

# Natural Language Processing HW4

Caden Daubach, Yunhao Yang

## Problem 1

- (a) Based on my exploration of the Stanford Parser, there were a few interesting things about the style of trees produced. First of all, the parser does not stick to binary rules as we see primarily in CKY. Instead for a sentence like "Did you go to the grocery store?", the Stanford parser ended up parsing the entire sentence with an SQ→VBD NP VP . initially creating a quaternary structure, and then proceeded at the end to parse the noun phrase "the grocery store" as NP → DET NN NN allowing for a trinary structure at the bottom. Additionally, for consecutive prepositional phrases the parser tended to take each following phrase as describing the previous phrase's object. For example, "I spotted the man with a telescope on the roof" would be parsed as "with a telescope" describing the man and "on the roof" as describing the telescope, creating a step-style structure for parsing consecutive prepositional phrases.

We also tested the Berkeley Neural Parser and observed several key structural patterns. The parser follows Penn Treebank conventions by creating a separate S node inside each subordinate clause (SBAR), so nested sentences like "The cat that ate the rat that stole the cheese ran away" have multiple embedded S structures rather than just one at the top. It correctly handles English's right-branching nature, attaching all relative clauses to the leftmost head noun. The parser also wraps wh-words into phrase-level constituents (WHNP) rather than leaving them as simple word-level tags.

- (b) Things it got wrong: For sentences like "She watched the man with the binoculars" for reasons described in part (a), it incorrectly parsed "with the binoculars" as a description of the man instead of the correct meaning describing how she watched the man. Also, in a sentence like "The man visited yesterday" the Stanford parser misclassifies "yesterday" as a Noun instead of its correct grammatical classification of an adverb or adverb phrase.

The parser from Berkeley failed on the garden-path sentence "The horse raced past the barn fell." It incorrectly treated "raced" as the main verb instead of recognizing it as a past participle in a reduced relative clause. The correct interpretation should be "The horse [that was] raced past the barn fell," where "fell" is the main verb.

Things it got right: The Stanford parser handled difficult parses including relative clauses like the sentence "The man who is a friend of mine visited yesterday". Additionally, the Stanford parser correctly groups modifiers like in the sentence "The big and friendly dog barked loudly". It correctly associated "big and friendly" with the noun phrase and "loudly" with the verb phrase.

The Berkeley parser correctly handled the inverted sentence "Never have I seen such a thing!" using the SINV label and proper constituent order (ADVP + auxiliary + subject + VP).

- (c) The sentence "The complex houses married and single soldiers and their families." expectedly tricks the parser. In reality, the primary noun phrase in this sentence is "The complex" and the verb is "houses" as in an apartment complex that houses soldiers. However the Stanford parser interprets this a complex house (NP) that gets married (VP) and then essentially gives up on the parse after that as the rest of the sentence will not make sense when interpreted in that way.

We also tested "The senator that the journalist that the editor hired interviewed resigned" with double center-embedding. The parser incorrectly tagged "interviewed" as a past participle (VBN) instead of a past tense verb (VBD), parsing "hired interviewed" as if it meant "hired [someone who was] interviewed" rather than recognizing that the journalist actively interviewed the senator. This error occurs because consecutive past tense verbs in nested clauses are extremely rare in Wall Street Journal training data, so the parser defaults to the more common "verb + past participle" pattern even though it creates an incomplete and semantically incorrect structure.

## Problem 2

- (a) I experimented with both constituency parsers (Berkeley Neural Parser) and dependency parsers (spaCy) to understand their different approaches to sentence structure. Constituency parsing builds hierarchical phrase structures with non-terminal nodes like NP, VP, and S, explicitly marking phrase boundaries and nested constituents. For example, it represents "The cat that ate the rat" with multiple embedded clauses, each having its own S node within SBAR structures. In contrast, dependency parsing is fundamentally verb-centric, treating the main verb as ROOT and connecting all other words through labeled grammatical relations like nsubj (subject), dobj (object), and prep (preposition). In "Never have I seen such a beautiful sunset," the dependency parser directly marks "seen" as ROOT with "I" as its subject and "Never" as a negation modifier, without creating intermediate phrase-level nodes.

Both parsing frameworks have distinct advantages for different purposes. Constituency parsing better captures phrase boundaries and hierarchical structure, making it useful for understanding how words group into constituents and for representing phenomena like VP-ellipsis. It also aligns with traditional linguistic theory from Chomsky's generative grammar. Dependency parsing, however, more directly represents grammatical relationships and works well across languages with different word orders, since dependency relations remain similar even when word order varies. The flatter structure of dependency trees ( $O(n)$  complexity) also makes them computationally more efficient than constituency trees, and the explicit labeling of grammatical functions makes it easier to extract information like "who did what to whom."

