Learning long-distance dependencies via probabilistic finite-state tree automata

Yang Wang and Roxane Martin
University of California, Los Angeles

Abstract

We investigate the role of Probabilistic Finite-State Tree Automata (PFSTAs) in modeling long-distance dependencies in natural language syntax, specifically focusing on the dependency between transitive verbs and their object noun-phrases (NP). We explore the use of PFSTAs, a probabilistic formal grammar that characterizes stochastic tree languages, to identify and differentiate structural relations that require satisfying dependencies. Our analysis also includes prepositions, which together with transitive verbs we refer to as NP-complement-selecting heads. We employ the Expectation-Maximization (EM) algorithm to learn a goal PFSTA from a set of unlabeled, incomplete trees. Our aim is to determine whether the EM learner can arrive at the PFSTA that correctly identifies and enforces wh-dependencies, given a randomly generated starting PFSTA and a treebank that abides by transitive verb licensing. Our results reveal that our learner can successfully represent long-distance wh-dependency when trained on generated treebanks, but struggles to learn when trained on trees parsed from CHILDES, a corpora of annotated child-directed speech. Our analysis sheds light on limitations and assumptions necessary when using computational models for language acquisition of long-distance dependencies.

1 Introduction

Natural language syntax can be described in terms of dependencies between elements and, often, these dependencies may span long-distances within a sentence. A frequently observed syntactic phenomenon is that of wh-fronting in wh-questions, where the wh-phrase moves out of its object position to the front of a question. Combining these two concepts gives us the relationship known as long-distance licensing. For example in English, a transitive verb requires an object, which is typically satisfied by an object noun-phrase (NP) immediately to the right of the verb, such as in the sentence, “I know Mary ate a pie.” However, this dependency can also be satisfied by the presence of a displaced wh-phrase in a non-local clause-initial position, if and only if the object NP is missing locally. For example, “I know what Mary ate” is grammatically correct because the wh-phrase “what” licenses the gap in the sentence where the object NP should be.

The relationship between the acquisition of verb-argument structure and of long-distance dependencies has been the focus of much experimental work in syntax acquisition, with researchers proposing various computational models for how this learning process might occur (e.g., Jin and Fisher, 2014; Gagliardi et al., 2016; Lidz et al., 2017; Perkins and Lidz, 2020). Significant analysis has been done on the structural relation between the moved wh-phrase and

the trace (or gap) it leaves behind, much of which indicates that the wh-phrase must c-command its trace. This paper aims to propose a novel computational model for how this necessary either-or dependency may be acquired by language learners.

2 Theory & Definitions

2.1 Finite-State Tree Automata

Finite-State Automata (FSTA) are formal grammars that can represent the either-or relations between these object requirements, so we employ it as a candidate structure for the adult English speaker's representation of transitive verb object requirement.

2.1 Probabilistic Finite-State Tree Automata

We explore Probalisitic Finite-State Tree Automata as a computational model for this long-distance dependency. In accordance with syntactic skeletons proposed in learnability analyses (Sakakibara, 1992 and Stabler, 1998), we assume that inputs to the learner consist of structures with branching hierarchies between lexical items, i.e. trees. The learner reaches a point where it can assign a basic syntactic tree to a sentence, but structural relations, specifically sisterhood and c-command, need to be identified and differentiated to ensure the dependency can be satisfied in the two different ways. The syntactic representations are encoded using PFSTAs, a probabilistic formal grammar that characterizes stochastic tree languages. They are mathematically related to context-free string languages (CFGs), but they are more linguistically intuitive. In PFSTAs, the primary representation of interest is tree-shaped, and state transitions are employed to track long-distance dependencies by representing "features" as states, enabling different features to percolate up in trees. Engelfriet (2015) provides the formal definition of a PFSTA, which is a mathematical model that generates a set of tree structures with probabilities that can be employed to capture the distributional patterns of natural language syntax.

