Abstract

We present PERIN, a novel permutation-invariant approach to sentence-to-graph semantic parsing. PERIN is a versatile, cross-framework and language independent architecture for universal modeling of semantic structures. Our system participated in the CoNLL 2020 shared task, Cross-Framework Meaning Representation Parsing (MRP 2020), where it was evaluated on five different frameworks (AMR, DRG, EDS, PTG and UCCA) across four languages. PERIN was one of the winners of the shared task. The source code and pre-trained models are available at http://www.github.com/ufal/perin.

1 Introduction

The aim of the CoNLL 2020 shared task, Cross-Framework Meaning Representation Parsing (MRP 2020; Oepen et al., 2020), is to translate plain text sentences into their corresponding graph-structured meaning representation.¹ MRP 2020 features five formally and linguistically different frameworks with varying degrees of linguistic and structural complexity:

- **AMR**: Abstract Meaning Representation (Banarescu et al., 2013);
- **DRG**: Discourse Representation Graphs (Abzianidze et al., 2017) provide a graph encoding of Discourse Representation Structure (Van der Sandt, 1992);
- **EDS**: Elementary Dependency Structures (Oepen and Lønning, 2006);
- **PTG**: Prague Tectogrammatical Graphs (Hajič et al., 2012);

These frameworks constitute the cross-framework track of MRP 2020, while the separate cross-lingual track introduces one additional language for four out of the five frameworks: Chinese AMR (Li et al., 2016), German DRG, Czech PTG and German UCCA (Hershcovich et al., 2019).

In agreement with the shared task objective to advance uniform meaning representation parsing across diverse semantic graph frameworks and languages, we propose a language and structure agnostic sentence-to-graph neural network architecture modeling semantic representations from input sequences.

The main characteristics of our approach are:

- **Permutation-invariant model**: PERIN is, to our best knowledge, the first graph-based semantic parser that predicts all nodes at once in parallel and trains them with a permutation-invariant loss function. Semantic graphs are naturally orderless, so constraining them to an artificial node ordering creates an unfounded restriction; furthermore, our approach is more expressive and more efficient than order-based auto-regressive models.

- **Relative encoding**: We present a substantial improvement of relative encodings of node labels, which map anchored tokens onto label strings (Straka and Straková, 2019). Our novel formulation allows using a richer set of encoding rules.

- **Universal architecture**: Our work presents a general sentence-to-graph pipeline adaptable for specific frameworks only by adjusting pre-processing and post-processing steps.

Our model was ranked among the two winning systems in both the cross-framework and the cross-lingual tracks of MRP 2020 and significantly advanced the accuracy of semantic parsing from the last year’s MRP 2019.

¹See http://mrp.nlpl.eu/2020/ for more details.
2 Related Work

Examples of general, formalism-independent semantic parsers are scarce in the literature. Hershco-ovich et al. (2018) propose a universal transition-based parser for directed, acyclic graphs, capable of parsing multiple conceptually and formally different schemes. Furthermore, several participants of MRP 2019 presented universal parsers. Che et al. (2019) improved uniform transition-based parsing and used a different set of actions for each framework. Lai et al. (2019) submitted a transition-based parser with shared actions across treebanks, but failed to match the performance of the other parsers. Straka and Straková (2019) presented a general graph-based parser, where the meaning representation graphs are created by repeatedly adding nodes and edges.

Graph-based parsers (McDonald and Pereira, 2006; Peng et al., 2017; Dozat and Manning, 2018; Cai and Lam, 2020) usually predict nodes in a sequential, auto-regressive manner and then connect them with a biaffine classifier. Unlike these approaches, our model infers all nodes in parallel while allowing the creation of rich intermediate representations by node-to-node self-attention.

Machine learning tools able to efficiently process unordered sets are gaining more attention in recent years. Qi et al. (2017) and particularly Zhang et al. (2019b) proposed permutation-invariant neural networks for point clouds, which are of great relevance to our system. Our work was further inspired by Carion et al. (2020), who utilize permutation invariance for object detection in a similar fashion to our sentence-to-graph generation.

