Proof net structure for neural Lambek categorial parsing

Aditya Bhargava and **Gerald Penn**
Department of Computer Science
University of Toronto
Toronto, ON, Canada M5S 3G4
{aditya, gpenn}@cs.toronto.edu

Abstract

In this paper, we present the first statistical parser for Lambek categorial grammar (LCG), a grammatical formalism for which the graphical proof method known as *proof nets* is applicable. Our parser incorporates proof net structure and constraints into a system based on self-attention networks via novel model elements. Our experiments on an English LCG corpus show that incorporating term graph structure is helpful to the model, improving both parsing accuracy and coverage. Moreover, we derive novel loss functions by expressing proof net constraints as differentiable functions of our model output, enabling us to train our parser without ground-truth derivations.

1 Introduction

In the family of categorial grammars, combinatory categorial grammar (CCG) has received by far the most attention in the computational linguistics literature. There exist algorithms for both mildly context-sensitive (*e.g.*, Kuhlmann and Satta, 2014) and context-free (typically CKY; Cocke and Schwartz, 1970; Kasami, 1966; Younger, 1967) CCG parsing, and there has been much research on statistical CCG parsers (*e.g.*, Clark and Curran, 2007; Lewis et al., 2016; Stanojević and Steedman, 2020). Another member of the categorial family, Lambek categorial grammar (LCG), has been less well-explored: LCG work has been primarily theoretical or focused on non-statistical parsing.

The recent lack of attention is likely due to two notable results: (1) LCG is weakly context-free equivalent (Pentus, 1997); and (2) LCG parsing is NP-complete (Pentus, 2006; Savateev, 2012). However, neither of these issues is necessarily *practically* relevant. Moreover, LCG presents a number of advantages and interesting properties. For example, LCG provides even greater syntax-semantics transparency than is the case for most CCG parsers because it does not invoke non-categorial rules, maintaining a consistent parsing framework. LCG's rules together define a calculus over syntactic categories that is a subset of linear logic (Girard, 1987).

LCG, like CCG or LTAG, is a highly lexicalized formalism: lexical categories encode substantial syntactic information, and as a result are themselves complex and structured. Despite this, the inner structure of the categories has not been strongly considered in parsers beyond evaluating the category for compatibility with a grammatical rule.

In this paper, we present the first statistical LCG parser. Unlike past parsers for CCG or LTAG, our parser explicitly incorporates structural aspects of the grammar. We base our system on *proof nets*, a graphical method for representing linear logic proofs that abstracts over irrelevant aspects, such as the order of application of logical rules (Girard, 1987; Roorda, 1992). This corresponds to the problem of spurious ambiguity, making proof nets an attractive choice for representing derivations.

Our work has two primary contributions. First, we introduce a self-attention–based LCG parsing model that incorporates proof net structure in multiple ways. We find that minding proof net structure enables us to define a model that is differentiable through this categorial structure down to the atomic categories of the grammar, improving parsing accuracy and coverage on an English LCG corpus.

Second, proof net constraints allow us to define novel grammatico-structural loss functions that can be used as training objectives. This enables us to train a parser *without ground-truth derivations* that has high coverage and even frequently includes the correct parse among the parses that it finds. Our analysis shows that all of our components contribute to the parser's performance, but that planarity information is especially important.

Proceedings of the 17th International Conference on Parsing Technologies (IWPT 2021), pages 13–25
Bangkok, Thailand (online), August 6, 2021.

$$\frac{\Delta \vdash X/Y \quad \Gamma \vdash Y}{\Delta, \Gamma \vdash X} /_{\mathrm{e}} \qquad \frac{\Gamma \vdash Y \quad \Delta \vdash X\backslash Y}{\Gamma, \Delta \vdash X} \backslash_{\mathrm{e}}$$

$$\frac{\Gamma, Y \vdash X}{\Gamma \vdash X/Y} /_{\mathrm{i}} \qquad \frac{Y, \Gamma \vdash X}{\Gamma \vdash X\backslash Y} \backslash_{\mathrm{i}}$$

$$\frac{}{X \vdash X}\ \text{axiom}$$

Figure 1: The rules of the associative Lambek calculus without product and allowing empty premises.

2 Background

2.1 Lambek categorial grammar

Lexical categories in LCG, like those of CCG, comprise an infinite set of categories that is formed by the closure of two binary connectives, the forward (/) and backward slash (\), on a small set of **atomic** (*i.e.*, primitive) categories, such as S and NP for sentences and noun phrases. The connectives both create functional categories, and they differ in which of a word or phrase a specified argument must appear. For example, (S\NP)/NP/NP represents a category that combines with two NPs to its right and one NP to its left to yield a valid S.[1] In English, this category might represent a ditransitive verb.

Figure 1 shows the rules of inference for **L***, the associative Lambek calculus without product and allowing empty premises. In L*, statements, called **sequents**, have (ordered) lists of categories as **antecedents** on the left of the turnstile and single categories as **consequents** on the right. The interpretation of a sequent is that its consequent can be derived from its antecedents. The rules have their **premises** above a bar, **conclusions** below, and a label for the rule to the right. Rules $/_{\mathrm{e}}$ and $\backslash_{\mathrm{e}}$ *eliminate* a slashed category, in that it is missing from their conclusions; rules $/_{\mathrm{i}}$ and $\backslash_{\mathrm{i}}$ *introduce* a new slashed functor in the consequent of their conclusions.

Each rule states that its concluding sequent is true (derivable) if and only if all of its premises are. X and Y are variables over categories (atomic or complex) while Δ and Γ are variables over possibly-empty[2] *lists* of categories. A typical application of LCG is to look up a category for each word in a sentence and then inquire whether the consequent S is derivable from the antecedent that lists these categories in the same order as their words.

While some of CCG's rules are not derivable in the Lambek calculus (*inter alia*, crossing composition and substitution), first-degree harmonic composition and type-raising are. At the same time, LCG's introduction rules cannot be derived by any CCG with finite rules (Zielonka, 1981).

Although LCG parsing is known to be an NP-complete problem (Pentus, 2006; Savateev, 2012), Fowler (2010) presented an algorithm that is exponential only in category *order*, a quantity that is bounded to small values in practice (Fowler, 2016).

2.2 Term graphs: enhanced proof nets

Our work in this paper is based on a variety of proof net known as term graphs. A **term graph** is a digraph that represents a sequent proof in the Lambek calculus. The atoms of the sequent correspond to vertices in the graph, and the internal structure of the lexical categories is represented by **regular** edges and **Lambek** edges between the vertices. Together, the vertices, regular edges, and Lambek edges are referred to as a **proof frame**, which is invariant across possible proofs for the sequent. A proof is represented by a proof frame plus by an additional set of regular edges between the vertices called a **linkage**. Different linkages correspond to different proofs, which in turn correspond to different syntactic parses. For a term graph to be valid, the frame-plus-linkage is subject to certain conditions, detailed below.[3]

To construct a proof frame for a sequent $A_1, A_2, \ldots, A_n \vdash B$, the categories in the sequent are first assigned positive or negative polarities. Each lexical category A_i of the antecedent is marked negative (A_i^-), while the consequent B is marked positive (B^+). Each polarized category is decomposed into its polarized atoms according to a set of recursively-applied rules. These rules also specify the regular and Lambek edges between the atoms, represented as solid and dashed edges, respectively. The lexical decomposition rules are:

$$(X/Y)^- \Rightarrow X^- \rightarrow Y^+ \qquad (X\backslash Y)^- \Rightarrow Y^+ \leftarrow X^-$$
$$(X\backslash Y)^+ \Rightarrow X^+ \dashrightarrow Y^- \qquad (X/Y)^+ \Rightarrow Y^- \dashleftarrow X^+$$

The total order of the frame (indicated left-to-right) is determined by the ordering of the lexical categories in the sequent together with the ordering specified in the decomposition rules above.

