An in-depth look at Euclidean disk embeddings for structure preserving parsing

Federico Fancellu* **Lan Xiao*†** **Allan Jepson** **Afsaneh Fazly**
Samsung AI Center, Toronto
{federico.f, allan.jepson, a.fazly}@samsung.com

Abstract

Preserving the structural properties of trees or graphs when embedding them into a metric space allows for a high degree of interpretability, and has been shown beneficial for downstream tasks (e.g., hypernym detection, natural language inference, multimodal retrieval). However, whereas the majority of prior work looks at using structure-preserving embeddings when *encoding* a structure given as input, e.g., WordNet (Fellbaum, 1998), there is little exploration on how to use such embeddings when *predicting* one. We address this gap for two structure generation tasks, namely dependency and semantic parsing. We test the applicability of *disk embeddings* (Suzuki et al., 2019) that has been proposed for embedding Directed Acyclic Graphs (DAGs) but has not been tested on tasks that generate such structures. Our experimental results show that for both tasks the original disk embedding formulation leads to much worse performance when compared to non-structure-preserving baselines. We propose enhancements to this formulation and show that they almost close the performance gap for dependency parsing. However, the gap still remains notable for semantic parsing due to the complexity of meaning representation graphs, suggesting a challenge for generating interpretable semantic parse representations.

1 Introduction

Numerous studies in NLP have focused on embedding linguistic elements into metric spaces, where instances are represented as vectors whose geometric distance reflects the semantic similarity among instances (Mikolov et al., 2013; Baroni et al., 2014; Pennington et al., 2014, *inter alia*). More recently, some have gone beyond embedding words and sequences, and explored the encoding of a hierarchy (e.g., WordNet, Fellbaum, 1998) through modelling its partial order structure (Vendrov et al., 2015; Lai and Hockenmaier, 2017; Vilnis et al., 2018). Consequently, given the size and depth of such structures, attention has shifted to geometric spaces (mostly hyperbolic) that could better represent order and containment relations (Nickel and Kiela, 2017; Ganea et al., 2018; Dong et al., 2018; Suzuki et al., 2019). Methods to embed elements of hierarchies are *structure-preserving*, and therefore interpretable, in that the relative position in the embedding space reflects the relation in the original hierarchy (e.g., parent–child relation). Most of these methods have been shown to be beneficial not only on tasks pertinent to the encoded hierarchy itself (e.g., hyponymy relations), but on downstream tasks including multimodal retrieval (Vendrov et al., 2015) and video understanding (Surís et al., 2021).

However, whereas there is a plethora of studies looking at preserving structure while *encoding* an hierarchy given as input, there is little to no exploration on how to do this while *predicting* one. In this work, we start one such exploration with the quintessential structure generation task in NLP: parsing. Given an input sentence, e.g., 'Anna asked Mary to stop', parsing is the task of transducing a natural language string into a structured linguistic representation (e.g., the AMR graph in Fig. 1(a)) that encodes either syntactic or semantic properties of the string. Recent neural network based approaches have achieved state-of-the-art performance while being able to generalize both across frameworks, as well as across trees and graphs (Zhang et al., 2019; Lindemann et al., 2020; Ozaki et al., 2020; Samuel and Straka, 2020; Procopio et al., 2021, *inter alia*). However, not much can be said about the representations these parsers learn since the parsers are not explicitly trained to preserve any of the geometric properties of their output structure, and as such are not interpretable. We believe that moving to interpretable, structured rep-

*Equal Contribution
†Now at Pinterest

Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, pages 283–295

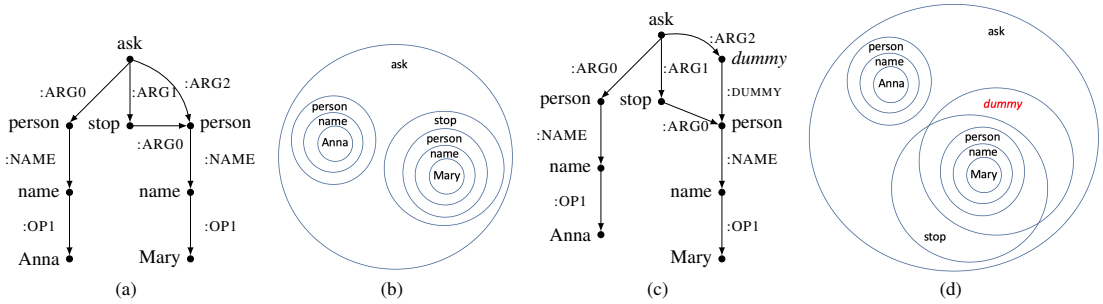


Figure 1: Abstract Meaning Representation (AMR) graph for the sentence 'Anna asked Mary to stop' (a), with a transitive closure between the nodes 'ask' and 'person'; however, given the corresponding disk embedding representation in (b), we cannot reconstruct such a relation. Therefore, we introduce a *dummy* node in every transitive closure (c) so that graph relations and disk embedding containment are bijective (d).

resentations would allow for a better diagnostic of what is learnt by a parser, while at the same time laying the foundations to connect 'deep' natural language understanding to other tasks, especially in the multimodal domain.

How can we build such interpretable representations? Does interpretability impact performance? Kádár et al. (2021) have attempted to answer these questions in the context of parsing into dependency *trees* by means of a structure-preserving loss that forces embedding distances to be isometric to tree distances. Whereas Kádár et al. show that a structure-preserving embedding leads to comparable performance with blackbox methods, their method does not generalize to graphs. This is of particular relevance to semantic parses that are often DAGs, and where it is unclear how to isometrically embed multiple paths between a pair of nodes.

To this end, we turn to a method that allows us to model transitive asymmetrical relations expressible as a DAG: *disk embeddings* (Dong et al., 2018; Suzuki et al., 2019). Disk embeddings represent DAGs as a series of concentric disks, each defined by a center vector and a radius (as the ones in Fig 1(b)), and have been shown to outperform other methods when encoding hypernym relations in the WordNet hierarchy. However, it is not clear whether such a method transfers well to more complex architectures where embeddings are contextualized given an input sentence, and structure prediction often interacts with predicting other elements of the tree or graph (node label, edge label, etc.).

In summary, our work makes the following contributions:

Disk embedding losses for tree and graph generation in parsing: we found the disk embedding loss formulation of Suzuki et al. (2019) to be sub-optimal w.r.t. parsing performance. We found that simply adding a positive margin already provides a large boost in performance, but the best performance is obtained when an auxiliary loss that considers local neighbourhood relations is added, as well as when parent-child relations are oversampled.

Interpretability in parsing (though at the cost of performance): through a comparison with non-interpretable approaches, we found that whereas for dependency trees the price to pay in terms of performance for interpretability is small, for semantic graphs the gap is higher, highlighting where future work should focus its efforts. Importantly, we found that most semantic parsing errors are local and specific to the parser we use, where the lack of explicit alignment between words and graph nodes poses a challenge for disk embeddings, especially in the case of named entity substructures.