Definition 1: PFSTA

A 4-place tuple ⟨Q, Σ, δ, I⟩ such that

- $I : Q \to [0, 1]$ *describes the probability of the initial state*
  $\sum_{q \in Q} I(q) = 1$
- $\delta : \bigcup_k (Q \times \Sigma_k \times Q^k) \to [0, 1]$, *the probability associated with each transition*
  $\forall q_0 \in Q, \Sigma_{\sigma \in \Sigma} \left(\Sigma_{p^k} \left(\delta \left(q_0, \sigma, p^k \right) \right) \right) = 1$

2.2 Expectation-Maximization

In order to learn a PFSTA from a set of unlabeled, incomplete trees, the learner relies on likelihood maximization. Specifically, the learner aims to find the best parameter values that maximize the likelihood of the input data. This is accomplished using a statistical learning algorithm called Expectation-Maximization (EM) (Dempster et al., 1977).

The EM algorithm is composed of two steps: the E-step and the M-step. During the E-step, the algorithm calculates the expected counts of each state and transition in the grammar given the incomplete trees. This is done using the over-under algorithm, which is another

dynamic programming algorithm that efficiently computes the probabilities of all possible subtrees in the incomplete trees. The under algorithm calculates the probability of a subtree rooted at a given node, while the over algorithm calculates the probability of the rest of the tree that does not include the subtree rooted at the given node. By combining the results of these two algorithms, the expected counts of each state and transition can be calculated.

During the M-step, the algorithm updates the parameters of the grammar based on the expected counts obtained from the E-step. Specifically, the algorithm finds the values of the parameters that maximize the expected log-likelihood of the data given the current grammar. Pseudocode for the specifics of the EM and over-under algorithms can be found in the Appendix.

3 Methods

Our goal is to determine whether the EM learner can arrive at the PFSTA that correctly identifies and enforces wh-dependency, given a treebank of trees that correctly satisfy transitive verb licensing, and a randomly generated starting PFSTA. In actuality, we are more concerned with the resulting FSTA (or even CFG) as the exact likelihoods of the transitions vary from treebank to treebank, so the interesting metric is which transitions survive and which are pruned by the EM algorithm. The goal FSTA and CFG are as seen in Figures 1 and 2 below.

Goal PFSTA:

Q: [0, 1, 2, 3, 4]
I: {1: 1.0}
Delta:
(0, *'WH'*, ()): 1.0
(1, '*', (0, 4)): 0.3069
(1, '*', (1, 1)): 0.1693
(1, '*', (2, 3)): 0.3968
(1, *'C'*, ()): 0.127
(2, *'V'*, ()): 1.0
(3, *'NP'*, ()): 1.0
(4, '*', (1, 4)): 0.3012
(4, '*', (2,)): 0.6988

Figure 1

CFG form:

Initial state: N
L → *WH*
N → L UL
N → N N
N → V NP
N → *C*
V → *V*
NP → *NP*
UL → N UL
UL → V

State assignment: {0: 'L', 1: 'N', 2: 'V', 3: 'NP', 4: 'UL'},
where L is the licensing state, N is the neutral state, V is the verb state, NP is the noun phrase state, and UL is the unlicensed state.
Unitalicized labels corresponds to states, italicized labels correspond to terminals.

Figure 2

3.1 Hypothesis Space Restriction

The purpose of assigning terminal states and assigning the root state is to restrict the hypothesis space available to the learner. This is the equivalent of assuming that at the language acquisition state when children are learning long-distance dependencies, they have already learned to differentiate which lexical items belong to which parts of speech and that as a whole, sentences must be grammatical.

To generate the starting PFSTA, we follow a random generation process with certain impositions. We start with the given terminal symbols and states, and assign terminal states. The initial state is also assigned, specifically the neutral state as the root state, which enforces a resolved dependency. The rest of the probabilities are randomly distributed such that the sum of transitions exiting any state sums to one. This process ensures that the starting PFSTA has a valid structure.

3.2 Treebank Generation

We developed training treebanks in two different ways: tree generation and natural language parsing. The treebank generation phase occurs via two different methods. First, we use random recursive generation of trees with the desired terminals and then only retaining trees that meet licensing rules, i.e. enforcing that *V* terminals occur only either with a *NP* terminal as a right sister or c-commanded by a *WH* terminal. The second method involves the generation of trees from a variation of the goal PFSTA. Both of these methods result in treebanks that follow the rules of wh-licensing but do not correspond to natural language sentences.