3 Methods

3.1 Graph Representation

All five semantic formalisms share the same representation via directed labeled multigraphs in the graph interchange format proposed by Kuhlmann and Oepen (2016). Universally, the semantic units are represented by nodes and the semantic relationships by labeled edges. Each node can be anchored to a (possibly empty) set of input characters, and can contain a (possibly empty) list of properties, each being an attribute-value pair.

We simplify this graph structure by turning the properties into graph nodes: every property \{attribute : value\} of node \(n\) is removed and a new node with label value is connected to the parent node \(n\) by an edge labeled with attribute; the anchors of the new node are the same as of its parent.\(^2\) Figure 4 illustrates this transformation together with other pre-processing steps (specific for each framework) explained in detail in Section 3.7.

Another change to the internal graph representation is the use of relative label encoding (discussed in Section 3.4), which substitutes the original node labels by lists of relative encoding rules.

3.2 Overall Architecture

A simplified illustration of the whole model can be seen in Figure 1. The input is tokenized, an encoder (Section 3.5) computes contextual embeddings of the tokens, and each embedded token \(e_t\) is then mapped onto \(Q\) queries by nonlinear transformations \(q_{i,t} = \tanh (W_t e_t + b_t), t \in \{1, \ldots, Q\}\), where \(W_t\) is a trainable weight matrix and \(b_t\)

\(^2\)“Nodeification” of properties was motivated by the nature of AMR graphs, where the properties are equivalent to instanced concepts/nodes (Banerjee et al., 2013). From a more practical viewpoint, it allows us to utilize a single classifier for both the node labels and the less-frequent properties, and to simplify the whole architecture.
is a trainable bias vector. After that, a decoder (Transformer with pre-norm residual connections (Nguyen and Salazar, 2019) and cross-attention) processes the queries, obtaining their final feature vectors $h_{i,t}$. These feature vectors are shared across all classification heads, each inferring specific aspects of the final meaning representation graph from them:

- **Relative encoding classifier** decides what node label should serve as the “answer” to each query; a query can also be denied (no node is created) when classified as “null”. Relative label prediction is described in detail in Section 3.4.3.
- **Anchor biaffine classifier** uses deep biaffine attention (Dozat and Manning, 2017) to create anchors between nodes and surface tokens – to be more precise, the biaffine attention processes the latent vectors of queries $h_{i,t}$ and tokens $e_j$, and predicts the presence of an anchor between every pair of them as a binary classification task.
- **Edge biaffine classifier** uses three biaffine attention modules to predict whether an edge should exist between a pair of nodes (presence binary classification), what label(s) should it have (label multi-class or multi-label classification, depending on the framework) and what attribute should it have (attribute multi-class classification) – in essence, this module is a simple extension of the standard edge classifier by Dozat and Manning (2018).
- **Property classifier** uses a linear layer followed by a sigmoid nonlinearity to identify nodes that should be converted to properties.
- **Top classifier** uses a linear layer followed by a softmax nonlinearity (where the probabilities are normalized across nodes) to detect the top node.

This section described all modules capable of handling different characteristics of meaning representation graphs. Not all of them appear in each framework – for example, AMR graphs do not need edge attributes, while UCCA graphs do not contain any properties. More details about specific framework configurations are given in Section 3.7.

### 3.3 Permutation-invariant Graph Generation

Semantic graphs are orderless, so it is unnatural to constrain their generation by some artificial node ordering. Traditionally, graph nodes have been predicted by a sequence-to-sequence model (Peng et al., 2017), with the nodes being generated in some hardwired order (Zhang et al., 2019a). Demanding a fixed node ordering causes the discontinuity issue (Zhang et al., 2019b): even when correct items are predicted, they are viewed as completely wrong if not in the expected order. We avoid this issue by using such a loss function and such a model that produce the same outcome independently on the node ordering (Zaheer et al., 2017).

#### 3.3.1 Permutation-equivariant Model

We transform the queries $q = \{q_i\}_{i=1}^N$ into hidden features $h = \{h_i\}_{i=1}^N$ in such a manner that any permutation $\pi \in \mathcal{S}_N$ of the input $\pi(q) = \{q_{\pi(i)}\}_{i=1}^N$ produces the same output $\pi(h) = \{h_{\pi(i)}\}_{i=1}^N$. The Transformer architecture (Vaswani et al., 2017) conveniently fulfills this requirement (assuming positional embeddings are not used). Furthermore, it can combine any pair of input items independently of their distance and in an efficient non-autoregressive way.

