Efficient Weighted Deduction Systems for Earley's Algorithm

Anonymous ACL submission

Abstract

The parsing algorithm of Earley (1970), as presented, has a runtime complexity of $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$ where N is the length of the sentence, $|\mathcal{G}|$ is the size of the grammar, and $|\mathcal{R}|$ is the number of productions in the grammar. This is unworkable for the large grammars that arise in natural language processing. Fortunately, the dynamic programming algorithm can be improved to run in time $\mathcal{O}\big(N^3|\mathcal{G}|\big)$, matching the complexity of running CKY on a binarized version of $\mathcal{G}$. Some of the necessary speed-ups have been presented in part or in full in various parts of the literature. However, there has been no unified, formal treatment that is written as a deduction system[1]or covers the weighted case. We present such a treatment in terms of five proof rules that can be used in weighted deduction, which refine Earley's PREDICT, SCAN and COMPLETE actions. We also provide a generalization of Earley's algorithm that uses a finite-state automaton to represent the grammar, and whose runtime is proportional to the size of the automaton (and the usual $\mathcal{O}\big(N^3\big)$ term), or more precisely the size of the portion of the automaton that is reached while parsing the input sentence. Further speed-ups can then be achieved by minimizing the automaton so that similar productions share transitions.

1 Introduction

Earley's algorithm (1970) was a landmark achievement in theoretical computer science. It was the first algorithm that could directly parse under an *arbitrary* context-free grammar in time $\mathcal{O}\big(N^3\big)$ (N being the length of the input string). Also, since it exhaustively filters rules by left context, it parses unambiguous grammars in $\mathcal{O}\big(N^2\big)$ time and a class of deterministic context-free grammars in $\mathcal{O}(N)$.

Earley's algorithm is well-known in computational linguistics and NLP, and not only because of its ability to directly handle unrestricted grammars and exploit their structure for potential speedups. Because it parses incrementally from left to right, it can be used for online sentence processing, maintaining a parse forest over the sentence prefix that has been seen so far—which supports incremental syntactic featurization and incremental semantic interpretation—as well as the set of grammatical next words.[2] It can be attractively extended to compute the probabilities of these next words (Stolcke, 1995), which is the standard way to compute autoregressive language model probabilities under a PCFG to support cognitive modeling (Hale, 2001), speech recognition (Roark, 2001), and neural generation of grammatical text (Shin et al., 2021).

The runtime of Earley's algorithm is cubic in the length of a sentence. However, an often overlooked aspect in the algorithm's analysis is the grammar constant. Earley's algorithm takes $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$ time where $|\mathcal{G}|$ is the size of the grammar (the total length of all productions) and $|\mathcal{R}|$ is the number of productions (Shieber et al., 1995; Stolcke, 1995; Goodman, 1999). When the grammar is taken to be constant, these terms are absorbed into the $\mathcal{O}$ operator. Indeed, Earley (1970)'s original paper did not discuss the grammar constant in its runtime analysis. However, natural language grammars can be very large (Dunlop et al., 2010). For example, the Berkeley grammar (Petrov et al., 2006), a learned grammar for the Penn Treebank (PTB) (Marcus et al., 1993), contains over one million productions. Grammars of such magnitude make the $\mathcal{O}(|\mathcal{G}||\mathcal{R}|)$ factor an intimidating prospect.

Fortunately, some massaging of Earley's dynamic program makes it run in $\mathcal{O}\big(N^3|\mathcal{G}|\big)$. This matches the runtime of the CKY parsing algorithm on a binarized version of $\mathcal{G}$ (when $\mathcal{G}$ can be binarized, i.e., it has no unary or nullary productions). However, the literature around Earley's

[1]That said, declarative formulations have been presented in other formats in the dissertations of Barthélemy (1993), de la Clergerie (1993), and Nederhof (1994).

[2]In a programming language editor, incremental interpretation can support syntax checking, syntax highlighting, and tooltips; next-word prediction can support autocomplete.

often misses this bound or over-complicates the explanation of the necessary speed-ups:

- Standard presentations of Earley's algorithm (e.g., the textbook treatment of Jurafsky and Martin, 2009, Section 13.4) typically give the runtime as $\mathcal{O}(N^3)$, eliding the grammar constant.
- The Graham–Harrison–Ruzzo (GHR) algorithm, a well known variant of Earley's, is a non-weighted recognizer that again runs in $\mathcal{O}(N^3|\mathcal{G}||\mathcal{R}|)$. The original exposition of Graham et al. (1980) did provide implementation details in their Section 3 that enables the algorithm to run in the improved $\mathcal{O}(N^3|\mathcal{G}|)$, but these details were not presented in the form of a deduction system and were apparently lost in retellings of the algorithm (Sikkel, 1993, page 112).
- Leermakers (1992) and Moore (2000) improve the efficiency of the COMPLETE rule, but not the PREDICT rule. This leads to an algorithm that runs in $\mathcal{O}(N^3|\mathcal{G}| + N^2|\mathcal{G}||\mathcal{R}|)$ time. This is still unusable for large grammars.
- Klein and Manning (2001a) propose an agenda-based Earley parser that distinguishes "active" and "passive" items (Gebhardt, 2015). This improves the runtime to $\mathcal{O}(N^3|\mathcal{G}||\mathcal{N}|)$ where $\mathcal{N}$ is the set of non-terminals in the grammar. The extra factor of $\mathcal{O}(|\mathcal{N}|)$ is still quite large and could have been dispensed with.

In this paper, we provide a simple deduction system, EARLEY–FAST, that can be used to discover all parses and compute their total semiring weight in $\mathcal{O}(N^3|\mathcal{G}|)$ time.[3] We compare EARLEY–FAST to the traditional deduction system, EARLEY. We discuss modifications in §5 and Apps. A and B that must be made to the grammar in order to avoid repeatedly deducing new proofs of an item (due to cyclic derivations of a string via unary and/or nullary rules), which can prevent the weighted version of Earley's algorithm from terminating.

We conduct a small empirical experiment using various grammars induced from the PTB and no pruning. For all of these grammars, EARLEY requires over 3 minutes to parse a single sentence from the PTB over 25 words, which is not practical. Nevertheless, we are able to parse sentences under 40 words using EARLEY–FAST in 7 seconds (again, without pruning). In §6, we additionally present a generalization in which dotted productions (within items of the deduction system) are replaced by states of a finite-state automaton (FSA). This generalization, which we call EARLEY–FSA, allows us to shrink the grammar constant further, yielding a further $2.5\times$ speed-up when using the PTB-induced grammars.

[3] In the weighted case (§5), this $\mathcal{G}$ is a preprocessed version of the grammar. Preprocessing takes additional time and may enlarge the grammar. Also, our runtimes assume that the semiring operations $\oplus$, $\otimes$, and * take $O(1)$ time each.

2 Weighted Context-Free Grammars

A **context-free grammar** (**CFG**) $\mathcal{G}$ is a tuple $\langle \mathcal{N}, \Sigma, \mathcal{R}, S \rangle$ where Σ is a finite set of **terminal** symbols, $\mathcal{N}$ is a finite set of **non-terminal** symbols with $\Sigma \cap \mathcal{N} = \emptyset$, $\mathcal{R}$ is a set of **productions** from a non-terminal to a sequence of non-terminals and terminals (i.e., $\mathcal{R} \subseteq \mathcal{N} \times (\mathcal{N} \cup \Sigma)^*$), and $S \in \mathcal{N}$ is the **start** symbol. We denote terminal symbols by lower-case letters ($a, b, \ldots$) and non-terminal symbols by upper-case letters ($A, B, \ldots$). We use a Greek letter (ρ, μ, or ν) to denote a sequence of non-terminals and terminals, i.e., an element of $(\mathcal{N} \cup \Sigma)^*$. Therefore, a production has the form $A \to \rho$. Note that ρ may be the empty sequence ϵ. We refer to $|\rho| \geq 0$ as the **arity** of the production and $|A \to \rho| \stackrel{\text{def}}{=} 1 + |\rho|$ as the **size** of the production. Productions of arity 0, 1, and 2 are referred to as **nullary**, **unary**, and **binary** productions respectively. We write $|\mathcal{G}| \stackrel{\text{def}}{=} \sum_{(A \to \rho) \in \mathcal{R}} |\rho|$ for the total **size** of the CFG. Therefore, if K is the maximum arity of a production, $|\mathcal{G}| \leq |\mathcal{R}|(1 + K)$.

