TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Theoretical Computer Science, Chair of Foundations of Programming

LEARNING PRUNING POLICIES FOR LINEAR CONTEXT-FREE REWRITING SYSTEMS

INF-PM-FPG

Andy Püschel

July 20, 2018

1 INTRODUCTION

Syntactiv parsing is the problem of determining the syntactic structure of a given sentence. If a sentence can be parsed, the algorithm returns the syntactic structure of the given sentence e.g. in form of derivation trees or even more information about the whole parsing process e.g. in form of parse charts. One of the parsing algorithms is the weighted deductive parsing algorithm, introduced by [Ned03], which takes as input a word and a grammar. The weighted deductive parsing algorithm uses different deduction rules to create constituents from single words in the sentence firstly, and secondly from already created constituents. Since the runtime varies in the size of the grammar (size of the set of rules), particular grammar constants and the sentence length, pruning policies will be a not indispensable feat. A pruning policy decides whether a certain constituent for a part of the sentence is likely to be a good constituent or not. The application of a given pruning policy can reduce the runtime of a parser drastically. Nevertheless, a given pruning policy is not always optimal for different sentences generated by the same grammar.

In this paper, we will introduce an approach to train such a pruning policy for PLCFRS. The main objective of this paper will be to introduce a method similar to [VE17] but for PLCFRS instead of PCFG.

After considering the prelimniaries in Section 2, we extend the weighted deductive parsing algorithm for PLCFRS by a pruning policy Section 3. In Section 4, the Locally Optimal Learning to Search training algorithm (LOLS), introduced by [Cha+15], is presented for pruning policies. Furthermore a metric will be provided which asks, how to evaluate (how to reward) a pruning policy. To improve the runtime of the training algorithm, we present the change propagation

algorithm, introduced by [VE17]. In Section 5, a description for our experimental setup in detail will be given and the results will be presented in Section 6. At last, we conclude our work in Section 7.

2 PRELIMINARIES

To ensure a common understanding of terms, some fundamental definitions and notations will be introduced in this section. For the formal language theory we use the definitions by [chomsky56] and [Iari90]. Firstly, we denote by $\mathbb{N}$ the set of natural numbers. Now, for each $k \in \mathbb{N}$, we define $[k] = \{1, \ldots, k\}$.

Definition 2.1 (Relation Product)**.** Let A, B, C be sets and $R \subseteq A \times B$, $S \subseteq B \times C$ be relations. We define the *relation product* as

$$R; S = \{(a, c) \in A \times C \mid \exists b \in B : (a, b) \in R \text{ and } (b, c) \in S\}.$$

Definition 2.2 (Alphabet)**.** An *alphabet* is a finite nonempty set.

Definition 2.3 (Sentence Length)**.** Let Σ be an alphabet and $w = w_1 \ldots w_n \in \Sigma^*$. The *length of* w is defined as $|w| = n$.

Definition 2.4. Let $l \in \mathbb{N}$, $X_1, \ldots, X_l$ be nonempty sets, $x = (x_1, \ldots, x_l) \in X_1 \times \ldots \times X_l$ be a tuple and $i \in [l]$. The *i-th element of* x is denoted by $x|_i = x_i$.

Definition 2.5 (Set of Trees)**.** Let Σ be an alphabet and H be a set disjoint from Σ. The *set of unranked trees over* Σ *indexed by* H, denoted by $T_\Sigma(H)$, is the smallest set U such that:

(i) $H \subseteq U$

(ii) $\forall k \in \mathbb{N}, \sigma \in \Sigma, \xi_1, \ldots, \xi_k \in U : \sigma(\xi_1, \ldots, \xi_k) \in U$

We denote $T_\Sigma(\emptyset)$ as T_Σ.

Definition 2.6 (Sorted Set)**.** Let S be a countable set (called sorts). An *S-sorted set* is a tuple $(A, sort)$ where

- A is a set and
- $sort : A \to S$.

$(A, sort)$ is called *finite* if for each $s \in S : A_s$ is finite, it is called *nonempty* if $\bigcup_{s \in S} A_s$ is nonempty and we write $a \in (A, sort)$ if $a \in A$. We denote $(A, sort)$ as A.

Definition 2.7 (LCFRS [VWJ87], [Kal12])**.** A *linear context-free rewriting system* is a tuple $G = (N, \Sigma, \Xi, P, S)$ where

- N is a finite nonempty $\mathbb{N}$-sorted set (nonterminal symbols),
- Σ is a finite set (terminal symbols) (with $\forall l \in \mathbb{N} : \Sigma \cap N_l = \emptyset$),
- Ξ is a finite nontempty set (variable symbols) (with $\Xi \cap \Sigma = \emptyset$ and $\forall l \in \mathbb{N} : \Xi \cap N_l = \emptyset$),
- P is a *set of production rules* of the form $\rho = \phi \to \psi$ where
  - $\phi = A(\alpha_1, \ldots, \alpha_l)$ (called left-hand side of ρ)
    where $l \in \mathbb{N}$, $A \in N_l$, $\alpha_1, \ldots, \alpha_l \in (\Sigma \cup \Xi)^*$ and
  - $\psi = B_1(X_1^{(1)}, \ldots, X_{l_1}^{(1)}) \ldots B_m(X_1^{(m)}, \ldots, X_{l_m}^{(m)})$ (called right-hand side of ρ)
    where $m \in \mathbb{N}$, $B_1 \in N_{l_1}, \ldots, B_m \in N_{l_m}$, $X_j^{(i)} \in \Xi$ for $1 \le i \le m, 1 \le j \le l_i$

and for every $X \in \Xi$ occurring in ρ we require that X occurs exactly once in the left-hand side of ρ and exactly once in the right-hand side of ρ, and

- $S \in N_1$ (initial nonterminal symbol).

A nonterminal symbol $A \in N$ has *fanout* l, denoted by $fanout(A)$, if $A \in N_l$. The fanout of a LCFRS G is defined as $fanout(G) = \max \bigcup_{A \in N} fanout(A)$.
A LCFRS is called *ordered* if for every $\rho = \phi \to \psi \in P$ and for each $X_1, X_2 \in V$ occurring in ϕ and $\psi : X_1$ preceeds X_2 in ϕ iff. X_1 preceeds X_2 in ψ.
A LCFRS is called *ε-free* if for every $\rho = \phi \to \psi \in P$ there are no ε-components in ϕ.
A LCFRS is called *simple* if for every $\rho = \phi \to \psi \in P$:

1. $\phi = A(\sigma)$ where $A \in N_1$, $\sigma \in \Sigma$ and $\psi = \varepsilon$ or
2. $\psi \neq \varepsilon$ and there is no $\sigma \in \Sigma$ that occurs in ϕ or in ψ.

