HUJI-KU at MRP 2020: Two Transition-based Neural Parsers

Ofir Arviv[*], **Ruixiang Cui**[**], and **Daniel Hershcovich**[**]

[*]Hebrew University of Jerusalem, School of Computer Science and Engineering
[**]University of Copenhagen, Department of Computer Science
ofir.arviv@mail.huji.ac.il, {rc,dh}@di.ku.dk

Abstract

This paper describes the HUJI-KU system submission to the shared task on Cross-Framework Meaning Representation Parsing (MRP) at the 2020 Conference for Computational Language Learning (CoNLL), employing TUPA and the HIT-SCIR parser, which were, respectively, the baseline system and winning system in the 2019 MRP shared task. Both are transition-based parsers using BERT contextualized embeddings. We generalized TUPA to support the newly-added MRP frameworks and languages, and experimented with multitask learning with the HIT-SCIR parser. We reached 4th place in both the cross-framework and cross-lingual tracks.

1 Introduction

The CoNLL 2020 MRP Shared Task (Oepen et al., 2020) combines five frameworks for graph-based meaning representation: EDS, PTG, UCCA, AMR and DRG. It further includes evaluations in English, Czech, German and Chinese. While EDS, UCCA and AMR participated in the 2019 MRP shared task (Oepen et al., 2019), which focused only on English, PTG and DRG are newly-added frameworks to the MRP uniform format.

For this shared task, we extended TUPA (Hershcovich et al., 2017), which was adapted as the baseline system in the 2019 MRP shared task (Hershcovich and Arviv, 2019), to support the two new frameworks and the different languages. In order to add this support, only minimal changes were needed, demonstrating TUPA's strength in parsing a wide array of representations. TUPA is a general transition-based parser for directed acyclic graphs (DAGs), originally designed for parsing UCCA (Abend and Rappoport, 2013). It was previously used as the baseline system in SemEval 2019 Task 1 (Hershcovich et al., 2019), and generalized to support other frameworks (Hershcovich et al., 2018a,b).

We also experimented with the HIT-SCIR parser (Che et al., 2019). This was the parser with the highest average score across frameworks in the 2019 MRP shared task, and has also since been applied to other frameworks (Hershcovich et al., 2020).

2 TUPA-MRP

TUPA (Hershcovich et al., 2017) is a transition-based parser supporting general DAG parsing. The parser state is composed of a buffer B of tokens and nodes to be processed, a stack S of nodes currently being processed, and an incrementally constructed graph G. The input to the parser is a sequence of tokens: $w_1, \dots, w_n$. A classifier is trained using an oracle to select the next transition based on features encoding the parser's current state, where the training objective is to maximize the sum of log-likelihoods of all gold transitions at each step.

The MRP variant (Hershcovich and Arviv, 2019) supports node and edge labels, as well as node properties and edge attributes. The code is publicly available.[1]

2.1 Transition set

The TUPA-MRP transition set, shown in Figure 1, is the same as the one used by Hershcovich and Arviv (2019). It includes the transitions SHIFT and REDUCE to manipulate the stack, NODE$_X$ to create nodes compositionally, CHILD$_X$ to create unanchored children, LABEL$_X$ to label nodes, PROPERTY$_X$ to set node properties, LEFT-EDGE$_X$ and RIGHT-EDGE$_X$ to create edges, ATTRIBUTE$_X$ to set edge attributes, SWAP to allow non-planar graphs and FINISH to terminate the sequence.

[1] https://github.com/danielhers/tupa/tree/mrp

Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing, pages 73–82
Online, Nov. 19-20, 2020. ©2020 Association for Computational Linguistics

Before Transition				Transition	After Transition				
Stack	**Buffer**	**N.**	**Edges**		**Stack**	**Buffer**	**Nodes**	**Edges**	**Extra Effect**
S	$x \mid B$	V	E	SHIFT	$S \mid x$	B	V	E	
$S \mid x$	B	V	E	REDUCE	S	B	V	E	
$S \mid x$	B	V	E	NODE$_X$	$S \mid x$	$y \mid B$	$V \cup \{y\}$	$E \mid (y,x)$	$\ell_E(y,x) \leftarrow X$
$S \mid x$	B	V	E	CHILD$_X$	$S \mid x$	$y \mid B$	$V \cup \{y\}$	$E \mid (x,y)$	$\ell_E(x,y) \leftarrow X$
$S \mid x$	B	V	E	LABEL$_X$	$S \mid x$	B	V	E	$\ell_V(x) \leftarrow X$
$S \mid x$	B	V	E	PROPERTY$_X$	$S \mid x$	B	V	E	$p(x) \leftarrow p(x) \cup \{X\}$
$S \mid y,x$	B	V	E	LEFT-EDGE$_X$	$S \mid y,x$	B	V	$E \mid (x,y)$	$\ell_E(x,y) \leftarrow X$
$S \mid x,y$	B	V	E	RIGHT-EDGE$_X$	$S \mid x,y$	B	V	$E \mid (x,y)$	$\ell_E(x,y) \leftarrow X$
S	B	V	$E \mid (x,y)$	ATTRIBUTE$_X$	S	B	V	$E \mid (x,y)$	$a(x,y) \leftarrow a(x,y) \cup \{X\}$
$S \mid x,y$	B	V	E	SWAP	$S \mid y$	$x \mid B$	V	E	
[root]	$\emptyset$	V	E	FINISH	$\emptyset$	$\emptyset$	V	E	terminal state