Interestingly, both parsers struggled with the same challenging constructions. The garden-path sentence "The horse raced past the barn fell" confused both frameworks: Berkeley's constituency parser

incorrectly tagged "interviewed" as a past participle (VBN) in the similar sentence about the senator and journalist, while spaCy's dependency parser wrongly chose "raced" as ROOT instead of "fell." This suggests that despite their different representational approaches, modern neural parsers face similar difficulties with rare syntactic constructions that don't match their training data patterns, particularly reduced relative clauses that create temporary ambiguity.

(b) We tested spaCy's Chinese dependency parser and found several issues:

First, ROOT selection is inconsistent across sentence types. While the parser correctly identifies verbs as ROOT in simple sentences like "He reads books in the library," it becomes unreliable with more complex constructions. In serial verb sentences such as "She sits on the chair watching TV," where two verbs appear without explicit conjunctions or infinitive markers, the parser struggles to determine which verb is primary. Unlike English, which uses "to" or "and" to clarify relationships between verbs, Chinese allows multiple verbs to appear consecutively, and their relationship must be inferred from context.

Second, the parser fails on long sentences with nested relative clauses. Consider the sentence meaning "The girl who came yesterday who wore red clothes is my sister." This sentence contains three verbs (came, wore, is), where only the copula "is" should be the main predicate. However, the parser cannot reliably identify this because Chinese relative clauses are marked only by the particle "de," which serves multiple functions—possession, modification, and relativization—without the explicit clause boundaries that English provides through "who," "that," or "which." The parser becomes confused by multiple instances of "de" and multiple verbs, often selecting an embedded verb as ROOT instead of the true main predicate.

Third, Chinese-specific grammatical constructions are poorly handled. The "ba" construction, which fronts objects before the verb (similar to "I the-book placed on the table"), is tagged with "X" (unknown category). This indicates that the Universal Dependencies annotation scheme lacks appropriate categories for Chinese-unique structures. Similarly, the "bei" passive construction shows inconsistent annotation patterns.

These parsing difficulties stem from fundamental typological differences. Chinese lacks morphological inflection—no tense markers, no agreement markers, no case marking—forcing parsers to rely entirely on word order and context. Moreover, Chinese allows extensive argument dropping: subjects and objects can be omitted when contextually clear, but parsers trained on explicit syntax struggle with these null elements. The ambiguity of the function particle "de," which English distributes across multiple distinct markers ('s, that, who, which, of), compounds the difficulty.

Compared to English constituency parsers like Berkeley, which benefit from clear phrase boundaries and explicit subordination markers, Chinese dependency parsers must make far more inference-based decisions. Despite using similar neural architectures and achieving comparable accuracy scores on

standard benchmarks, Chinese parsers make qualitatively different types of errors—particularly confusion about clause structure and verb relationships rather than the attachment ambiguities common in English parsing.

## Problem 3

(a) Correctness:

Our weighted Earley parser keeps track of each item’s best derivation and total weight by maintaining a single canonical record for each state  $(A \rightarrow \alpha \cdot \beta, i)$ . Each item is represented as:

```
Item(rule, dot_position, start_position, weight, backpointer)
```

where **weight** stores the negative log-probability  $(-\log_2 P(\text{derivation}))$ , and **backpointer** records the previous item and constituent used to extend the derivation.

Items are uniquely identified by their tuple **(rule, dot\_position, start\_position)**. Each column of the chart maintains a hash index:

```
_index[(rule, dot_position, start_position)]  $\rightarrow$  item_index
```

This enables constant-time duplicate detection.

Whenever a new derivation is generated, we check if it already exists in the index:

- If it is new, we add it to the agenda.
- If it exists but the new item has a lower weight, we replace the existing one. If the replaced item has already been processed, we move it back to the unprocessed region by updating the pointer.

This follows Section B.2 of the reading: when a lower-weight duplicate appears, the parser reprocesses it to ensure that the improvement propagates to its customers. As a result, each item’s stored weight always reflects the best derivation discovered so far, and the chart ultimately represents the highest-probability parse tree.

(b) Time complexity:

**Space complexity.** The chart has  $n + 1$  columns (for positions  $0 \dots n$ ). Each column contains at most  $O(n)$  unique items, since each represents a distinct combination of (rule, dot, start). Thus, the total space complexity is  $O(n^2)$ .