3.2 CHILDES Treebank Parsing

The CHILDES treebank is a well-known dataset, containing transcripts of conversations between children and their caregivers. These conversations have been parsed and annotated into tree form. A subset of these treebanks contain trace; these traces capture the movement of a wh-word from its original position to a position within a question, providing insight into how interrogative structures manipulate word order.

Our learner takes as input a corpus of binary trees with all inner nodes labeled as '*' and terminals *WH, V, NP*, and *C*. The CHILDES corpora annotation is much richer than this, so we are tasked with converting the CHILDES trees to our desired format by cleaning and binarizing. Our criteria for transformation from a parse to a tree containing a wh-dependency are as follows (A step by step explanation of a full parse can be found in the Appendix):

- Contains WHNP annotation marker (rather than WHVP, for instance)
- Contains trace
- *V* & *NP*-trace are *direct* ordered sisters
    - Excludes ditransitive verbs with trace in direct object (in second NP)
- *V* is a lexical verb (not auxiliary)
- WHNP and trace are coindexed

After parsing the CHILDES treebank we discovered a large discrepency between the number of wh-questions expected in CHILDES corpus and the number identified by our parser, likely due to our strict structural requirements when parsing.

Based on Pearl & Sprouse (2013), we expected approximately 15-20% of the trees in our parsed CHILDES trees to include relevant wh-dependencies. However, once we apply the filters of our parsing we identified only about 11% of relevant dependencies as seen in the statistics in Figure 3.

Even with this filtering, we still identified 2638 trees that our parser did not transform into a wh-tree. Upon investigation, we found the majority of wh-questions 'missed' in parsing to

occur for three reasons: 1) ditransitive verbs 2) copular verbs, and 3) verbs with their object trace nested in prepositional phrases. Ditransitive verbs give evidence for both *V-NP* and *WH-V* constructions (as can be seen in Example 1); in the current parsing mechanism they produce trees containing V-NP which are also necessary for our learner (and children) to learn the short distance configuration of this dependency.

Total number of trees: 46740
WHNP trees: 8790 → 18.81%
with object question: 8090 → 17.31%
with trace: 5274 →11.28%
with lexical verb: 5253 → 11.24%
WH trees: 2615 → 5.59%

Figure 3

what does he call you t
('ROOT', ('SBARQ', ('WHNP-1', ('WP', 'what')), ('SQ', ('VP', ('AUX', 'does'), ('NP', ('PRP', 'he')), ('VP', ('VB', 'call'), ('NP', ('PRP', 'you')), ('NP', ('-NONE-ABAR-WH', '*t*-1'))))), ('.', '?')))

Example 1

Since instead identifying these as *WH-V* constructions would bar them from providing *V-NP* information, for the current study, we will continue to parse questions ditransitive verbs as statements.

Although the latter two scenarios above do not concern transitive verbs, they have the long-distance licensing structure we are interested in investigating: they either require an *NP* complement or must be c-commanded by a wh-phrase. However, allowing copular verbs to act as *V*s in our structure introduces the question of how to deal with situations such as Example 2 below. Due to the inversion that occurs with question formation, the trace is clearly not the direct sister of 'is,' but the answer to the question would become the 'object' of the copula. With the current parsing, this sentence gives us a *V-NP* tree, but not a *WH-V* tree.

what is that t
('ROOT', ('SBARQ', ('WHNP-1', ('WP', 'what')), ('SQ', ('VP', ('COP', 'is'), ('NP', ('NN', 'that')), ('NP', ('-NONE-ABAR-WH-', '*t*-1')))), ('.', '?')))

Example 2

Despite the large number of CHILDES parses that contain such inverted questions, we did not adjust our definition of an NP-complement-selector to include these copulas as they fail the simple test of the NP-seeking head having its trace or object as a direct right sister.