Figure 1: The TUPA-MRP transition set, from Hershcovich and Arviv (2019). We write the stack with its top to the right and the buffer with its head to the left; the set of edges is also ordered with the latest edge on the right. NODE, LABEL, PROPERTY and ATTRIBUTE require that $x \neq \text{root}$; CHILD, LABEL, PROPERTY, LEFT-EDGE and RIGHT-EDGE require that $x \notin w_{1:n}$; ATTRIBUTE requires that $y \notin w_{1:n}$; LEFT-EDGE and RIGHT-EDGE require that $y \neq \text{root}$ and that there is no directed path from y to x; and SWAP requires that $\mathrm{i}(x) < \mathrm{i}(y)$, where $\mathrm{i}(x)$ is a running index for nodes. ℓ_E and ℓ_V are respectively the edge and node labeling functions. $p(x)$ is the set of node x's properties, and $a(x,y)$ is the set of edge (x,y)'s attributes.

2.2 Transition Classifier

To predict the next transition at each step, TUPA uses a BiLSTM module followed by an MLP and a softmax layer for classification (Kiperwasser and Goldberg, 2016). The BiLSTM module is applied before the transition sequence starts, running over the input tokenized sequence. It consists of a pre-BiLSTM MLP with feature embeddings (§2.3) and pre-trained contextualized BERT (Devlin et al., 2019) embeddings concatenated as inputs, followed by (multiple layers of) a bidirectional recurrent neural network (Schuster and Paliwal, 1997; Graves, 2008) with a long short-term memory cell (Hochreiter and Schmidhuber, 1997).

Whenever a LABEL$_X$/PROPERTY$_X$/ATTRIBUTE$_X$ transition is selected, an additional classifier is evoked with the set of possible label/property/attribute values for the currently parsed framework, respectively, as possible outputs. This hard separation is made due to the large number of node labels and properties in the MRP frameworks.

2.3 Features

In both training and testing, we use vector embeddings representing the lemmas, coarse POS tags (UPOS) and fine-grained POS tags (XPOS). These feature values are provided by UDPipe as companion data by the task organizers. In addition, we use punctuation and gap type features (Maier and Lichte, 2016), and previously predicted node and edge labels, node properties, edge attributes and parser actions. These embeddings are initialized randomly (Glorot and Bengio, 2010).

To the feature embeddings, we concatenate numeric features representing the node height, number of parents and children, and the ratio between the number of terminals to total number of nodes in the graph G (Hershcovich et al., 2017). Numeric features are taken as they are, whereas categorical features are mapped to real-valued embedding vectors. For each non-terminal node, we select a *head terminal* for feature extraction, by traversing down the graph, selecting the first outgoing edge each time according to alphabetical order of labels.

2.4 Intermediate Graph Representation

We mostly reuse Hershcovich and Arviv (2019)'s internal representation of MRP graphs in TUPA, where top nodes and anchoring are combined into the graph by adding a virtual root node and virtual terminal nodes, respectively, during preprocessing. Similarly, we introduce placeholders in the node labels and properties matching the tokens they are aligned to, and collapse AMR name properties. In the case of DRG and PTG, the newly added frameworks, where graphs may contains cycles, we break those cycles in order for them to be parseable by TUPA, which supports general DAG parsing. Only 0.27% of the DRG graphs in the provided dataset are cyclic. In the case of PTG, 33.97% are cyclic. Figure 2 shows an example PTG graph, and Figure 3 the graph in TUPA's intermediate representation. As the latter demon-