2 Disk Embeddings: Background

As discussed above, disk embeddings (Dong et al., 2018; Suzuki et al., 2019) provide an interpretable model for transitive asymmetrical relations (i.e., partially ordered sets or *posets*), such as those represented by a DAG. They are a general framework that allows to embed posets in a (quasi-)metric space. Let's define (X, d) as a quasi-metric space with distance d and a *closed disk* $D(\mathbf{x}, r) = \{p \in X \mid d(p, \mathbf{x}) \leq r\}$, where $\mathbf{x}$ is the *center* and r the radius. We can express the containment relationship for two disks[1] as

$$D(\mathbf{c}_i, r_i) \supset D(\mathbf{c}_j, r_j) \iff d(\mathbf{c}_i, \mathbf{c}_j) < r_i - r_j. \quad (1)$$

[1]Even though multi-dimensional disks are technically balls, we will refer to them as disks throughout the paper.

Given a set of such disks, the ordering provided by this subset relationship provides a poset.[2]

Disk embeddings seek to maintain *order isomorphism* between the posets of a graph G (X_G, $\preceq_G$) and their disk embedding representation (X_ϕ, $\preceq_\phi$). Such an isomorphism exists if there is a bijective function $f : X_G \rightarrow X_\phi$ such that $x_G \preceq y_G \iff x_\phi \preceq y_\phi$. In our case the bijective function f is modelled via a neural network architecture introduced in § 3.

To achieve isomorphism, we make use of the *protrusion* ℓ_{ij} of disk $x_j = D(\mathbf{c}_j, r_j)$ with respect to disk $x_i = D(\mathbf{c}_i, r_i)$ as the degree of containment:

$$\ell_{ij} = d(\mathbf{c}_i, \mathbf{c}_j) - r_i + r_j. \tag{2}$$

It follows from Eq. (1) that $\ell_{ij} < 0$ (*negative* protrusion) if and only if $x_j \subset x_i$. Moreover, the protrusion ℓ_{ij} provides a continuous measure of the degree of containment. Specifically it equals the maximum signed distance of points in disk $D(\mathbf{c}_j, r_j)$ from the boundary of disk $D(\mathbf{c}_i, r_i)$. Here $\ell_{ij} < 0$ indicates that $D(\mathbf{c}_j, r_j)$ is entirely contained inside $D(\mathbf{c}_i, r_i)$ (and, indeed, with a margin equal to $-\ell_{ij}$). While $\ell_{ij} > 0$ indicates that some point in $D(\mathbf{c}_j, r_j)$ is outside $D(\mathbf{c}_i, r_i)$ by a distance equal to ℓ_{ij}.

Recovering DAGs from disk embeddings. We wish to be able to represent a DAG with a disk embedding and also be able to recover the edges of that DAG from its disk embedding. However, recovering the edges of the original DAG is not always possible. To see this, suppose we have a disk embedding formed by a 1-1 mapping of the nodes of a DAG to disks.[3] Moreover, suppose the mapping is an order isomorphism, so the partial ordering is preserved.

It is natural to consider an edge from disk x_i to disk x_j if and only if $x_j \subset x_i$ and there is no other intervening node (i.e., there is no x_k such that $x_j \subset x_k \subset x_i$). This process recovers a DAG with the same partial ordering as the original DAG, but it may be missing edges. This issue occurs, for example, for the disk embedding shown in Fig. 1b, which represents the same partial ordering as the DAG in Fig. 1a. However, in decoding this disk embedding, we would not decode the edge from 'ask' to 'person', since 'stop' is an intervening node.

We remedy this problem by adding as many *dummy* nodes as necessary to the original DAG (as illustrated in Fig. 1c and Fig. 1d). Specifically, for any edge (n_i, n_j) in the original DAG that can be removed without changing the partial ordering, we create a new *dummy* node m_{ij}, and replace that edge by the two edges (n_i, m_{ij}) and (m_{ij}, n_j). Every edge in the modified graph is then required to reproduce the implied partial ordering, and this modified DAG is therefore recoverable from an order isomorphism.

3 Disk Embedding: Model

The disk embedding module takes as input the hidden representations of a sentence encoder (LSTM or transformer); we discuss how these representations are obtained in the context of dependency and semantic parsing in § 4.1 and 4.2 respectively. This input undergoes a linear transformation followed by a $\mathrm{LeakyReLU}$ activation to obtain X_g, that is then used to learn disk embeddings.

To learn the centers C and the radii R of the disks, we pass X_g through a n-layer MLP (where n=2 in our case) as follows:

$$C = f_a(W_c^{(n)}(X_g^{(n-1)}) + b_c) \tag{3a}$$

$$R = f_a(W_r^{(n)}(X_g^{(n-1)}) + b_r) \tag{3b}$$

where $X_g^{(n-1)}$ are the input representations from the previous layer (X_g^0 is the input representation X_g), $W^{(n)}$ are the weights for the nth MLP layer, b_c and b_r are the biases, and f_a is a non-linearity activation function ($\mathrm{LeakyReLU}$ in our case).

Centers and radii C and R are used to compute the *protrusion* values ℓ_{ij} as in Eq. 2. We then use the ℓ_{ij} values in a contrastive loss with a margin α as follows:

$$\mathcal{L}_c = \sum_{(i,j)\in\mathcal{P}} [\ell_{ij}]_+ + \sum_{(i,j)\in\mathcal{N}} [\alpha - \ell_{ij}]_+ \tag{4}$$

where [x]$_+$ is the function $\max$(0,x) and $\mathcal{P}$ and $\mathcal{N}$ represent the sets of positive and negative pairs, respectively, where positives are pairs of nodes in an ancestor–descendant relation, and negatives include all other pairs of nodes. Recall that $\ell_{ij} < 0$ if node i is the ancestor of j, and > 0 otherwise, hence for the positive samples we only incur in a loss if the $\ell_{ij} > 0$.

Given that we have access to both positive and negative instances (node pairs in an ancestor–descendant relationship vs. all the rest), we can

[2]The manner in which individual edges in a DAG are represented in a disk embedding is more subtle and is discussed later.

[3]Note that disk embeddings must necessarily be acyclic due to the subset relations.

also formulate a Maximum Likelihood Estimation (MLE) binary cross-entropy loss, where we directly maximize the probability of a pair of nodes i and j to be either in an ancestor–descendant relationship or not.

$$a_{ij} = \text{sigmoid}(-\ell_{ij}), \quad (5)$$

$$\mathcal{L}_{MLE} = -\sum_{\substack{i,j \in V, \\ i \neq j}} \varrho_{ij} \log a_{ij} + (1 - \varrho_{ij}) \log(1 - a_{ij}), \quad (6)$$

where $a_{i,j}$ is the probability of i and j being in an ancestor–descendant relationship, and is obtained by applying a sigmoid function to ℓ_{ij} over two classes (ancestor–descendant vs. other). ϱ_{ij} is an indicator function that is 1 if i and j are in an ancestor–descendant relationship.