We choose to handle prepositions differently. Taking Example 3 below:

```
what  does  he  look  like  t
('ROOT', ('SBARQ', ('WHNP-1', ('WP', 'what')), ('SQ', ('VP', ('AUX', 'does'), ('NP', ('PRP', 'he')),
('VP', ('VB', 'look'), ('PP', ('IN', 'like'), ('NP', ('-NONE-ABAR-WH-', '*t*-1')))))), ('.', '?')))
```

Example 3

For sentences of this structure we instead allow the *PP* head (in Example 3 the node ('IN', 'like')) to become the *V* head and then continue parsing. The original verb is 'ignored' in favor of treating the trace as the complement of the preposition. Since our learner not concerned with identifying categories of verbs/parts of speech, these can also be serve as evidence for learning the long-distance wh-dependency. Therefore, we expand our criteria and parsing mechanism to allow preposition-trace pairs to be interpreted as gaps requiring wh-licensing.

4 Results

4.1 Generated Treebank Trials

Our learner was consistently able to converge to the goal PFSTA using a generated treebank, created via one of the two methods described earlier. The successful runs typically consisted of the best likelihood from 10 random initializations, which were trained on treebanks of 100 to 200 generated trees. These treebanks were created with an enforced maximum depth of 6, and the initializations imposed the hypothesis space restrictions discussed earlier: assigned terminal states and an assigned initial state. The learner typically converged in 10 to 15 iterations of the EM algorithm. The percentage of types of trees in a treebank of 100 generated trees were as follows: *WH* transitions (33%), *NP* transitions (14%), and *C* only (59%). The average depth of a tree containing a *WH* terminal was 4.34, and the overall average tree depth was 2.83. The resulting learned PFSTA from one of these successful runs can be seen in Figure 4.

```
Q: [0, 1, 2, 3, 4]
I: {1: 1.0}
Delta:
(0, 'WH', ()): 1.0
(1, '*', (0, 4)): 0.3023
(1, '*', (1, 1)): 0.1953
(1, '*', (2, 3)): 0.3767
(1, 'C', ()): 0.1256
(2, 'V', ()): 1.0
(3, 'NP', ()): 1.0
(4, '*', (1, 4)): 0.3229
(4, '*', (2,)): 0.6771
```

Figure 4

4.2 CHILDES Treebank Trials

When our learner was given a random sample of 100 trees from the CHILDES dataset, it did not converge to the desired FSTA. The percentages of the different types of trees in the sample were 0.06 for *WH* sentences, 0.33 for *V-NP* sentences, and 0.61 for *C*-only trees. The treebank summaries in the results showed that there were only 12 *WH* transitions in the sample, compared to 75 *NP* transitions and 123 *C* transitions. On average, the depth of the *WH* trees was 5.67, while the average depth of all the trees was 3.05. These results suggest that not enough

trees in the CHILDES dataset are able to teach wh-licensing or perhaps even V-NP pairing, making it difficult for the learner to converge to the desired PFSTA.

5 Discussion

The over-under algorithm has shown promise in tracking long-distance dependencies. The algorithm's capability of detecting certain structural relations has been demonstrated on schematic, generated data within a restricted hypothesis space.

However, an interesting question remains: why is the expectation-maximization algorithm paired with over-under values able to track long-distance licensing when trained on generated treebanks, but not when trained on CHILDES? Since the hypothesis space is identically restricted in both cases, the learner's ability to converge to the desired FSTA must be due to properties of each of the treebanks.

On average, CHILDES trees are deeper, especially on wh-questions, and the distribution of *WH* trees to *V-NP* trees to *C*-only trees differs as well. There are far fewer wh-transitions in the treebanks sampled from CHILDES, as can be seen in the treebank statistics laid out in the Results section. This suggests that the issue with learning from the CHILDES data is potentially that not enough available trees are able to teach wh-dependency. Our learner made certain assumptions about syntactic structures that children had learned prior to learning wh-dependencies, for instance, the difference between transitive and intransitive verbs. It may be necessary to expand on this assumption and assume that children also learn to differentiate which verbs are solely transitive or might be ditransitive before learning wh-licensing. Similarly, we may perhaps need to assume that children need to learn subject auxiliary inversion first as this would increase the number of trees interpreted by the learner as wh-trees significantly. For both these cases, point of additional assumptions would be to have further information about wh-phrase movement.