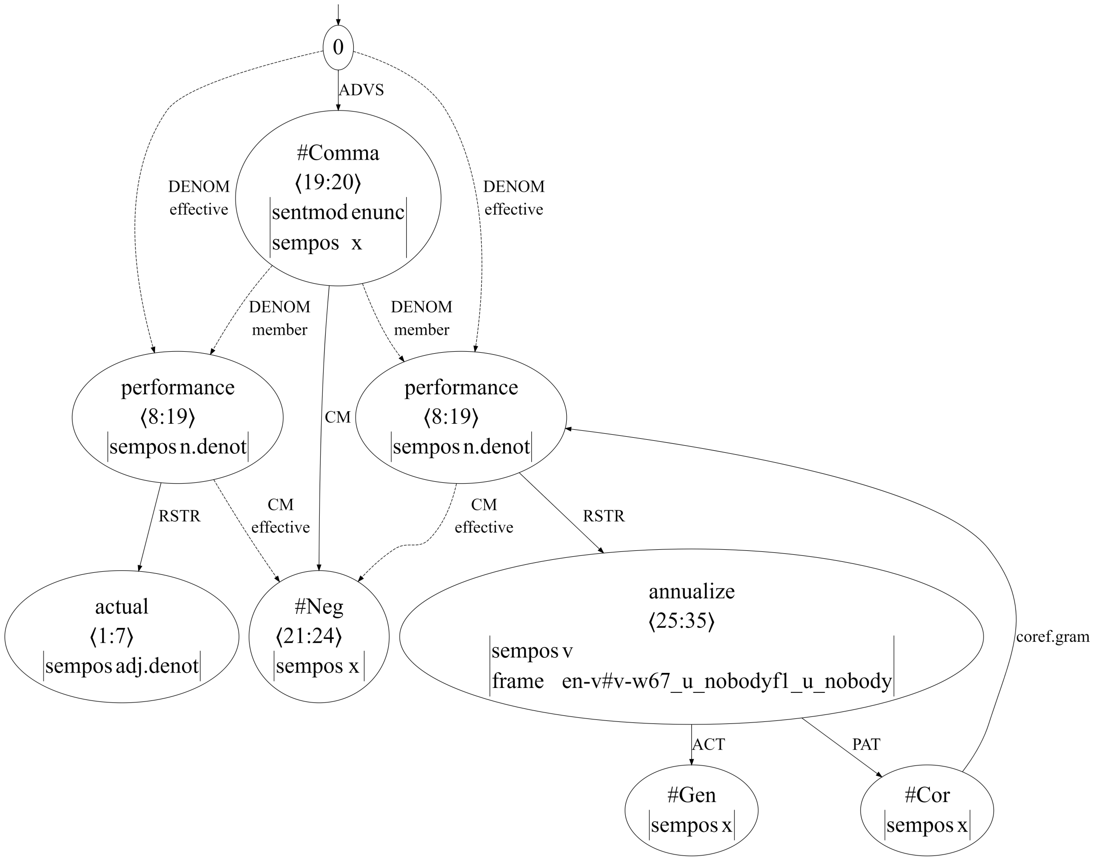

Figure 2: PTG graph, in the MRP formalism, for the sentence "*Actual performance, not annualized". Edge labels are shown on the edges. Node labels are shown inside the nodes, along with any node properties (in the form |property value|). Anchoring is also provided for PTG.

strates, cycles are broken by removing an arbitrary edge in the cycle (the coref.gram edge in this case).

2.5 Constraints

As each framework has different constraints on the allowed graph structures, we apply these constraints separately for each one. During training and parsing, the relevant constraint set rules out some of the transitions according to the parser state.

Some constraints are task-specific, others are generic. For the new frameworks, DRG and PTG, all the constraints, except for one (PTG being multigraph), are derived from the graph properties as defined by their component pieces.[2] For example, both require node labels, but only PTG requires node properties. No new types of constraints were needed to be added to TUPA to support these frameworks.

2.6 Training details

The model is implemented using DyNet v2.1 (Neubig et al., 2017).[3] Unless otherwise noted, we use the default values provided by the package. We use the same hyperparameters as Hershcovich and Arviv (2019), without any hyperparameter tuning on the CoNLL 2020 data.

We use the weighted sum of last four hidden layers of a BERT (Devlin et al., 2019) pre-trained model[4] as extra input features, summing over wordpiece vectors to get word representations.

2.7 Cross-framework track

In the cross-framework track, we use the English bert-large-cased pre-trained encoder, and train separate TUPA models for each of the PTG, UCCA, AMR and DRG frameworks. Table 1 shows the number of training epochs per framework, as well as validation and evaluation results.