For a given $\mathcal{G}$, we write $\mu \Rightarrow \nu$ to denote that $\mu \in (\mathcal{N} \cup \Sigma)^*$ can be rewritten into ν by a single production of $\mathcal{G}$. For example, $A\,B \Rightarrow \rho\,B$ expands A into ρ using the production $A \to \rho$. The reflexive transitive closure of this relation, $\stackrel{*}{\Rightarrow}$, then denotes rewriting by any sequence of zero or more productions: for example, $A\,B \stackrel{*}{\Rightarrow} \rho\,\mu\,\nu$.

A **derivation tree** of $\mathcal{G}$ is a finite rooted ordered tree T whose leaves are labeled with elements of Σ and whose internal nodes are labeled with elements of $\mathcal{N}$, such that (1) the root is labeled with S and (2) if an internal node labeled with A has a sequence of children labeled with ρ, then $A \to \rho$ is in $\mathcal{R}$. Given an **input sentence** $\mathbf{x} \in \Sigma^*$ of length N, we write $\mathcal{T}_{\mathbf{x}}$ for its set of derivation trees, that is, all trees with leaf sequence $\mathbf{x}$. $\mathcal{T}_{\mathbf{x}}$ is countable and possibly infinite. It is non-empty iff $S \stackrel{*}{\Rightarrow} \mathbf{x}$, with each $T \in \mathcal{T}_{\mathbf{x}}$ serving as a witness that $S \stackrel{*}{\Rightarrow} \mathbf{x}$.

We will also consider **weighted CFGs**, in which each production $A \to \rho$ is additionally equipped with a **weight** $w(A \to \rho) \in \mathbb{W}$ where $\mathbb{W}$ is

	EARLEY	EARLEY–FAST
Domains	$i, j, k \in \{0, \ldots, N\} \quad A, B \in \mathcal{N} \cup \{\widehat{S}\} \quad a \in \Sigma \quad \rho, \mu, \nu \in (\mathcal{N} \cup \Sigma)^*$	
Items	$[i, j, A \to \mu \bullet \nu] \quad [j, k, a] \quad A \to \rho$	$[i, j, A \to \mu \bullet \nu] \quad [j, k, a] \quad A \to \rho$ $[i, j, A \to \bullet \star] \quad [i, j, A \to \star \bullet]$
Axioms	$A \to \rho, \forall (A \to \rho) \in \mathcal{R}$ $[k-1, k, x_k], \forall k \in \{1, \ldots, N\}$ $[0, 0, \widehat{S} \to \bullet S]$	$A \to \rho, \forall (A \to \rho) \in \mathcal{R}$ $[k-1, k, x_k], \forall k \in \{1, \ldots, N\}$ $[0, 0, S \to \bullet \star]$
Goal	$[0, N, \widehat{S} \to S \bullet]$	$[0, 0, S \to \star \bullet]$
Rules	PRED: $\dfrac{B \to \rho}{[j, j, B \to \bullet \rho]}\ [i, j, A \to \mu \bullet B\, \nu]$	PRED1: $\dfrac{}{[j, j, B \to \bullet \star]}\ [i, j, A \to \mu \bullet B\, \nu]$ PRED2: $\dfrac{B \to \rho}{[j, j, B \to \bullet \rho]}\ [j, j, B \to \bullet \star]$
	SCAN: $\dfrac{[i, j, A \to \mu \bullet a\, \nu] \quad [j, k, a]}{[i, k, A \to \mu\, a \bullet \nu]}$	SCAN: $\dfrac{[i, j, A \to \mu \bullet a\, \nu] \quad [j, k, a]}{[i, k, A \to \mu\, a \bullet \nu]}$
		COMP1: $\dfrac{[j, k, B \to \rho \bullet]}{[j, k, B \to \star \bullet]}$
	COMP: $\dfrac{[i, j, A \to \mu \bullet B\, \nu] \quad [j, k, B \to \rho \bullet]}{[i, k, A \to \mu\, B \bullet \nu]}$	COMP2: $\dfrac{[i, j, A \to \mu \bullet B\, \nu] \quad [j, k, B \to \star \bullet]}{[i, k, A \to \mu\, B \bullet \nu]}$

Table 1: Deduction systems for Earley (1970)'s algorithm (EARLEY) and our faster algorithm (EARLEY–FAST).

the set of values of a **closed semiring** $\mathcal{S} \stackrel{\text{def}}{=} \langle \mathbb{W}, \oplus, \otimes, *, \mathbb{0}, \mathbb{1} \rangle$. A semiring $\mathcal{S}$ has two operators, $\oplus$, which is associative and commutative, and $\otimes$, which is associative and distributes over $\oplus$. Additionally, the semiring contains values $\mathbb{0}, \mathbb{1} \in \mathbb{W}$ such that $\mathbb{0}$ is an identity for $\oplus$ and annihilator for $\otimes$ and $\mathbb{1}$ is an identity for $\otimes$. A closed semiring additionally has an operator $*$ satisfying the axiom $(\forall w \in \mathbb{W})\ w^* = \mathbb{1} \oplus w \otimes w^* = \mathbb{1} \oplus w^* \otimes w$. The interpretation is that w^* returns the infinite sum $\mathbb{1} \oplus w \oplus (w \otimes w) \oplus (w \otimes w \otimes w) \oplus \cdots$.

For any A and ρ, any derivation tree T that establishes $A \stackrel{*}{\Rightarrow} \rho$ can be given a weight

$$w(T) \stackrel{\text{def}}{=} \bigotimes_{(A \to \rho) \in T} w(A \to \rho) \tag{1}$$

where $A \to \rho$ ranges over the productions associated with the internal nodes of T.[4] We then write $w\left(A \stackrel{*}{\Rightarrow} \rho\right)$ for the total weight of all such derivations. In particular, the goal of a **weighted recognizer** is to find the total weight of all derivation trees of a given input sentence $\mathbf{x}$:

$$\mathrm{Z}_{\mathbf{x}} \stackrel{\text{def}}{=} w\left(S \stackrel{*}{\Rightarrow} \mathbf{x}\right) = \bigoplus_{T \in \mathcal{T}_{\mathbf{x}}} w(T) \tag{2}$$

[4]The productions appear in this product in the order of a prefix traversal of T; this is important if $\otimes$ is not commutative.

An ordinary unweighted recognizer is the special case where $\mathbb{W}$ is the boolean semiring, so $\mathrm{Z}_{\mathbf{x}} = \text{true}$ iff $S \stackrel{*}{\Rightarrow} \mathbf{x}$ iff $\mathcal{T}_{\mathbf{x}} \neq \emptyset$. A **parser** is a recognizer that returns one or more derivation trees—this can be achieved using the derivation semiring (Goodman, 1999) or by storing the provenance of each derived item.

3 Earley's Algorithm

We describe Earley's algorithm using a **deduction system**, which is effectively a version of the sequent calculus (Pierce, 2002) that is often employed in the presentation of parsing algorithms and other logic programs (Pereira and Shieber, 1987). Much is known about how to execute (Goodman, 1999), transform (Eisner and Blatz, 2007), and neuralize (Mei et al., 2020) deduction systems.