Future directions for this research include parsing the discussed ditransitives and inverted questions such that they provide evidence for wh-licensing, introducing more complex terminal labeling and associated PFSTAs to differentiate transitive verbs, ditransitive verbs, copulas, and prepositions. Furthermore, expanding the investigation of wh-dependencies to include subject gap wh-questions would provide additional insight on the acquisition of syntactic structures with wh-movement.

6 References

Annie Gagliardi, Tara M Mease, and Jeffrey Lidz. 2016. *Discontinuous development in the acquisition of filler-gap dependencies: Evidence from 15-and 20-month-olds*. Language Acquisition, 23(3):234–260.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. *Maximum likelihood from incomplete data via the em algorithm*. Journal of the Royal Statistical Society: Series B (Methodological), 39(1):1–22.

Edward Stabler. 1998. *Acquiring languages with movement*. Syntax, 1:72–97.

Haifeng Lee, Keshu Zhang, Tao Jiang. 2005. *The Regularized EM Algorithm*, AAAI-05, 807-812

Jeffrey Lidz, Aaron Steven White, and Rebecca Baier. 2017. *The role of incremental parsing in syntactically conditioned word learning*. Cognitive Psychology, 97:62–78.

Joost Engelfriet. 2015. *Tree automata and tree grammars*. arXiv preprint arXiv:1510.02036.

Kyong-sun Jin and Cynthia Fisher. 2014. *Early evidence for syntactic bootstrapping: 15-month-olds use sentence structure in verb learning*. BUCLD 38 Online Proceedings Supplement.

Laurel Perkins and Jeffrey Lidz. 2020. *Filler-gap dependency comprehension at 15 months: The role of vocabulary*. Language Acquisition, 27(1):98–115.

Lisa Pearl & Jon Sprouse (2013): *Syntactic Islands and Learning Biases: Combining Experimental Syntax and Computational Modeling to Investigate the Language Acquisition Problem*, Language Acquisition, 20:1, 23-68

Yasubumi Sakakibara. 1992. *Efficient learning of context-free grammars from positive structural examples*. 97(1):23–60.

7 Appendix

A. Over-Under & Expectation Maximization algorithms[1]

Algorithm 1: Over-Under algorithm

Input: $maxIteration$,
A randomly initialized PFSTA
$G = \langle Q, \Sigma, I^0, \delta^0 \rangle$
Output: A learnt PFSTA
$G_f = \langle Q, \Sigma, I^f, \delta^f \rangle$
Data: A set of unlabeled trees
$T = \{t_1, t_2, ..., t_n\}$

Function `E-step` (G, T) **:**
  $\mathrm{E}_I, E_\delta = \{\}, \{\}$
  for $i \leftarrow 1$ **to** n **do**
    for $q \in Q$ **do**
      $\mathrm{E}_I[q, t_i] = \dfrac{I(q)\mathcal{U}(t_i, q)}{Pr(t_i)}$
    for $(q, x, [p_1, p_2, p_3 ....p_k]) \in \delta$ **do**
      $y := (q, x, [p_1, p_2, p_3 ....p_k])$
      $\mathrm{E}_\delta[y, t_i] := 0$
      for $k \in ADDRESS(t_i)$ **do**
        if *the node at* $k = x$ **then**
          C = context at k in t_i
          st = subtrees at k in t_i
          $\mathrm{E}_\delta[y, t_i]$+=
          $\dfrac{\mathcal{O}(C, q)\delta(y)\prod_z \mathcal{U}(st_z, p_z)}{Pr(t_i)}$
  return E_I, E_δ

Function `M-step` $(G,\ E_I,\ E_\delta)$ **:**
  for $q \in Q$ **do**
    $I(q) = \dfrac{\sum_{t_i} E_I[q, t_i]}{\sum_{q' \in Q} \sum_{t_i} E_I[q', t_i]}$
  for $(q, x, [p_1, p_2, p_3 ....p_k]) \in \delta$ **do**
    $y := (q, x, [p_1, p_2, p_3 ....p_k])$
    $\delta(y) =$
    $\dfrac{\sum_{t_i} E_\delta[y, t_i]}{\sum_{x' \in \Sigma} \sum_{s_1, ... s_{k'} \in Q} \sum_{t_i} E[(q, x', [s_1, ... s_{k'}]), t_i]}$
  $\mathrm{G}^{new} = \langle Q, I, \delta, \Sigma \rangle$
  return G^{new}