[2] http://mrp.nlpl.eu/2020/index.php?page=15

[3] http://dynet.io

[4] https://github.com/huggingface/pytorch-transformers

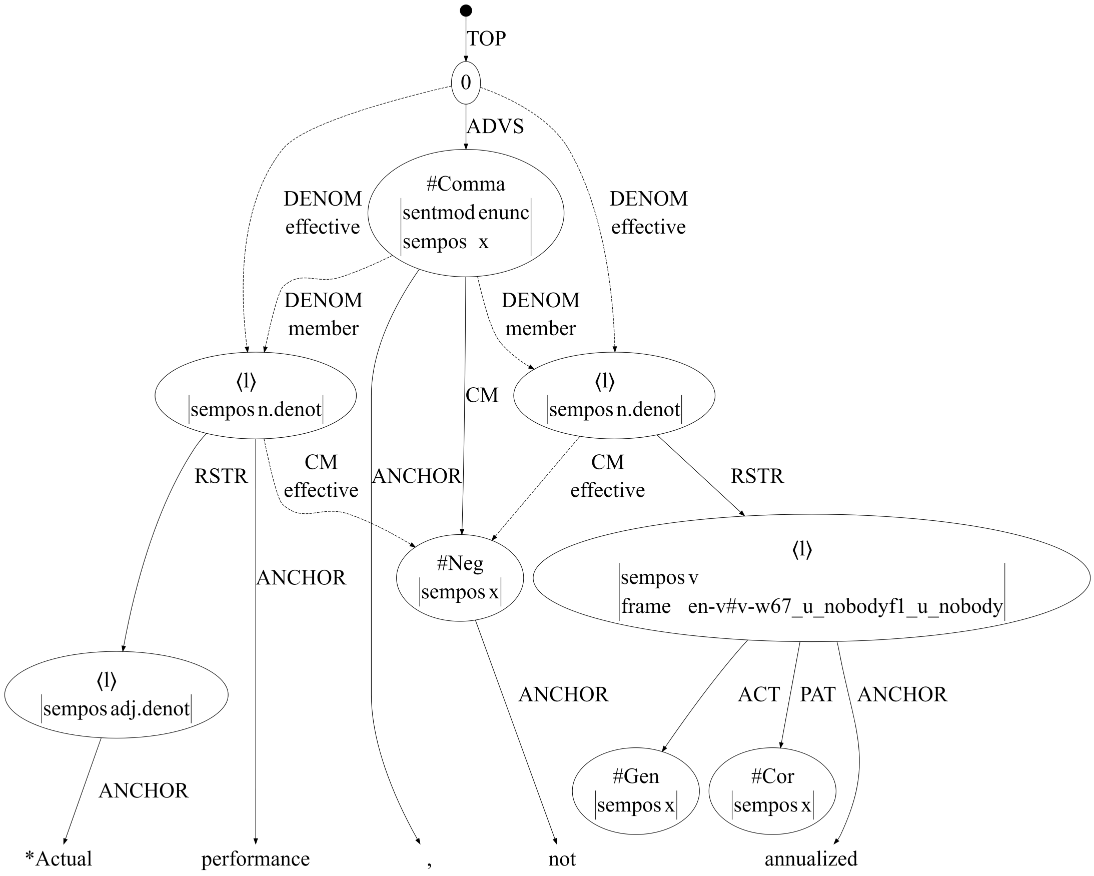

Figure 3: Converted PTG graph in the TUPA intermediate graph representation. Same as in the intermediate graph representation for all frameworks, it contains a virtual root node attached to the graph's top node with a TOP edge, and virtual terminal nodes corresponding to text tokens, attached according to the anchoring with ANCHOR edges. Same as for all frameworks with node labels and properties (i.e., all but UCCA), labels and properties are replaced with placeholders corresponding to anchored tokens, where possible. The placeholder $\langle\ell\rangle$ corresponds to the concatenated lemmas of anchored tokens. For graphs containing cycles, like this one, the cycles are broken by removing an arbitrary edge in the cycle (the `coref.gram` edge in this case).

2.8 Cross-lingual track

For the cross-lingual track, as a generic contextualized encoder that supports many languages, we use multilingual BERT (`bert-base-multilingual-cased`) and train the models exactly the same as in the cross-framework track (separate model for each framework's respective monolingual dataset from the cross-lingual track), for Czech PTG and Chinese AMR.

For German DRG, as the provided dataset contains a relatively small amount of examples, 1575 as opposed to 6606 in English DRG (from the cross-framework track), we first pre-train a model on the DRG data in English and then fine-tune it on the DRG German dataset, in this case using mBERT to facilitate cross-lingual transfer. Surprisingly, this improves our validation F1 score only by 0.013 points as opposed to training on the German dataset only, showing that the contribution of cross-lingual transfer is limited (but at least not detrimental) with this architecture and data sizes.

3 HIT-SCIR Parser

The HIT-SCIR parser (Che et al., 2019) is a transition-based parser, which extended previous parsers by employing stack LSTM (Dyer et al., 2015) to allow computing homogeneous operation within a batch efficiently, and by adopting and fine-tuning BERT (Devlin et al., 2019) embedding for effectively encoding contextual information. The parser is implemented in the AllenNLP framework (Gardner et al., 2018). It supports parsing DM, PSD, UCCA, EDS and AMR, all included in the 2019 MRP shared task. The official dataset would be pre-processed for system input and post-processed for output.

In our experiment, we modified the HIT-SCIR

Track	Framework	System	# Epochs	Best Epoch	Validation F1	Eval F1	Rank	Best System
CF	EDS	HIT-SCIR	6	2	0.82	0.80	5	0.94 (H)
CF	PTG	TUPA	32	19	0.53	0.54	4	0.89 (H)
CF	UCCA	TUPA	99	66	0.79	0.73	4	0.76 (Ú)
CF	UCCA	HIT-SCIR	6	3	0.78			
CF	AMR	TUPA	8	2	0.44	0.52	5	0.82 (H)
CF	DRG	TUPA	200	99	0.52	0.63	5	0.94 (Ú)
CL	PTG	TUPA	20	13	0.60	0.58	4	0.91 (Ú)
CL	UCCA	HIT-SCIR	13	6	0.77	0.75	4	0.81 (Ú)
CL	UCCA	TUPA	100	95	0.43			
CL	AMR	TUPA	21	12	0.44	0.45	4	0.80 (H)
CL	DRG	TUPA	100	(*) 68	0.52	0.62	4	0.93 (H)
CL	DRG	TUPA	100	81	0.51			
CF	Overall					0.64	4	0.86 (H&Ú)
CL	Overall					0.60	4	0.85 (H&Ú)

Table 1: Training details and official evaluation MRP F-scores. For comparison, the highest score achieved for each framework and evaluation set is shown: H stands for Hitachi (Ozaki et al., 2020) and Ú for ÚFAL (Na and Min, 2020). HIT-SCIR for English UCCA (CF) and TUPA for German UCCA (CL), both in gray, were not used in the submission, since their validation F1 were lower than the other system. For German DRG (CL) we trained 2 parsers: one on only the CL DRG dataset (in grey), not used in the submission, and another (*) trained on the English DRG dataset in per-training. The number of epochs does not include pre-training on English DRG.