Definition 2.8 (yield [KM10])**.** Let G be a LCFRS where $G = (N, \Sigma, \Xi, P, S)$ and $A \in N$. We define $yield : N \to \bigcup_{l \in \mathbb{N}} (\Sigma^*)^l$ to be the smallest family $(Y_A \mid A \in N)$, such that:

(i) For every rule $\rho = A(\alpha_1, \ldots, \alpha_l) \to \varepsilon \in P : (\alpha_1, \ldots, \alpha_l) \in Y_A$.

(ii) For every rule $\rho = A(\alpha_1, \ldots, \alpha_l) \to B_1(X_1^{(1)}, \ldots, X_{l_1}^{(1)}) \ldots B_m(X_1^{(m)}, \ldots, X_{l_m}^{(m)}) \in P$ and for every $(\beta_{i,1}, \ldots, \beta_{i,l_i}) \in Y_{B_i}$ for $i \in [m]$
$(f(\alpha_1), \ldots, f(\alpha_l)) \in Y_A$ where f is defined as $f : (\Sigma \cup \Xi)^* \to \Sigma^*$ with:

$$f(x) = \begin{cases} \varepsilon & \text{if } x = \varepsilon \\ x & \text{if } x \in \Sigma \\ \beta_{i,j} & \text{if } x \in \Xi, \text{ for } i \in [m] \text{ and } j \in [l_i] : x = X_j^{(i)} \\ f(y)f(z) & \text{if } x = yz \text{ and } y, z \in \Sigma^+. \end{cases}$$

Definition 2.9 (Language [KM10])**.** Let G be a LCFRS with $G = (N, \Sigma, \Xi, P, S)$. The *language of* G, denoted by $L(G)$, is defined as:

$$L(G) = \{w \mid (w) \in yield(S)\}$$

Definition 2.10 (Substitution)**.** Let X be a finite set, Σ be an alphabet and $x_i \in X$, $\gamma_i \in \Sigma^*$, $\alpha \in (X \cup \Sigma)^*$. We denote the *substitution of variables x_i by γ_i in α* by $\alpha[x_i/\gamma_i]$.

Definition 2.11 (Derivation relation)**.** Let $G = (N, \Sigma, \Xi, P, S)$ be a LCFRS,
$\rho = A(\alpha_1, \ldots, \alpha_l) \to B_1(X_1^{(1)}, \ldots, X_{l_1}^{(1)}) \ldots B_m(X_1^{(m)}, \ldots, X_{l_m}^{(m)}) \in P$ for $m \geq 0$ and $K = \Sigma \cup N \cup \hat{\{}\ (\ \hat{,}\ )\ \hat{,}\ \hat{\}}$. The *derivation relation*, denoted by $\overset{\rho}{\Rightarrow}_G$, is the binary relation $\overset{\rho}{\Rightarrow}_G \subseteq (K^* \times K^*)$:

$$\begin{aligned} \overset{\rho}{\Rightarrow}_G = \{ & A(\gamma_1^{(0)}, \ldots, \gamma_{l_0}^{(0)})k,\ B_1(\gamma_1^{(1)}, \ldots, \gamma_{l_1}^{(1)}) \ldots B_m(\gamma_1^{(m)}, \ldots, \gamma_{l_m}^{(m)})k \\ & \mid k \in K^*, \forall 0 \leq i \leq m, 1 \leq j \leq l_i, \gamma_j^{(i)} \in \Sigma^* : \gamma_j^{(0)} = \alpha_j[X_j^{(i)}/\gamma_k^{(i)}]\} \end{aligned}$$

Definition 2.12 (Derivation)**.** Let $G = (N, \Sigma, \Xi, P, S)$ be a LCFRS, $n \in \mathbb{N}$ and $\rho_1, \ldots, \rho_n \in P$. A *derivation* is a sequence of rules $d = \rho_1 \ldots \rho_n \in P^*$ such that $\overset{\rho_1}{\Rightarrow}_G ; \ldots ; \overset{\rho_n}{\Rightarrow}_G \neq \emptyset$. We denote $\overset{\rho_1}{\Rightarrow}_G ; \ldots ; \overset{\rho_n}{\Rightarrow}_G$ by $\overset{d}{\Rightarrow}_G$.

Definition 2.13 (Set of derivations)**.** Let $G = (N, \Sigma, \Xi, P, S)$ be a LCFRS, $A \in N$, $l = sort(A)$ and $(w_1, \ldots, w_l) \in (\Sigma^*)^l$. The *set of derivations from A to* $(w_1, \ldots, w_l)$, denoted by $D_G(A, w_1, \ldots, w_l)$, is defined as

$$D_G(A, w_1, \ldots w_l) = \{d \mid A(w_1, \ldots, w_l) \overset{d}{\Rightarrow}_G \varepsilon\}.$$

We use the shorthand D_G for the *set of all derivations from S*, i.e.:

$$D_G = \bigcup_{w \in \Sigma^*} D_G(S, w).$$

Definition 2.14 (dtree)**.** Let $G = (N, \Sigma, \Xi, P, S)$ be a LCFRS, $w \in \Sigma^*$ be a sentence and $d = \rho_1 \dots \rho_n \in D_G(S, w)$ be a derivation. We define $dtree : D_G(S, w) \to T_N(\Sigma)$ by

$$dtree(d) = t(d)$$

where $t : P^* \to T_N(\Sigma) \times P^*$ is a partial function, such that

$$t(\rho_1 \dots \rho_n) = \begin{cases} undefined & \text{if } n = 0 \\ (A(\alpha), \rho_2, \dots \rho_n) & \text{if } \rho_1 = A(\alpha) \to \varepsilon \\ (A(\xi_1, \dots, \xi_m), d_m) & \text{if } \rho_1 = A(\alpha_1, \dots, \alpha_m) \to \psi, \text{ where } (\xi_1, d_1) = t(\rho_2 \dots \rho_n), \\ & (\xi_2, d2) = t(d1), \dots, (\xi_m, d_m) = t(d_{m-1}). \end{cases}$$

Definition 2.15 (PLCFRS [KSK06])**.** Let G be a *LCFRS*. A *probabilistic linear context-free rewriting system* (*PLCFRS*) is a tuple (G, p) where

- $G = (N, \Sigma, \Xi, P, S)$ is a LCFRS with $N = (N', sort)$ and
- $p : P \to [0, 1]$ is a *probability distribution*.