Function `Main` (G, T) **:**
  $iter = 0$
  Repeat until converge
    $\mathrm{E}_I, E_\delta =$ `E-step`(G, T)
    $G =$ `M-step`(G, E_I, E_δ)
    $i+ = 1$
  while $iter \leq maxIteration$

[1] This pseudocode comes directly from what Yang shared with me on Overleaf.

B. CHILDES treebank cleaning:

Below is the series of function calls to transform a CHILDES tree into one we can feed to our learner:

```
remove_animacy(t)                       # remove animacy annotations
remove_trailing_hyphen(t)               # remove trailing hyphens on labels
tuple_tree = from_tuple(t)              # convert from tuple to tree
clean_labels(tuple_tree)                # clean NP, V, WH, and trace labels
drop_punctuation(tuple_tree)            # drop punctuation
tree = collapse_unary(tuple_tree)       # collapse unary branches and
                                          complex V,NP
star_nodes(tree)                        # star all inner nodes
tree = binarize(tree)                   # binarize tree
drop_traces(tree)                       # drop traces
tree.set_address('')                    # set addresses
assign_addresses(tree)
```

Additional description of the parsing process:

The `remove_trailing_hyphen` function takes a string t and removes the trailing hyphen if there is one. The `remove_animacy` function takes the resulting string from `remove_trailing_hyphen` and removes any instances of animacy annotation from the string.

The `from_tuple` function takes the string t and converts it to a tree represented as a tuple.

The `investigate_clean_labels` function takes the tuple tree `tuple_tree` and performs two operations: it rewrites any verb (V) that is the sister of a noun phrase (NP) as a complementizer (C), and it only keeps wh-movement phrases (WHs) that have a trace (i.e., it removes WHs that are not accompanied by a trace).

The `drop_punctuation` function takes the tuple tree tuple_tree and removes any nodes that represent punctuation marks (such as periods or commas).

The `collapse_unary` function takes the tuple tree `tuple_tree` and collapses any unary branches (i.e., branches with only one child) that have the same label. Additionally, it also collapses certain complex structures involving a verb (V) and a noun phrase (NP).

The `star_nodes` function takes the tree tree and "stars" all of the internal nodes of the tree (i.e., all nodes except the leaves). This involves adding a special symbol (an asterisk) to the label of each internal node. The resulting tree is not assigned to a variable since the function modifies the input tree directly.

The `binarize` function takes the tree tree and "binarizes" it, which involves introducing new nodes and branches to ensure that each node in the tree has at most two children.

The `drop_traces` function takes the tree tree and removes any traces (i.e., nodes with a special label that indicate where a moved constituent originally came from). The two lines of code set the address of the root node of the tree to an empty string, and then call the

`assign_addresses` function on the tree. The `assign_addresses` function assigns unique addresses to each node in the tree, based on its position in the tree structure.

C. CHILDES runs (goal PFSTA was *not* learned):

Q: [0, 1, 2, 3, 4]
I: {1: 1.0}
Delta:
(0, '*', (1, 1)): 0.0908
(0, '*', (2, 3)): 0.0159
(0, '*WH*', ()): 0.8934
(1, '*', (0, 4)): 0.0759
(1, '*', (1, 1)): 0.0993
(1, '*', (1, 4)): 0.1302
(1, '*', (2, 3)): 0.0161
(1, '*C*', ()): 0.6785
(2, '*V*', ()): 1.0
(3, '*NP*', ()): 1.0
(4, '*', (1, 1)): 0.2181
(4, '*', (1, 4)): 0.5099
(4, '*', (2, 3)): 0.1106
(4, '*', (2,)): 0.1613
CFG form:
I: N
L → N N
L → V NP
L → *WH*
N → L UL
N → N N
N → N UL
N → V NP
N → *C*
V → *V*
NP → *NP*
UL → N N
UL → N UL
UL → V NP
UL → V