A linkage for a proof frame consists of directed edges called **links** from positive vertices to negative vertices of the same atomic category. Valid linkages form perfect matchings: each vertex in the frame

[1] Although LCG's usual notation employs these connectives slightly differently, we use CCG notation here.

[2] There are calculus variants that disallow such empty lists.

[3] See (Fowler, 2009, 2016) for full details.

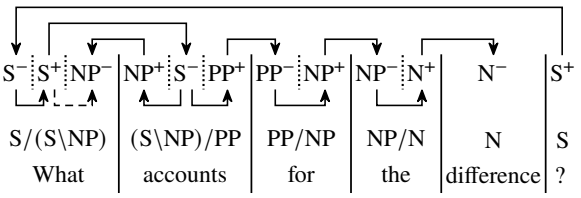

Figure 2: An example term graph. Dotted vertical lines delimit polarized atoms within a word; solid vertical lines mark lexical boundaries. The linkage is shown above the atoms; the proof frame edges are shown below them, with solid regular edges and dashed Lambek edges. The consequent category is aligned with sentence-final punctuation for convenience. This single term graph represents multiple spuriously ambiguous derivations.

has exactly one link, and that link is outgoing for positive vertices and incoming for negative vertices.

A term graph represents a proof in L* (Fowler, 2009), and therefore also an LCG parse, so long as it meets the following conditions:

T1. The linkage is half-planar; *i.e.*, the links can be drawn above the linearly-ordered vertices without crossing.

T2. Treating links as regular edges, the graph is regular-acyclic; *i.e.*, there are no cycles containing only regular edges.

T3. For each Lambek edge $\langle i, j \rangle$, there exists a regular path from i to j.

A term graph that satisfies these conditions is called **L*-integral**. Figure 2 shows an example term graph.

3 Neural network LCG parsing

For categorial and other highly-lexicalized grammatical formalisms, the standard approach to statistical syntactic parsing separates the problem into two steps: (1) a supertagger assigns lexical categories to the words in the input sentence; then (2) a parser uses the supertagger's predictions to produce a predicted parse for the sentence. With proof nets, the lexical categories uniquely determine the proof frame, so supertagging can be seen as predicting a proof frame for the sentence. The second step then corresponds to predicting the linkage for the proof frame. Of course, the linkage must, together with the proof frame, yield an L*-integral term graph.

Our work in this paper focuses on the latter component. Our parser is constructed such that we can separate its aspects that incorporate term graph structure and constraints from a "baseline" model which uses almost no such information. To provide a broad overview, our baseline model runs a Transformer encoder stack (Vaswani et al., 2017) over the proof frame vertices. The top encoder block is truncated, omitting everything after and including the softmax, which directly yields scores for every pair; we mask these scores so that only valid links (*i.e.*, those from positive vertices to negative vertices of the same atomic category) are considered.

We next detail our baseline model in Section 3.1 and then our various methods for incorporating term graph structure in Sections 3.2–3.4. Note that we use named tensor notation (Chiang et al., 2021) in the mathematical descriptions.

3.1 Baseline: parsing by predicting links

3.1.1 Parser inputs

Our baseline model takes as input a sentence, an associated proof frame, and an alignment between the words in the sentence and the polarized atoms that are the proof frame's vertices. We represent each word as a vector of size $|\mathsf{vec}|$ so that a sentence of length $|\mathsf{words}|$ is represented as a matrix $\boldsymbol{H} \in \mathbb{R}^{\mathsf{words} \times \mathsf{vec}}$. For a grammar with atomic categories $\mathcal{T} = \{t_1, t_2, \ldots, t_{|T|}\}$, the set of possible polarized atoms is $\mathcal{P}_\mathcal{T} = \{t_1^+, t_1^-, t_2^+, t_2^-, \cdots, t_{|\mathcal{T}|}^+, t_{|\mathcal{T}|}^-\}$; the proof frame vertices are thus each represented as one-hot vectors of width $|\mathsf{pt}| = |\mathcal{P}_\mathcal{T}| = 2|\mathcal{T}|$. A proof frame with $|\mathsf{vtx}|$ vertices is represented by stacking its vertex's vectors, forming a matrix $\boldsymbol{N} \in \{0,1\}^{\mathsf{vtx} \times \mathsf{pt}}$ with $\sum_{\mathsf{pt}} \boldsymbol{N} = 1$. The vtx axis is ordered according to the vertices' total order. Finally, the word-vertex alignment is represented as a matrix $\boldsymbol{M} \in \{0,1\}^{\mathsf{vtx} \times \mathsf{words}}$ where $\boldsymbol{M}_{\mathsf{vtx}(i),\mathsf{words}(j)} = 1$ if and only if vertex i corresponds to word j.

3.1.2 Transformer encoder stack

The Transformer encoder stack as defined by Vaswani et al. (2017) adds positional encoding vectors to the model inputs. In our case, we have two input sequences (polarized atoms and word vectors) of differing lengths, along with an alignment between them. We include the positional encoding vectors over the *word* positions as inputs to the encoder, and apply *relative* positional attention (Dai et al., 2019) during the self-attention step over the *polarized atom* positions. We found this combination most effective during development.

More precisely, we add the usual sinusoidal positional encoding vectors (Vaswani et al., 2017) $\boldsymbol{P}_\mathrm{w} \in \mathbb{R}^{\mathsf{words} \times \mathsf{vec}}$ to the word vectors and map the result to the vertex indices. We embed the polarized atoms $\boldsymbol{N}$ via trainable matrix $\boldsymbol{A} \in \mathbb{R}^{\mathsf{pt} \times \mathsf{vec}}$ and add them to their corresponding word vectors to yield

inputs $\boldsymbol{X}_0 \in \mathbb{R}^{\mathsf{vtx} \times \mathsf{vec}}$ for the encoder stack:

$$[\mathrm{PE}(p)]_{\mathsf{vec}(i)} = \begin{cases} \sin\left(\dfrac{p-1}{10^{4(i-1)/|\mathsf{vec}|}}\right) & i \text{ odd} \\ \cos\left(\dfrac{p-1}{10^{4(i-2)/|\mathsf{vec}|}}\right) & i \text{ even} \end{cases}$$

$$[\boldsymbol{P}_\mathrm{w}]_{\mathsf{words}(p)} = \mathrm{PE}(p)$$

$$\boldsymbol{X}_0 = (\boldsymbol{H} + \boldsymbol{P}_\mathrm{w}) \underset{\mathsf{words}}{\odot} \boldsymbol{M} + \boldsymbol{N} \underset{\mathsf{pt}}{\odot} \boldsymbol{A}$$

The Transformer encoder stack consists of L encoder layers. Denoting the input to layer l as $\boldsymbol{X}_{l-1}$, with $\boldsymbol{X}_0$ as above, each layer computes $\boldsymbol{X}_l$ as:

$$\boldsymbol{T}_l = \mathrm{SelfAttn}_l(\boldsymbol{X}_{l-1}) + \boldsymbol{X}_{l-1}$$
$$\boldsymbol{X}_l = \mathrm{FFN}_l(\boldsymbol{T}_l) + \boldsymbol{T}_l$$

with FFN defined as in (Vaswani et al., 2017).[4]

From a given input sequence, standard self-attention computes query, key, and value tensors. Although all three tensors derive from the same input sequence, the key and value tensors function as "memory" tensors, so their sequence axis is a "lookup" axis distinct from that of the query tensors. In our case, the input sequence axis is vtx, so we preserve this distinction by renaming the vtx axis to vtx′ for the key and value tensors.