#### 3.3.2 Permutation-invariant Loss

The hidden features $h$ are further refined into predictions $\hat{y} = f_\theta(h)$ by the classification heads. In order to create a permutation-invariant loss function, i.e., a function $\mathcal{L}(\pi(\hat{y}), y)$ giving the same result for every $\pi \in \mathcal{S}_N$, we find a permutation $\pi^* \in \mathcal{S}_N$ assigning each query to its most similar node. After permuting the targets according to $\pi^*$, the standard losses can be computed, because they are no longer dependent on the original ordering of $\hat{y}$ and $y$.

3Unfortunately, the uniqueness of $\pi^*$ is not always guaranteed: given that the proposed matching depends only on labels and anchors, there might be multiple equivalent nodes (considering only labels and anchors). We break ties between such nodes by also minimizing the likelihood of their edges across all their permutations.

To find the minimizing permutation $\pi^*$, we start by extending the (multi)set of target nodes $y$ by “null” nodes (denoted as $\emptyset$) in order to fulfill $|\hat{y}| = |y|$. When classified as “null” during inference, the query is denied and omitted from further processing. The permutation $\pi^*$ is then defined as

$$\pi^* = \arg \max_{\pi \in \mathcal{S}_N} \sum_{i=1}^N p_{\text{match}}(\hat{y}_{\pi^*(i)}, y_{\pi(i)}),$$

where the matching score $p_{\text{match}}$ is composed of a label score and the geometric mean (GM) of
anchor scores of all input tokens $T$. The label score of the $j$th query and the $j$th node is defined as the predicted probability of the target $j$th label; the anchor score of the $i$th query, $j$th node and a token $t \in T$ is defined as the predicted probability of the actual (non)existence of an anchor between $t$ and the $j$th node:

$$
\begin{align*}
    p_{\text{match}} &= p_{\text{label}} \cdot p_{\text{anchor}} \\
    p_{\text{label}}(\hat{y}_i, y_j) &= \mathbb{1}_{y_j^\text{label} \neq \emptyset} P(y_j^\text{label}|h_i; \theta) \\
    p_{\text{anchor}}(\hat{y}_i, y_j) &= \text{GM}_{t \in \text{tokens}} P(\mathbb{1}_{t \in y_j^\text{anchors}}|t, h_i; \theta).
\end{align*}
$$

We use the geometric mean to keep the anchor score $p_{\text{anchor}}$ magnitude independent of the number of tokens, and therefore have a similar weight as the label score $p_{\text{label}}$.

The optimal matching $\pi^*$ can be efficiently computed by the Hungarian algorithm (Kuhn, 1955) in $O(n^3)$. As a result, every query is assigned either to a regular node or to a “null” node $\emptyset$. An illustration of a matching between queries and target nodes is presented in Figure 2.

The loss functions for the queries are computed with respect to the matched nodes. After finding $\pi^*$, we permute all target nodes and compute the classification losses in the standard “order-based” way (i.e., by minimizing the cross-entropy between the predictions and the corresponding targets). The losses of queries matched to the “null” nodes are ignored, except for their relative label loss $\ell_{\text{label}}$, which pushes these queries to predict $\emptyset$ as their label. The label loss is further altered by the focal loss factor (Lin et al., 2017) to mitigate the imbalance of labels introduced by extending the targets with the “null” nodes.

### 3.3.3 Anchor Masking

During the early experiments with this architecture, we noticed that nodes tend to be generated from their anchored tokens (or more precisely from the queries of their anchored tokens), after the outputs stabilize during first epochs. We employ this observation to create an inductive bias by limiting the possible pairings to occur only between target nodes and predictions from their anchored tokens. Formally, this is achieved by setting

$$
p_{\text{anchor}}(\hat{y}_i, y_j) = \varepsilon,$$

if the $j$th node is not anchored to the $i$th token, with $\varepsilon$ being a small positive constant close to 0.