MRP 2019 parser to support the 2020 data for English EDS (for the cross-framework track) and German UCCA (for the cross-lingual track). We also explored the possibilities of employing multi-task learning with the parser (§5). A repository containing our modified version of the parser is publicly available.[5]

3.1 Transition set

Che et al. (2019) defined a different transition set per framework, according to framework's characteristics. As UCCA and EDS are already targets of 2019 MRP shared task, we inherit the existing transition sets for both frameworks. For UCCA, the transition system was modelled after that of the UCCA-specific (not MRP generic) TUPA (Hershcovich et al., 2017), which includes Shift, Reduce, Node$_X$, Left-Edge$_X$, Right-Edge$_X$, Left-Remote$_X$, Right-Remote$_X$ and Swap.

The parser's EDS transition set is based on Buys and Blunsom (2017)'s work, from which Node-Start$_X$ and Node-End are two steps to create concept nodes and form node alignment. Apart from these two, Shift, Reduce, Left-Edge$_X$, Right-edge$_X$, Drop, Pass and Finish are also used to represent EDS transition process.

3.2 Transition Classifier

The parser state is represented by (S, L, B, E, V), where S is a stack holding processed words, L is a list holding words popped out of S that will be pushed back in the future, and B is a buffer holding unprocessed words. E is a set of labeled dependency arcs. V is a set of graph nodes include concept nodes and surface tokens. Transition classifier takes S, L, B and also the action history as input, all are modeled with stack LSTM, and outputs an action. The input to the parser is a sequence of BERT embedding. A transition classifier takes S, L, B and the action history as inputs and maximizes the log-likeihood of the correct action given the current state using an oracle to get the correct action.

3.3 Preprocessing

MRP 2019 provided companion data (containing the results of syntactic preprocessing) in both `CoNLL-U` and `mrp` formats. However, this year's task only provides `mrp`-formatted companion data. Since the HIT-SCIR 2019 parser can only take `CoNLL-U`-formatted companion data, we update it to allow converting companion data provided by 2020 MRP shared task from `mrp` format to `CoNLL-U` format.

[5] https://github.com/ruixiangcui/hit-scir-mrp2020

3.4 Anchoring

The parser itself is also modified to support the MRP 2020 task. For EDS parsing specifically, in this year's task's provided data, anchoring for a token containing spaces, such as an integer number followed by a fraction number (e.g., "3 1/2") is treated as one token, while the original parser's node anchoring treats the two parts separately. Another example would be: "x-Year-to-date 1988 figure includes Volkswagen domestic-production through July." In this sentence, "x-Year-to-date 1988" is marked as a node anchored from characters 2 to 26, but the provided companion data treats "x-Year-to-date" as anchored from characters 0 to 14 as the corresponding token anchor. To handle these cases, we allow the parsing system to perform partial node alignment regardless of overlapping token anchors.

3.5 Constraints

The second problem we encounter when parsing EDS is that there are a few instances that are too short, and no valid actions can be performed according to the existing transition system. In this case, we allow the FINISH action, adding it directly to the allowed action set when no valid action exists, with the effect that the transition sequence is terminated and the current graph is returned.

3.6 Training

We train the modified HIT-SCIR parser on English and German UCCA (in the cross-framework and cross-lingual tracks, respectively) and English EDS (in the cross-lingual track). The training time is 2 days 1 hour for English UCCA, 22 hours for German UCCA, and 4 days 6 hours for English EDS. The training details are shown in Table 1. Since HIT-SCIR parser's validation score on cross-framework UCCA is 0.01 lower than TUPA, we opt for TUPA in that category. Hyperparameters are taken directly from Che et al. (2019).

4 Results

Table 1 presents the averaged scores on the test sets in the official evaluation, for our submission and for the best-performing system in each framework and evaluation set.

Validation vs. evaluation scores. The validation scores of 5 out of the 9 parsers is lower than their evaluation score: CF PTG by 0.01 F1 points, CF AMR by 0.08, CF DRG by 0.11, CL AMR by 0.01 and CL DRG by 0.1. We hypothesize it is due to the randomness in the evaluation metric: the MRP scorer uses a search algorithm to find a correspondence relation between the gold-standard and system graphs that maximizes tuple overlap. This search algorithm runs for a limited number of iterations. In order to decrease its running time, we used a lower limit on its parameters (10 random restarts, 5,000 iterations) than the default (20 random restarts, 50,000 iterations), which may have affected the accuracy of our validation score and potentially our system performance.