A deduction system derives **items** using **deductive rules**. Items represent propositions; the rules are used to derive all propositions that are true. A deductive rule is of the form

$$\text{EXAMPLE:}\ \frac{X \qquad Y \qquad \cdots}{Z}$$

where EXAMPLE is the name of the rule, the 0 or more items above the bar are called **antecedents**, and the single item below the bar is called a **consequent**. Antecedents may also be written to the side

of the bar; these are called **side conditions** and will be handled differently for weighted parsing in §5. **Axioms** are special rules that have no antecedents; as a shorthand, we omit the bar in this case and simply write the consequent.

Our unweighted recognizer will determine whether a certain **goal item** is provable by a certain set of deductive rules from axioms that encode $\mathcal{G}$ and $\mathbf{x}$. The deduction system is set up so that this is the case iff $S \overset{*}{\Rightarrow} \mathbf{x}$. The provability of an item under a deduction system can be generically solved through forward chaining. An unweighted parser is a version of this method that does some extra bookkeeping and thus is able to return one or more actual proofs of the goal item, which correspond to derivation trees. In general, a **proof tree** (or just **proof**) d of an item is a tree-structured proof of that item.

Pereira and Warren (1983) first presented Earley's algorithm as a deduction system, shown as EARLEY in Table 1. Some items have the form $[i, j, A \to \mu \bullet \nu]$. The **span** (i, j) refers to a contiguous segment $\mathbf{x}_{i:j} \overset{\text{def}}{=} x_{i+1} \cdots x_j$ of the input sentence $\mathbf{x}$. $A \to \mu\nu$ is a production in $\mathcal{G}$, and $\bullet$ marks a position in the production. The item $[i, j, A \to \mu \bullet \nu]$ is derivable only if the grammar has a production $A \to \mu\,\nu$ such that $\mu \overset{*}{\Rightarrow} \mathbf{x}_{i:j}$. Therefore, $\bullet$ indicates the progress we have made through the production. An item with nothing to the right of $\bullet$, e.g., $[i, j, A \to \rho \bullet]$ is called **completed**. The grammar $\mathcal{G}$ is encoded by axioms $A \overset{*}{\Rightarrow} \rho$ that correspond to the productions of the grammar. The input sentence $\mathbf{x}$ is encoded by axioms of the form $[k-1, k, a]$ where $a \in \Sigma$; this axiom is true iff $\mathbf{x}_{k-1:k} = x_k = a$.[5]

While $\mu \overset{*}{\Rightarrow} \mathbf{x}_{i:j}$ is a necessary condition for $[i, j, A \to \mu \bullet \nu]$ to be provable, it is not sufficient. For efficiency, the EARLEY deduction system is cleverly constructed so that this item is provable iff[6] it can appear in a proof of some string that begins with $\mathbf{x}_{0:j}$, and thus might appear in a derivation of $\mathbf{x}$.[7]

[5] All methods in this paper can be also applied directly to lattice parsing, in which i, j, k range over states in an acyclic lattice of possible input strings, and 0 and N refer to the unique initial and final states. A lattice edge from j to k labeled with terminal a is then encoded by the axiom $[j, k, a]$.

[6] This characterization assumes that every non-terminal $B \in \mathcal{N}$ can be expanded by some rule $B \to \rho$. If not, B is **useless** and can be safely eliminated from the grammar, along with any production in $\mathcal{R}$ that mentions B. This may create new useless non-terminals that can be eliminated in turn.

[7] Earley (1970) also generalized the algorithm to prove this item only if it can appear in a proof of some string that begins with $\mathbf{x}_{0:(j+\Delta)}$, for a fixed Δ. This is lookahead of Δ tokens.

Including $[0, 0, \widehat{S} \to \bullet\, S]$ as an axiom in the system effectively causes forward chaining to start looking for a derivation at position 0. The system has proved $S \overset{*}{\Rightarrow} \mathbf{x}$ if it can derive the goal item $[0, N, \widehat{S} \to S \bullet]$, where $N = |\mathbf{x}|$. These two items conveniently pretend that the grammar has been augmented with a new start symbol $\widehat{S}$ that only rewrites according to the single production $\widehat{S} \to S$.

Earley's algorithm has three deduction rules: PREDICT, SCAN, and COMPLETE. We describe each of these rules and their runtime in detail. Additionally, we discuss how past work has improved on the runtime of the different rules in §3.4. We analyze the runtime of the deduction system following McAllester (2002): for each deduction rule, we examine the domain size of its free variables.

3.1 Predict

To look for constituents of type B starting at position j, using the rule $B \to \rho$, we need to derive $[j, j, B \to \bullet\, \rho]$. Earley's algorithm imposes $[i, j, A \to \mu \bullet B\, \nu]$ as a side condition, so that we only start looking if such a constituent B could be combined with some item to its left.[8]

$$\textsc{Pred}: \frac{B \to \rho}{[j, j, B \to \bullet\, \rho]}\ [i, j, A \to \mu \bullet B\, \nu]$$

Runtime analysis. PRED has four free variables: indices i and j with a domain size of $N+1$, dotted production $A \to \mu \bullet B\, \nu$ with domain size $|\mathcal{G}|$, and production rule $B \to \rho$ with a domain size of $|\mathcal{R}|$. Therefore, its total runtime is $\mathcal{O}\big(N^2|\mathcal{G}||\mathcal{R}|\big)$.

3.2 Scan

If we have derived an incomplete item $[i, j, A \to \mu \bullet\, a\, \nu]$, we can advance the dot if the next terminal symbol is a:

$$\textsc{Scan}: \frac{[i, j, A \to \mu \bullet\, a\, \nu] \quad [j, k, a]}{[i, k, A \to \mu\, a \bullet \nu]}$$

This makes progress toward completing the A.

Runtime analysis. SCAN has four free variables: indices i, j, and k with a domain size of $N+1$, and dotted production $A \to \mu \bullet B\, \nu$ with domain size $|\mathcal{G}|$. However, since we consider terminal symbols to have a span width of 1, it is the case that $j = k-1$ and so SCAN has a total runtime of $\mathcal{O}\big(N^2|\mathcal{G}|\big)$.

[8] Minnen (1996) and Eisner and Blatz (2007) explain that this side condition is an instance of the "magic sets" technique that filters some unnecessary work from a bottom-up algorithm (Ramakrishnan, 1991).

3.3 Complete

Recall that having $[i, j, A \to \mu \bullet B\,\nu]$ allowed us to start looking for a B at position j (PRED). Once we have found a complete B by deriving $[j, k, B \to \rho \bullet]$, we can advance the dot in the former rule:

$$\textsc{Comp:}\ \frac{[i, j, A \to \mu \bullet B\,\nu] \quad [j, k, B \to \rho \bullet]}{[i, k, A \to \mu\, B \bullet \nu]}$$

Runtime analysis. COMP has five free variables: indices i, j, and k with a domain size of $N + 1$, dotted production $A \to \mu \bullet B\,\nu$ with domain size $|\mathcal{G}|$, and the completed production $B \to \rho$ with a domain size of $|\mathcal{R}|$. Therefore, COMP will have a total runtime of $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$.

3.4 Previous Speed-ups

Putting the above steps together, the total runtime of the EARLEY algorithm is $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$. In addition, the number of possible derived items is $\mathcal{O}\big(N^2|\mathcal{G}|\big)$, which is a bound on the space needed by the algorithm to store the items that have been derived so far and index them for fast lookup (McAllester, 2002; Eisner et al., 2005). We briefly discuss past approaches used to improve the asymptotic efficiency.

Leermakers (1992) noted that in an item of the form $[i, j, A \to \mu \bullet \nu]$, the sequence μ is irrelevant to subsequent deductions. Therefore, he suggested (in effect) replacing μ with a generic placeholder $\star$. This merges items that had only differed in their μ values, so the algorithm processes fewer items. This technique can also be seen in Moore (2000) and Klein and Manning (2001a,b). Importantly, this means that each non terminal only has one completed state, $[j, k, B \to \star \bullet]$, for each span. This improves the runtime of Earley's to $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{N}| + N^2|\mathcal{G}||\mathcal{R}|\big)$. Our §4.2 will give a version of the trick that only gets this effect. The full version of Leermakers (1992)'s trick is subsumed by our generalized approach in §6.