Structure auxiliary loss. A contrastive or MLE loss optimizes a *global* pair-wise containment relationship between nodes; however, this might come at the expense of *local* relations, which we still seek to preserve. In order to incorporate information about local neighbourhood structure, we propose a (local) structure-preserving loss. The goal of this loss is to maximize the probability of correctly predicting the local relation between two nodes out of a class of 6 relations, namely, $\mathcal{C} = \{$parent, grandparent, child, grandchild, sister, other$\}$, where 'other' represents all relations with a path length greater than 2.[4] To predict these relations we start by passing X_g through a biaffine transform to obtain s_{ij}; this is the same as the biaffine function used in dependency and semantic parsing for edge prediction and since our parsers are two such systems, we reuse their implemented biaffine function. During training, we seek to predict the correct relationship between all pairs of nodes using a cross-entropy loss. Computation is as follows:

$$X_b^1, X_b^2 = \text{ReLU}(W_b^{(1)} X_g + b_b^{(1)}) \quad (7)$$

$$s_{ij} = X_b^{1T} U X_b^2 + W_b^{(2)} X_b^1 + W_b^{(3)} X_b^1 + b_b^{(2)} \quad (8)$$

$$\mathcal{L}_{struc} = \sum_{\substack{i,j \in V, \\ i \neq j}} -\log(\text{softmax}(W_b^{(4)} s_{ij} + b_b^{(3)})) \quad (9)$$

where X_b^1, X_b^2 are separate representations for ancestors and descendants respectively, whose dimensionality is half that of X_g, $W_b^{(n)}$ and U are the weight and b_b the bias. s_{ij} is the output of the biaffine transform, on top of which a linear followed by a softmax transform are applied to obtain the probability for the nodes i and j having the relation class $c \in \mathcal{C}$.

When used, the structure auxiliary loss is added to either the constrastive or the MLE loss to obtain the overall loss for a sentence, normalized by a factor T. We use the number of node pairs as T as opposed to the number of nodes, since we observe that the former achieves consistently better performance.

$$\mathcal{L} = \frac{1}{T}(\mathcal{L}_{c/MLE} + \mathcal{L}_{struc}) \quad (10)$$

Decoding into parent–child relations. At test time we recover the parent–child relations from the protrusion values ℓ_{ij} by first using Eq. 5 to get ancestor probabilities a_{ij}. Using this equation, negative protrusion values are transformed to high ancestor probabilities (≥ 0.5) and positive ones to low probabilities (< 0.5). We then identify the direct parent of each node as the ancestor with no intervening node:

$$p_{ij} = a_{ij}[1 - \max_{k \in V \setminus \{i,j\}}[a_{ik} a_{kj}]] \quad (11)$$

Note that for dependency parsing the parent–child probabilities p_{ij} are fed as input to a Maximum Spanning Tree (MST) decoding algorithm to obtain the final tree.

4 Parsing Systems

Our baseline dependency and semantic parsing both determine edge presence between nodes using the biaffine formulation of Dozat et al. (2017) (see Eq. 7-9), which predicts the most likely parent for each node, along with the grammatical relation between each pair of head and dependent, conditioned on an encoded hidden representation. We replace this with the disk embedding module described in § 3, leaving all other modules unchanged.

4.1 Dependency parsing

We use the dependency parser of Kádár et al. (2021),[5] which has been shown to perform on par with SOTA systems while relying on structure-preserving, interpretable representations. To generate the hidden representations $h_1, h_2 ... h_{|S|}$ for a given sentence S=$w_1 ... w_{|S|}$, a highway-BiLSTM encoder (Srivastava et al., 2015) takes as input a sequence of $|S|$ embeddings $\mathbf{x}_1, \ldots, \mathbf{x}_{|S|}$, where

[4] We also experimented with maximizing the probability of parent–child relations vs. others ($|\mathcal{C}| = 2$) as well as including sisters relations ($|\mathcal{C}| = 3$) but found that a 6-way classification consistently resulted in the best performance.

[5] The parser of Kádár et al. (2021) is based on the parser of (Qi et al., 2018) whose codebase is provided at `https://github.com/stanfordnlp/stanfordnlp`.

each $\mathbf{x}_i$ is a concatenation of word-level, character-level, part-of-speech and morphological feature embeddings for w_i. We use pre-trained word2vec (Mikolov et al., 2013) and fastText embedding (Bojanowski et al., 2017) to initialize word embeddings, whereas the remaining embeddings are trained from scratch. A BiLSTM decoder then applies an MLP on top of the encoded representation to generate the input to either the biaffine or the disk embedding module.

4.2 Semantic parsing

Unlike dependencies, there is no one-to-one correspondence between words in a sentence and nodes in an AMR graph. A common solution is to use an auto-regressive parser (e.g., Zhang et al., 2019; Bevilacqua et al., 2021) that decodes nodes following an arbitrary graph linearization. However, semantic parses, just like dependency trees, are inherently orderless, which is why we opted for a non-autoregressive parser that predicts all nodes in parallel at once.[6] We use the PERIN parser (Samuel and Straka, 2020) to parse sentences into AMR graphs.[7] The parser generates graphs in two steps: first, a transformer encoder followed by a transformer decoder take pre-trained XLM-R embeddings (Conneau et al., 2019) as input to generate the hidden representations $h_1 ... h_{|S|}$.

Unlike dependency parsing, the alignment between words and nodes in a graph is missing, so the cross-entropy loss w.r.t. node labels cannot be computed directly. The alignments have to be bijective (one hidden state to one node only) but should still accommodate for many-to-many correspondences, as in the case of named entities (e.g., 'Mary') that are mapped to entire subgraphs (e.g., person → name → Mary). To meet both conditions, each h_i is transformed into k representations via a function $\phi : h_i \rightarrow \hat{y}_i$ with $\hat{y}_i \in \mathbb{R}^{dk}$. The parser finds the best alignment between the set of vectors $\hat{\mathcal{Y}} = \{\hat{y}_i\}$, and the set of target nodes $\mathcal{Y}$ by scoring all permutations $\Pi(\hat{\mathcal{Y}})$ and selecting the one, $\pi*$, that maximizes the probability of a vector $\hat{y}_i$ to correspond to the label of a node y, i.e., $p(y^{label})$; see Eq. 12. Note that we require the two sets $\mathcal{Y}$ and $\hat{\mathcal{Y}}$ to be the same size, and as such, we extend the set $\mathcal{Y}$ of target nodes with NULL tokens; in practice this means that the words aligned to NULL are dropped.

$$\pi^* = \arg\max_{\pi \in \Pi} \sum_{i=1}^{|S| \times k} \mathbb{1}_{[y^{label} \neq \text{NULL}]} p(y^{label}_{\pi(i)} | \hat{y}_i; \theta) \quad (12)$$

where θ are the model parameters. Given a permutation π^*, the parser then computes the weighted sum of five different losses: the node label, the edge presence, the edge label, the property (see below), and the top node loss. For more detail, we refer the reader to the original paper (Samuel and Straka, 2020).