### 3.4 Relative Label Encoding

Similarly to Straka et al. (2019); Straka and Straková (2019), we use relative encodings for the prediction of node labels: instead of direct classification of label strings, we utilize rules specifying how to transform anchored surface tokens into the semantic labels. For example, in Figure 1, the anchored token “diving” is transformed into “dive” by using a relative encoding rule deleting its last three characters and appending a character “e”. Such a rule could be also employed for predicting a node anchored to “taking” or “giving”. Relative encoding of labels is thus able to reduce the number of classification targets and generalize outside of the set of “absolute” label strings seen during training. Alternatively, the relative encoding can be seen as an extension of the pointer networks (Gu et al., 2016), which also decides how to post-process the copied tokens. Table 1 demonstrates how the relative encoding rules reduce the number of targets that need to be classified.

---

**Figure 2:** Example of a matching between queries and target nodes during training. Every input token is mapped onto $Q$ (2 in this case) queries $q_{i,j}$, which are decoded into node predictions $y_{i,j}$. These predictions are paired with the ground truth nodes $y$, as in Equation 1. Then, the loss functions are computed with respect to the paired target nodes. Queries without any match should be classified as “null” nodes. When classified as “null” during inference, the query is not turned into any node (the query is denied).
Table 1: The numbers of absolutely and relatively encoded node labels. Relative encodings lead to a significant reduction of classification targets in an order of magnitude across all frameworks. Note that node labels are the union of labels and property values (except for PTG), as described in Section 3.1.

<table>
<thead>
<tr>
<th>framework</th>
<th>language</th>
<th># labels strings</th>
<th># encoding rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMR</td>
<td>English</td>
<td>27,049</td>
<td>4,385</td>
</tr>
<tr>
<td></td>
<td>Chinese</td>
<td>30,949</td>
<td>2,560</td>
</tr>
<tr>
<td>DRG</td>
<td>English</td>
<td>5,715</td>
<td>684</td>
</tr>
<tr>
<td></td>
<td>German</td>
<td>1,905</td>
<td>918</td>
</tr>
<tr>
<td>EDS</td>
<td>English</td>
<td>31,933</td>
<td>1,322</td>
</tr>
<tr>
<td>PTG</td>
<td>English</td>
<td>39,336</td>
<td>529</td>
</tr>
<tr>
<td></td>
<td>Czech</td>
<td>38,448</td>
<td>1,321</td>
</tr>
</tbody>
</table>

### 3.4.1 Minimal Encoding Rule Set

Naturally, a label can be generated from anchored tokens in multiple ways. Unlike previous works that needed some heuristic to select a single rule from all suitable ones (Straka and Straková, 2019), we do not constraint the space of the possible rules much. Instead, we construct the final set of encoding rules to be the smallest possible one capable of encoding all labels.

Formally, let $S$ be an arbitrary class of functions transforming a list of text strings (anchored tokens) into another string (node label), and let $N$ be the set of all nodes from the training set. For $n \in N$, denote $n_\ell$ the anchored surface tokens and $n_\ell$ the target label string. Then the set of applicable rules for the node $n$ is $S_n = \{ r \in S | r(n_\ell) = n_\ell \}$. Our goal is to find the smallest subclass $S^* \subseteq S$ capable of encoding all node labels, in other words a subclass $S^*$ satisfying

$$\forall n \in N : S^* \cap S_n \neq \emptyset.$$ 

This formulation is equivalent to the minimal hitting set problem. Therefore, we can find the optimal solution of our problem by reducing it to a weighted MaxSAT formula in CNF: every $S_n = \{ r_1, r_2, \ldots, r_k \}$ becomes a hard clause $(r_1 \lor r_2 \lor \ldots \lor r_k)$ and every $r \in S$ becomes a soft clause $(\neg r)$. We then submit this formula to the RC2 solver (Ignatiev et al., 2019) to obtain the minimal set of rules. Note that although solving this problem can take up to several hours, it needs to be done only once and then cached for all the training runs.

### 3.4.2 Space of Relative Rules

Our space of relative rules $S$ consists of four disjoint subclasses:

1. **token rules** are represented by seven-tuples $(d_l, d_r, s, r_l, r_r, a_l, a_r)$ and process a list of anchored tokens $n_r$ by first deleting the first $d_l$ and the last $d_r$ tokens, then by concatenating the remaining ones into one text string with the separator $s$ inserted between them, followed by removing the first $r_l$ and last $r_r$ characters and finally by adding the prefix $a_l$ and suffix $a_r$.