We employ relative positional encoding following Dai et al. (2019), allowing our model to directly learn to attend to polarized atoms at positions relative to a given atom. The *relative* positional vectors are represented as a tensor $\boldsymbol{P}_\mathrm{v} \in \mathbb{R}^{\mathsf{vtx} \times \mathsf{vtx}' \times \mathsf{vec}}$ so that $[\boldsymbol{P}_\mathrm{v}]_{\mathsf{vtx}(i),\mathsf{vtx}'(j)} = \mathrm{PE}(i-j)$ is the encoding of position j (on the key/value axis) *relative to* position i (on the query axis).

For multi-headed self-attention, each encoder layer l computes $|\mathsf{heads}|$ attention heads, each of width $|\mathsf{hdim}|$. We thus have trainable parameters $\boldsymbol{W}_{\mathrm{q},l}, \boldsymbol{W}_{\mathrm{k},l}, \boldsymbol{W}_{\mathrm{v},l}, \boldsymbol{W}_{\mathrm{r},l} \boldsymbol{W}_{\mathrm{o},l} \in \mathbb{R}^{\mathsf{heads} \times \mathsf{hdim} \times \mathsf{vec}}$ and $\boldsymbol{b}_{\mathrm{k},l}, \boldsymbol{b}_{\mathrm{r},l} \in \mathbb{R}^{\mathsf{heads} \times \mathsf{hdim}}$ with which we compute the query, key, value, relative position encoding, and attention score tensors $\boldsymbol{Q}_l, \boldsymbol{K}_l, \boldsymbol{V}_l, \boldsymbol{R}_l$, and $\boldsymbol{S}_l$ as:

$$\boldsymbol{Q}_l = \boldsymbol{W}_{\mathrm{q},l} \underset{\mathsf{vec}}{\odot} \boldsymbol{X}_{l-1} \tag{1}$$

$$\boldsymbol{K}_l = \boldsymbol{W}_{\mathrm{k},l} \underset{\mathsf{vec}}{\odot} [\boldsymbol{X}_{l-1}]_{\mathsf{vtx} \to \mathsf{vtx}'} \tag{2}$$

$$\boldsymbol{V}_l = \boldsymbol{W}_{\mathrm{v},l} \underset{\mathsf{vec}}{\odot} [\boldsymbol{X}_{l-1}]_{\mathsf{vtx} \to \mathsf{vtx}'}$$

$$\boldsymbol{R}_l = \boldsymbol{W}_{\mathrm{r},l} \underset{\mathsf{vec}}{\odot} \boldsymbol{P}_\mathrm{v}$$

[4] We apply layer normalization (Ba et al., 2016) as well, but omit it here for concision. We use the "pre-norm" application order (Wang et al., 2019; Nguyen and Salazar, 2019).

$$\boldsymbol{S}_l = \frac{(\boldsymbol{Q}_l + \boldsymbol{b}_{\mathrm{k},l}) \underset{\mathsf{hdim}}{\odot} \boldsymbol{K}_l + (\boldsymbol{Q}_l + \boldsymbol{b}_{\mathrm{r},l}) \underset{\mathsf{hdim}}{\odot} \boldsymbol{R}_l}{\sqrt{|\mathsf{hdim}|}} \tag{3}$$

where $\mathsf{vtx} \to \mathsf{vtx}'$ denotes renaming axis vtx to vtx′. Next, with final trainable parameter $\boldsymbol{W}_{\mathrm{o},l} \in \mathbb{R}^{\mathsf{heads} \times \mathsf{hdim} \times \mathsf{vec}}$, we compute the $\mathrm{SelfAttn}_l$ output:

$$\mathrm{SelfAttn}_l(\boldsymbol{X}_{l-1}) = \boldsymbol{W}_{\mathrm{o},l} \underset{\substack{\mathsf{hdim} \\ \mathsf{heads}}}{\odot} \left(\underset{\mathsf{vtx}'}{\mathrm{softmax}}(\boldsymbol{S}_l) \underset{\mathsf{vtx}'}{\odot} \boldsymbol{V}_l \right) \tag{4}$$

Each encoder layer includes all of these steps except for the final layer $l = L$, where we omit Equation 4.

Finally, we apply a mask $\boldsymbol{F} \in \{0, \infty\}^{\mathsf{vtx} \times \mathsf{vtx}'}$ to ensure that only edges from positive atoms to negative atoms of the same category are considered:

$$\boldsymbol{F}_{\mathsf{vtx}(i),\mathsf{vtx}'(j)} = \begin{cases} 0 & \text{if } \mathrm{atom}(i) = \mathrm{atom}(j) \\ & \text{and } \mathrm{pol}(i) > \mathrm{pol}(j), \\ \infty & \text{otherwise} \end{cases}$$

$$\boldsymbol{S} = \underset{\mathsf{heads}}{\mathrm{mean}}(\boldsymbol{S}_L) - \boldsymbol{F} \tag{5}$$

where $\mathrm{atom}(i)$ returns the category of vertex i and:

$$\mathrm{pol}(i) = \begin{cases} 1 & \text{if vertex } i \text{ is positive,} \\ -1 & \text{if vertex } i \text{ is negative.} \end{cases}$$

3.1.3 Linkage loss function

Given the predicted score matrix $\boldsymbol{S}$, it still remains to specify how to predict candidate linkages. We first note the problem with which we are presented at this stage is exactly that of finding the max-weight (or min-cost) perfect bipartite matching. Ideally, $\boldsymbol{S}$ will provide scores that, when optimized over, yield the desired matching, *i.e.*, the ground-truth linkage.

As we aim to train our parser on a corpus with ground-truth linkages using gradient descent, our perfect matching algorithm must be differentiable for training so that gradients can be back-propagated through it from the loss function; we use Sinkhorn's algorithm (Sinkhorn and Knopp, 1967) with temperature, also known as SoftAssign (Kosowsky and Yuille, 1994; Gold and Rangarajan, 1996b). The procedure, which alternates normalizing the rows and columns of $\exp(\boldsymbol{S}/\tau)$, $\tau > 0$, converges to a doubly-stochastic matrix. In the limit of $\tau \to 0$, this converges to the optimal matching (Mena et al., 2018), thereby providing a means of computing the optimal linkage. Moreover, with the addition of standard Gumbel noise, $\boldsymbol{S}$ can be seen to parameterize a distribution over permutation matrices, with the Sinkhorn operator then functioning as a means of sampling from this distribution.