We say (G, p) is *proper* if for every $l \in \mathbb{N}, A \in N_l$:

$$\sum_{(A(\alpha_1,\dots,\alpha_l)\to\psi)\in P} p(A(\alpha_1, \dots, \alpha_l) \to \psi) = 1.$$

We say (G, p) is *consistent* if

$$\sum_{d \in D_{(G,p)}} p(d) = 1 \qquad \text{where for each } d = \rho_1 \dots \rho_n : \qquad p(d) = \prod_{i=1}^{n} p(\rho_i).$$

Definition 2.16 (Spans)**.** We define the mapping $span : \mathcal{P}(\mathbb{N}) \setminus \{\emptyset\} \to (\mathcal{P}(\mathbb{N}) \setminus \{\emptyset\})^*$ with

$$span(U) = U_1 \dots U_l$$

where $l \in \mathbb{N}$ and $U_1, \dots, U_l \subseteq U$, such that

- for all $i \in [l] : U_i = \{j, j + 1, \dots, k\}$ for $j, k \in \mathbb{N}, j \leq k$ and $(k + 1) \notin U$,
- $U = \bigcup_{i=1}^{l} U_i$ and
- $\forall x, y \in [l] : x < y$ iff. $min(U_x) < min(U_y)$.

Definition 2.17 (Derivation Graph)**.** Let (G, p) be a simple ordered ε-free PLCFRS where $G = (N, \Sigma, \Xi, P, S)$ and $w \in L(G)$. A *derivation graph over* (G, p) *and* w is a tuple $H = (V, E, b, \omega)$ where

- $V = \mathcal{P}([|w|]) \setminus \{\emptyset\}$ (vertices),
- E is a finte set of hyperedges of the form $\eta\langle\wp\rangle \to \theta$ where $\eta \in V \times N$, $\wp \in [0, 1]$ and $\theta \in (V \times N)^*$.
- $b : E \to \{0, 1\}$ is a mapping and
- $\omega : V \times N \to E \times [0, 1]$ is a partial mapping (witness function).

We define $\mathcal{I}_{(G,p)}(w)$ as the set of all derivation graphs over (G, p) and w. We call a $H \in \mathcal{I}_{(G,p)}(w)$ pruned derivation graph.
For every rule $e = \eta\langle\wp\rangle \to \theta \in E$ we call $\eta_e = \eta$ the head of e, $\wp_e = \wp$ the probability of e and $\theta_e = \theta$ the tail of e.
Let $v \in V$ be a vertex and $A \in N$. We define $In(v, A)$ as the *set of incoming hyperedges of v and A*, i.e.:

$$In(v, A) = \{e \in E \mid \eta_e = (v, A)\}$$

and $Out(v, A)$ as the *set of outgoing hyperedges of v and A*, i.e.:

$$Out(v, A) = \{e \in E \mid \text{ where } (v, A) \text{ occurs in } \theta_e\}.$$

The set E is called *consistent* if, for each $e \in E$, we have:

- $e = (\{i\}, A)\langle p(\rho)\rangle \to \varepsilon$ for some production $\rho = A(w_i) \to \varepsilon \in P$, or
- $(\bigcup_{i=1}^{m} u_i, A)\langle p(\rho)\rangle \to (u_1, B_1) \dots (u_m, B_m)$ for some production $\rho = A(\alpha_1, \dots, \alpha_l) \to B_1(X_1^{(1)}, \dots, X_{l_1}^{(1)}) \dots B_m(X_1^{(m)}, \dots, B_m(X_{l_m}^{(m)}) \in P$ and for every $(u_1, B_1)\langle\wp_1\rangle \to \theta_1, \dots, (u_m, B_m)\langle\wp_m\rangle \to \theta_m \in E$ where $\alpha_1, \dots, \alpha_l \in \Sigma^*$, $m \in \mathbb{N}$, $j \in [m]$, $k \in [l_j]$, $\wp_j \in [0, 1]$: if $\forall q \in [l] : spans(v)|_q = \bigcup_{X_k^{(j)} \in \alpha_q} spans(u_j)|_k$ and $\forall j' \in [m], j' \neq j : u_j$ and $u_{j'}$ are pairwise disjoint.

The function ω is called *consistent* if

$$\omega(v, A) = (f(v, A), g(v, A)) \qquad \text{where } f : V \times N \to E \text{ and } g : V \times N \to [0, 1] \text{ with}$$

$$f(v, A) = \begin{cases} undefined & \text{if } In(v, A) = \emptyset \\ \operatorname{argmax}_{e \in In(v,A)} \wp \cdot b(e) & \text{if } e = (v, A)\langle\wp\rangle \to \varepsilon \\ \operatorname{argmax}_{e \in In(v,A)} \wp \cdot b(e) \cdot \prod_{i=1}^{m} g(u_i, B_i) & \text{if } e = (v, A)\langle\wp\rangle \to (u_1, B_1) \dots (u_m, B_m) \text{ for } m \in \mathbb{N}, m \geq 1 \end{cases}$$

$$g(v, A) = \begin{cases} undefined & \text{if } In(v, A) = \emptyset \\ \max_{e \in In(v,A)} \wp \cdot b(e) & \text{if } e = (v, A)\langle\wp\rangle \to \varepsilon \\ \max_{e \in In(v,A)} \wp \cdot b(e) \cdot \prod_{i=1}^{m} g(u_i, B_i) & \text{if } e = (v, A)\langle\wp\rangle \to (u_1, B_1) \dots (u_m, B_m) \text{ for } m \in \mathbb{N}, m \geq 1 \end{cases}$$

H is called *consistent* if E and ω are consistent. The set of all consistent derivation graphs over (G, p) and w is denoted by $\mathcal{H}_{(G,p)}(w)$. H is called *maximal* if there is no $H' \in \mathcal{H}_{(G,p)}(w)$ with a larger set of hyperedges.