2. **lemma rules** are created similarly to the token rules, but use the provided lemmas instead of tokens;

3. **number rules** transform word numerals into digits – for example, tokens [“forty”, “two”] become “42”;

4. **absolute rules** use the original label string $n_\ell$, without taking into account any anchored tokens $n_\ell$; they serve as the fallback rules when no relative encodings are applicable.

### 3.4.3 Prediction of Relative Rules

Even with the minimal set of rules $S^*$, multiple rules may be applicable to a single node. Therefore, the prediction of relative rules is a multi-label classification problem. The target distribution for a node $n$ over all $r \in S^*$ is defined as follows:

$$P(r|n) = \begin{cases} 1 & \text{if } r \in S_n; \\ \frac{1}{|S^* \cap S_n|}, & \text{otherwise.} \end{cases}$$

The label loss $\ell_{\text{label}}$ is then calculated as the cross-entropy between the target and the predicted distributions.

We use mixture of softmaxes (MoS) to mitigate the softmax bottleneck (Yang et al., 2018) that arises when multiple hypotheses can be correctly applied to a single input. MoS allows the model to consider $K$ different hypotheses at the same time and weight them relatively to their plausibility.

Formally, let $h_q$ be the final latent vector for query $q$ and let $W_k$, $b_k$, $w_c$, $b_c$, $w_r$, $b_r$ be the trainable weights. Then, the estimated MoS distribution

---

4To show a real example of a *token rule* from EDS, the rule $(0, 1, +, 0, 0, \_a_1)$ maps tokens (“at”, “the”, “very”, “least”, “,”) into the label “at+the+very+least_a_1”.

4The target distribution is further modified by label smoothing (Szegedy et al., 2016) for better regularization.
of relative rules $P_\theta(r|n)$ is defined as follows:

$$x_k = \tanh(W_k h_i + b_k)$$

$$\pi_k = \frac{\text{sigmoid}(h^T_i w_k + b_k)}{\sum_{k'} \text{sigmoid}(h^T_i w_{k'} + b_{k'})}$$

$$P_\theta(r|n) = \sum_{k=1}^{K} \pi_k \text{softmax}(x^T_k w_r + b_r).$$

### 3.5 Finetuning XLM-R

To obtain rich contextual embeddings for each input token, we finetune the pretrained multilingual model XLM-R (Conneau et al., 2020). The architecture of the encoder is presented in Figure 3.

#### 3.5.1 Contextual Embedding Extraction

Different layers in BERT-like models represent varying levels of syntactic and semantic knowledge (van Aken et al., 2019), raising a question of which layer (or layers) should be used to extract the embeddings from. Following Kondratyuk and Straka (2019), we solve this problem by a purely data-driven approach and compute the weighted sum of all layers. Formally, let $\ell_t$ be the intermediate output from the $l^{th}$ layer and let $w_t$ be a trainable scalar weight. The final contextual embedding is then calculated as

$$e = \sum_{l=1}^{L} \text{softmax}(w_t) e_l.$$

Note that each input token can be divided into multiple subwords by the XLM-R tokenizer. To obtain a single embedding for every token, we sum the embeddings of all its subwords. Finally, the contextual embeddings are normalized with layer normalization (Ba et al., 2016) to stabilize the training.$^6$

#### 3.5.2 Finetuning Stabilization

Given the large number of parameters in the pretrained XLM-R model, we employ several stabilization and regularization techniques in attempt to avoid overfitting.

We start by dividing the model parameters into two groups: the finetuned XLM-R and the rest of the network. Both groups are updated with AdamW optimizer (Loshchilov and Hutter, 2019), and their learning rate follows the inverse square root schedule with warmup (Vaswani et al., 2017). The learning rate of the finetuned encoder is frozen for the first 2000 steps before the warmup phase starts (Howard and Ruder, 2018). The warmup is set to 6000 steps for both groups, while the learning rate peak is $6 \cdot 10^{-5}$ for the XLM-R and $6 \cdot 10^{-4}$ for the rest of the network. The weight decay for XLM-R, $10^{-2}$, is considerably higher compared to $1.2 \cdot 10^{-6}$ used in the rest of the network (Devlin et al., 2019).