While the GHR algorithm—a modified version of Earley's algorithm—is commonly known to be $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$, Graham et al. (1980, Section 3) provide a detailed exploration of the low-level implementation of their algorithm that enables it to be run in $\mathcal{O}\big(N^3|\mathcal{G}|\big)$ time. This explanation spans 20 pages and includes techniques similar to those mentioned in §4, as well as discussion of data structures. To the best of our knowledge, these details have not been carried forward in subsequent presentations of GHR (Stolcke, 1995; Goodman, 1999).

4 An Improved Deduction System

Our EARLEY–FAST deduction system, shown in the right column of Table 1, shaves a factor of $\mathcal{O}(\mathcal{R})$ off the runtime of EARLEY. We introduce items $[i, j, A \to \bullet\star]$ and $[i, j, A \to \star\bullet]$ that will be used to speed up PRED (§4.1) and COMP (§4.2) respectively. We can also use these items to replace the goal item and the axiom that used $\widehat{S}$; the extra $\widehat{S}$ symbol is no longer needed. In the remainder of this section, we describe our new deduction rules for COMP and PRED. (SCAN is unchanged.)

4.1 Predict

We split PRED into two rules: PRED1 and PRED2. The first rule, PRED1, creates an item that gathers together all requests to look for a given nonterminal B starting at a given position j:

$$\textsc{Pred1:}\ \frac{}{[j, j, B \to \bullet\star]}\ [i, j, A \to \mu \bullet B\,\nu]$$

There are three free variables in the rule: indices i and j, and dotted production $A \to \bullet\, B\,\nu$. Therefore, PRED1 has a total runtime of $\mathcal{O}\big(N^2|\mathcal{G}|\big)$.

The second rule, PRED2, expands the item into commitments to look for each specific kind of B:

$$\textsc{Pred2:}\ \frac{[j, j, B \to \bullet\star]}{[j, j, B \to \bullet\rho]}\ B \to \rho \in \mathcal{R}$$

PRED2 has two free variables: index j and production $B \to \rho$. Therefore, PRED2 has a runtime of $\mathcal{O}(N|\mathcal{G}|)$, leading to both and so the two rules together have a runtime of $\mathcal{O}\big(N^2|\mathcal{G}|\big)$.

4.2 Complete

We speed up COMP in a similar fashion to PRED. We split COMP into two rules: COMP1 and COMP2. The first rule, COMP1, gathers all completed B constituents over a given span into a single item:

$$\textsc{Comp1:}\ \frac{[j, k, B \to \rho \bullet]}{[j, k, B \to \star\bullet]}$$

We have three free variables: indices j and k, and completed production $B \to \rho$. Therefore, COMP1 has a total runtime of $\mathcal{O}\big(N^2|\mathcal{G}|\big)$.

The second rule, COMP2, attaches the resulting complete items to any incomplete items that predicted them:

$$\textsc{Comp2:}\ \frac{[i, j, A \to \mu \bullet B\,\nu] \quad [j, k, B \to \star\bullet]}{[i, k, A \to \mu\, B \bullet \nu]}$$

We have four free variables: indices i, j, and k, and dotted production $A \to \mu \bullet B\,\nu$. Therefore, COMP2 has a total runtime of $\mathcal{O}\big(N^3|\mathcal{G}|\big)$ and so

the two rules together have a runtime of $\mathcal{O}(N^3|\mathcal{G}|)$. This speed-up to the COMPLETE step is an instance of the fold transform (Tamaki and Sato, 1984).

5 Semiring-Weighted Parsing

We have so far presented Earley's algorithm and our improved deduction system in the unweighted case. This is equivalent to running a weighted algorithm in the boolean semiring. However, we are often interested in weighted recognition under an arbitrary closed semiring (Mohri, 1997). In weighted deduction, each axiom has a weight, and the weight of a proof tree is the product under $\otimes$ of the weights of its axioms (which in our case are input words and CFG productions), in the left-to-right order in which they are encountered in the proof tree. General algorithms for weighted deduction (Goodman, 1999; Eisner et al., 2005) can be used to find the total weight under $\oplus$ of all proofs of the goal item (as in (2)).

To solve the weighted CFG recognition problem using weighted deduction, we can continue to use the deduction systems in Table 1. Axioms of the form $A \to \rho$ should inherit their weight from the corresponding grammar production, i.e., $w(A \to \rho)$. All other axioms have weight ①. The weight of a proof tree of the goal item (according to the weighted deduction system) is now the weight of the corresponding derivation tree (according to the weighted CFG), so the total weight of all such proofs is $Z_{\mathbf{x}}$ as desired.

The deduction systems presented in §3 and §4 work for any semiring-weighted CFG. Unfortunately, the forward-chaining algorithm for weighted deduction (e.g., Eisner et al., 2005) may not terminate if the system permits *cyclic* proofs, where an item can participate in one of its own proofs.[9] In this case, the algorithm will merely approach the correct value of $Z_{\mathbf{x}}$ as it discovers deeper and deeper proofs of the goal item. Cyclicity in our system can arise from sets of unary productions such as $\{A \to B, B \to A\} \subseteq \mathcal{R}$, or equivalently, from $\{A \to B\,C, B \to A\} \subseteq \mathcal{R}$ where $C \overset{*}{\Rightarrow} \epsilon$ (which is possible if $\mathcal{R}$ contains $C \to \epsilon$ or other nullary productions). We take the approach of eliminating problematic unary and nullary productions from the weighted grammar without changing $Z_{\mathbf{x}}$ for any $\mathbf{x}$. We provide methods to do this in App. A and App. B respectively.[10] The runtime of the weighted deduction systems is then the same as in the unweighted case, where $|\mathcal{G}|$ now refers to the size of the modified grammar. The elimination of some productions can increase $|\mathcal{G}|$, but we explain how to limit this effect.

Once cyclic proofs are impossible, there exists a topologically sorted order of the items. Visiting the items in this order lets us compute $Z_{\mathbf{x}}$ with the same asymptotic complexity as unweighted parsing. Goodman (1999, Section 5) discusses execution strategies, including a generic method to dynamically discover a topologically sorted order (though we can specify one statically for our systems).

6 Earley's Algorithm Using an FSA

In this section, we present a generalization of EARLEY–FAST that can parse with any **weighted finite-state automaton** (**WFSA**) grammar $\mathcal{M}$ in $\mathcal{O}(N^3|\mathcal{M}|)$. Here $\mathcal{M}$ is a WFSA that encodes the CFG productions as follows. For any $\rho \in (\Sigma \cup \mathcal{N})^*$ and any $A \in \mathcal{N}$, for $\mathcal{M}$ to accept the string $\rho\,\widehat{A}$ with weight $w \in \mathbb{W}$ is tantamount to having the production $A \to \rho$ in the CFG with weight w.[11]

This presentation has two advantages over a standard CFG. First, $\mathcal{M}$ can be compiled from user-friendly specifications like NP $\to$ Det? Adj* N$^+$ PP*, which specifies infinitely many productions with unboundedly long right-hand-sides ρ (although $\mathcal{M}$ still only describes a context-free language). Second, productions with similar right-hand-sides may share partial paths in $\mathcal{M}$, which means that a single item can efficiently represent many dotted productions.

Our WFSA grammar is similar to a **recursive transition network** or **RTN** grammar (Woods, 1970). Adapting Earley's algorithm to RTNs was discussed by Jr. and Brown (1981), Kochut (1983), Leermakers (1989), and Perlin (1991). Klein and

[9] In general, even acyclic weighted deduction can fail to terminate, if the deduction system can derive infinitely many distinct items. But the number of derived items in our case is $\mathcal{O}(N^2|\mathcal{G}|)$, as noted earlier, and therefore finite.

