BME-TUW at SR'20: Lexical grammar induction for surface realization

Gábor Recski[1], **Ádám Kovács**[1,2], **Kinga Gémes**[1,2], **Judit Ács**[2], **András Kornai**[3]

[1]TU Wien
firstname.lastname@tuwien.ac.at

[2]Dept. of Automation and Applied Informatics, Budapest U of Technology
lastname.firstname@aut.bme.hu

[3]SZTAKI Institute of Computer Science
andras@kornai.com

Abstract

We present a system for mapping Universal Dependency structures to raw text which learns to restore word order by training an Interpreted Regular Tree Grammar (IRTG) that establishes a mapping between string and graph operations. The reinflection step is handled by a standard sequence-to-sequence architecture with a biLSTM encoder and an LSTM decoder with attention. We modify our 2019 system (Kovács et al., 2019) with a new grammar induction mechanism that allows IRTG rules to operate on lemmata in addition to part-of-speech tags and ensures that each word and its dependents are reordered using the most specific set of learned patterns. We also introduce a hierarchical approach to word order restoration that independently determines the word order of each clause in a sentence before arranging them with respect to the main clause, thereby improving overall readability and also making the IRTG parsing task tractable. We participated in the 2020 Surface Realization Shared task, subtrack T1a (shallow, closed). Human evaluation shows we achieve significant improvements on two of the three out-of-domain datasets compared to the 2019 system we modified. Both components of our system are available on GitHub under an MIT license.

1 Introduction

This paper describes our participation in the T1a subtask of the 2020 Surface Realization Shared Task (Mille et al., 2020), which requires systems to map Universal Dependency graphs of lemmata to raw sentences by reconstructing word forms and word order. We extend the word order restoration component of Kovács et al. (2019) by making it sensitive to lemmata in addition to parts-of-speech and by performing the word order restoration step separately for each clause of a sentence. The paper is structured as follows: Section 2 briefly describes the data and the task as well as the architecture that we modify. Section 3 presents our new method for rule generation for creating lexical grammars, Section 4 describes the hierarchical approach for breaking up the word order restoration task into clause-level subtasks. Section 6 presents a summary of both automatic and human evaluations, Section 7 discusses potential future work. The IRTG-based system for word order restoration can be downloaded from `https://github.com/adaamko/surface_realization`, the inflection system was trained using the framework under `https://github.com/juditacs/deep-morphology`. Both components are available for download under an MIT license.

Proceedings of the Third Workshop on Multilingual Surface Realisation (MSR'20), pages 21–29
Barcelona, Spain (Online), December 12, 2020.

2 Background

2.1 SRST

The 2020 Surface Realization Shared Task (Mille et al., 2020) involves mapping Universal Dependency representations to raw text. The input data in the shallow track consists of UD-annotated sentences for 11 languages, without word order information and without inflected word forms, i.e. they are typed dependency graphs of lemmata of a sentence. The task is to reproduce the original sentences, which involves reconstructing word order and inflected word forms. The primary method of evaluation is human judgments of readability and fidelity of output sentences to the original, but automatic metrics are also used to measure the similarity between predicted and gold standard sentences.

The system presented in this paper, a substantial extension of the solution of Kovács et al. (2019), offers a fully rule-based, explainable, and language-independent solution for the word order restoration component of the shallow surface realization task and uses an independent ML system for reconstructing word forms. In last year's evaluation, the top rule-based systems were consistently outperformed by the best-performing IMS (Yu et al., 2019) and CMU (Du and Black, 2019) systems (see Mille et al. (2019) for evaluation details). Both of these solutions rely on deep learning models for the word order restoration component of the task. Still, an important similarity between their approaches and ours is that they also take the *divide-and-conquer* approach of determining word order separately for each head and all its dependents. In future work this will allow detailed comparison between the atomic decisions of our system and the state-of-the-art (see Section 7 for some proposals).

2.2 IRTGs for surface realization

The system presented in this paper is based on the architecture presented in Kovács et al. (2019), which involves the induction of an Interpreted Regular Tree Grammar (Koller, 2015) whose rules establish a mapping between operations of i) a string algebra that concatenates words and phrases of a sentence and ii) an algebra over graphs that builds UD structures by merging nodes and subgraphs. We briefly outline the architecture of this system and refer the reader to the original paper for details.