Dropout of entire intermediate XLM-R layers results in additional regularization – we drop each layer with 10% probability by replacing $w_t$ with $-\infty$ during the final contextual embedding computation (Section 3.5.1). Inter-layer and attention dropout rates are the same as during the XLM-R pretraining.$^7$

#### 3.6 Balanced Loss Weights

Semantic parsing is an instance of multi-task learning, where each task $t \in T$ can have conflicting needs and where the task losses $\ell_t$ can have different magnitudes. The overall loss function $\mathcal{L}$ to be optimized therefore consists of the weighted sum of partial losses $\ell_t$:

$$\mathcal{L}(f_{\theta}(x), y) = \sum_{t \in T} w_t \ell_t(f_{\theta}(x), y).$$

Finding optimal values for the loss weights $w_t$ is extremely complicated. This issue is usually resolved either by (suboptimally) setting all weights equally to 1 or by a thorough grid search. However, the

---

$^6$A side effect of the normalization step is that the subword summation is equal to the more common subword average (Zhang et al., 2019a).

$^7$Due to the space constrains, all hyperparameters for each training configuration (together with the source code and pretrained models) are published at https://github.com/ufal/perin.
complexity of the grid search grows exponentially with $|T|$ and would need to be performed independently for all nine combinations of frameworks and languages.

A more feasible solution is to set the weights adaptively according to a data-driven metric as in [2018]. We follow [2018], who balance the magnitudes of gradients $\|\nabla_{\theta_s} w_i \ell_i\|_2$, where $\theta_s$ are the weights of the shared part of the network. That magnitude is made proportional to the ratio of the current loss and its initial value: when $\ell_i$ decreases relatively quickly, its strength gets reduced to leave more space for the other tasks. Consequently, the loss weights $w_i$ are not static, but change throughout the training to balance the individual gradient norms. Figure 5 shows an example of the balancing dynamics.

3.7 Framework Specifics

3.7.1 AMR

AMR is a Flavor 2 framework, which means its nodes are not anchored to the surface forms. We instead exploit the general algorithm for the minimal encoding rule set (Section 3.4.1) to create artificial anchors: considering all possible one-to-one anchors $a \in A_n$ for each node $n$, we infer all compatible rules $S_n = \bigcup_{a \in A_n} S_n^a$, and find the minimal set of rules $S^*$. The artificial anchors of a node $n$ are then defined as $\{a \in A_n | S_n^a \cap S^* \neq \emptyset\}$. Consequently, our parser does not need any approximate anchoring (because we instead compute an anchoring minimizing the number of relative rules).

On the other hand, Chinese AMR graphs contain anchors (they are actually of Flavor 1), therefore, the described procedure is applied only on English AMR.

AMR graphs also contains inverted edges that transform them into tree-like structures. The inverted edges are marked by modified edge labels (for example, ARGO becomes ARGO-of). We normalize the graphs back into their original non-inverted form, making them more uniform, simplifying edge prediction to become more local and independent of the global graph structure. An example of AMR pre-processing is shown in Figure 4.

Considering the fact that every node is artificially anchored to at most a single token, the anchor classifier is not needed, if anchor masking is used (Section 3.3.3). Finally, AMR parsing does not employ the edge attribute classifier.
<table>
<thead>
<tr>
<th>System</th>
<th>AMR\textsuperscript{eng}</th>
<th>DRG\textsuperscript{eng}</th>
<th>EDS\textsuperscript{eng}</th>
<th>PTG\textsuperscript{eng}</th>
<th>UCCA\textsuperscript{eng}</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>HUJI-KU (Arviv et al., 2020)</td>
<td>52.36%</td>
<td>62.75%</td>
<td>79.68%</td>
<td>53.76%</td>
<td>72.91%</td>
<td>64.29%</td>
</tr>
<tr>
<td>Hitachi (Ozaki et al., 2020)</td>
<td>81.54%</td>
<td>93.19%</td>
<td>93.56%</td>
<td>88.73%</td>
<td>75.07%</td>
<td>86.42%</td>
</tr>
<tr>
<td>HIT-SCIR (Dou et al., 2020)</td>
<td>69.80%</td>
<td>89.07%</td>
<td>87.40%</td>
<td>84.26%</td>
<td>74.76%</td>
<td>81.06%</td>
</tr>
<tr>
<td>ÚFAL PERIN</td>
<td>80.23%</td>
<td>94.16%</td>
<td>92.73%</td>
<td>88.44%</td>
<td>76.40%</td>
<td>86.39%</td>
</tr>
<tr>
<td>ÚFAL PERIN*</td>
<td>80.23%</td>
<td>94.16%</td>
<td>92.73%</td>
<td>89.19%</td>
<td>76.40%</td>
<td>86.54%</td>
</tr>
</tbody>
</table>