[10] App. B may be a contribution of this paper, as we were unable to find a correct construction in the literature.

[11] Unless $\otimes$ is non-commutative (e.g., a derivation semiring). In contrast to footnote 4, we must now interpret (1) as multiplying the rule probabilities in a kind of *infix* order. Suppose a derivation tree for $A \overset{*}{\Rightarrow} \mathbf{x}$ uses a WFSA path at the root that accepts $BC\widehat{A}$ with weight w. Recursively let w_B and w_C be the weights of the child subderivations, rooted at B and C. Then the overall weight of the derivation of A will not be $w \otimes w_B \otimes w_C$ (prefix order), but rather $w_1 \otimes w_B \otimes w_2 \otimes w_C \otimes w_3$. Here we have factored the path weight w into $w_1 \otimes w_2 \otimes w_3$, which are respectively the weights of the subpath up through B, the subpath from there up through C, and the subpath from there to the end.

Domains	$i, j, k \in \{0, \ldots, N\} \quad A \in \mathcal{N} \quad a \in \Sigma \quad q, q' \in \mathcal{Q}$
Items	$[i, j, q] \quad [i, j, q?] \quad [i, j, a] \quad [i, j, A \to \bullet\star] \quad [i, j, A \to \star\bullet] \quad q \overset{a}{\rightsquigarrow} q' \quad q \overset{A}{\rightsquigarrow} q' \quad q \overset{A}{\rightsquigarrow} \star \quad q \overset{*\widehat{A}}{\rightsquigarrow} \star$
Axioms	$[0, 0, q], \forall q \in \mathcal{I} \qquad [j, j+1, x_k], \forall k \in \{1, \ldots, N\} \qquad [0, 0, S \to \bullet\star]$
Goals	$[0, N, S \to \star\bullet]$
Rules	PRED1: $\dfrac{[i, j, q] \quad q \overset{A}{\rightsquigarrow} \star}{[j, j, A \to \bullet\star]}$ COMP1: $\dfrac{[j, k, q] \quad q \overset{\widehat{A}}{\rightsquigarrow} q' \quad q' \in \mathcal{F}}{[j, k, A \to \star\bullet]}$
	PRED2: $\dfrac{q \in \mathcal{I}}{[j, j, q?]}$ COMP2: $\dfrac{[i, j, q] \quad q \overset{A}{\rightsquigarrow} q' \quad [j, k, A \to \star\bullet]}{[i, k, q'?]}$
	SCAN: $\dfrac{[i, j, q] \quad q \overset{a}{\rightsquigarrow} q' \quad [j, k, a]}{[i, k, q'?]}$ EPSILON: $\dfrac{[i, j, q] \quad q \overset{\epsilon}{\rightsquigarrow} q'}{[i, j, q'?]}$ FILTER: $\dfrac{[j, k, q?]}{[j, k, q]} \dfrac{[j, j, A \to \bullet\star]}{q \overset{*\widehat{A}}{\rightsquigarrow} \star}$

Table 2: EARLEY–FSA, a variant of EARLEY–FAST in which FSA states replace dotted productions.

Manning (2001b) used a weighted version for PTB parsing. None of them spelled out a deduction system, however.

Also, an RTN is a collection of productions of the form $A \to \mathcal{M}_A$, where for $\mathcal{M}_A$ to accept ρ is tantamount to having $A \to \rho$ in the CFG. Thus an RTN uses one FSA per non-terminal. Our innovation is to use one WFSA for the entire grammar, specifying the left-hand-side non-terminal as a final symbol. Thus, to allow productions $A \to \mu\,\nu$ and $B \to \mu\,\nu'$, our single WFSA can have paths $\mu\,\nu\,\widehat{A}$ and $\mu\,\nu'\,\widehat{B}$ that share the μ prefix. This allows our EARLEY–FSA to match the μ prefix only once, in a way that could eventually result in completing either an A or a B (or both). This concept of sharing a left context was first introduced by Jr. and Brown (1981).

We write $|\mathcal{M}|$ for the number of edges in $\mathcal{M}$. A traditional weighted CFG $\mathcal{G}$ can be easily encoded as a WFSA $\mathcal{M}$ with $|\mathcal{M}| = |\mathcal{G}|$, by creating a weighted path of length k and weight w for each CFG production of size k and weight w, terminating in a final state, and then merging the initial states of these paths into a single state that becomes the initial state of the resulting WFSA. The paths are otherwise disjoint. Importantly, this WFSA can then be determinized and minimized (Mohri, 2002) to potentially reduce the number of edges and thus speed up parsing (Klein and Manning, 2001b).

In general, however, the grammar can be specified by any WFSA $\mathcal{M}$—not necessarily deterministic. This could be compiled from weighted regular expressions, or an encoded Markov model trained on observed productions (Collins, 1999), or be obtained by merging states of another WFSA grammar (Stolcke and Omohundro, 1994) in order to smooth its weights and speed it up.

The WFSA has states $\mathcal{V}$ and weighted transitions (or edges) $\mathcal{E}$, over an alphabet $\mathcal{A}$ consisting of $\mathcal{N} \cup \Sigma$ together with hatted non-terminals like $\widehat{A}$. Its initial and final states are denoted by $\mathcal{I} \subseteq \mathcal{V}$ and $\mathcal{F} \subseteq \mathcal{V}$, respectively. We denote an edge of the WFSA by $(q \overset{a}{\rightsquigarrow} q') \in \mathcal{E}$ where $q, q' \in \mathcal{V}$ and $a \in \mathcal{A} \cup \{\epsilon\}$. This corresponds to an axiom with the same weight as the edge. $q \in \mathcal{I}$ corresponds to an axiom whose weight is the initial-state weight of q. The axiom $q \in \mathcal{F}$ actually means (more generally) that q has an ϵ-path to a final state; its weight is the total weight of all such paths.

For a state $q \in \mathcal{V}$ and symbol $a \in \mathcal{N}$, the precomputed side condition $q \overset{A}{\rightsquigarrow} \star$ is true iff there exists a state $q' \in \mathcal{V}$ such that the transition $q \overset{A}{\rightsquigarrow} q'$ exists in $\mathcal{E}$. Additionally, the precomputed side condition $q \overset{*\widehat{A}}{\rightsquigarrow} \star$ is true if there exists a path starting from q that will eventually read $\widehat{A}$.

The EARLEY–FSA deduction system is given in Table 2 and has a runtime of $\mathcal{O}\big(N^3|\mathcal{M}|\big)$. It is similar to EARLEY–FAST, where the dotted rules have been replaced by WFSA states. However, unlike a dotted rule, a state does not specify a PREDICTed left-hand-side non-terminal. As a result, when we "advance the dot" to a new state q, we build an item $[j, k, q?]$ that is annotated with a question mark. This mark represents the fact that although q is compatible with several left hand sides A (those for which $q \overset{*\widehat{A}}{\rightsquigarrow} \star$ is true), the left context $\mathbf{x}_{0:j}$ might not call for any of those non-terminals. If it does,

then the new FILTER rule will remove the question mark, allowing further progress.

As before, we must eliminate unary and nullary rules before parsing; App. C explains how to do this with a WFSA grammar. In addition, although Table 2 allows the WFSA to contain ϵ-transitions, App. C explains how to eliminate *ϵ-cycles* in the WFSA, which could prevent us from converging, for the usual reason that an item $[i, j, q]$ could participate in its own derivation. Afterwards, there is again a toposorted order in which the deduction engine can attempt to build items (Goodman, 1999).