CF vs. CL tracks. Surprisingly, the CL track scores are mostly on-par with the CF tack ones, even though the CL parsers were often trained on significantly less examples. While the CF UCCA training dataset contains 6,872 examples and the CL UCCA contains only 3,713, both parsers gained similar scores. Similarly, the CF DRG dataset contains 6,606 example, while the the CL DRG contains only 1,575. TUPA trained only on the 1,575 examples gained a similar score to the CF one, while training on less then a fourth of the examples. The CF PTG dataset contains 42,024 examples. And while the CL PTG contains a lower, however similar, amount (39,560), it got a higher score (0.07 F1 point in validation, and 0.04 in evaluation). And while the CL AMR dataset is only a third of the CF AMR datsaet (16,529 and 57,885 examples respectively), both parser gained the same validation score. However, the evaluation score of the CF AMR is higher by 0.07 F1 points. This could be possibly attributed to our MRP scorer low iteration limit.

5 Multitask Cross-Framework Parsing

In addition to training separate models per framework and language, we also experiment with training multitask cross-framework parsers, using a neural architecture with parameter sharing (Peng et al., 2017, 2018; Hershcovich et al., 2018a; Lindemann et al., 2019; Hershcovich and Arviv, 2019). We use the HIT-SCIR parser as a basis, with different variations of shared architecture on top of it. For our experiments we choose the UCCA and EDS frameworks. The code is pub-

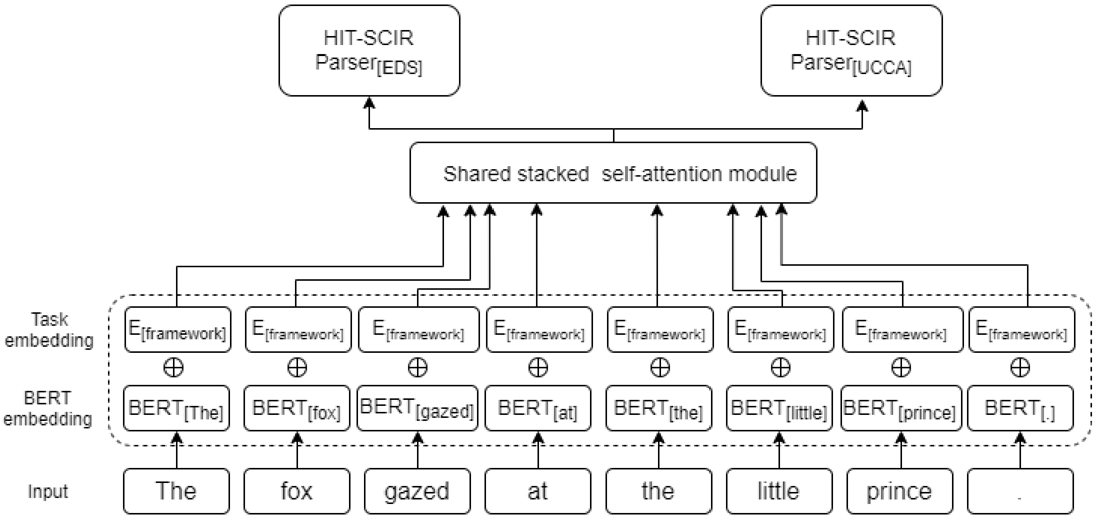

Figure 4: Illustration of the first variant of the HIT-SCIR multitask model, parsing the sentence "The fox gazed at the little prince." Top: Dedicated HIT-SCIR parsers for each framework. Bottom: Encoder architecture. BERT embeddings are extracted for each token and are concatenated with framework-specific learned embedding. Vector representations for the input tokens are then computed by a shared stacked self-attention encoder. The encoded vectors are then fed to a framework-specific HIT-SCIR parsers as input tokens.

Hyperparameter	**Value**
Task embedding dim	20
Shared encoder	
Input dim	1024
Framework-specific encoder	
Input dim	768
Both encoders	
Input dim	768
Projection dim	512
Feedforward hidden dim	512
# layers	3
# attention heads	8

Table 2: HIT-SCIR multitask model hyperparameters.

licly available.[6]

5.1 Model

We try two different sharing architectures. In both architectures, both frameworks share a stacked self-attention encoder (see Table 2 for details). In the first variation, we additionally use task embeddings; in the second, we use task-specific encoders instead.

Task embedding. In the first sharing architecture, both frameworks share a stacked self-attention encoder whose input is a BERT embedding concatenated with a learned task embedding of dimension 20. This has been shown to help in shared architecture multitask models (Sun et al., 2020), as well as cross-lingual parsing models, where a language embedding is used (Ammar et al., 2016; de Lhoneux et al., 2018). In our case, the "task" has two possible values, namely UCCA and EDS. The output of the shared encoder is then fed into two separate "decoders", which are HIT-SCIR parser transition classifiers. We use one for each framework, whose architecture and hyperparameters are the same as in the single task setting. Figure 4 illustrates this architecture.

