.9**	21,954	41.2	41.5
PRS(2)	4,062	8.0	**65.3**	5,726	**11.3**	12.2
PRS(3)	1,066	2.2	**53.7**	1,636	**3.4**	3.8
PRS(4)	541	1.2	**53.7**	654	**1.4**	1.7
LCS	44,454	5.9	**89.4**	92,587	**12.2**	18.4

Table 2: Sentence prediction results.

Prefix	Shares of UAL , the parent	PRD(3)	PRS(3)
ITSG	company of United Airlines ,	–	–
SRILM	company , which is the	–	–
Goldstd	of United Airlines , were extremely active all day Friday .		
Prefix	PSE said it expects to report earnings of $ 1.3 million to $ 1.7 million , or 14		
ITSG	cents a share ,	–	+
SRILM	% to $ UNK	–	–
Goldstd	cents to 18 cents a share .		

Table 3: Examples comparing sentence predictions for ITSG and SRILM (UNK: unknown word).

on the basis of context-free rules. These are augmented with a large number of non-local features (e.g., grandparent categories). Our approach avoids the need for such additional features, as TSG fragments naturally contain non-local information. Roark's parser outperforms ours in both full-sentence and incremental F-score (see Section 4), but cannot be used for sentence prediction straightforwardly: to obtain a prediction for the next word, we would need to compute an argmax over the whole vocabulary, then iterate this for each word after that (the same is true for the parsers of Schuler et al., 2010 and Demberg et al., 2014). Most incremental dependency parsers use a discriminative model over parse actions (Nivre, 2007), and therefore cannot predict upcoming words either (but see Huang and Sagae 2010).

Turning to the literature on sentence prediction, we note that ours is the first attempt to use a parser for this task. Existing approaches either use n-gram models (Eng and Eisner, 2004; Bickel et al., 2005) or a retrieval approach in which the best matching sentence is identified from a sentence collection given a set of features (Grabski and Scheffer, 2004). There is also work combining n-gram models with lexical semantics (Li and Hirst, 2005) or part-of-speech information (Fazly and Hirst, 2003).

In the language modeling literature, more sophisticated models than simple n-gram models have been developed in the past few years, and these could potentially improve sentence prediction. Examples include syntactic language models which have applied successfully for speech recognition (Chelba and Jelinek, 2000; Xu et al., 2002) and machine translation (Schwartz et al., 2011; Tan et al., 2011), as well as discriminative language models (Mikolov et al., 2010; Roark et al., 2007). Future work should evaluate these approaches against the ITSG model proposed here.

6 Conclusions

We have presented the first incremental parser for tree substitution grammar. Incrementality is motivated by psycholinguistic findings, and by the need for real-time interpretation in NLP. We have shown that our parser performs competitively on both full sentence and sentence prefix F-score. We also introduced sentence prediction as a new way of evaluating incremental parsers, and demonstrated that our parser outperforms an n-gram model in predicting more than one upcoming word.

The performance of our approach is likely to improve by implementing better binarization and more advanced smoothing. Also, our model currently contains no conditioning on lexical information, which is also likely to yield a performance gain. Finally, future work could involve replacing the relative frequency estimator that we use with more sophisticated estimation schemes.

Acknowledgments

This work was funded by EPSRC grant EP/I032916/1 "An integrated model of syntactic and semantic prediction in human language processing". We are grateful to Brian Roark for clarifying correspondence and for guidance in using his incremental parser. We would also like to thank Katja Abramova, Vera Demberg, Mirella Lapata, Andreas van Cranenburgh, and three anonymous reviewers for useful comments.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The theory of parsing, translation, and compiling*. Prentice-Hall, Inc., Upper Saddle River, NJ.

Gerry T. M. Altmann and Yuki Kamide. 1999. Incremental interpretation at verbs: Restricting the domain of subsequent reference. *Cognition*, 73:247–264.

Steffen Bickel, Peter Haider, and Tobias Scheffer. 2005. Predicting sentences using n-gram language models. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 193–200. Vancouver.

Rens Bod. 1995. The problem of computing the most probable tree in data-oriented parsing and stochastic tree grammars. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, pages 104–111. Association for Computer Linguistics, Dublin.

Rens Bod, Khalil Sima'an, and Remko Scha. 2003. *Data-Oriented Parsing*. University of Chicago Press, Chicago, IL.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. *Computer Speech and Language*, 14:283–332.