As noted above, we can speed up EARLEY–FSA by reducing the size of the WFSA. Unfortunately, minimization of general FSAs is NP-hard. However, we can at least seek the minimal *deterministic* WFSA $\mathcal{M}'$ such that $|\mathcal{M}'| \leq |\mathcal{M}|$, at least in most semirings (Mohri, 2000; Eisner, 2003). The determinization (Aho et al., 1986) and minimization (Aho and Hopcroft, 1974; Revuz, 1992) algorithms for the boolean semiring are particularly well-known. Minimization merges states, which results in merging items, much as when EARLEY–FAST merged items that had different pre-dot symbols (Leermakers, 1992; Moore, 2000).

Another advantage of the WFSA presentation of Earley's is that it makes it simple to express a tighter bound on the runtime. Much of the grammar size $|\mathcal{G}|$ or $|\mathcal{M}|$ is due to terminal symbols that are not used at most positions of the input. Suppose the input is an ordinary sentence (one word at each position, unlike the lattice case in footnote 5), and suppose c is a constant such that no state q has more than c outgoing arcs labeled with the same terminal symbol $a \in \Sigma$. Then when SCAN tries to extend $[i, j, q]$, it considers at most c arcs. Thus, the $\mathcal{O}(|\mathcal{M}|)$ factor in our runtime (where $|\mathcal{M}| = |\mathcal{E}|$) can be replaced with $\mathcal{O}(|\mathcal{V}| \cdot c + |\mathcal{E}_{\mathcal{N}}|)$, where $\mathcal{E}_{\mathcal{N}} \subseteq \mathcal{E}$ is the set of edges that are *not* labeled with terminals.

7 Practical Runtime of Earley's

We empirically measure the runtimes of EARLEY, EARLEY–FAST, and EARLEY–FSA. We use the tropical semiring to find the highest-weighted derivation trees. We use two grammars that were extracted from the PTB: Markov-order-2 (M2) and Parent-annotated Markov-order-2 (PM2).[12] For each grammar, we ran our parsers [13](using the tropical semiring) on 100 randomly selected sentences of 5 to 40 words from the PTB test-set (mean 21.4, stdev 10.7), although we omitted sentences of length > 25 from the EARLEY graph as it was too slow (> 3 minutes per sentence). The full results are displayed in App. D. The graph shows that EARLEY–FAST is roughly $20\times$ faster at all sentence lengths. We obtain a further speed-up of $2.5\times$ by switching to EARLEY–FSA.

8 Conclusion

In this pedagogical paper, we have shown how the runtime of Earley's algorithm is reduced to $\mathcal{O}\big(N^3|\mathcal{G}|\big)$ from the naive $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$. We presented this dynamic programming algorithm as a deduction system, which splits prediction and completion into two steps each, in order to share work among related items. To further share work, we generalized Earley's algorithm to work with a grammar specified by a weighted FSA. We showed how to generalize these methods to semiring-weighted grammars by correctly transforming the grammars to eliminate cyclic derivations. We demonstrated that these speedups are effective in practice.

We remark on two useful extensions. Stolcke (1995)'s algorithm computes the total weight of all sentences with a given *prefix*; this can be arranged by augmenting our deduction system. We would also like to recover parse trees under the *original* grammar (before unary and nullary rules were eliminated). This can be done by constructing a derivation semiring such that $\mathrm{Z}_{\mathbf{x}}$ gives the best parse tree along with its weight, or alternatively a representation of the possibly infinite forest of all parse trees.

We intend this work to serve as a clean reference for those who wish to efficiently implement an Earley-style parser or develop related incremental parsing methods. For example, our deductive systems could be used as the starting point for neural models of incremental processing (in which an item's vector-space representation is computed from its derivations, along with its weight), or for extensions to more powerful grammar formalisms.

[12] Available at https://code.google.com/archive/p/bubs-parser/. M2 contains 13,893 non-lexicon rules and 52,009 lexicon rules. PM2 contains 25,919 non-lexicon rules and 52,009 lexicon rules. The downloaded grammars did not have nullary rules or unary chains.

[13] We will release our Cython implementation upon publication. A fast implementation of Earley's algorithm is reported by Polat et al. (2016) but does not appear to be public.

References

Alfred V. Aho and John E. Hopcroft. 1974. *The Design and Analysis of Computer Algorithms*. Pearson Education.

Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. 1986. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science / World student series edition. Addison-Wesley.

John Aycock and R. Nigel Horspool. 2002. Practical Earley parsing. *The Computer Journal*, 45(6):620–630.

François Barthélemy. 1993. *Outils pour lÁnalyse Syntaxique Contextuelle*. Ph.D. thesis, University of Orléans.

Eric V. de la Clergerie. 1993. *Automates a piles et programmation dynamique DyAlog: une application a la programmation en logique*. Ph.D. thesis, University Paris VII.

Michael J. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Aaron Dunlop, Nathan Bodenstab, and Brian Roark. 2010. Reducing the grammar constant: an analysis of CYK parsing efficiency. Technical report, Oregon Health & Science University.

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

Jason Eisner. 2003. Simpler and more general minimization for weighted finite-state automata. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 64–71.

Jason Eisner and John Blatz. 2007. Program transformations for optimization of parsing algorithms and other weighted logic programs. In *Proceedings of FG 2006: The 11th Conference on Formal Grammar*, pages 45–85. CSLI Publications.

Jason Eisner, Eric Goldlust, and Noah A. Smith. 2005. Compiling comp ling: Weighted dynamic programming and the Dyna language. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 281–290, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Kilian Gebhardt. 2015. *Training of hybrid grammars for the generation of discontinuous phrase structures and non-projective dependency structures*. Ph.D. thesis, Technische Universität Dresden.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–606.

Susan L. Graham, Michael A. Harrison, and Walter L. Ruzzo. 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–462.

John Hale. 2001. A probabilistic Earley parser as a psycholinguistic model. In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2007. *Introduction to Automata Theory, Language, and Computation*, 3 edition. Pearson international edition. Addison-Wesley.

Mark Johnson. 2000. Inside-outside (computer program).

Paul Walton Purdom Jr. and Cynthia A. Brown. 1981. Parsing extended *LR(k)* grammars. *Acta Informatica*, 15:115–127.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing*, 2 edition. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Dan Klein and Christopher D. Manning. 2001a. Parsing and hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies*, pages 123–134, Beijing, China.

Dan Klein and Christopher D. Manning. 2001b. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn Treebank. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 338–345, Toulouse, France. Association for Computational Linguistics.

Krzysztof Kochut. 1983. Towards the elastic ATN implementation. In *The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks*, pages 175–214. Springer.

René Leermakers. 1989. How to cover a grammar. In *27th Annual Meeting of the Association for Computational Linguistics*, pages 135–142, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

René Leermakers. 1992. A recursive ascent Earley parser. *Information Processing Letters*, 41(2):87–91.

Daniel J. Lehmann. 1977. Algebraic structures for transitive closure. *Theoretical Computer Science*, 4(1):59–76.

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

David A. McAllester. 2002. On the complexity analysis of static analyses. *Journal of the ACM*, 49(4):512–537.

Hongyuan Mei, Guanghui Qin, Minjie Xu, and Jason Eisner. 2020. Neural Datalog through time: Informed temporal modeling via logical specification. In *Proceedings of the 37th International Conference on Machine Learning*.

Guido Minnen. 1996. Magic for filter optimization in dynamic bottom-up processing. In *Proceedings of the 34th conference on Association for Computational Linguistics*, pages 247–254.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Mehryar Mohri. 2000. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 324:177–201.

Mehryar Mohri. 2002. Generic ε-removal and input ε-normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143.

Robert C. Moore. 2000. Improved left-corner chart parsing for large context-free grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies*, pages 171–182, Trento, Italy. Association for Computational Linguistics.

Mark J. Nederhof. 1994. *Linguistic Parsing and Program Transformations*. Ph.D. thesis, University of Nijmegen.

Mark-Jan Nederhof and Giorgio Satta. 2008. Computing partition functions of PCFGs. *Research on Language and Computation*, 6(2):139–162.