Importantly for training with gradient descent, the operations of Sinkhorn's algorithm are fully differentiable. For a given doubly-stochastic output matrix from Sinkhorn's algorithm, the negative log likelihood $\mathfrak{J}_{\text{NLL}}$ of the ground-truth linkage $\mathcal{L} = \{\langle i_1, j_1\rangle, \langle i_2, j_2\rangle, \ldots, \langle i_{|\mathsf{vtx}|/2}, j_{|\mathsf{vtx}|/2}\rangle\}$ is a natural choice of loss function for training:

$$\boldsymbol{Z} = \underset{\mathsf{vtx},\mathsf{vtx}'}{\text{Sinkhorn}}(\exp(\boldsymbol{S}/\tau)) \tag{6}$$

$$\mathfrak{J}_{\text{NLL}}(\mathcal{L}, \boldsymbol{Z}) = -\underset{\langle i,j\rangle\in\mathcal{L}}{\text{mean}}\left(\ln[\boldsymbol{Z}]_{\mathsf{vtx}(i),\mathsf{vtx}'(j)}\right) \tag{7}$$

Our base model uses this loss function and is trained with the ground-truth linkages as targets.

3.2 Modelling term graph structure

The model just described is a straightforward application of attention scores and Sinkhorn's algorithm to the problem of finding linkages for a proof frame. However, the only place where the structured nature of the proof net is exploited is in the polarity and category restrictions on the matching; other relevant characteristics are not directly taken into account, such as the proof frame or the validity conditions. Given that the validity conditions cannot even be evaluated without the proof frame edges, we hypothesize that including knowledge of the proof frame structure will help the model to select valid linkages, or even the correct one. Similarly, encoding information about the validity conditions themselves may also be beneficial. We therefore incorporate term graph structure into our model in a number of ways, which we now describe.

3.2.1 Regular and Lambek edges

As is, the parser does not have knowledge of the internal structure of the lexical categories; while it receives as input the atomic category and polarity of each vertex in the proof frame, it has no knowledge of the regular and Lambek edges. We hypothesize that incorporating this structure will boost parser performance, as the edges provide crucial information about which links combinations fail to satisfy the validity conditions for term graphs.

To encode these edges in the parser, we alter the inputs to the encoder's attention blocks. We represent the regular edges as an adjacency matrix $\boldsymbol{E}_{\text{R}} \in \{0,1\}^{\mathsf{vtx}\times\mathsf{vtx}'}$ where $[\boldsymbol{E}_{\text{R}}]_{\mathsf{vtx}(i),\mathsf{vtx}'(j)} = 1$ if and only if the proof frame has a regular edge from (negative) vertex i to (positive) vertex j. Lambek edges are represented similarly as an adjacency matrix $\boldsymbol{E}_{\text{L}}$. For each encoder layer l, we introduce four new transformation matrices $\boldsymbol{W}_{\text{q,R},l}, \boldsymbol{W}_{\text{q,L},l}, \boldsymbol{W}_{\text{k,R},l}, \boldsymbol{W}_{\text{k,L},l} \in \mathbb{R}^{\mathsf{heads}\times\mathsf{hdim}\times\mathsf{vec}}$ and alter Equations 1 and 2:

$$\begin{aligned}
\boldsymbol{Q}_l &= \boldsymbol{W}_{\text{k},l} \underset{\mathsf{vec}}{\odot} \boldsymbol{X}_{l-1} \\
&\quad + \left[\left(\boldsymbol{W}_{\text{q,R},l} \underset{\mathsf{vec}}{\odot} \boldsymbol{X}_{l-1}\right) \underset{\mathsf{vtx}}{\odot} \boldsymbol{E}_{\text{R}}\right]_{\mathsf{vtx}'\to\mathsf{vtx}} \\
&\quad + \left(\boldsymbol{W}_{\text{q,L},l} \underset{\mathsf{vec}}{\odot} [\boldsymbol{X}_{l-1}]_{\mathsf{vtx}\to\mathsf{vtx}'}\right) \underset{\mathsf{vtx}'}{\odot} \boldsymbol{E}_{\text{L}} \\
\boldsymbol{K}_l &= \boldsymbol{W}_{\text{q},l} \underset{\mathsf{vec}}{\odot} [\boldsymbol{X}_{l-1}]_{\mathsf{vtx}\to\mathsf{vtx}'} \\
&\quad + \left[\left(\boldsymbol{W}_{\text{k,R},l} \underset{\mathsf{vec}}{\odot} [\boldsymbol{X}_{l-1}]_{\mathsf{vtx}\to\mathsf{vtx}'}\right) \underset{\mathsf{vtx}'}{\odot} \boldsymbol{E}_{\text{R}}\right]_{\mathsf{vtx}\to\mathsf{vtx}'} \\
&\quad + \left(\boldsymbol{W}_{\text{k,L},l} \underset{\mathsf{vec}}{\odot} \boldsymbol{X}_{l-1}\right) \underset{\mathsf{vtx}}{\odot} \boldsymbol{E}_{\text{L}}
\end{aligned}$$

Per adjacency matrix, this alteration first computes a transformation of the input for both the query and key aspects of the self-attention transformation. Multiplying by the adjacency matrix then, for each vertex i, sets the value to be equal to the sum of the values of either i's out-neighbours or i's in-neighbours, depending on the particular term. This serves as a form of message passing along the graph edges, similar to some methods in graph-based neural networks (Gilmer et al., 2017).

3.2.2 Planarity-aware attention

Condition T1 requires that term graph linkages be half-planar. We include planar crossing information in the attention scores $\boldsymbol{S}_l$ in Equation 3 for each vertex pair by subtracting the mean attention score of conflicting vertex pairs. More formally, let $\mathcal{X}_{ij}$ denote the set of vertex pairs between which a link would cross with a link between vertex pair (i, j) in the half-plane above the linearly ordered vertices of the term graph. Then we adjust $\boldsymbol{S}_l$ as follows:

$$\begin{aligned}
\mathcal{X}'_{ij} &= \left\{(k,m) \;\middle|\; \begin{array}{l} k < i < m < j \\ \text{or } i < k < j < m \end{array}\right\} \\
\mathcal{X}_{ij} &= \mathcal{X}'_{ij} \cup \{(m,k) \mid (k,m) \in \mathcal{X}'_{ij}\} \qquad (8) \\
[\mu_{\text{cross}}(\boldsymbol{A})]_{\mathsf{vtx}(i),\mathsf{vtx}'(j)} &= \underset{(k,m)\in\mathcal{X}_{ij}}{\text{mean}}[\boldsymbol{A}]_{\mathsf{vtx}(k),\mathsf{vtx}'(m)} \\
\boldsymbol{S}'_l &= \frac{(\boldsymbol{Q}_l + \boldsymbol{b}_{\text{k},l}) \underset{\mathsf{hdim}}{\odot} \boldsymbol{K}_l + (\boldsymbol{Q}_l + \boldsymbol{b}_{\text{r},l}) \underset{\mathsf{hdim}}{\odot} \boldsymbol{R}_l}{\sqrt{|\mathsf{hdim}|}} \\
\boldsymbol{S}_l &= \boldsymbol{S}'_l - \mu_{\text{cross}}(\boldsymbol{S}'_l)
\end{aligned}$$

3.2.3 Edge filtering

In Equation 5, a mask is applied to the candidate link scores produced by the model to enforce the

category and polarity constraints. We augment this mask in two ways. First, we disallow *necessarily* non-planar links, *i.e.*, links that cannot be in any planar linkage. Penn (2004) defined a context-free grammar for *building* planar linkages; we use this CFG with the inside-outside algorithm (Baker, 1979) to identify whether candidate links each exist in any spanning planar linkage.

Second, we disallow intra-word links, *i.e.*, links between any two vertices that map to the same word. Some of these links are permissible according to the rules of L*, but do not occur in our corpus; moreover, their linguistic utility is unclear. Overall, we expect that these restrictions on allowable links will help reduce the size of the search space, thereby improving system performance.