**Time complexity:** Each of the three operations of Earley’s algorithm—PREDICT, SCAN, and ATTACH—runs in constant or linear time per item and collectively yield the standard  $O(n^3)$  time complexity. Weighted parsing introduces reprocessing when lower-weight duplicates are found, but because

this “move back” operation is  $O(1)$ , the expected runtime remains close to  $O(n^3)$  even though worst-case reprocessing can exceed that.

Constant-time push and duplicate check: The `Agenda` class uses a hash map to maintain constant-time duplicate detection:

```
_index : Dict[(rule, dot, start), int]
```

Each push or update is therefore  $O(1)$ , including duplicate checking. This is crucial to maintaining cubic time, since a linear search over previously added items would increase the runtime beyond  $O(n^3)$ .

**Extra credit discussion.** Reprocessing is needed because a better derivation can appear after an item has already been processed. One way to avoid it is to compute items in increasing weight order, similar to Dijkstra’s algorithm. Such a modification could achieve  $O(n^3 \log n)$  or even  $O(n^3)$  runtime with additional indexing structures, though it increases implementation complexity.

## Problem 4

We implemented two optimization techniques from the reading: Batch Duplicate Check (E.1) and Indexing Customers (E.5).

The Batch Duplicate Check optimization addresses redundant prediction operations. In the naive implementation, when multiple items need the same nonterminal at a position, the parser attempts to predict all rules for that nonterminal repeatedly. We maintain a set `_predicted` in each column that tracks which nonterminals have already been predicted. Before predicting rules for nonterminal  $A$  at position  $i$ , we check if  $A \in \text{cols}[i].\_predicted$ . If yes, we skip the prediction entirely since all rules are already in the column. This ensures each nonterminal is predicted at most once per position.

The Indexing Customers optimization improves the attach operation. Originally, when attaching a completed constituent, the parser searches linearly through all items in a column to find customers. We add a dictionary `_customers` to each `Agenda` that maps symbols to items waiting for them: `_customers[symbol] → [items]`. When pushing an item, we add it to `_customers[item.next_symbol()]`. During attach, instead of `self.cols[mid].all()`, we use `self.cols[mid].get_customers(item.rule.lhs)` to directly access only relevant customers. This reduces the search from  $O(n)$  to  $O(k)$  where  $k$  is the actual number of customers, typically much smaller than column size  $n$ .

Version	Real Time	User Time	Speedup
<code>parse.py</code> (original)	17.296s	17.101s	1.0×
<code>parse2.py</code> (E.1 + E.5)	2.161s	2.125s	<b>8.0×</b>

Table 1: Performance on first 3 sentences of `wallstreet.sen` using PyPy3

The optimizations achieved an 8× speedup on the first three sentences. For the full `wallstreet.sen` (9 sentences), `parse2.py` completed in approximately 6 minutes total. The first three sentences finished in

under 2 seconds, sentences 4-6 took 6-16 seconds each, and the two longest sentences (40 and 38 tokens with deep embedding) required 2-3 minutes each.

Examining the parses in `wallstreet.par` reveals several patterns. Sentence 1 ("John is happy .") produces a clean parse with weight 34.224, correctly identifying the copular construction with ADJP-PRD. Sentence 3 successfully handles an embedded wh-question with weight 94.581, properly nesting "what General Motors has done" as a complement of "wondering." However, longer sentences show structural challenges. Sentence 6 (weight 212.545) involves complex quantifier phrases "4% to 5% annualized increases" where multiple PP attachments create ambiguity. The parser resolves these based on training data frequencies rather than semantic plausibility.

The longest sentence (Sentence 7, 40 tokens, weight 349.133) demonstrates PCFG limitations with seven levels of VP embedding in the participial phrase "running the combined companies, projected to have nearly 22 million subscribers..." While grammatically correct under Penn Treebank conventions, the deep right-branching structure reflects the parser's inability to use semantic information for structural preferences. Throughout the parses, we observe Penn Treebank idiosyncrasies like function tags (-PRD, -NOM, -ADV), separate PUNC constituents, and NPR structures for proper names, all faithfully reproduced from the training corpus.

Compared to neural parsers from Problem 1, our PCFG parser produces structures that are statistically optimal under the training distribution but sometimes differ from human intuitions. Neural parsers can incorporate lexical semantics and context through learned representations, while our parser relies purely on grammar rule probabilities. Despite this limitation, the optimizations make it practical to parse moderately complex Wall Street Journal sentences, though very long or deeply embedded constructions remain computationally expensive.