In our work, we solely focus on the edge presence classifier, that is a biaffine function followed by a cross-entropy loss, as stated at the beginning of this section. Note that there could be multiple nodes with the same label; in particular, this is the case for *properties* that are subgraphs describing named entities containing the same semantic constants (e.g., 'person' and 'name' for the named entities 'Anna' and 'Mary' in Fig. 1). An infelicitous consequence of the formula in Eq. 12 is that 'Anna' (or 'Mary') can be assigned either of the 'name' or 'person' nodes. To solve this problem, the parser decides which mapping is optimal by scoring edge attachments for all permutations of property nodes and selects the argmax. We will refer to this problem as the *edge permutation problem* when analyzing the errors of the parser in § 8.

5 Data and settings

We use the English-EWT section of Universal Dependencies (UD; Nivre et al., 2020) for dependency parsing, and the AMR2.0 dataset (Knight et al., 2017) for semantic parsing.

Ablations. For both tasks, we perform an ablation study on the development data to understand the impact of the choice of the loss function, sampling method, distance function and related settings (see Table 1). Specifically, we consider the following losses: the *original* loss formulation of Suzuki et al. (2019) and an extended version where a margin is added to the positive pairs (+*pos margin*; Eq. 4). Additionally, we report results for the MLE loss (Eq. 6), and for when we add our auxiliary structure loss (+*struct*; Eq. 9) to the contrastive or MLE losses.

In Eq. 3a–3b, centers and radii are computed separately, whereas in the original implementation of Suzuki et al. they are learnt jointly as a single vector in which the radius is the last dimension. To understand whether capturing the interaction

[6] Nonetheless, our disk embedding module is parser agnostic and could be applied to auto-regressive models as well.

[7] https://github.com/ufal/perin

between centers and radii helps with learning better disk embeddings, we create a (+*shared*) alternative where we jointly learn the centres and radii as in the original implementation. Finally, we build on the intuition of centers and radii informing each other, and propose an additional option on top of *shared*, (+*iter2* and +*iter3*), where we train an MLP to take C, R and $X_g^{(n-1)}$ as input and iteratively refine the centers and radii k times (where $k \in \{2,3\}$).

All the settings described assume that we include all negative instances (node pairs that are not in an ancestor–descendant relation) as part of the loss computation; however this might lead to a class imbalance problem as there are many more negatives than there are positive instances ($\sim$ 10:1 ratio). We therefore experiment with different sampling methods, including removing descendant–ancestor negatives altogether. We found that only one sampling method lead to performance improvement, i.e., oversampling of parent–child nodes (+*sampling*). In practice, we multiply the loss of every parent–child pair by a factor (here 2) in order to penalize errors coming from such pairs. We further test the joint impact of this sampling approach together with the auxiliary structure loss (+*both*).

Eq. 4 uses a constant margin α for negative samples. However, one can formulate a *tailored* lower bound on the margin, σ_{ij}, that depends on the relationship between node i and j. We formally prove the existence of such a lower bound in Appendix B, and provide an algorithm to identify it (Algorithm 1). In practice, however, the use of this tailored margin did not result in a performance improvement for either the dependency or the semantic parsing. For completeness, we include the results in Appendix C, Table 4 (+*tailored*).

For all combinations, we compare two distance functions: ℓ_2 and ℓ_1 norms. We include a list of hyperparameter values in Appendix A.

Comparison with baselines. We then compare the performance of our best systems (on test data) against the dependency parser of parser of Kádár et al. (2021), and the semantic parser of Samuel and Straka (2020), respectively. We found that $\sim$ 3% of the instances on the test set contain cycles which cannot be modelled by disk embeddings. To assess whether this impacts performance, we also provide results when removing instances containing cycles.

6 Evaluation

For both dependency and semantic parsing tasks, we compare our model output (a tree or a graph) to a reference parse.

For dependency parsing, we report accuracy, calculated using two standard measures: Unlabelled Attachment Score (UAS), that is the percentage of tokens that are assigned the correct head; and Labelled Attachment Score (LAS) that is the percentage of tokens that are assigned the correct head and the correct grammatical relation. We use UAS for model selection.

For semantic parsing, we use Mtool[8] to report a set of fine-grained F1 scores that reflect the performance of each classifier in the PERIN parser. *edge* (presence), (node) *label*, *top* (node) reflect the losses introduced in § 4.2; *prop*(erty prediction) represents a dedicated score on how well we predict named entity subgraphs (the *properties*), as well as how well we connect them to the rest of the graph.[9] Finally, *all* is a weighted average of these F1 scores.[10]

All results are reported as an average of 3 runs, along with standard deviations.

7 Results

The disk embedding loss formulation of Suzuki et al. (2019) performs considerably worse than all other settings. Results in Table 1 show that for both semantic and dependency parsing, adding a margin to positive instances (i.e., pushing positives below a margin $-\alpha$) leads to a considerable boost in performance. Compare the row of results for *original(ℓ_2)* with the rows for different variations of *pos margin(ℓ_2)*.

The structure auxiliary loss helps with performance, often in conjunction with oversampling parent–child pairs. Results in Table 1 show that injecting information on the local neighbourhood structure, together with giving more weight to parent–child relations helps both dependency and semantic parsing; see results in the row corresponding to *pos margin(ℓ_1)+both*. On top of these settings, we also test whether performance can be further improved by having a shared representation

[8] https://github.com/cfmrp/mtool

[9] Property scores can overlap with edge presence scores in that they also assess edge prediction but only for property nodes.

[10] We use Mtool instead of SMATCH scoring (Cai and Knight, 2013) since Mtool provides a more fine-grained evaluation of the parser performance.