Table 2: The all F1 scores and a macro-average total score of the shared task systems. PERIN is our official shared task submission and PERIN* is a post-competition submission with a fixed bug. The best results are typeset in **bold**. The top table contains the cross-framework scores on English treebanks, while the bottom table presents the cross-lingual ones.

### 3.7.2 DRG

Since the DRG graphs are also of Flavor 2, they are pre-processed similarly to English AMR. Additionally, we reduce all nodes representing binary relations into labeled edges between the corresponding discourse elements.

Nodes in German DRG graphs are labeled in English, which decreases the applicability of relative encoding. Therefore, we employ the opus-mt-de-en (Tiedemann and Thottingal, 2020) machine translation model from Huggingface’s transformers package (Wolf et al., 2019) to translate the provided lemmas from German to English, before computing the relative encoding rules.

DRG parsing does not make use of anchor and edge attribute classifiers, just like AMR parsing.

### 3.7.3 EDS

EDS graphs are post-processed to contain a single continuous anchor for every node. The EDS parser contains all the classification modules described in Section 3.2, except for the edge attribute classifier.

### 3.7.4 PTG

Properties in PTG graphs are not converted into nodes as in other frameworks, but are directly predicted from latent vectors $h_q$ by multi-class classifiers (one for each property type). Additionally, the frame properties are selected only from frames listed in CzEngVallex (Urešová et al., 2015).

We utilize all classification heads except for the top node classification, because PTG graphs contain special `<TOP>` nodes, which make the separate top prediction redundant.

### 3.7.5 UCCA

We augment the UCCA nodes by assigning them leaf and inner labels. Additionally, the inner nodes are anchored to the union of anchors of their children. Therefore, the nodes can be differentiated by the permutation-invariant loss (Section 3.3.2).

The UCCA parser does not have the property classifier and the top classifier, where the latter is not needed, because the top node can be inferred from the structure of the rooted UCCA graphs.

### 4 Results

We present the overall results of our system in Table 2 and Table 3. Both tables contain F1 scores obtained using the official MRP metric. Table 2 shows the all F1 scores for the individual frameworks together with the overall averages for the cross-framework and cross-lingual tracks. Macro-averaged results (across all nine frameworks) for the different MRP metrics are displayed in Table 3.

Note that our original submission (denoted as PERIN) contained a bug in anchor prediction for Chinese AMR and both PTG frameworks. The bug caused the nodes to get anchored to at least one token. We submitted a fixed version called PERIN* in the post-competition evaluation and compare it with the original one in Table 2.

According to the official whole-percent-only all F1 score, our competition submission reached tied first place in both the cross-lingual and the cross-framework track, with its performance virtually 8

*Fine-grained results for each framework are available in the task overview by Oepen et al. (2020).
HUJI-KU (Arviv et al., 2020) 85.85% 22.80% 29.48% 46.79% 61.35% 7.67% 62.43%
Hitachi (Ozaki et al., 2020) 95.67% 68.93% 48.89% 61.95% 80.14% 24.93% 85.81%
Hit-SCIR (Dou et al., 2020) 94.37% 61.84% 30.80% 52.18% 71.41% 22.51% 75.66%
UFAL PERIN* 94.20% 70.36% 49.34% 63.45% 79.68% 27.07% 86.26%