Inspecting Figure 2 exemplifies how these extra filters can be useful. Disallowing necessarily non-planar links eliminates a candidate link from the NP^+ of "for" to the NP^- of "What" as it would prevent the NP^+ and S^- of "accounts" as well as the NP^- of "the" each from having any planar links. This then implies that there must be a link from the NP^+ of "for" to the NP^- of "the", since that is the only remaining option. Similarly, disallowing the S^+ of "What" from linking to its own S^- immediately implies that it must then link to the S^- of "accounts" while also preemptively preventing a violation of condition T2.

3.3 k-best linkages

While Sinkhorn (with Gumbel noise) provides differentiable sampling of matchings, it has two noteworthy drawbacks. First, it sometimes does not converge to an exact permutation and gets stuck with some values very close to 0.5 (Guigues, 2020), requiring some means of or discretizing such cases (*e.g.*, Gold and Rangarajan, 1996a). This does not pose a potential issue for our parser during training, since $\mathfrak{J}_{\text{NLL}}$ does not require a permutation matrix. During inference, however, the parser needs to be able to produce a discrete result as its output parse.

Second, Sinkhorn makes it difficult at best to retrieve multiple matchings from the distribution. Without Gumbel noise (and with sufficiently small τ), it will converge to the *best* permutation, but one cannot specifically retrieve the second-best (etc.) matchings from this. Sampling (via the addition of Gumbel noise) may yield *multiple* matchings, but there is no guarantee of their overall rank; furthermore, if the input matrix represents a very concentrated distribution, retrieving further matchings may require inordinate sampling rounds.

Since there can be multiple valid parses for a sentence, a parser should ideally be able to return multiple parses if they exist. Moreover, there is no guarantee that the predicted linkage $\boldsymbol{Z}$ in Equation 6 will yield an L*-integral term graph, so it is worthwhile to be able to evaluate alternatives. We therefore use Murty's algorithm (Murty, 1968), a k-best optimal matching algorithm, to produce k candidate linkages from $\boldsymbol{S}$. We stably sort the linkages according to the number of term graph conditions that they violate when combined with the input proof frame, allowing fractional violations of condition T3. Enabling the production of multiple candidate parses also makes it possible for the parser to return the correct parse when it otherwise might not have done so.

3.4 L* structural loss

In contrast to other statistical parsers, the system presented thus far does not have any explicit encoding of the rules of the grammar. Since the negative log-likelihood loss function (Equation 7) is based only on the ground-truth linkage, it is not clear how well the model will be able to generalize and return multiple valid linkages when applicable, rather than linkages most similar to the correct one. We therefore introduce loss function terms that directly encode the term graph validity conditions, and posit that they will help the parser produce linkages that, with the input proof frame, yield an L*-integral term graph. These novel loss functions also enable training our model without ground-truth derivations.

For condition T1, we define the planarity loss function $\mathfrak{J}_{\text{T1}}$ as a function of the post-Sinkhorn matrix $\boldsymbol{Z}$ from Equation 6 so that each link in each pair of crossing links is penalized in proportion to the scores given to the pair:

$$\mathfrak{J}_{\text{T1}}(\boldsymbol{Z}) = \sum_{i,j} \left(\boldsymbol{Z}_{\text{vtx}(i),\text{vtx}'(j)} \sum_{(k,m)\in\mathcal{X}_{ij}} \boldsymbol{Z}_{\text{vtx}(k),\text{vtx}'(m)} \right)$$

where $\mathcal{X}_{ij}$ is defined as in Equation 8. Minimizing $\mathfrak{J}_{\text{T1}}$ then corresponds to minimizing the scores assigned to crossing links.

The remaining loss terms require further computation. Note that conditions T2 and T3 both express constraints on the (non-)existence of certain *regular* paths; the latter is already stated as such while the former can be equivalently restated as barring regular paths from any vertex to itself. Checking

for the existence or absence of paths between two vertices of a graph requires traversing graph edges. As graph traversal corresponds to multiplication by the graph's adjacency matrix, this presents a differentiable means of computing the extent to which a candidate term graph meets conditions T2 and T3.

For an arbitrary weighted graph G with vertices $\mathcal{V}$, denote by $\mathcal{W}_{i,j,n}$ the set of all walks of length n from vertex i to vertex j. By definition, each walk $w \in \mathcal{W}_{i,j,n}$ is a sequence of n edges, *i.e.*, $w = (\langle k_1, l_1\rangle, \langle k_2, l_2\rangle, \ldots, \langle k_n, l_n\rangle), k_1 = i, l_n = j$. Let $\boldsymbol{A}$ denote the adjacency matrix of G; then the matrix power $\boldsymbol{A}^n$ represents the sum (over walks) of walk edge products, *i.e.*, $(\boldsymbol{A}^n)_{i,j} = \sum_{w \in \mathcal{W}_{i,j,n}} \prod_{\langle k,l\rangle \in w} (\boldsymbol{A})_{k,l}$. If edges in G have only positive weights, then $(\boldsymbol{A}^n)_{i,j} = 0$ if and only if there are no walks of length n from vertex i to vertex j. It then follows that $(\sum_{n=1}^{N} \boldsymbol{A}^n)_{i,j} = 0$ if and only if there are no walks of length $\leq N$ from vertex i to vertex j. Since a cycle in G can have length at most $|\mathcal{V}|$, G is therefore acyclic if and only if $(\sum_{n=1}^{N} \boldsymbol{A}^n)_{i,i} = 0 \,\forall i \in \mathcal{V}, N \geq |\mathcal{V}|$.

Returning to term graphs, for condition T2 this means that we can detect regular cycles in a candidate graph with $|\mathcal{V}|$ vertices by constructing an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ from the regular edges of the proof frame together with the candidate linkage, computing $\sum_{n=1}^{|\mathcal{V}|} \boldsymbol{A}^n$, and then verifying that the diagonal entries are all zero. For condition T3, we can similarly inspect the entries corresponding to the parent and child nodes of all Lambek edges and verify that they are all one, indicating that a regular path exists. We can now see how to specify these conditions as loss functions:

$$\boldsymbol{G} = \sum_{n=1}^{|\text{vtx}|} \boldsymbol{A}^n = \sum_{n=1}^{|\text{vtx}|} (\boldsymbol{E}_\text{R} + \boldsymbol{Z})^n$$

$$\mathfrak{J}_\text{T2}(\boldsymbol{G}) = \sum_{i=1}^{|\text{vtx}|} (\boldsymbol{G}_{i,i})^2$$

$$\mathfrak{J}_\text{T3}(\boldsymbol{G}) = \sum_{i=1}^{|\text{vtx}|} \sum_{j=1}^{|\text{vtx}|} (\boldsymbol{E}_\text{L})_{i,j} (\boldsymbol{G}_{i,j} - 1)^2$$

Minimizing $\mathfrak{J}_\text{T2}$ and $\mathfrak{J}_\text{T3}$ correspond to minimizing violations of conditions T2 and T3, respectively.

We refer to the sum of these three loss functions $\mathfrak{J}_{\text{L}^*} = \mathfrak{J}_\text{T1} + \mathfrak{J}_\text{T2} + \mathfrak{J}_\text{T3}$ as the **(L*) structural loss**.