	dependency	*semantic*				
system	UAS	edge	label	prop	top	all
Kádár et al. (2021)	91.62(±.33)	-	-	-	-	-
Samuel and Straka (2020)	-	70.14(±.36)	88.15(±.05)	87.34(±.25)	90.37(±.11)	80.28(±.21)
original(ℓ_2)	70(±.94)	46.98(±2.23)	87.70(±.14)	58.36(±4.63)	89.44(±.69)	67.19(±1.85)
pos margin(ℓ_2)	87.64(±.15)	58.29(±1.19)	87.77(±.21)	80.23(±2.92)	90.03(±.35)	73.80(±1.27)
+struct	88.63(±.21)	61.33(±1.07)	86.04(±.07)	81.95(±1.15)	88.12(±1.17)	74.36(±.9)
+sampling	88.79(±.37)	59.58(±.23)	87.76(±.4)	80.78(±.78)	89.59(±.64)	74.38(±.15)
+both	89.63(±.16)	62.47(±.33)	86.15(±.18)	81.57(±2.88)	89.91(±.17)	75.26(±.4)
pos margin(ℓ_1)	87.88(±.21)	58.1(±.28)	87.50(±.09)	80(±2.69)	89.02(±1.14)	73.68(±.18)
+struct	89.23(±.31)	62.14(±.49)	86.2(±.26)	81.58(±2.31)	88.57(±1.32)	75.02(±.52)
+sampling	88.91(±.05)	59.49(±.38)	87.7(±.4)	80.52(±1.43)	89.40(±1.59)	74.36(±.31)
+both	90.08(±.21)	63.78(±.07)	86(±.20)	83.12(±.28)	88.93(±1.1)	75.83(±.02)
+both+shared	90.02(±.01)	63.75(±.5)	86.01(±.12)	83.04(±1.31)	89.88(±.7)	75.89(±.15)
+both+shared(iter2)	**90.27**(±.01)	63.22(±.8)	86.83(±.21)	83.73(±1.46)	88.98(±2.1)	**76.51**(±.6)
+both+shared(iter3)	90.12(±.12)	63.57(±1.6)	86.76(±.13)	83.34(±3.21)	90.61(±.36)	76.24(±1.21)
MLE	85.61(±2.63)	56.52(±.12)	87.71(±.11)	79.15(±2.3)	90.20(±.1)	72.60(±.7)
+struct	87.98(±.46)	59.74(±.08)	85.61(±.16)	80.37(±1.63)	90.57(±.2)	73.42(±.92)

Table 1: Results for dependency and semantic parsing on *dev* set to evaluate the impact of different settings and loss functions.

of radii and centers, as well as by iteratively refining this. Results show that doing so only leads to a slight improvement. Finally, we can see that all our contrastive loss formulations perform better than our MLE loss, and that performance is comparable with ℓ_2 and ℓ_1 distance. Further experiments and analyses use the best setting of *pos margin(ℓ_1)+both+shared(iter2)*.

To have interpretability one has to pay a price in performance. Table 2 shows results on the *test* set for both dependency and semantic parsing, for the SOTA parsers, as well as our best settings. As can be seen, whereas the loss in performance for dependency parsing is rather small, the gap is wider for semantic parsing. We also notice that removing instances with cycles does not change this gap, and as such we can conclude that the fact that disk embeddings cannot model these instances is not responsible for a drop in performance. We investigate this gap in performance for semantic parsing as part of our analysis in the following section.

system	*dependency*		*semantic*	*semantic* (w/o cycles)
	UAS	LAS		
K(2021)	90.93(±.08)	89.91(±.06)	-	-
SS(2020)	-	-	79.89(±.12)	78.86(±.09)
ours	89.17(±.02)	87.69(±.01)	75.21(±.19)	74.54(±.23)

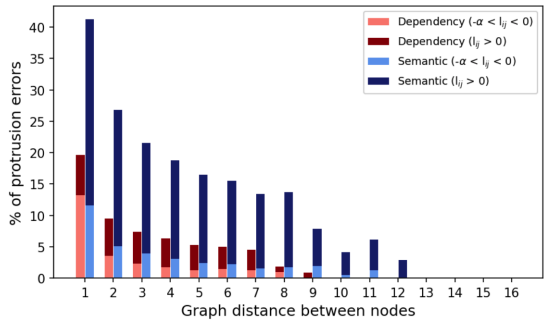
Table 2: Comparison between our best dependency and semantic parsers with a disk embedding loss, *ours*, and the baselines, namely K(2021) (Kádár et al., 2021) and SS(2020) (Samuel and Straka, 2020). Results are on the *test* set.

8 Analysis

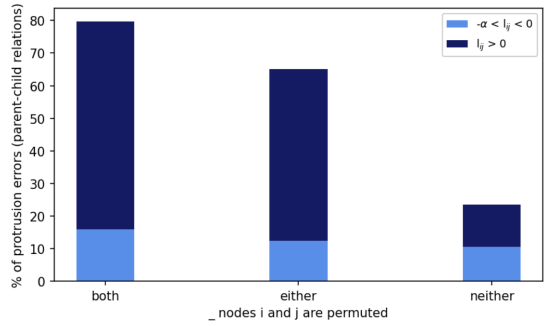
Do errors correlate with the (graph) distance and the relation between nodes? Now that we have access to interpetable representations, we can inspect which relations in a parse tree/graph are challenging to embed correctly. We begin by defining two types of errors w.r.t. the protrusion ℓ_{ij} for a predicted edge (i, j): one where $\ell_{ij} > 0$ and one where $-\alpha < \ell_{ij} < 0$ (note that a correct $\ell_{ij} < -\alpha$). We use 0 instead of α as the cut-off point because, although not below the desired margin, the $\mathrm{sigmoid}$ in Eq. 5 will still correctly predict the disk for i containing that of j. We plot these errors against the graph distance between all node pairs,[11] and report the % of errors over the total number of node pairs for different values of graph distance. Fig. 2(a) shows that there is an inverse correlation between errors and distance, with parent–child (distance of 1) and grandparent–grandchild relations (distance of 2) displaying the highest numbers of incorrect protrusions; this is particularly striking in the case of semantic parsing and we elaborate more on this when discussing the role of edge permutations below. However, we can see that a large number of incorrect protrusions fall in $(-\alpha, 0)$, especially in the case of dependency parsing, for which the $\mathrm{sigmoid}$ will still predict a correct containment.

What makes semantic parsing harder? We

[11] In a graph, where there could be multiple paths between a pair of nodes, we take the length of the longest path as the graph distance.



(a)



(b)

Figure 2: A detailed breakdown of % protrusion error (i.e., $\ell_{ij} > -\alpha$) for dependency and semantic parsing, for different values of graph distance (a); (b) shows this % for parent–child pairs (pairs with distance 1), categorized according to whether both, either or neither nodes are subjected to edge permutation.

start answering this question by looking at the results on the dev set where properties (*prop*) are the ones whose performance is impacted the most. We referred to this in § 4.2 as the *edge permutation* problem for nodes in named entity substructures which pose a challenge in the absence of explicit alignment information; we hypothesize this might cause a drop in performance and if so, we expect performance to drop more when there are more permutations. Fig. 3 shows that there is indeed an effect of the number of permutations on performance, with the baseline system performing better on instances with a large number of permutations. Interestingly, in the absence of permutations, our parser performs comparably to the baseline system.

Edge permutations could also be the main reason behind the large % of local errors in Fig. 2(a). To confirm this, we take a closer look at the breakdown of % protrusion errors for parent–child pairs according to whether both, either or neither node is subjected to permutation. Fig. 2(b) shows that when both or either node is permuted, the containment relationship is usually incorrectly predicted. Note, however, that whereas the number of these incorrect predictions are large, the overall performance is not overly affected, because when matching a predicted and a gold graph, we look at the node label and not at the node id. Using the example in Fig. 1(b), from a matching perspective, we would obtain the same graph swapping the parent node 'name' of 'Anna' with the one of 'Mary'.