Table 3: Overall results for different MRP metrics, macro-averaged over all frameworks and languages. The best results are typeset in **bold**.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Tops</th>
<th>Labels</th>
<th>Properties</th>
<th>Anchors</th>
<th>Edges</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFAL PERIN*</td>
<td>89.53%</td>
<td>93.45%</td>
<td>94.34%</td>
<td>93.40%</td>
<td>90.74%</td>
<td>92.73%</td>
</tr>
<tr>
<td>w/o MoS</td>
<td>88.04%</td>
<td>93.39%</td>
<td>93.79%</td>
<td>93.48%</td>
<td>90.76%</td>
<td>92.65%</td>
</tr>
<tr>
<td>w/o focal loss</td>
<td>89.08%</td>
<td>93.33%</td>
<td>93.59%</td>
<td>93.21%</td>
<td>90.46%</td>
<td>92.46%</td>
</tr>
<tr>
<td>BERT encoder</td>
<td>89.95%</td>
<td>92.97%</td>
<td>94.74%</td>
<td>92.92%</td>
<td>89.84%</td>
<td>92.27%</td>
</tr>
<tr>
<td>w/o balanced losses</td>
<td>89.23%</td>
<td>92.28%</td>
<td>94.46%</td>
<td>92.12%</td>
<td>89.19%</td>
<td>91.60%</td>
</tr>
</tbody>
</table>

Table 4: Ablation study showing MRP scores of different configurations on EDS. The top row contains the submitted configuration without any changes; then we report the results for 1) label classifier without the mixture of softmaxes (MoS); 2) label loss not multiplied by the focal loss coefficient; 3) encoder with finetuned BERT-large (English) instead of multilingual XLM-R and 4) constant loss weights, equally set to 1.0.

<table>
<thead>
<tr>
<th>System</th>
<th>AMR\textsuperscript{eng}</th>
<th>EDS\textsuperscript{eng}</th>
<th>UCCA\textsuperscript{eng}</th>
</tr>
</thead>
<tbody>
<tr>
<td>best from MRP 2019</td>
<td>73.11%</td>
<td>92.55%</td>
<td>82.61%</td>
</tr>
<tr>
<td>UFAL PERIN*</td>
<td>78.43%</td>
<td>95.17%</td>
<td>82.71%</td>
</tr>
</tbody>
</table>

Table 5: The last year’s shared task had three frameworks – English AMR, EDS and UCCA – in common with MRP 2020. All parsers were evaluated on The Little Prince dataset, the first row shows the F1 scores of the best performing parser for each framework (Oepen et al., 2019).

identical to the system by Hitachi (Ozaki et al., 2020). Our bugfixed submission reached the first rank in both tracks, improving the cross-lingual score by nearly one percent point. Our system excels in label prediction, which might suggest the effectiveness of the relative label encoding. Furthermore, our system surpasses the best systems from the last year’s semantic shared task, MRP 2019 (Oepen et al., 2019), by a wide margin – as can be seen in Table 5.

PERIN falls short in AMR\textsuperscript{eng} parsing by 1.31%. On closer inspection, this follows from the inferior edge accuracy on this framework – the difference to Hitachi is 4.56% on AMR\textsuperscript{eng} and 2.78% on AMR\textsuperscript{zh}. Furthermore, Hitachi is better in all aspects of EDS\textsuperscript{eng} and DRC\textsuperscript{deu}. On the other hand PERIN consistently beats Hitachi in both PTG and both UCCA frameworks. We hope that combining the strengths of these two parsers will help to further advance the state of meaning representation parsing.

4.1 Ablation Experiments

We conducted several additional experiments to evaluate the effects of various components of our architecture. The results are summarized in Table 4. We have decided to use EDS for these experiments because – in our eyes – it represents the “average” framework without any significant irregularities.

The experiments show that using the mixture of softmaxes for label prediction does not have a substantial effect and can be potentially omitted to reduce the parameter count. On the other hand, the inferior results of the model with constant equal loss weights demonstrate the importance of balancing them.

5 Conclusion

We introduced a novel permutation-invariant sentence-to-graph semantic parser called PERIN. Given its state-of-the-art performance across a number of frameworks, we believe permutation-invariant node prediction might be the first step in a promising direction of semantic parsing and generally of graph generation.

Acknowledgments

This work has been supported by the Grant Agency of the Czech Republic, project EXPLORE LUSyD (GX20-16891X). It has also been supported by the Ministry of Education, Youth and Sports of the Czech Republic, Project No. LM2018101 LINDAT/CLARIAH-CZ.
References


Jan Hajic, Eva Hajicová, Iarmila Panevová, Petr Šgall, Ondrej Bojar, Silvie Cinková, Eva Fucíková, Marie