Task-specific encoders. In the second architectures, both frameworks share a stacked self-attention encoder whose input is a BERT embedding, and in addition each framework has another stacked self-attention encoder of it own, similar in concept to Peng et al. (2017, 2018)'s FREDA1 architecture (which, however, used BiLSTMs), also employed by Hershcovich et al. (2018a); Lindemann et al. (2019). The outputs of these encoders are processed the same as in the first variation (task-specific decoders). Figure 5 illustrates this architecture.

5.2 Training details

Each training batch contains examples from a single framework, while the model is alternating between the different batch types. As the EDS training dataset is much larger than the UCCA one,

[6] https://github.com/OfirArviv/hit-scir-mrp2020/tree/multitask

			Validation	Validation	Validation
Sharing architecture	**# Epochs**	**Best Epoch**	**Average F1**	**UCCA F1**	**EDS F1**
Shared encoder					
+ task embedding	13	2	0.55	0.68	0.43
+ task specific encoders	13	4	0.38	0.49	0.27

Table 3: HIT-SCIR multitask model training details and scores.

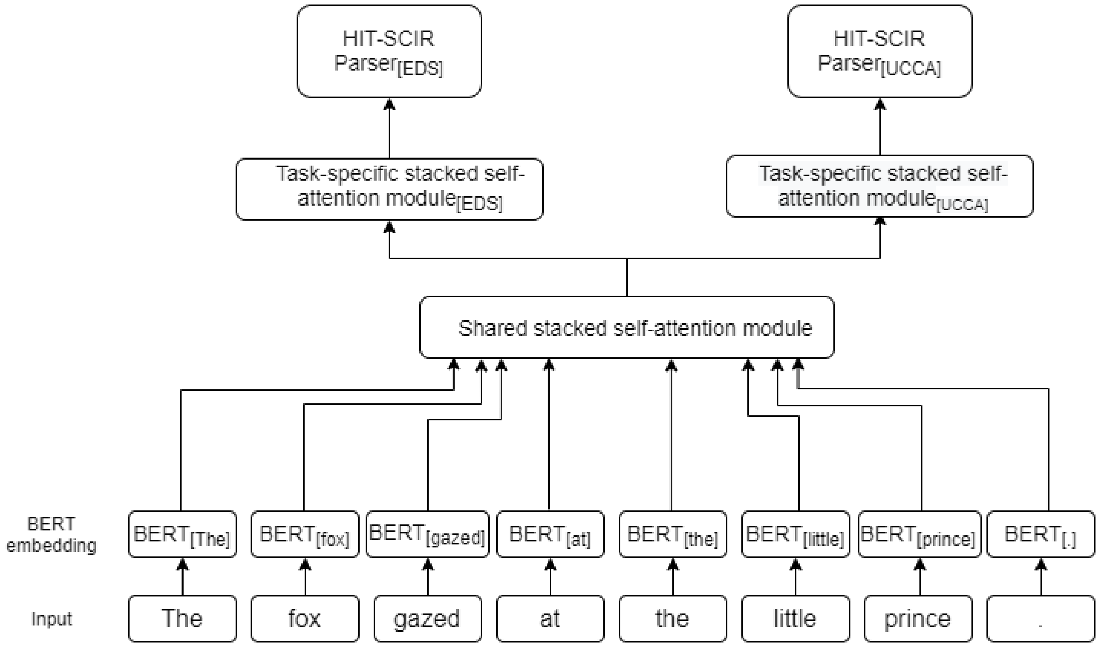

Figure 5: Illustration of the second variant of the HIT-SCIR multitask model, parsing the sentence "The fox gazed at the little prince." Top: Dedicated HIT-SCIR parsers for each framework. Bottom: Encoder architecture. BERT embeddings are extracted for each token. Vector representations for the input tokens are then computed by a shared stacked self-attention encoder and by a framework-specific self-attention encoder. The encoded vectors are then fed to a framework-specific HIT-SCIR parsers as input tokens.

we balance them out by training the same number of examples from each framework in each epoch. Due to time constraints we tried out only a single set of hyperparameters, chosen arbitrarily without tuning. We select the epoch with the best average MRP F-score on a validation set, which is the union of both validation sets of EDS and UCCA.

5.3 Results

Table 3 presents the average scores on the validation sets for multitask trained models. The multitask HIT-SCIR consistently falls behind the single-task one, for each framework separately and in the overall scores; but it is clear that our first multitask architecture (with task embedding) outperforms the second one (with task-specific encoders).

5.4 Discussion

Previous results on multitask MRP showed mixed results, some showing improved performances (Peng et al., 2017; Hershcovich et al., 2018a; Lindemann et al., 2019). Others failed to show improvements (Hershcovich and Arviv, 2019), and argued that the large multitask models were underfitting due to insufficient training. In our case, however, the multitask models underperform despite reaching convergence.

We hypothesize that with better hyperparameters or different sharing architectures, more favorable results could be obtained. However, it is possible that multitask learning would be more helpful in a factorization-based parser (Peng et al., 2017; Lindemann et al., 2019), where inference is global and more uniform across frameworks. A transition-based parser may be less suited for utilizing information from different tasks that have different transition systems, as in the HIT-SCIR parser. Adapting it to have a more uniform transition system, like TUPA does, could facilitate cross-framework sharing. Alternatively, improving TUPA's training efficiency would also enable such experimentation.