Fernando C. N. Pereira and Stuart M. Shieber. 1987. *Prolog and Natural-Language Analysis*. Number 10 in CSLI Lecture Notes. Center for the Study of Language and Information.

Fernando C. N. Pereira and David H. D. Warren. 1983. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Mark Perlin. 1991. LR recursive transition networks for Earley and Tomita parsing. In *29th Annual Meeting of the Association for Computational Linguistics*, pages 98–105, Berkeley, California, USA. Association for Computational Linguistics.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia. Association for Computational Linguistics.

Benjamin C. Pierce. 2002. *Types and Programming Languages*. MIT Press.

Sinan Polat, Merve Selcuk-Simsek, and Ilyas Cicekli. 2016. A modified earley parser for huge natural language grammars. *Res. Comput. Sci.*, 117:23–35.

Raghu Ramakrishnan. 1991. Magic templates: A spellbinding approach to logic programs. *Journal of Logic Programming*, 11(3-4):189–216.

Dominique Revuz. 1992. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1):181–189.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1&2):3–36.

Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Punta Cana.

Klaas Sikkel. 1993. *Parsing Schemata*. Ph.D. thesis, University of Twente, Enschede, Netherlands.

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

Andreas Stolcke and Stephen M. Omohundro. 1994. Best-first model merging for hidden Markov model induction. Technical Report ICSI TR-94-003, ICSI, Berkeley, CA.

Hisao Tamaki and Taisuke Sato. 1984. Unfold/fold transformation of logic programs. In *Proceedings of the Second International Logic Programming Conference, Uppsala University, Uppsala, Sweden, July 2-6, 1984*, pages 127–138.

Robert Endre Tarjan. 1981a. Fast algorithms for solving path problems. *Journal of the ACM*, 28(3):594–614.

Robert Endre Tarjan. 1981b. A unified approach to path problems. *Journal of the ACM*, 28(3):577–593.

William A. Woods. 1970. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606.

A Eliminating Unary Cycles

Stolcke (1995, Section 4.5) addresses the problem of unary production cycles by modifying the deduction rules.[14] He assumes use of the real semiring, where $\oplus = +$ and $\otimes = \times$. Here, inverting a single $|\mathcal{N}| \times |\mathcal{N}|$ matrix suffices to compute the total weight of all rewrite sequences $A \overset{*}{\Rightarrow} B$, known as **unary chains**, for each ordered pair $A, B \in \mathcal{N}^2$. His modified rules then ignore the original unary productions and refer to these weights instead.

We take a similar approach, but instead describe it as a transformation of the weighted grammar, leaving the deduction system unchanged. We also generalize to other closed semirings. Finally, we do not collapse all unary chains as Stolcke (1995) does, but only those subchains that can appear on cycles. This prevents the grammar size from blowing up more than necessary (recall that the parser's runtime is proportional to grammar size). For example, if the unary productions are $A_i \to A_{i+1}$ for all $1 \leq i < K$, then there is no cycle and our transformation leaves these $K - 1$ productions unchanged, rather than replacing them with $K(K-1)/2$ new unary productions that correspond to the possible chains $A_i \overset{*}{\Rightarrow} A_j$ for $1 \leq i \leq j \leq K$.

Given a weighted CFG $\mathcal{G} = \langle \mathcal{N}, \Sigma, \mathcal{R}, S, w \rangle$, consider the weighted graph whose vertices are $\mathcal{N}$ and whose weighted edges $A \to B$ are given by the unary productions $A \to B$. (This graph may include self-loops such as $A \to A$.) Its strongly connected components (SCCs) can be found in linear time and thus in $\mathcal{O}(|\mathcal{G}|)$ time. For any A and B in the same SCC, $w(A \overset{*}{\Rightarrow} B) \in \mathbb{W}$ denotes the total weight of all rewrite sequences of the form $A \overset{*}{\Rightarrow} B$ (including the 0-length sequence with weight ①, if $A = B$). For an SCC of size K, there are K^2 such weights and they can be found in total time $\mathcal{O}(K^3)$ by the Kleene–Floyd–Warshall algorithm (Lehmann, 1977; Tarjan, 1981b,a). In the real semiring, this algorithm corresponds to using Gauss-Jordan elimination to invert $I - E$, where E is the weighted adjacency matrix of the SCC (rather than of the whole graph as in Stolcke (1995)). In the general case, it computes the infinite matrix sum $I \oplus E \oplus (E \otimes E) \oplus \cdots$ in closed form, with the help of the * operator of the closed semiring.

We now construct a new grammar $\mathcal{G}' = \langle \mathcal{N}', \Sigma, \mathcal{R}', \overline{S}, w' \rangle$ that has no unary cycles, as follows. For each $A \in \mathcal{N}$, our $\mathcal{N}'$ contains two non-terminals, $\overline{A}$ and $\underline{A}$. For each ordered pair of non-terminals $A, B \in \mathcal{N}^2$ that fall in the same SCC, $\mathcal{R}'$ contains a production $\overline{A} \to \underline{B}$ with $w'\left(\overline{A} \to \underline{B}\right) = w\left(A \overset{*}{\Rightarrow} B\right)$. For every rule $A \to \rho$ in $\mathcal{R}$ that is *not* of the form $A \to B$ where A and B fall in the same SCC, $\mathcal{R}'$ also contains a production $\underline{A} \to \overline{\rho}$ with $w'(\underline{A} \to \overline{\rho}) = w(A \to \rho)$, where $\overline{\rho}$ is a version of ρ in which each non-terminal B has been replaced by $\overline{B}$. Finally, as a constant-factor optimization, $\overline{A}$ and $\underline{A}$ may be merged back together if A formed a trivial SCC with no self-loop: that is, remove the weight-① production $\overline{A} \to \underline{A}$ from $\mathcal{R}'$ and replace all copies of $\overline{A}$ and $\underline{A}$ with A throughout $\mathcal{G}'$.

[14] Johnson (2000) provides an implementation of CKY (and the inside-outside algorithm) that allows unary productions and handles unary cycles in a similar way.

B Eliminating Nullary Productions

We eliminate nullary productions from the weighted grammar in order to avoid cycles. This must be done *before* eliminating unary cycles (App. A), since eliminating nullary productions can create new unary productions. Hopcroft et al. (2007, Chapter 7.1.3) explain how to do this in the unweighted case. Stolcke (1995, Section 4.7.4) sketches a generalization to the probability semiring, but it also uses the non-semiring operations of division and subtraction (and is not clearly correct). We therefore give an explicit general construction.

While we provide a method that handles nullary productions by modifying the grammar, it is also possible to instead modify the algorithm to allow advancing the dot over **nullable** non-terminals, i.e., non-terminals A such that the grammar allows $A \overset{*}{\Rightarrow} \epsilon$ (Aycock and Horspool, 2002).

Our first step, like Stolcke's, is to compute the "null weight" $e_A \overset{\text{def}}{=} w\left(A \overset{*}{\Rightarrow} \epsilon\right)$ for each $A \in \mathcal{N}$. Although a closed semiring does not provide an operator for this summation, these values are a solution to the system of $|\mathcal{N}|$ polynomial equations[15]

$$e_A = \bigoplus_{(A \to B_1 \cdots B_n) \in \mathcal{R}} w(A \to B_1 \cdots B_n) \otimes \bigotimes_{i=1}^{n} e_{B_i} \qquad (3)$$

A solution should exist for the sum in (2) to be well-defined in the first place. If so, a solution can normally be found in practice by initializing all $e_A = ⓪$ and then iteratively recomputing them, using the equations above, until numerical convergence. Nederhof and Satta (2008) review some other methods for the case of the real semiring.

[15] If $(A \to \epsilon) \in \mathcal{R}$, it will be covered by the case $n = 0$.