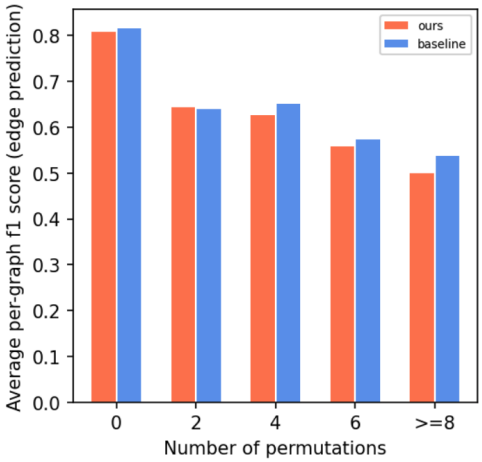


Figure 3: Performance on edge prediction for our disk embedding formulation vs. the baseline semantic parser. Instances are divided based on the number of edge permutations computed (with 1 meaning no permutations), as shown on the X-axis.

We also analyze the difference in performance between trees and graphs, as well as the effect of the number of nodes on performance; due to space limitations we include these results in Appendix D.1. In Appendix D.3, we discuss whether the size of the parses warrants moving to the hyperbolic space.

9 Conclusions and Future Work

We have explored disk embeddings as a means to obtain interpretable representations when training a parser that preduces a tree/graph. We showed that previously proposed disk embedding formulations are sub-optimal for the task of parsing, and accordingly explored alternatives that improve parsing performance. Nonetheless, our results suggest that we still need to pay a cost in performance to attain interpretability; this cost is small when parsing into trees, but notable for graphs. We also speculate that this cost might be due to properties of the parser we use, especially in cases where the absence of an alignment between words in a sentence and nodes in a graph allows for many permutations.

Our work can be considered as a first attempt to bring parsing and 'deep' natural language understanding into the realm of interpretability and representation learning, so that trees and graphs could be used in downstream tasks (e.g., image retrieval), similarly to how word embeddings have been used.

Acknowledgments

We thank the three anonymous reviewers for their useful comments. Research was conducted at the Samsung AI Centre Toronto and funded by Samsung Research, Samsung Electronics Co.,Ltd

References

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247.

Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. 2021. One spring to rule them both: Symmetric amr semantic parsing and generation without a complex pipeline. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*.

Tiansi Dong, Chrisitan Bauckhage, Hailong Jin, Juanzi Li, Olaf Cremers, Daniel Speicher, Armin B Cremers, and Jörg Zimmermann. 2018. Imposing category trees onto word-embeddings using a geometric construction. In *International Conference on Learning Representations*.

Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada. Association for Computational Linguistics.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press.

Octavian Ganea, Gary Bécigneul, and Thomas Hofmann. 2018. Hyperbolic entailment cones for learning hierarchical embeddings. In *International Conference on Machine Learning*, pages 1646–1655. PMLR.

Ákos Kádár, Lan Xiao, Mete Kemertas, Federico Fancellu, Allan Jepson, and Afsaneh Fazly. 2021. Dependency parsing with structure preserving embeddings. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1684–1697.

Kevin Knight, Bianca Badarau, Laura Banarescu, Claire Bonial, Madalina Bardocz, Kira Griffitt, Ulf Hermjakob, Daniel Marcu, Martha Palmer, Tim O'Gorman, et al. 2017. Abstract meaning representation (amr) annotation release 2.0. Technical report, Technical Report LDC2017T10, Linguistic Data Consortium, Philadelphia, PA.

Alice Lai and Julia Hockenmaier. 2017. Learning to predict denotational probabilities for modeling entailment. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 721–730.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2020. Fast semantic parsing with well-typedness guarantees. *arXiv preprint arXiv:2009.07365*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119. Curran Associates, Inc.

Maximillian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. *Advances in neural information processing systems*, 30:6338–6347.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: An evergrowing multilingual treebank collection. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4034–4043, Marseille, France. European Language Resources Association.

Hiroaki Ozaki, Gaku Morio, Yuta Koreeda, Terufumi Morishita, and Toshinori Miyoshi. 2020. Hitachi at mrp 2020: Text-to-graph-notation transducer. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 40–52.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Luigi Procopio, Rocco Tripodi, and Roberto Navigli. 2021. Sgl: Speaking the graph languages of semantic parsing via multilingual translation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 325–337.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D Manning. 2018. Universal dependency parsing from scratch. In *CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*.

David Samuel and Milan Straka. 2020. Ufal at mrp 2020: Permutation-invariant semantic parsing in perin. *arXiv preprint arXiv:2011.00758*.

Rupesh Kumar Srivastava, Klaus Greff, , and Jürgen Schmidhuber. 2015. Highway networks. In *Proceedings of the Deep Learning Workshop at the International Conference on Machine Learning*.

Dídac Surís, Ruoshi Liu, and Carl Vondrick. 2021. Learning the predictability of the future. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12607–12617.

Ryota Suzuki, Ryusuke Takahama, and Shun Onoda. 2019. Hyperbolic disk embeddings for directed acyclic graphs. In *International Conference on Machine Learning*, pages 6066–6075. PMLR.

Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. 2015. Order-embeddings of images and language. *arXiv preprint arXiv:1511.06361*.

Luke Vilnis, Xiang Li, Shikhar Murty, and Andrew McCallum. 2018. Probabilistic embedding of knowledge graphs with box lattice measures. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 263–272.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. Broad-coverage semantic parsing as transduction. *arXiv preprint arXiv:1909.02607*.

A Hyperparameters

Hyperparameters for the disk embedding modules of both dependency and semantic parsing are listed in Table 3; all hyperparameters *not* related to the disk embedding module are the same as in the original implementations. Both were tuned separately on the *+pos margin(ℓ_2)* system. All models were trained on a single TitanX GPU v100.

	semantic	dependency
batch size	8	5000
accumulation step	4	-
decoder lr	6e-4	1e-3
layers (disk MLP)	2	
center dim.	800	
weight init.	-0.01∼0.01	
activation (Eq. 3a-3b)	leakyReLU	
dropout	0.0	
margin	1	2

Table 3: Hyperparameters used in the disk embedding module for semantic and dependency parsing.

B Tailored bounds on protrusions

Suppose we have a DAG, $G = (V, E)$, with an associated order-isomorphic disk embedding for a given margin of $\alpha > 0$. That is, given any two distinct nodes a and b in the DAG we must have $|\ell_{ab}| \geq \alpha$. Here we prove a stronger lower bound of the form $|\ell_{ab}| \geq \sigma_{ab}\alpha$ where $\sigma_{ab} \geq 1$ depends on the relationship of a and b in the DAG. Moreover, the signs of ℓ_{ab} are such that $\ell_{ab} \leq -\sigma_{ab}\alpha$ when a is an ancestor of b, and $\ell_{ab} \geq \sigma_{ab}\alpha$ otherwise.