6 Conclusion

We have presented TUPA-MRP and a modified HIT-SCIR parser, which constitute the HUJI-KU submission in the CoNLL 2020 shared task on Cross-Framework Meaning Representation. TUPA is a general transition-based DAG parser with a uniform transition system, which is easily adaptable for multiple frameworks. We used it for parsing in both the cross-framework and the cross-lingual tracks, adapting it for the newly introduced frameworks, PTG and DRG. HIT-SCIR is a transition-based parser with framework-specific transition systems, which we adapted for this year's shared task and used for English EDS and UCCA parsing in the cross-framework track. The HIT-SCIR parser was additionally used in experimenting on multitask learning, with negative results for that approach.

Future work will tackle the MRP task with more modern transition-based-like parser architectures, such as pointer networks (Ma et al., 2018), which have so far only been applied to bilexical framworks, i.e., flavor-0 SDP (Fernández-González and Gómez-Rodríguez, 2020).

Acknowledgments

We are grateful for the valuable feedback from the anonymous reviewers.

References

Omri Abend and Ari Rappoport. 2013. Universal Conceptual Cognitive Annotation (UCCA). In *Proc. of ACL*, pages 228–238.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Many languages, one parser. *Transactions of the Association for Computational Linguistics*, 4:431–444.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1215–1226, Vancouver, Canada. Association for Computational Linguistics.

Wanxiang Che, Longxu Dou, Yang Xu, Yuxuan Wang, Yijia Liu, and Ting Liu. 2019. HIT-SCIR at MRP 2019: A unified pipeline for meaning representation parsing via efficient training and effective encoding. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 76–85, Hong Kong. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL*, pages 4171–4186.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependeny parsing with stack long short-term memory. In *Proc. of ACL*, pages 334–343.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Transition-based semantic dependency parsing with pointer networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7035–7046, Online. Association for Computational Linguistics.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Alex Graves. 2008. Supervised sequence labelling with recurrent neural networks. *Ph. D. thesis*.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for UCCA. In *Proc. of ACL*, pages 1127–1138.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018a. Multitask parsing across semantic representations. In *Proceedings of the 56th Meeting of the Association for Computational Linguistics*, pages 373 – 385, Melbourne, Australia.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018b. Universal dependency parsing with a general transition-based DAG parser. In *Proc. of CoNLL UD Shared Task*, pages 103–112.

Daniel Hershcovich, Zohar Aizenbud, Leshem Choshen, Elior Sulem, Ari Rappoport, and Omri Abend. 2019. SemEval-2019 task 1: Cross-lingual semantic parsing with UCCA. In *Proc. of SemEval*, pages 1–10.

Daniel Hershcovich and Ofir Arviv. 2019. TUPA at MRP 2019: A multi-task baseline system. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 28–39, Hong Kong. Association for Computational Linguistics.

Daniel Hershcovich, Miryam de Lhoneux, Artur Kulmizev, Elham Pejhan, and Joakim Nivre. 2020. Køpsala: Transition-based graph parsing via efficient training and effective encoding. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 236–244, Online. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*, 4:313–327.

Miryam de Lhoneux, Johannes Bjerva, Isabelle Augenstein, and Anders Søgaard. 2018. Parameter sharing between dependency parsers for related languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4992–4997, Brussels, Belgium. Association for Computational Linguistics.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2019. Compositional semantic parsing across graphbanks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4576–4585, Florence, Italy. Association for Computational Linguistics.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

Wolfgang Maier and Timm Lichte. 2016. Discontinuous parsing with continuous trees. In *Proc. of Workshop on Discontinuous Structures in NLP*, pages 47–57.

Seung-Hoon Na and Jinwoo Min. 2020. JBNU at MRP 2020: AMR parsing using a joint state model for graph-sequence iterative inference. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 83 – 87, Online.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The dynamic neural network toolkit. *CoRR*, abs/1701.03980.

Stephan Oepen, Omri Abend, Lasha Abzianidze, Johan Bos, Jan Hajič, Daniel Hershcovich, Bin Li, Tim O'Gorman, Nianwen Xue, and Daniel Zeman. 2020. MRP 2020: The Second Shared Task on Cross-framework and Cross-Lingual Meaning Representation Parsing. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 1 – 22, Online.

Stephan Oepen, Omri Abend, Jan Hajič, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, Jayeol Chun, Milan Straka, and Zdeňka Urešová. 2019. MRP 2019: Cross-framework Meaning Representation Parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Computational Natural Language Learning*, pages 1 – 27, Hong Kong, China.

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, Online.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep multitask learning for semantic dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2037–2048, Vancouver, Canada. Association for Computational Linguistics.

Hao Peng, Sam Thomson, Swabha Swayamdipta, and Noah A. Smith. 2018. Learning joint semantic parsers from disjoint data. In *Proc. of NAACL-HLT*.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Y. Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, H. Wu, and Haifeng Wang. 2020. Ernie 2.0: A continual pre-training framework for language understanding. In *AAAI*.