We now modify the grammar as follows. We adopt the convention that for a production $A \to \rho$ that is not yet in $\mathcal{R}$, we consider its weight to be $w(A \to \rho) = \text{⓪}$, and increasing this weight by any non-⓪ amount adds it to $\mathcal{R}$. For each non-terminal B such that $e_B \neq \text{⓪}$, let us assume the existence of an auxiliary non-terminal $B_{\neq\epsilon} \notin \mathcal{N}$ such that $B_{\neq\epsilon} \overset{*}{\not\Rightarrow} \epsilon$ but $\forall \mathbf{x} \neq \epsilon$, $w\Big(B_{\neq\epsilon} \overset{*}{\Rightarrow} \mathbf{x}\Big) = w\Big(B \overset{*}{\Rightarrow} \mathbf{x}\Big)$. We iterate this step: as long as we can find a production $A \to \mu\, B\, \nu$ in $\mathcal{R}$ such that $e_B \in \mathcal{N}$, we modify it to the more restricted version $A \to \mu\, B_{\neq\epsilon}\, \nu$ (keeping its weight), but to preserve the possibility that $B \overset{*}{\Rightarrow} \epsilon$, we also increase the weight of the shortened production $A \to \mu\, \nu$ by $w(A \to \mu\, B\, \nu) \otimes e_B$.

A production $A \to \rho$ where ρ includes k non-terminals will be gradually split up by the above procedure into 2^k productions, in which each non-terminal B has been either specialized to $B_{\neq\epsilon}$ or removed. In particular, we can see from (3) that $w(A \to \epsilon) = e_A$. So far we have preserved all weights $w\Big(A \overset{*}{\Rightarrow} \mathbf{x}\Big)$, provided that the auxiliary non-terminals behave as assumed. But for each A we now remove $A \to \epsilon$ from $\mathcal{R}$, and since A can no longer rewrite as ϵ, we rename all other rules $A \to \rho$ to $A_{\neq\epsilon} \to \rho$. This closes the loop by defining the auxiliary non-terminals as desired.

Finally, since S is the start symbol, we add back $S \to \epsilon$ (with weight e_S) as well as adding the new rule $S \to S_{\neq\epsilon}$ (with weight ①). Thus (as in Chomsky Normal Form), the only nullary rule is now $S \to \epsilon$, which may be needed to generate the 0-length sentence. We now have a new grammar with non-terminals $\mathcal{N}' = \{S\} \cup \{B_{\neq\epsilon} : B \in \mathcal{N}\}$. To simplify the names, we can rename the start symbol S to $\widehat{S}$ and then drop the $_{\neq\epsilon}$ subscripts.[16]

C Handling Nullary and Unary Productions in an FSA

We can handle nullary productions by directly adapting the construction of App. B to the WFSA case. Indeed, the WFSA version is simpler to express. For each arc $q \overset{B}{\rightsquigarrow} q'$ such that $B \in \mathcal{N}$ and $e_B \neq \text{⓪}$, we replace the B label of that arc with $B_{\neq\epsilon}$ (preserving the arc's weight), and add a new arc $q \overset{\epsilon}{\rightsquigarrow} q'$ of weight e_B. We then define a new WFSA $\mathcal{M}' = (\mathcal{M} \cap \neg\mathcal{M}_{\text{bad}}) \cup \mathcal{M}_{\text{good}}$, where $\mathcal{M}_{\text{bad}}$ is an unweighted FSA that accepts exactly those strings of the form $\widehat{A}$ (i.e., nullary productions), $\neg$ takes the unweighted complement, and $\mathcal{M}_{\text{good}}$ is a WFSA that accepts exactly strings of the form $\widehat{S}$ (with weight e_S) and $S_{\neq\epsilon}\widehat{S}$ (with weight ①). As this construction introduces new ϵ arcs, it should precede the elimination of ϵ-cycles.

Notice that in the example of App. B where a production $A \to \rho$ was replaced with up to $2^k - 1$ variants, the WFSA construction efficiently shares structure among these variants. It adds at most k edges at the first step and at most doubles the total number of states through intersection with $\neg\mathcal{M}_{\text{bad}}$.

Similarly, we can handle unary productions by directly adapting the construction of App. A to the WFSA case. We first extract all weighted unary rules by intersecting $\mathcal{M}$ with the unweighted language $\{B\widehat{A} : A, B \in \mathcal{N}\}$ (and determinizing the result so as to combine duplicate rules). Exactly as in App. A, we construct the unary rule graph and compute its SCCs along with weights $w\Big(A \overset{*}{\Rightarrow} B\Big)$ for all A, B in the same SCC. We modify the WFSA by underlining all hatted non-terminals $\widehat{A}$ and overlining all non-terminals B. Finally, we define our new WFSA grammar $(\mathcal{M} \cap \neg\mathcal{M}_{\text{bad}}) \cup \mathcal{M}_{\text{good}}$. Here $\mathcal{M}_{\text{bad}}$ is an unweighted FSA that accepts exactly those strings of the form $\overline{B}\underline{\widehat{A}}$ and $\mathcal{M}_{\text{good}}$ is a WFSA that accepts exactly strings of the form $\underline{B}\overline{\widehat{A}}$ such that A, B are in the same SCC, with weight $w\Big(A \overset{*}{\Rightarrow} B\Big)$.

Following each construction, non-terminal names can again be simplified as in Apps. A and B.

Finally, §6 mentioned that we must eliminate ϵ-cycles from the FSA. The algorithm for doing so (Mohri, 2002) is fundamentally the same as our method for eliminating unary rule cycles from a CFG (App. A), but now it operates on the graph whose edges are ϵ-transitions of the FSA, rather than the graph whose edges are unary rules of the CFG.

[16] We can also iteratively remove any useless non-terminals (footnote 6), which correspond to non-terminals that *only* rewrote as ϵ in the original grammar.

D Runtime Experiment Results

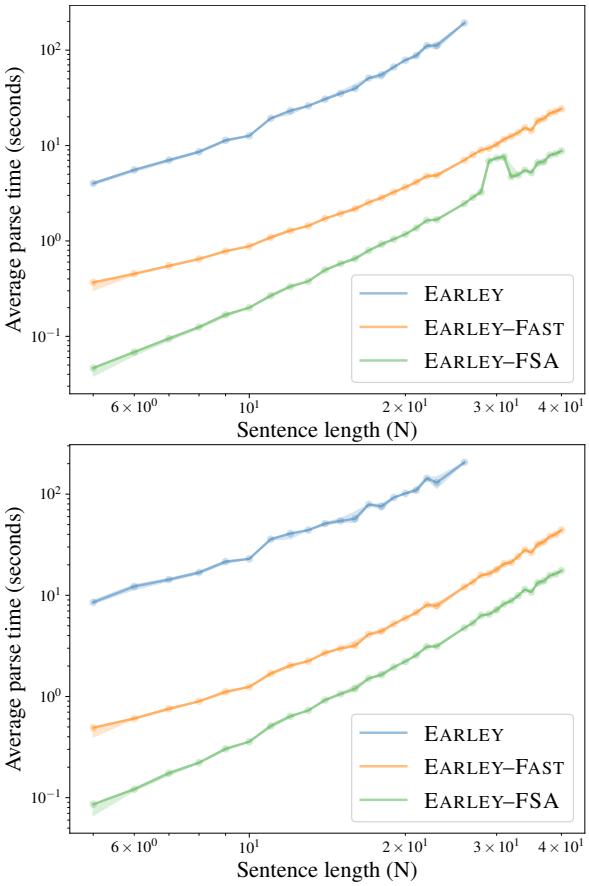

Figure 1: Average parse time per sentence for 100 randomly selected sentences of 5–40 words on the M2 grammar (left) and PM2 grammar (right). As all these algorithms are worst-case cubic in N, each curve on these log-log plots is bounded above by a line of slope 3, but the lower lines have better grammar constants. The experiment was conducted using a Cython implementation on an Intel(R) Core(TM) i7-7500U processor with 16GB RAM.