Suppose a is an ancestor of b, that is, for some $k > 0$, there is a path $(n_k, n_{k-1}, \ldots, n_0)$ in the DAG with $a = n_k$, $b = n_0$. If there are several such paths from a to b we choose one that is the longest. Then, since we have assumed the corresponding disk embedding satisfies the margin α, we have that n_i is an ancestor of n_{i-1} and therefore $\ell_{n_i,n_{i-1}} \leq -\alpha$. Therefore

$$\sum_{i=1}^{k} \ell_{n_i,n_{i-1}} \leq -k\alpha. \tag{13}$$

Moreover from the definition of $\ell_{n_i,n_{i-1}}$ in (2), we have

$$\begin{aligned}\sum_{i=1}^{k} \ell_{n_i,n_{i-1}} &= \sum_{i=1}^{k} d(c_i, c_{i-1}) - \sum_{i=1}^{k} (r_i - r_{i-1}) \\ &= \sum_{i=1}^{k} d(c_i, c_{i-1}) - (r_k - r_0) \\ &\geq d(c_k, c_0) - (r_k - r_0) \qquad (14a) \\ &= \ell_{n_k,n_0} \equiv \ell_{a,b}. \qquad (14b)\end{aligned}$$

Here we have used the triangle inequality in (14a). Together (14b) and (13) imply

$$\ell_{a,b} \leq -\sigma_{ab}\,\alpha \text{ with } \sigma_{ab} = k, \tag{15}$$

where k is the maximum length of any path from a to b.

For the protrusion from the descendant b to the ancestor a, namely $\ell_{b,a}$, we use

$$\ell_{b,a} \geq \sigma_{b,a}\,\alpha, \text{ for } \sigma_{b,a} = \sigma_{a,b}. \tag{16}$$

Here (16) is a simple consequence of (15) and the relation

$$\ell_{n_i n_j} + \ell_{n_j n_i} = 2d(\mathbf{c}_i, \mathbf{c}_j) \geq 0, \tag{17}$$

which follows easily from (2).

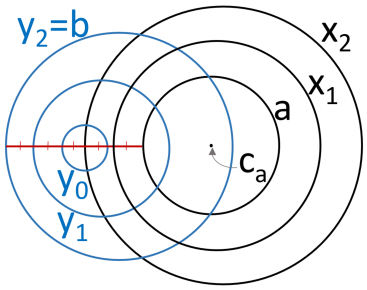


Figure 4: An example of a tight "other" relationship between nodes a and b in the situation described in Theorem B.1. Here $|P_1| = |P_2| = 2$, and $\ell_{a,b}$, is depicted by the length of the red line, where each hash mark denotes a subsegment of length α. Note the lower bound, namely $\ell_{a,b} = [|P_1| + |P_2| + 1]\alpha$, is achieved.

The remaining case is when a and b are in an "other" relationship, that is, they are not in an ancestor–descendant relationship (or vice versa). For this case it is useful to first define a *feasible pair of paths* (see Fig. 4). Define (P_1, P_2) to be a feasible pair of paths for nodes a and b if P_1 is a path $(x_i, x_{i-1}, \ldots, x_0)$ ending at $x_0 = a$, P_2 is a path $(y_j, y_{j-1}, \ldots, y_0)$ starting at $y_j = b$, and where no node in either path is a descendant of a

node in the other path. The following theorem provides a lower bound on the protrusion ℓ_{ab} in terms of such a feasible pair (P_1, P_2).

Theorem B.1. Suppose P_1 and P_2 are a feasible pair of paths for nodes a and b, as described above. Then

$$\ell_{ab} \geq [|P_1| + |P_2| + 1]\,\alpha, \tag{18}$$

where $|P|$ denotes the number of edges (i.e., the length) of the path P.

Proof. Since b is an ancestor of y_0, the bound (15) implies $\ell_{b,y_0} \leq -|P_2|\alpha$. Moreover, since x_i is neither an ancestor nor descendant of y_0, we have $\ell_{x_i,y_0} \geq \alpha$. By subtracting these two inequalities we find that

$$\ell_{x_i,y_0} - \ell_{b,y_0} \geq [|P_2| + 1]\,\alpha. \tag{19}$$

From (2) we find

$$\begin{aligned}
[|P_2| + 1]\alpha &\leq \ell_{x_i,y_0} - \ell_{b,y_0} && \text{(20a)}\\
&= d(c_{x_i}, c_{y_0}) - d(c_b, c_{y_0}) - r_{x_i} + r_b && \text{(20b)}\\
&\leq d(c_{x_i}, c_b) - r_{x_i} + r_b && \text{(20c)}\\
&= \ell_{x_i,b}, && \text{(20d)}
\end{aligned}$$

where we have used the triangle inequality in (20c). Similarly, since the path P_1 starts at x_i and ends at $x_0 = a$ we have

$$\ell_{x_i,a} \leq -|P_1|\alpha.$$

Moreover, all the disks x_k, for $k = 0, \ldots, i$ are in the other relationship to node b. Subtracting this inequality from (20) then gives

$$\begin{aligned}
[|P_1| + |P_2| + 1]\alpha &\leq \ell_{x_i,b} - \ell_{x_i,a} && \text{(21a)}\\
&= d(c_{x_i}, c_b) - d(c_{x_i}, c_a) - r_a + r_b && \text{(21b)}\\
&\leq d(c_a, c_b) - r_a + r_b && \text{(21c)}\\
&= \ell_{a,b}, && \text{(21d)}
\end{aligned}$$

where we have again used the triangle inequality in (21c). Eqn. (21) is the desired result. □

We apply Theorem B.1 by defining σ_{ab}, in the case a is neither an ancestor nor descendant of b, to be the maximum lower bound (18) over all feasible pairs, (P_1, P_2), for these nodes a and b. That is

$$\sigma_{ab} = \max_{(P_1,P_2)} \{|P_1| + |P_2| + 1\}. \tag{22}$$

Given a DAG, Algorithm 1 below computes this σ_{ab} for two nodes a and b in an "other" relationship.