4 Related work

A key aspect of our parser is that it makes use of a structured decomposition of lexical categories in categorial grammars. In this sense, our work follows up on the intuition of recent "constructive" supertaggers, which have been explored for a type-logical grammar (Kogkalidis et al., 2019) and for CCG (Bhargava and Penn, 2020; Prange et al., 2021). Such supertaggers construct categories out of the atomic categories of the grammar; this challenges the classical approach to supertagging, where lexical categories are treated as opaque, rendering the task of supertagging equivalent to large-tagset POS tagging. With this view, it becomes possible for novel categories to be produced; furthermore, the supertaggers are better able to incorporate prediction history and thereby produce grammatical outputs (Bhargava and Penn, 2020).

Recently, Kogkalidis et al. (2020) proposed a system for parsing a "type-logical" grammar that is essentially a modal, non-directional extension of LCG. The Dutch grammar they used is substantially different from our grammar: their connectives are both modal and non-directional; in addition, they have far more atomic categories. While their model is similar to our baseline (Section 3.1), our work here differs substantially in that we incorporate proof-net structural elements and validity conditions, and our system is able to return multiple linkages (Sections 3.2–3.4). Our approach also enables ground-truth–free training.[5]

Lastly, our trained parser operates in polynomial time. Since LCG parsing is NP-complete, our work adds to the body of recent work applying neural networks to NP-hard combinatorial optimization problems to yield polynomial-time approximate solvers (*e.g.*, Li et al., 2018; Gannouni et al., 2020; Sultana et al., 2020; Cappart et al., 2021).

5 Experiments

5.1 Data

We train our models on LCGbank, a semi-automatic conversion of CCGbank to LCG (Fowler, 2016). This conversion necessitated adjusting for instances of CCG's crossing rules that are not permitted in LCG, as well as providing fully categorial parses for the cases in CCGbank where non-categorial rules are used (*e.g.*, unary type-changing).[6] LCGbank also omits features on its categories and includes

[5] Although Kogkalidis et al. (2020) describe their model's training as "end-to-end", their approach is perhaps better described as *joint* training. A truly end-to-end system would allow *differentiation* through the supertagger/proof frame construction, which remains a topic for further investigation.

[6] Refer to (Fowler, 2016) for further conversion details.

Category	Count	Category	Count
NP	1,356,438	,	32
S	563,390	RRB	26
N	419,766	:	16
PP	48,642	.	2
conj	844	LRB	2

Table 1: Counts of atomic categories in LCGbank.

the 274 sentences that were originally excluded from CCGbank. These adjustments substantially increase the number of lexical categories in LCGbank compared to CCGbank. Without the features, CCGbank has 476 unique lexical categories, while LCGbank has 987. Decomposed, the categories yield 10 atomic categories, shown in Table 1.

We follow the CCGbank/PTB tradition of using section 0 for development/validation and section 23 for testing, yielding 1,921 and 2,414 sentences, respectively. For training, however, we use all of the remaining data (sections 1–22 and 24), in contrast to the usual training set for CCGbank (sections 2–21). This is simply to make full use of all available data and yields 44,833 sentences for training.

5.2 Model & training details

We implement our model with PyTorch (Paszke et al., 2019) and PyTorch Lightning (Falcon et al., 2019). Including the truncated top layer, we use three encoder layers (*i.e.*, $L = 3$) with $|\mathsf{vec}| = 384$ and $|\mathsf{heads}| = 4$. We use ReLU activations (Nair and Hinton, 2010) throughout. Parameters are initialized according to PyTorch's defaults. Our transformer uses normalization before each layer rather than after (Wang et al., 2019; Nguyen and Salazar, 2019). To represent the lexical inputs, we use a distilled (Sanh et al., 2019) version of Roberta (Liu et al., 2019) as provided by the Hugging Face Transformers library (Wolf et al., 2020).

For our inside-outside algorithm implementation, we adopt Rush's (2020) overall method for adapting it to GPU matrix operations. We implement the intensive parts of the algorithm as custom CUDA kernels that operate on packed Booleans. We use the fastmurty library (Motro and Ghosh, 2019) for the k-best matchings algorithm.

Training examples are sorted by output sequence length to yield efficient batches; the ordering of the batches is shuffled every epoch. We clip gradients, scaling accordingly, if the sum of gradient norms exceeds 1. We train our models with the AdamW optimizer (Loshchilov and Hutter, 2019; Kingma and Ba, 2014) for 40 epochs, halving the learning rate when performance reaches a plateau with patience of three epochs. We keep the model weights from the epoch with the best development set performance. We report results averaged over three training runs with different random seeds.

We tune hyperparameters with Optuna (Akiba et al., 2019), using the tree-structured Parzen estimator (Bergstra et al., 2011) for sampling and asynchronous successive halving (Karnin et al., 2013; Jamieson and Talwalkar, 2016; Li et al., 2020) for pruning. The initial learning rate and weight decay coefficients are sampled from log-uniform distributions on $[10^{-5}, 10^{-2})$ and $[10^{-7}, 10^{-2})$, respectively. The dropout rate is sampled from a uniform distribution on $[0, 0.6)$. We also use dropout on the input lexical tokens, sampled uniformly on $[0, 0.1)$.

5.3 Experimental conditions and evaluation

We evaluate four conditions: (1) the baseline model (Section 3.1) trained only with $\mathfrak{J}_{\text{NLL}}$; (2) the improved model (Section 3.2) trained only with $\mathfrak{J}_{\text{NLL}}$; (3) the improved model trained with both $\mathfrak{J}_{\text{NLL}}$ and $\mathfrak{J}_{\text{L}^*}$; and (4) the improved model trained only with $\mathfrak{J}_{\text{L}^*}$. The latter condition is trained without ground truth while the others are trained with it. Since comparing the two cases would be unfair (especially on a measure such as sentence accuracy), we evaluate them separately. To evaluate the effect of allowing k-best linkages (Section 3.3), we evaluate all conditions with both $k = 1$ and $k = 512$. Note that with $k = 1$, our baseline model is similar in design to that of Kogkalidis et al. (2020), with minor differences such as model sizes and vector embedding details; this represents the closest point of comparison while controlling for our other model aspects as well as our grammar and corpus.

We evaluate our parser using four measures: (1) **link accuracy**, the percentage of positive vertices that were assigned their correct negative vertex; (2) **sentence accuracy**, the percentage of sentences with 100% link accuracy; (3) **coverage**, the percentage of sentences for which an L*-integral linkage was found; and (4) the average number of unique parses (*i.e.*, L*-integral) found per sentence.

When used for computing $\mathfrak{J}_{\text{NLL}}$, we use Sinkhorn temperature $\tau = 0.01$. We use a separate temperature parameter τ_{L^*} when computing $\mathfrak{J}_{\text{L}^*}$. The intuition behind this is that because $\mathfrak{J}_{\text{NLL}}$

	Condition	LAcc	SAcc	Cov	Parses
$k = 1$	Baseline	97.7	86.2	97.3	—
	+TG	97.9	87.4	98.4	—
	+TG+$\mathfrak{J}_{\text{L}^*}$	97.9	87.2	98.7	—
$k = 512$	Baseline	97.9	87.7	99.8	5.9
	+TG	98.0	88.2	99.96	5.7
	+TG+$\mathfrak{J}_{\text{L}^*}$	98.0	87.8	99.96	5.8

Table 2: Link accuracies (LAcc), sentence accuracies (SAcc), coverage (Cov), and average number of parses per sentence for the $\mathfrak{J}_{\text{NLL}}$-trained systems on the LCGbank test set. +TG refers to the model changes of Section 3.2. +TG+$\mathfrak{J}_{\text{L}^*}$ includes the model changes and is trained with both $\mathfrak{J}_{\text{NLL}}$ and $\mathfrak{J}_{\text{L}^*}$. All values are averages over three random seeds.

permits one specific parse while $\mathfrak{J}_{\text{L}^*}$ permits multiple parses, they may require different levels of uncertainty in their corresponding doubly-stochastic matrices. We treat τ_{L^*} as a hyperparameter with initial values sampled log-uniformly on $[0.01, 10)$.