Definition 2.18 (gtree)**.** Let $G = (N, \Sigma, \Xi, P, S)$, $w \in \Sigma^*$, $H = (V, E, b, \omega) \in \mathcal{I}_{(G,p)}(w)$ be a pruned derivation graph and $F = \{\#failure\#\}$, such that $F \cap \Sigma = \emptyset$. We define the mapping $gtree : \mathcal{I}_{(G,p)}(w) \to T_N(\Sigma \cup F)$ as

$$gtree(H) = \begin{cases} t([|w|], S) & \text{if } t([|w|], S) \in T_N(\Sigma) \\ \#failure\# & \text{otherwise} \end{cases}$$

where $t : V \times N \to T_N(\Sigma \cup F)$ with

$$t(v, A) = \begin{cases} w_i & \text{if } \theta_{\omega(v,A)|_1} = \varepsilon \text{ and for } i \in [|w|] : v = \{i\} \\ A(t(u_1, B_1), \dots, t(u_m, B_m)) & \text{if } \theta_{\omega(v,A)|_1} = (u_1, B_1) \dots (u_m, B_m) \\ \#failure\# & \text{otherwise.} \end{cases}$$

Definition 2.19 (Corpus)**.** Let (G, p) be a PLCFRS with $G = (N, \Sigma, \Xi, P, S)$ and $X \subset \Sigma^*$, $Y \subset T_N(\Sigma)$ be a finite sets. An $X \times Y$*-corpus*, denoted by c, is a mapping:

$$c : X \times Y \to \mathbb{R}_{\geq 0}.$$

We define the *support of* c, denoted by $supp(c)$, and the *size of* c, denoted by $|c|$ as

$$supp(c) = \{(x, y) \in X \times Y \mid c(x, y) > 0\} \qquad \text{and}$$
$$|c| = \sum_{(x,y)\in X\times Y} c(x, y) \qquad \text{respectively.}$$

3 WEIGHTED DEDUCTIVE PARSING FOR PLCFRS

In this section, we define a parser with the ability to take pruning policies into account. The parsers output is a pruned derivation graph. An weighted deductive parse algorithm for simple ordered ε-free PLCFRS (G, p) is used as a parsing algorithm for this purpose. The deduction system and parts of the parser are taken from [KM10]. We assume that every rule $\rho \in P$ for $G = (N, \Sigma, \Xi, P, S)$ has either the form $A(a) \to \varepsilon$ for $A \in N_1$ and $a \in \Sigma$, or the form $\phi \to \psi$ where $\psi \neq \varepsilon$ and no $a \in \Sigma^*$ occurs in ϕ. Furthermore, we define states for every iteration in the weigthed deductive CYK-like algorithm and actions which lead from one state to the next state. In this section, we will refer to a PLCFRS (N, Σ, Ξ, P, S) and to a sentence $w \in \Sigma^*$ as (G, p) and w, respectively.

Definition 3.1 (State)**.** A *state*, denoted by s, is a tuple $s = (H, e)$ where

- $H \in \mathcal{I}_{(G,p)}(w)$ is a pruned derivation graph and
- $e \in E$ is a hyperedge.

We denote the set of all states by $\mathcal{S}$.

Definition 3.2 (Set of actions)**.** The *set of actions*, denoted by $\mathcal{A}$, is the binary set

$$\mathcal{A} = \{keep, prune\}.$$

Definition 3.3 (Trajectory)**.** For each $0 \leq t \leq n$, let $a_t \in \mathcal{A}$ be an action and $s_{t+1} \in \mathcal{S}$ be a state. A *trajectory* is a sequence $\tau = s_0 a_0 s_1 a_1 \ldots s_n$ of alternating states and actions. We define $|\tau| = n$ as the trajectory length.

Definition 3.4 (Pruning policy)**.** Let $H = (V, E, b, \omega) \in \mathcal{I}_{(G,P)}(w)$ be a derivation graph. A *pruning policy* π is a mapping:

$$\pi : E \times \Sigma^* \to \mathcal{A}$$

The pruning policy decides what action $a \in \mathcal{A}$ should be chosen for each hyperedge (item) for a sentence.

To create the corresponding hyperedges (items) in the parsing process, we define a deduction systems with two deduction rules:

SCAN: $\dfrac{}{(\{i\}, A)\langle p(\rho)\rangle \to \varepsilon}$ if $\rho = A(w_i) \to \varepsilon$ for $i \in [|w|]$

RULE: $\dfrac{(v_1, B_1)\langle \wp_1\rangle \to \theta_1, \ldots, (v_m, B_m)\langle \wp_m\rangle \to \theta_m}{(\bigcup_{i=1}^{m} u_i, A)\langle p(\rho)\rangle \to (u_1, B_1,) \ldots (u_m, B_m)}$

if $\rho = A(\alpha_1, \ldots, \alpha_l) \to B_1(X_1^{(1)}, \ldots, X_{l_1}^{(1)}) \ \ldots \ B_m(X_1^{(m)}, \ldots, B_m(X_{l_m}^{(m)})$ where $\alpha_1, \ldots, \alpha_l \in \Xi^*$, $m \in \mathbb{N}$, $j \in [m]$, $k \in [l_j]$, $\wp_j \in [0, 1]$, $\forall q \in [l]$: $spans(\bigcup_{i=1}^{m} u_i)|_q = \bigcup_{X_k^{(j)} \in \alpha_q} spans(u_j)|_k$ and $\forall j' \in [m], j' \neq j : u_j$ and $u_{j'}$ are pairwise disjoint

Algorithm 1 Weighted deductive parsing for PLCFRS

Input: PLCFRS (G, p) with $G = (N, \Sigma, V, P, S)$,
sentence $w = w_1 \dots w_n$,
pruning policy π
Output: pruned derivation graph $H \in \mathcal{I}_{(G,p)}(w)$

function PARSE($(G, p), w, \pi$)
  $V := \mathcal{P}([|w|]) \setminus \{\emptyset\}$ ▷ initialize set of vertices
  for $v \in V$ **do**
    for $A \in N$ **do**
      $\omega(v, A) := \mathit{undefined}$ ▷ initialize the witness function
    end for
  end for
  $Q := \{\}$
  Add each item generated by SCAN to Q
  $E := \{\}$ ▷ initialize set of hyperedges
  for $e \in Q$ **do**
    $b(e) := 1$
    UPDATE($\omega, \eta_e|_1, \eta_e|_2$)
  end for
  while $Q \neq \emptyset$ **do**
    $e := \operatorname{argmax}_{e' \in Q}(\wp_e \cdot \prod_{(u,B)\in\theta_e} \omega(u, B)|_2)$
    $Q := Q \setminus \{e\}$
    $E := E \cup \{e\}$
    for each e' deduced from e and other items in E by RULE **do**
      if $\pi(e', w) = \mathit{keep}$ **then**
        UPDATE($\omega, \eta_{e'}|_1, \eta_{e'}|_2$)
        $b(e') := 1$ ▷ define mapping for e'
        if $e' \notin E \cup Q$ **then**
          $Q := Q \cup \{e'\}$
        end if
      end if
    end for
  end while
  return (V, E, b, ω)
end function

function UPDATE(ω, v, A)
  for $i \in \mathit{In}(v, A)$ **do** ▷ $i = (v_i, A_i)\langle\wp_i\rangle \to \theta_i$: hyperedge
    $s := \wp_i \cdot b(i) \cdot \prod_{(u_i,B_i)\in\theta_i} \omega(u_i, B_i)_2$
    if $s > \omega(v_i, A_i)|_2$ or $\omega(v_i, A_i) = \mathit{undefined}$ **then**
      $\omega(v_i, A_i) := (i, s)$ ▷ redefine mapping for v_i and A_i
    end if
  end for
end function