Algorithm 1: compute_margin($\mathcal{G}$, n_i, n_j)

Input: graph $\mathcal{G}$, nodes n_i and n_j in 'other' relation
Output: maximum lower bound σ_{ij}
Initialize q with all ancestors of n_i, including n_i;
Initialize $\sigma_{i,j} = 1$;
while *q is not empty* **do**
  n_y = q.pop();
  if *n_j is reachable from n_y* **then**
    continue;
  else
    $\mathcal{G}'$ = copy($\mathcal{G}$);
    Remove all descendants of n_y in $\mathcal{G}'$;
    l_{gj} = 0;
    for *each descendant n_w of n_j in $\mathcal{G}'$* **do**
      if *longest_path(n_w, n_j) > l_{gj}* **then**
        l_{gj} = longest_path(n_w, n_j);
        $n_g = n_w$;
    end
    l_{yi} = longest_path(n_i, n_y);
    σ_{ij} = max(σ_{ij}, $l_{yi} + l_{gj} + 1$);
  end
end

C Using a tailored bound on protrusion

Eq. 4 can then modified to include σ_{ij} as shown below:

$$\mathcal{L}_c = \sum_{(i,j)\in P} [\sigma_{ij}\alpha + l_{ij}]_+ + \sum_{(i,j)\in N} [\sigma_{ij}\alpha - l_{ij}]_+ \tag{23}$$

Note that in this formulation a margin is used for positive samples as well. Results in Table 4 shows that a tailored margin, even in combination with the auxiliary structure loss as well as parent–child oversampling, does not lead to any gain over our best system (*pos margin(ℓ_1)+both*).

D Analysis

D.1 Is performance worse for larger trees/graphs?

Instances with no permutations might also be easier because they are shallower or contain less nodes. Fig. 5 shows that there is an effect due to the number of nodes. However Fig. 6 shows that performance starts to diverge at more than 25 nodes which doesn't fully explain performance for instance with less than 10 permutations.

D.2 Is there a difference in performance between trees and graphs?

Given the difference in performance between dependency and semantic parsing in Table 1, one can hypothesize that it is easier to preserve the structure of trees than graphs. To answer this question, we

	dependency	*semantic*				
system	UAS	edge	label	prop	top	all
Kádár et al. (2021)	91.62(±.33)	-	-	-	-	-
Samuel and Straka (2020)	-	70.14(±.36)	88.15(±.05)	87.34(±.25)	90.37(±.11)	80.28(±.21)
original(ℓ_2)	70(±.94)	46.98(±2.23)	87.70(±.14)	58.36(±4.63)	89.44(±.69)	67.19(±1.85)
pos margin(ℓ_1)*+both*	90.08(±.21)	63.78(±.07)	86(±.20)	83.12(±.28)	88.93(±1.1)	**75.83**(±.02)
+tailored (ℓ_2)	88.6(±.65)	52.13(±1.02)	88.79(±.07)	74.09(±2.2)	89.66(±.92)	69.95(±.33)
+struct	89.21(±.05)	61.37(±.35)	86.67(±1.75)	81.86(±1.31)	88.97(±.92)	74.70(±.31)
+sampling	89.14(±.91)	54.47(±1.12)	87.94(±.1)	76(±1.15)	88.93(±1.56)	70.91(±.38)
+both	89.87(±.64)	62.91(±.34)	86.91(±.01)	83.42(±.42)	89.2(±.89)	75.54(±.35)
+tailored (ℓ_1)	88.65(±.16)	52.38(±.58)	88.84(±.25)	73.44(±1.78)	89.81(±.87)	69.72(±.71)
+struct	89.4(±.04)	61.92(±.17)	86.73(±.15)	82.40(±.62)	89.64(±1.62)	75.05(±.06)
+sampling	89.12(±.43)	53.98(±.68)	87.78(±.11)	75.98(±1.92)	89.71(±.34)	70.82(±.18)
+both	89.92(±.4)	62.89(±.65)	86.89(±.11)	82.61(±1.02)	89.11(±.83)	75.60(±.43)

Table 4: Results for the *+tailored* setting for dependency and semantic parsing on *dev* set. Performance of the baseline parsers, the *original* formulation (Suzuki et al., 2019), as well as our best system (*pos margin*(ℓ_1)*+both*, see Table 1) are also reported for comparison.

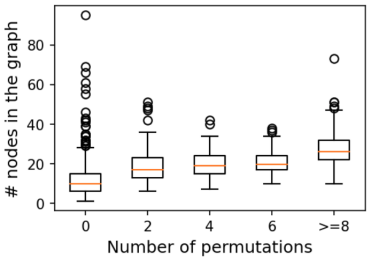


Figure 5: Analysis of number of permutations w.r.t. the number nodes in a graph.

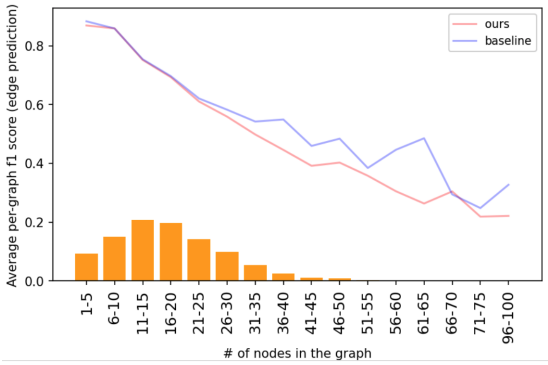


Figure 6: Analysis of the number of nodes in a graph w.r.t. parse prediction performance per instance for the baseline biaffine and the disk embedding system. In orange, an histogram over the proportion of instances in a particular size bin is also reported.

first divide the AMR dev set into instances whose gold parse is a tree vs. those that are graph. We then compare the average of the per-instance F1 scores for edge presence as given by the predictions of the baseline system (Samuel and Straka, 2020) vs. those of our best disk embedding model. We confirm that indeed graphs are harder, and, in line with the results of dependency parsing, the gap between the baseline system and our disk embedding formulation is larger for graphs (Δ =1.95) than trees (Δ =0.77).

D.3 Do we need to move to hyperbolic space?

To answer this question, we plot the difference between the radii for all pairs of ancestor–descendant disks ($|r_i - r_j|$), as well as the norm of the difference between the center of a disk and the mean of all centers of the disks in a graph ($||c_i - \bar{c}||$). We hypothesize that moving to the hyperbolic space is justified if an exponential growth is observed when the graphs get larger. However, Fig. 7 shows that this is not the case.

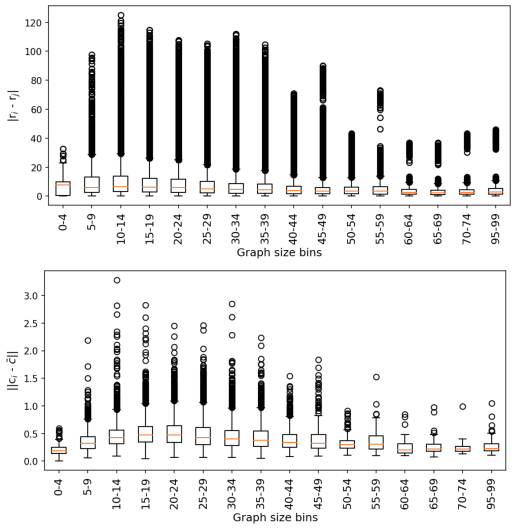


Figure 7: Boxplots analysis the absolute difference between radii of two disks (above) and the norm of the difference between the center of a disk in a graph and the average over the all disk centers in the same graph (below). Both are plotted against the number of nodes in a graph on the x axis.