Vera Demberg and Frank Keller. 2008. Data from eye-tracking corpora as evidence for theories of syntactic processing complexity. *Cognition*, 101(2):193–210.

Vera Demberg, Frank Keller, and Alexander Koller. 2014. Parsing with psycholinguistically motivated tree-adjoining grammar. *Computational Linguistics*, 40(1). In press.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

John Eng and Jason M. Eisner. 2004. Radiology report entry with automatic phrase completion driven by language modeling. *Radiographics*, 24(5):1493–1501.

Afsaneh Fazly and Graeme Hirst. 2003. Testing the efficacy of part-of-speech information in word completion. In *Proceedings of the EACL Workshop on Language Modeling for Text Entry Methods*, pages 9–16. Budapest.

Joshua Goodman. 1996. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, pages 177–183. Association for Computational Linguistics, Santa Cruz.

Korinna Grabski and Tobias Scheffer. 2004. Sentence completion. In *Proceedings of the 27th Annual International ACM SIR Conference on Research and Development in Information Retrieval*, pages 433–439. Sheffield.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086. Association for Computational Linguistics, Uppsala.

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

Jianhua Li and Graeme Hirst. 2005. Semantic knowledge in a word completion task. In *Proceedings of the 7th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 121–128. Baltimore.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Tomas Mikolov, Martin Karafiat, Jan Cernocky, and Sanjeev. 2010. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association*, pages 2877–2880. Florence.

Joakim Nivre. 2007. Incremental non-projective dependency parsing. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 396–403. Association for Computational Linguistics, Rochester.

Slav Petrov. 2009. *Coarse-to-Fine Natural Language Processing*. Ph.D. thesis, University of California at Bekeley, Berkeley, CA.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguististics*, 27:249–276.

Brian Roark, Asaf Bachrach, Carlos Cardenas, and Christophe Pallier. 2009. Deriving lexical and syntactic expectation-based measures for psycholinguistic modeling via incremental top-down parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 324–333. Association for Computational Linguistics, Singapore.

Brian Roark, Murat Saraclar, and Michael Collins. 2007. Discriminative n-gram language modeling. *Computer Speech and Language*, 21(2):373–392.

D. J. Rosenkrantz and P. M. Lewis. 1970. Deterministic left corner parsing. In *Proceedings of the 11th Annual Symposium on Switching and Automata Theory*, pages 139–152. IEEE Computer Society, Washington, DC.

Federico Sangati, Willem Zuidema, and Rens Bod. 2010. Efficiently extract recurring tree fragments from large treebanks. In Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the 7th InternationalConference on Language Resources and Evaluation*. European Language Resources Association, Valletta, Malta.

Yves Schabes. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

William Schuler, Samir AbdelRahman, Tim Miller, and Lane Schwartz. 2010. Broad-coverage parsing using human-like memory constraints. *Computational Linguististics*, 36(1):1–30.

Lane Schwartz, Chris Callison-Burch, William Schuler, and Stephen Wu. 2011. Incremental syntactic language models for phrase-based translation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 620–631. Association for Computational Linguistics, Portland, OR.

Khalil Sima'an. 1996. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of the 16th Conference on Computational Linguistics*, pages 1175–1180. Association for Computational Linguistics, Copenhagen.

Andreas Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.

Andreas Stolcke. 2002. SRILM – an extensible language modeling toolkit. In *Proceedings International Conference on Spoken Language Processing*, pages 257–286. Denver, CO.

Scott C. Stoness, Joel Tetreault, and James Allen. 2004. Incremental parsing with reference interaction. In Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman, editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 18–25. Association for Computational Linguistics, Barcelona.

Ming Tan, Wenli Zhou, Lei Zheng, and Shaojun Wang. 2011. A large scale distributed syntactic, semantic and lexical language model for machine translation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 201–210. Association for Computational Linguistics, Portland, OR.

Michael K. Tanenhaus, Michael J. Spivey-Knowlton, Kathleen M. Eberhard, and Julie C. Sedivy. 1995. Integration of visual and linguistic information in spoken language comprehension. *Science*, 268:1632–1634.

Peng Xu, Ciprian Chelba, and Frederick Jelinek. 2002. A study on richer syntactic dependencies for structured language modeling. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 191–198. Association for Computational Linguistics, Philadelphia.