In Algorithm 1 we calculate the pruned derivation graph $H \in \mathcal{I}_{(G,p)}(w)$ for a given pruning policy π, PLCFRS (G, p) and a sentence w. We call $\hat{d}$ the best derivation for (G, p) and $w \in L(G)$ if

$$\hat{d} = \underset{d \in D_G(S,w)}{\operatorname{argmax}}\ p(d).$$

Hence, the best derivation can be obtained by calling the function $gtree$ on the maximal derivation graph $H' = (V, E, b, \omega) \in \mathcal{H}_{(G,p)}(w)$, if for each $e \in E$ we have $b(e) = 1$:

$$\hat{d} = gtree(H').$$

The maximal derivation graph contains every possible set of positions for the sentence w connected by the hyperedges restricted by the rules of the given grammar. Algorithm 1 can be subdivided into two steps.

The algorithm starts with initializing the future set of vertices V. However, the set of hyperedges is initialized with the empty set (E) and the witness function ω is initialized with $undefined$ for every possible nonterminal A and vertex v (lines 3 - 7). The set Q will be regarded as a queue of items (hyperedges) and is initialized with every item constructed by the SCAN deduction rule. The items generated by SCAN should never be pruned. For every item e in the queue at this time, ω adn b will be updated for η_e (lines 11 - 14).

In the next step, the algorithm extracts an item from the queue and adds it to the set of hyperedges. For every extracted item (e in line 16), new items (e' in line 19) will be constructed by the RULE deduction rule. However, only the resulting items by the RULE deduction rule where e occurs in the premise of RULE are considered as the new items. If the pruning policy π decides to keep a new item e', the mappings ω and b will be updated for $\eta_{e'}$ and e', respectively. Moreover, item e' will be added to the queue if it was not already in the queue or in the set of hyperedges.

The algorithm ends if the queue is exhausted (empty) and returns the resulting derivation graph afterwards.

4 TRAINING WITH LOLS

In Algorithm 1 we computed the derivation graph for a given pruning policy π. In this section, a method is introduced to train a pruning policy such that the resulting derivations over sentences have high accuracy and the computation has a relatively low runtime. Before the Locally Optimal Learning to Search LOLS algorithm ([Cha+15]) is introduced, further definitions are needed.

Definition 4.1 (Reward function)**.** Let (G, p) be a PLCFRS with $G = (N, \Sigma, \Xi, P, S)$, $w \in \Sigma^*$ and F be a finite set, such that $\Sigma \subseteq F$. The *reward function* is the mapping:

$$r : \mathcal{I}_{(G,p)}(w) \times T_N(\Sigma) \to \mathbb{R}$$

which is defined as: $r(H, \xi) = accuracy(gtree(H), \xi) - \lambda * runtime(H)$ where

- $accuracy : T_N(F) \times T_N(\Sigma) \to \mathbb{R}$,
- $runtime : \mathcal{I}_{(G,p)}(w) \to \mathbb{R}$ and
- $\lambda \geq 0$ is a paramter which determines the amount of accuracy to be sacrificed to reduce the runtime by one unit.

The reward function calculates the reward for a given derivation graph and a gold derivation given by a $X \times Y$-corpus such that $X \subset \Sigma^*$ and $Y \subset T_N(\Sigma)$.

Definition 4.2 (Empirical value)**.** Let (G, p) be a PLCFRS with $G = (N, \Sigma, \Xi, P, S)$, π be a pruning policy and c be an $X \times Y$-corpus such that $X \subset \Sigma^*$ and $Y \subset T_N(\Sigma)$. The *empirical value of* π, denoted by $\mathcal{R}(\pi)$, is the average reward on the training set:

$$\mathcal{R}(\pi) = \frac{1}{|c|} \sum_{(w,\xi) \in supp(c)} r(\textsc{Parse}((G, p), w, \pi), \xi) \cdot c(w, \xi).$$

For a given PLCFRS (G, p) and a corpus c, the goal is to find a pruning policy $\hat{\pi}$ such that

$$\hat{\pi} = \operatorname*{argmax}_{\pi} \mathcal{R}(\pi).$$

For the LOLS algorithm, we consider the following four functions as given:

- ROLL-IN takes as input a pruning policy π, PLCFRS, a word w and a gold tree ξ (derivation tree). The function works like the Algorithm 1 but instead of returning a pruned derivation graph, the function returns a complete trajectory τ.
- ROLL-OUT computes the reward for a given pruning policy π, a gold tree ξ and a chosen action $\bar{a}_t$ in state s_t.
- TRAIN determines a pruning policy π depending on a set of pairs consisting of a state s_t and the rewards $\vec{r}$ after executing an action.
- INITIALIZEPOLICY initializes the pruning policy π.

Algorithm 2 LOLS algorithm for learning to prune by [VE17] and [Cha+15]

Input: PLCFRS (G, p) with $G = (N, \Sigma, \Xi, P, S)$,
$X \times Y$-corpus c such that $X \subset \Sigma^*$ and $Y \subset T_N(\Sigma)$
Output: pruning policy π

```
1:  function LOLS((G, p), c)
2:      π_1 := INITIALIZEPOLICY(...)
3:      for i := 1 to n do                          ▷ n : number of iterations
4:          Q_i := ∅                                ▷ Q_i : set of state-reward tuples
5:          for (w, ξ) ∈ supp(c) do
6:              τ := ROLL-IN((G, p), w, π_i, ξ)     ▷ τ = s_0 a_0 s_1 a_1 ... s_T : trajectory
7:              for t := 0 to |τ| − 1 do
8:                  for ā_t ∈ A do                  ▷ intervention
9:                      r⃗_t[ā_t] := ROLL-OUT(π_i, s_t, ā_t, ξ)
10:                 end for
11:                 Q_i := Q_i ∪ {(s_t, r⃗_t)}
12:             end for
13:         end for
14:         π_{i+1} := TRAIN(⋃_{k=1}^{i} Q_k)       ▷ dataset aggregation
15:     end for
16:     return argmax_{π_j : 1≤j≤n} R(π_j)
17: end function
```