For the condition that includes both $\mathfrak{J}_{\text{NLL}}$ and $\mathfrak{J}_{\text{L}^*}$, we linearly combine the two to obtain the final objective function $\mathfrak{J} = \alpha\,\mathfrak{J}_{\text{NLL}} + (1-\alpha)\,\mathfrak{J}_{\text{L}^*}$. We tune α as a hyperparameter as well, with initial uniform sampling on $[0.05, 0.95)$.

5.4 Results

5.4.1 Training with ground truth

Table 2 shows the performance of the systems trained against gold-standard linkages. Evaluating multiple linkages from the single score matrix $\boldsymbol{S}$ is clearly beneficial on all accounts. In particular, doing so yields almost complete coverage for all cases, but especially for our two improved versions. The accuracies improve as well; since our sorting of multiple candidates linkages is stable, the improvements to sentence accuracy come from cases where the correct parse was scored higher than other valid parses, but lower than some invalid parses. Here, filtering the list using the term graph validity conditions is clearly useful.

Incorporating term graph structure in the model as described in Section 3.2 improves performance as well, though not by as much as evaluating multiple linkages. While we expected the number of parses per sentence found by the parser to increase due to the presence of grammatico-structural information, in fact it returned fewer parses. With $\mathfrak{J}_{\text{NLL}}$ as the sole training objective, the model uses this extra information solely to increase its performance as measured by that objective. Interestingly, adding $\mathfrak{J}_{\text{L}^*}$ to the model improvements seems to *decrease* accuracy, nearly to the baseline's level for the $k = 512$ case. Coverage remains high, however. In this case, we believe that the two training objectives are somewhat conflicting, with $\mathfrak{J}_{\text{NLL}}$ pushing the model towards the correct linkage but $\mathfrak{J}_{\text{L}^*}$ equally preferring other valid linkages.

5.4.2 Training without ground truth

Training a model without ground-truth linkages impairs system performance substantially, as expected: the model has no signal guiding it to the correct linkage, nor differentiating the correct linkage from other valid ones. With $k = 1$, the system achieves 91.2% coverage on the LCGbank test set.

With $k = 512$, this increases substantially to 96.2%. Here the parser finds an average of 5.9 parses/sentence. Since it did not find a single valid parse for 3.8% of sentences, the number of parses found for *covered* sentences is 6.2. This is further in line with the idea that $\mathfrak{J}_{\text{L}^*}$ "pulls" the model away from the correct parse in the direction of other (valid) parses.

Since the loss function cannot distinguish correct linkages from other valid ones, this configuration cannot be expected to select the correct linkage. Nonetheless, the correct parse appears in the system's set of output parses for 79.0% of sentences, appearing at the top (*i.e.*, the correct sentence is given the highest score) for 53.4% of sentences with $k = 1$ and 54.9% of sentences with $k = 512$.

5.5 Analysis

Finally, we conduct a post-hoc ablation study for the ground-truth–free condition. For each ablated as, we adjust the model or loss function accordingly, and then retrain the model from scratch using the same hyperparameters as the original model. Table 3 shows the results, comparing coverage of the ablated versions with that of the original.

We see that removing all planarity information (*i.e.*, the link filtering, the planarity-aware attention, and the planarity loss term $\mathfrak{J}_{\text{T1}}$) is disastrous; this condition has by far the largest drop in coverage. This is especially notable as LCG proof nets must be half-planar due to the non-commutativity of L*; this useful constraint is not present in type-logical grammars that do not have this property, such as that employed by Kogkalidis et al. (2020).

Other decreases range from moderate to small:

Condition	$k = 1$	$k = 512$
+TG+$\mathfrak{J}_{L^*}-\mathfrak{J}_{NLL}$	91.2	96.2
$-\mathfrak{J}_{T1}$	84.5	95.1
$-\mathfrak{J}_{T2}$	72.9	92.9
$-\mathfrak{J}_{T3}$	70.6	93.8
−RL	89.0	95.9
−IW	81.1	91.0
−IW−NP	73.9	85.6
−RL−IW−NP−PA	74.9	90.7
−IW−NP−PA$-\mathfrak{J}_{T1}$	19.2	44.7

Table 3: LCGbank test set coverage under various ground-truth–free training conditions. $-\mathfrak{J}_x$ removes loss term x; −RL removes regular and Lambek edges; −IW removes the filter on intra-word links; −NP removes the filter on non-planar links; −PA removes planarity-aware attention. In contrast to Table 2, here the ablated versions (all but the first line) are results from one single training run each.

- All three loss terms are important, with coverage decreasing notably upon ablation; the decrease is lowest for $\mathfrak{J}_{T1}$, suggesting that its removal is partially ameliorated by the other sources of planarity information in the model.
- Removing the regular and Lambek edge information decreases coverage by a small amount.
- Filtering out intra-word links is surprisingly important; we had suspected that, since the model has information about which words are the same for given atomic category pairs, it would learn to avoid them. If the filter on non-planar links is also removed, coverage drops further. Removing planarity-aware attention and the proof frame edge information (*i.e.*, stripping down to the baseline system of Section 3.1, but here training with the structural loss only) strangely slightly *restores* coverage.

6 Conclusion and future work

We have presented an LCG parser with multiple novel techniques, including neural term graph structure and structural constraint encodings, novel loss functions derived from LCG term graph validity conditions, and a self-attention–based system for returning and efficiently evaluating k-best matchings. Evaluating on a corpus of English LCG proof nets, we found our improvements to be effective, especially the k-best matchings. Our loss functions, furthermore, enable training an LCG parsing model without ground-truth derivations or linkages. Analysis shows that planarity conditions are especially important, but that all of our alterations contribute to the parser's improved performance.

As we saw in Table 2, combining $\mathfrak{J}_{NLL}$ and $\mathfrak{J}_{L^*}$ seems to be detrimental to parser accuracy. The two loss terms have seemingly conflicting objectives, with the former concentrating probability mass around a single solution and the latter spreading probability mass over multiple solutions. We believe it would be worthwhile to explore combining these two in a more congruent manner.

Since our model allows differentiating through the structure of lexical categories, the obvious next step is to incorporate a supertagger and pass gradients down to it. As it stands, supertaggers have rudimentary knowledge of their context, with no notion of how the atomic subcategories of one category might combine with those of another. A tight coupling of the techniques we proposed here with an appropriately designed supertagger would yield a true end-to-end differentiable LCG parser.

Lastly, we believe further investigation of structural constraints and objectives to be promising. Although we still relied on supertags from the corpus, our results with the grammatico-structural loss functions demonstrate the training of a high-coverage parser with a decreased annotation burden. Techniques such as those presented here suggest a potential path to parsing with lower data requirements, or perhaps even to structured, formalism-driven unsupervised parsing.