Algorithm 2 represents a computation to train a pruning policy π for a given PLCFRS (G, p) where $G = (N, \Sigma, \Xi, P, S)$. We consider an initial policy π_1 (line 2) which should be trained. For each pruning policy π_i and each sentence-tree pair (w, ξ) in the corpus, such that $\xi \in \{dtree(d) \mid d \in D_{(G,p)}(S, w)\}$, the algorithm computes the trajectory τ with the function ROLL-IN$((G, p), w, \pi_i)$ (line 6). Note, we chose the initial policy π_1 such that PARSE$((G, p), w, \pi_1) \in$

$\mathcal{H}_{(G,p)}(w)$. Afterwards, the algorithm computes for each action $\bar{a}_t$ (*keep* or *prune*) in the trajectory τ the vector of rewards for each action $\bar{a}_t \in \mathcal{A}$ with the function ROLL-OUT($\pi_i, s_t, \bar{a}_t$) (line 7 - 12). The tuples consisting of a state s_t and the rewards $\vec{r}$ after executing an action $\bar{a}_t$ are stored in a set Q_i. A new pruning policy π_{i+1} is obtained by the function TRAIN and considering the union of all Q_i to ensure a eventually converging sequence of pruning policies (dataset aggregation by [RGB10]). The best pruning policy π is determined by finding the argument π_j such that $\mathcal{R}(\pi_j)$ is maximized.

We observe that for PLCFRS (G, p) where $G = (N, \Sigma, \Xi, P, S)$, a sentence $w = w_1 \dots w_n$ and $H = (V, E, b, \omega) \in \mathcal{I}_{(G,p)}(w)$ the trajectory τ has length $|\tau| = |E| - |\{e \in E \mid \theta_e = \varepsilon\}|$, since the items generated by the SCAN deduction rule will not be pruned and therefore can be ignored. $|\tau|$ equals the number of actions in the trajectory and therefore the number of executions of the function ROLL-OUT.

To reduce the runtime of Algorithm 2, we will improve the ROLL-OUT function. Instead of starting over at state s_t and computing the derivation graph after flipping the pruning decision $\bar{a}_t$ (line 9) like the naive ROLL-OUT algorithm (not improved function as stated in Section 4), we compute the new pruned derivation graph H via the change propagation algorithm (Algorithm 3).

For the change propagation algorithm (Algorithm 3), we use two different functions, CHANGE and RECOMPUTE. In the function RECOMPUTE, we calculate the maximal score s for a vertex-nonterminal pair (v, A) such that:

$$s = \max_{i=(v,A)\langle\wp\rangle \to \theta \in In(v,A)} \wp \cdot b(e) \cdot \prod_{(u,B)\in\theta} \omega(u, B)|_2 \quad \text{where } \theta \text{ has the form } (u_1, B_1) \dots (u_m, B_m).$$

The hyperedge $i = (v, A)\langle\wp\rangle \to \theta$ with its score s are remembered as witnesses for the vertex-nonterminal pair (v, A) (lines 24 - 30). However, we have to consider three cases for the function CHANGE. CHANGE computes the new hypergraph after flipping the pruning bit $\hat{b}$ for an item (hyperedge) $b(e) = \hat{b}$ (line 2). At first, the witness function ω will be redefined for the arguments v and A to *NULL* and eventually added to the working queue Q. As long as the working queue is not empty, the function extracts the first vertex-nonterminal pair (u, B) and checks whether the witness of (u, B) is known. If there is no current witness for the said pair, the hyperedge as well as the new value for the said pair are recomputed (line 8 - 10). Afterwards, the function calculates for every outgoing hyperedge from (u, B) ($o = (v_o, A_o)\langle\wp_o\rangle \to \theta_o \in Out(u, B)$) the score

$$s = \max_{o=(v_o,A_o)\langle\wp_o\rangle \to \theta_o \in Out(u,B)} \wp_o \cdot b(o) \cdot \prod_{(u_o,B_o)\in\theta_o} \omega(u_o, B_o)|_2 \quad \text{where } \theta \text{ has the form } (u_1, B_1) \dots (u_m, B_m),$$

remembers the hyperedge o and the new score s as witnesses for the vertex-nonterminal pair (v_o, A_o) and adds the the pair to the working queue (lines 11 - 15). For the case, the score s for any hyperedge $o = (v_o, A_o)\langle\wp_o\rangle \to \theta_o \in Out(u, B)$ is smaller than $\omega(v_o, A_o)|_2$, the function will redefine ω for the arguments v_o and A_o to *NULL* and adds the vertex-nonterminal pair (v_o, A_o) to the working queue (lines 16 - 19). In any other case, no further computations are executed and the working queue will be reduced by one vertex-nonterminal pair. The algorithm ends if there are no more pairs in the working queue.

5 SETUP

For this work, an application for learning to prune with PLCFRS is provided.[1] In our setup we will use the first 1000 sentences from the TIGER corpus to induce a grammar (G, p) with the disco-dop library[2] developed by [CSB16].

[1] project on GitHub: `https://github.com/AndyPueschel/LearningToPruneLCFRS`
[2] library on GitHub: `https://github.com/andreasvc/disco-dop`

Algorithm 3 change propagation algorithm by [VE17] and [AL09]

Input: pruned derivation graph $H = (V, E, \nu, \omega) \in \mathcal{I}_{(G,p)}(w)$,
current hyperedge $e = (v, A)\langle \wp \rangle \to \theta \in E$
new value $\hat{b} \in \{0, 1\}$

1: **function** CHANGE(H, e, b)
2: $b(e) := \hat{b}$ ▷ redefine mapping for e
3: $\omega(v, A) := NULL$ ▷ redefine mapping for v and A
4: $Q := \{(v, A)\}$
5: **while** $Q \neq \emptyset$ **do**
6: Choose (u, B) from Q
7: $Q := Q \setminus \{u, B)\}$
8: **if** $\omega(u, B) = NULL$ **then**
9: RECOMPUTE(u, B)
10: **end if**
11: **for** $o \in Out(u, B)$ **do** ▷ $o = (v_o, A_o)\langle \wp_o \rangle \to \theta_o$: hyper edge
12: $s := \wp_o \cdot b(o) \cdot \prod_{(u_o, B_o) \in \theta_o} \omega(u_o, B_o)|_2$
13: **if** $s > \omega(v_o, A_o)|_2$ **then**
14: $\omega(v_o, A_o) := (o, s)$ ▷ redefine mapping for v_o and A_o
15: $Q := Q \cup \{(v_o, A_o)\}$
16: **else if** $\omega(v_o, A_o) = (e, p)$ and $p < \omega(v_o, A_o)|_2$ **then**
17: $\omega(v_o, A_o) := NULL$ ▷ redefine mapping for v_o and A_o
18: $Q := Q \cup \{(v_o, A_o)\}$
19: **end if**
20: **end for**
21: **end while**
22: **end function**

23: **function** RECOMPUTE(v, A)
24: **for** $i \in In(v, A)$ **do** ▷ $i = (v_i, A_i)\langle \wp_i \rangle \to \theta_i$: hyperedge
25: $s := \wp_i \cdot b(i) \cdot \prod_{(u_i, B_i) \in \theta_i} \omega(u_i, B_i)|_2$
26: **if** $s > \omega(v_i, A_i)|_2$ **then**
27: $\omega(v_i, A_i) := (i, s)$ ▷ redefine mapping for v_i and A_i
28: **end if**
29: **end for**
30: **end function**

We use the first 67 sentences in the TIGER corpus with length 5 or less for the trainings corpus. These sentences are part of the set from where the grammar is induced. The grammar weights p for the rules in G will never be retrained, i.e. for every iteration in the LOLS algorithm the same probability distribution p will be used over again.

Since only 67 sentences are used for the trainigns algorithm, concrete items (the states with their rewards) would create a very meager decision making for the pruning policy. A solution is the abstraction of such items via feature functions to more abstract items so that the pruning policy is able to decide whether an item should be pruned or not, even though the concrete item was not part of the trainings set. The training itself (TRAIN in Section 4) is considered as a black box trainings process. We use appropriate functions from the pandas[3] and sklearn[4] library. In our experimental design, we consider the hyperedges of $H \in \mathcal{I}_{(G,p)}(w)$ for a sentence w as items where the feature function will be applied on. We consider an item as a hyperedge $e = (v_e, A_e)\langle\wp_e\rangle \to \theta_e$ and we use the following feature functions for the transformation:

- *label* retrieves the nonterminal A_e of an item e.
- *length* retrieves the sentence's length.
- *boundary words* retrieves for every continuous interval in the cover of the item the two boundary words for at most $fanout(G)$ many intervals.
- *boundary words conjunction* retrieves for every continuous interval in the cover of the item and for the two boundary words of the interval at positions i and k the four word conjunctions at positions $(i-1, i)$, $(k, k+1)$, (i, k) and $(i-1, k+1)$ for at most $fanoutG$) many intervals.
- *word shape conjunction* retrieves for every continuous interval in the cover of the item and for the two boundary words of the interval at positions i and k the four conjunctions of word shapes at positions $(i-1, i)$, $(k, k+1)$, (i, k) and $(i-1, k+1)$ after mapping each word to one of the shapes (uppercase, lowercase, numeric, special) for at most $fanout(G)$ many intervals.
- *span size* retrieves the number of positions in the cover of an item.
- *span length* retrieves the length of the cover of an item while taking the minimum and maximum position in the cover to account.
- *width bucket* retrieves the width category after mapping the sentence length to one of the categories $(2, 3, 4, 5, [6, 10], [11, 20], [20, \infty))$.

In case of overstepping the dimension of the sentence (words at unknown sentence positions) for $i-1$ and $k+1$ in the boundary words conjunction and word shape conjunction features, a special character sequence will be used instead.

6 RESULTS

For the experiment, we have to define how the functions *accuracy* and *runtime* will compute their results. Therefore, more definitions are needed.

Definition 6.1 (Recall [Pow11])**.** Let (G, p) be a PLCFRS with $G = (N, \Sigma, \Xi, P, S)$, $w \in \Sigma^*$ be a sentence and $\xi, \zeta \in T_N(\Sigma)$ be the derivation tree and gold tree, respectively. The *recall of* ξ *related to* ζ, denoted by $\mathfrak{r}(\xi, \zeta)$, is the number of gold constitutents in ζ appearing as vertices in the derivation ξ devided by the number of all constituents in ζ.

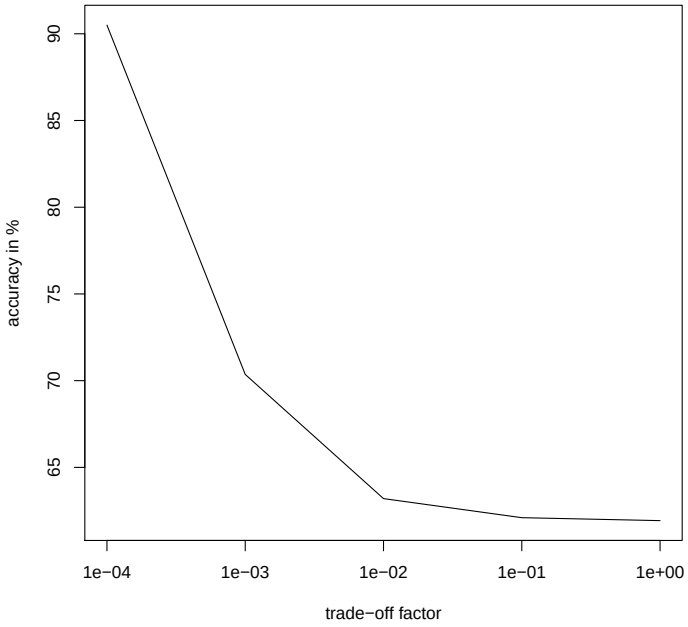
[3] Python Data Analysis Library: `https://pandas.pydata.org/`
[4] machine learning in python: `http://scikit-learn.org/stable/`

Definition 6.2 (Precision [Pow11])**.** Let (G, p) be a PLCFRS with $G = (N, \Sigma, \Xi, P, S)$, $w \in \Sigma^*$ be a sentence and $\xi, \zeta \in T_N(\Sigma)$ be the derivation tree and gold tree, respectively. The *precision of ξ related to ζ*, denoted by $\mathfrak{p}(\xi, \zeta)$, is the number of gold constituents in ζ which appear as vertices in ξ devided by the number of all vertices in ξ.