Acknowledgements

We thank Elizabeth Patitsas as well as our anonymous reviewers for their feedback. This research was enabled in part by support provided by NSERC, SHARCNET, and Compute Canada.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 2623–2631, New York, NY, USA. Association for Computing Machinery.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. In *arXiv:1607.06450 [Cs, Stat]*.

J. K. Baker. 1979. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, volume 24, pages 2546–2554. Curran Associates, Inc.

Aditya Bhargava and Gerald Penn. 2020. Supertagging with CCG primitives. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 194–204, Online. Association for Computational Linguistics.

Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. 2021. Combinatorial optimization and reasoning with graph neural networks. *arXiv:2102.09544 [cs, math, stat]*.

David Chiang, Alexander M. Rush, and Boaz Barak. 2021. Named Tensor Notation. *arXiv:2102.13196 [cs]*.

Stephen Clark and James R. Curran. 2007. Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33(4):493–552.

John Cocke and J. T. Schwartz. 1970. Programming Languages and Their Compilers: Preliminary Notes, Second Revised Version. Technical report, Courant Institute of Mathematical Sciences, New York University.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

William A. Falcon, Jirka Borovec, Adrian Wälchli, Nic Eggert, Justus Schock, Jeremy Jordan, Nicki Skafte, Ir1dXD, Vadim Bereznyuk, Ethan Harris, Tullie Murrell, Peter Yu, Sebastian Præsius, Travis Addair, Jacob Zhong, Dmitry Lipin, So Uchida, Shreyas Bapat, Hendrik Schröter, Boris Dayma, Alexey Karnachev, Akshay Kulkarni, Shunta Komatsu, Martin.B, Jean-Baptiste Schiratti, Hadrien Mary, Donal Byrne, Cristobal Eyzaguirre, cinjon, and Anton Bakhtin. 2019. PyTorch Lightning.

Timothy A. D. Fowler. 2009. Term Graphs and the NP-Completeness of the Product-Free Lambek Calculus. In *Formal Grammar*, pages 150–166, Berlin, Heidelberg. Springer Berlin Heidelberg.

Timothy A. D. Fowler. 2010. A Polynomial Time Algorithm for Parsing with the Bounded Order Lambek Calculus. In *The Mathematics of Language*, Lecture Notes in Computer Science, pages 36–43, Berlin, Germany. Springer.

Timothy A. D. Fowler. 2016. *Lambek Categorial Grammars for Practical Parsing*. Ph.D. thesis, University of Toronto.

Aymen Gannouni, Vladimir Samsonov, Mohamed Behery, Tobias Meisen, and Gerhard Lakemeyer. 2020. Neural combinatorial optimization for production scheduling with sequence-dependent setup waste. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2640–2647, Toronto, Canada. IEEE.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *ICML'17*, pages 1263–1272, Sydney, NSW, Australia. JMLR.org.

Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science*, 50(1):1–101.

Steven Gold and Anand Rangarajan. 1996a. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388.

Steven Gold and Anand Rangarajan. 1996b. Softassign versus softmax: Benchmarks in combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 8. MIT Press.

Laurent Guigues. 2020. Concerning Iterative Graph Normalization and Maximum Weight Independent Sets. *arXiv:2012.07764 [cs, math]*.

Kevin Jamieson and Ameet Talwalkar. 2016. Non-stochastic Best Arm Identification and Hyperparameter Optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 240–248, Cadiz, Spain. PMLR.

Zohar Karnin, Tomer Koren, and Oren Somekh. 2013. Almost Optimal Exploration in Multi-Armed Bandits. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1238–1246, Atlanta, USA. PMLR.

T. Kasami. 1966. An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages. Technical Report R-257, University of Illinois Coordinated Science Laboratory.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference for Learning Representations*, San Diego, USA.

Konstantinos Kogkalidis, Michael Moortgat, and Tejaswini Deoskar. 2019. Constructive Type-Logical Supertagging With Self-Attention Networks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 113–123, Florence, Italy. Association for Computational Linguistics.

Konstantinos Kogkalidis, Michael Moortgat, and Richard Moot. 2020. Neural Proof Nets. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 26–40, Online. Association for Computational Linguistics.

J.J. Kosowsky and A.L. Yuille. 1994. The invisible hand algorithm: Solving the assignment problem with statistical physics. *Neural Networks*, 7(3):477–490.

Marco Kuhlmann and Giorgio Satta. 2014. A New Parsing Algorithm for Combinatory Categorial Grammar. *Transactions of the Association for Computational Linguistics*, 2:405–418.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG Parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231, San Diego, California. Association for Computational Linguistics.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. 2020. A System for Massively Parallel Hyperparameter Tuning. In *Proceedings of Machine Learning and Systems*, volume 2, pages 230–246, Austin, USA.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *Proceedings of the 7th International Conference on Learning Representations*.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. 2018. Learning Latent Permutations with Gumbel-Sinkhorn Networks. In *Proceedings of the 6th International Conference on Learning Representations*, Vancouver, Canada.

Michael Motro and J. Ghosh. 2019. Scaling data association for hypothesis-oriented MHT. In *22nd International Conference on Information Fusion (FUSION)*, pages 1–8, Ottawa, Canada. IEEE.

Katta G. Murty. 1968. An Algorithm for Ranking all the Assignments in Order of Increasing Cost. *Operations Research*, 16(3):682–687.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, Haifa, Israel. Omnipress.

Toan Q. Nguyen and Julian Salazar. 2019. Transformers without Tears: Improving the Normalization of Self-Attention. In *Proceedings of the 16th International Workshop on Spoken Language Translation 2019*, Hong Kong. Zenodo.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, volume 32, pages 8026–8037. Curran Associates, Inc.

Gerald Penn. 2004. A Graph-Theoretic Approach to Sequent Derivability in the Lambek Calculus. *Electronic Notes in Theoretical Computer Science*, 53:274–295.

Mati Pentus. 1997. Product-Free Lambek Calculus and Context-Free Grammars. *The Journal of Symbolic Logic*, 62(2):648–660.

Mati Pentus. 2006. Lambek calculus is NP-complete. *Theoretical Computer Science*, 357(1-3):186–201.

Jakob Prange, Nathan Schneider, and Vivek Srikumar. 2021. Supertagging the Long Tail with Tree-Structured Decoding of Complex Categories. *Transactions of the Association for Computational Linguistics*, 9:243–260.

Dirk Roorda. 1992. Proof Nets for Lambek Calculus. *Journal of Logic and Computation*, 2(2):211–231.

Alexander Rush. 2020. Torch-Struct: Deep Structured Prediction Library. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 335–342, Online. Association for Computational Linguistics.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. In *arXiv:1910.01108 [Cs]*, Vancouver, Canada.

Yury Savateev. 2012. Product-free Lambek calculus is NP-complete. *Annals of Pure and Applied Logic*, 163(7):775–788.

Richard Sinkhorn and Paul Knopp. 1967. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348.

Miloš Stanojević and Mark Steedman. 2020. Max-Margin Incremental CCG Parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4111–4122, Online. Association for Computational Linguistics.

Nasrin Sultana, Jeffrey Chan, A. K. Qin, and Tabinda Sarwar. 2020. Learning to Optimise General TSP Instances. *arXiv:2010.12214 [cs, math]*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning Deep Transformer Models for Machine Translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

Wojciech Zielonka. 1981. Axiomatizability of Ajdukiewicz-Lambek Calculus by Means of Cancellation Schemes. *Mathematical Logic Quarterly*, 27(13-14):215–224.