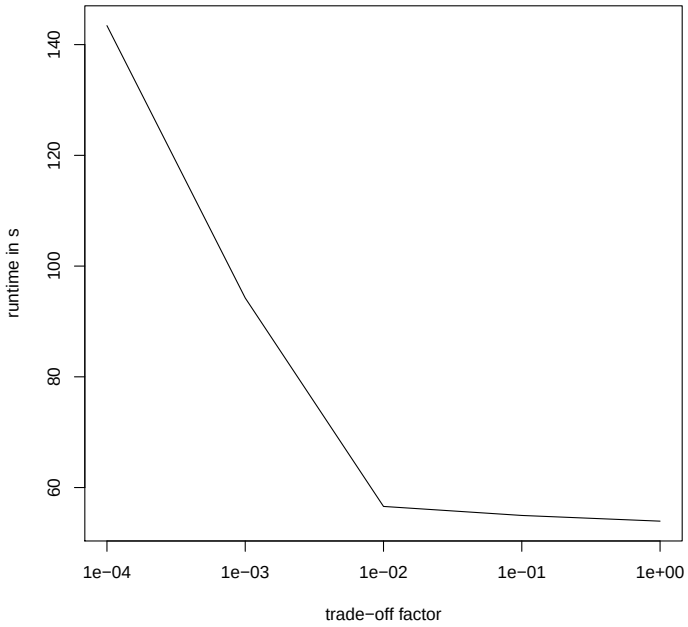
Definition 6.3 (F_1 score [Pow11])**.** Let (G, p) be a PLCFRS with $G = (N, \Sigma, \Xi, P, S)$, $w \in \Sigma^*$ be a sentence, and $\xi, \zeta \in T_N(\Sigma)$ be the derivation tree and gold tree, respectively. The F_1 *score* is the mapping $F_1 : T_N(\Sigma) \times T_N(\Sigma) \to \mathbb{R}_{\geq 0}$:

$$F_1(\xi, \zeta) = 2 \cdot \frac{\mathfrak{r}(\xi, \zeta) \cdot \mathfrak{p}(\xi, \zeta)}{\mathfrak{r}(\xi, \zeta) + \mathfrak{p}(\xi, \zeta)}.$$

The *accuracy* will be the F_1 measure and the *runtime* will be the number of hyperedges in the set E for a given derivation graph $H = (V, E, b, \omega) \in \mathcal{I}_{(G,p)}(w)$. The experiment will be run with different values for the trade-off factor λ, namely $1, 0.1, 0.01, 0.001$ and 0.0001.



(a) accuracy for λ



(b) runtime for λ

Figure 1: runtime and accuracy for given *lambda*

Figure 1 shows the impact of λ for the average accuracy and the average runtime. While the values for high λ varies slightly, an enormous difference can be observed for small λ (0.0001 and 0.001). Considering the smallest and the largest λ in this experimental design (0.0001 and 1), a reduction of the runtime by almost 30 seconds can be achieved if we use large λ for pruning. However, we will lose almost 30% accuracy as well.

7 CONCLUSION AND OUTLOOK

In this work, we presented a parse chart-like construct for PLCFRS in form of (pruned) derivation graphs and an approach to train pruning policies for PLCFRS by using an user-defined trade-off factor. We investigated the impact of the trade-off factor in case of the accuracy and in case of the runtime for weighted deductive parsing for PLCFRS. The parser works with PLCFRS with any fanout size and the training allows to use redefined features or self defined features.

For the calculation of the reward, we used one fixed computation method, schematically $reward = accuracy - \lambda * runtime$, as proposed in [VE17]. However, it could be interesting to see in future work, how the resulting pruning policies will change if not only the trade-off factor λ would be a variable but the computing method for the reward as well. In the described setup, we used the fixed features as defined in chapter 6. New features could be added easily for investigating its impact on the pruning policies. Furthermore, the used parser works in an exhausting manner, speed ups in the parsing process can be reached by implementing an non-exhaustive parser or even one with taking normalized PLCFRS into account.

8 REFERENCES

[AL09] Umut A. Acar and Ruy Ley-Wild. "Self-adjusting Computation with Delta ML". In: *Advanced Functional Programming: 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*. Ed. by Pieter Koopman, Rinus Plasmeijer, and Doaitse Swierstra. Springer Berlin Heidelberg, 2009, pp. 1–38. ISBN: 978-3-642-04652-0. DOI: 10.1007/978-3-642-04652-0_1.

[Cha+15] Kai-Wei Chang et al. "Learning to Search Better than Your Teacher". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. Ed. by David Blei and Francis Bach. JMLR Workshop and Conference Proceedings, 2015, pp. 2058–2066.

[CSB16] Andreas van Cranenburgh, Remko Scha, and Rens Bod. "Data-Oriented Parsing with discontinuous constituents and function tags". In: *Journal of Language Modelling* 4.1 (2016), pp. 57–111. URL: http://dx.doi.org/10.15398/jlm.v4i1.100.

[Kal12] Laura Kallmeyer. *Parsing Beyond Context-Free Grammars*. Springer Publishing Company, Incorporated, 2012. ISBN: 3642264530, 9783642264535.

[KM10] Laura Kallmeyer and Wolfgang Maier. "Data-driven Parsing with Probabilistic Linear Context-free Rewriting Systems". In: *Proceedings of the 23rd International Conference on Computational Linguistics*. COLING '10. Beijing, China: Association for Computational Linguistics, 2010, pp. 537–545. URL: http://dl.acm.org/citation.cfm?id=1873781.1873842.

[KSK06] Yuki Kato, Hiroyuki Seki, and Tadao Kasami. "Stochastic Multiple Context-free Grammar for RNA Pseudoknot Modeling". In: *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms*. TAGRF '06. Sydney, Australia: Association for Computational Linguistics, 2006, pp. 57–64. ISBN: 1-932432-85-X. URL: http://dl.acm.org/citation.cfm?id=1654690.1654698.

[Ned03] Mark-Jan Nederhof. "Weighted deductive parsing and Knuth's algorithm". In: *Computational Linguistics* 29.1 (2003), pp. 135–143.

[Pow11] David M. W. Powers. "Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation". In: *Journal of Machine Learning Technologies* 2.1 (2011), pp. 37–63. ISSN: 2229-3981 & 2229-399X.

[RGB10] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "No-Regret Reductions for Imitation Learning and Structured Prediction". In: *CoRR* abs/1011.0686 (2010).

[VE17] Tim Vieira and Jason Eisner. "Learning to Prune: Exploring the Frontier of Fast and Accurate Pasring". In: *Transactions of the Association for Computational Linguistics (TACL)* 5 (Feb. 2017).

[VWJ87] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. "Characterizing Structural Descriptions Produced by Various Grammatical Formalisms". In: *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*. ACL '87. Stanford, California: Association for Computational Linguistics, 1987, pp. 104–111. DOI: 10.3115/981175.981190. URL: https://doi.org/10.3115/981175.981190.