An IRTG rule is a (possibly weighted) rewrite rule of a Regular Tree Grammar that is mapped to an arbitrary number of named *interpretations*, each of which are operations of an algebra with the same arity as the RTG rule. Thereby any particular derivation of an IRTG grammar deterministically maps to a sequence of operations in each of the interpretation algebras. Parsing an object of one algebra involves finding the IRTG derivation(s) of the highest likelihood that would generate this object, while *decoding* is the subsequent construction of an object in another algebra using this sequence. The word reordering task is then equivalent to parsing graphs and decoding strings using an IRTG with two interpretations.

UD graphs have the property that the indegree of each node is exactly 1, i.e. each word is the dependent of exactly one other word in the sentence (this is achieved by introducing the dummy `ROOT` element that governs the root word in the sentence). This allows us to straightforwardly model the construction of UD graphs as a sequence of operations that each connect a word to all its dependents. The word order restoration task can then be modeled as the correspondence between a sequence of operations in a graph algebra building such an UD subgraph and the string operations concatenating the words of this subgraph in a particular order. Since the system operates on POS-tags, not word forms, an example would be the steps for building the UD subgraph `VERB` $\xrightarrow{\text{nsubj}}$ `NOUN`, which in a model for English would be expected to correspond to the concatenation operations yielding the string `NOUN VERB`. An IRTG rule encoding this correspondence can be seen in Figure 1: an RTG rule followed by its two interpretations, i.e. operations in a string algebra and an s-graph algebra as they are to be applied to the string and s-graph objects corresponding to each of the nonterminals on the right hand side of the RTG rule. For lack of space we do not describe the operation of the s-graph algebra in detail, but see the original system description (Kovács et al., 2019) for an overview and Koller and Kuhlmann (2011) or Courcelle and Engelfriet (2012) for more formal explanations. Grammars are built by inducing IRTG rules from sentences in the training data, and rule weights correspond to the observed frequencies of

individual patterns. Parsing and decoding are performed using the open-source `alto` library[1] (Gontrum et al., 2017).

```
VERB -> _nsubj(VERB, NOUN)
[string] *(?2, ?1)
[ud] f_dep1(merge(merge(?1, "(r<root> :nsubj d1<dep1>)"), r_dep1(?2)))
```

Figure 1: Example of an IRTG rule

Our main contribution is a number of substantial modifications of the grammar induction process of the above system, which we shall present in detail in Sections 3 and 4. First, the system presented above does not have access to lemmata and operates on graphs of POS-tags only, therefore it cannot learn about constructions that are typical of a lexical item but not its grammatical class. Second, its handling of unseen patterns is rudimentary: if a particular pattern of head POS and set of dependent POS tags has not been observed in the training data, the system will employ binary branching rules, i.e. the derivation will attach dependents to the head one-by-one. In the next sections we develop an architecture for inducing a lexical grammar that relies on lemma-level observations in the training data to cover each input UD graph with the most specific rules possible.

3 Lexical rule induction

3.1 Motivation

This section will describe our extension of the grammar induction system of Kovács et al. (2019) to include rules operating on lemmata in addition to those that refer to part-of-speech tags only. In modeling the construction of UD structures we preserve the fundamental assumption that the directed graphs of words are built as a series of operations which connect some set of dependents to their common head. In other words, the first argument of each rule in our grammar corresponds to a node from which outgoing dependency edges are to be added pointing to the nodes corresponding to the remaining arguments. The rule presented in Figure 1, which establishes an `nsubj` relation between a pair of words with the POS-tags `VERB` and `NOUN`, is recursive; the nonterminal `VERB` appearing on both sides of the RTG rule ensures that an arbitrary number of rules can be applied that connect dependents to the same head. Thus, all possible derivations of a given UD graph will have in common that for each word they eventually create the subgraph consisting of that word and all its dependents, but the number of possible RTG derivations that can generate a subgraph with $N + 1$ nodes is 2^N, while the number of IRTG derivations are doubly exponential since a rule with arity k can have $k!$ different string interpretations. The complexity of the parsing problem for a given sentence was reduced in the original system by including in the grammar only those patterns that we have observed in the training data (i.e. a rule such as that in Figure 1 will be added to our grammar if and only if the subgraph `VERB` $\xrightarrow{nsubj}$ `NOUN` was observed in the training data without additional edges outgoing from the verb head), a constraint that would most likely cause data sparsity issues if we were to extend the process from POS-tags to word forms without fundamental modifications.

Our new approach to grammar induction differs from that of Kovács et al. (2019) in multiple ways. First, by including rules that can take lemmas instead of POS-tags as some or all of its arguments, the number of patterns matching a subgraph of $N+1$ nodes increases from 2^N to 3^N and the number of IRTG rules that under our original approach would be included in a grammar pruned for a sentence of average size would no longer be tractable. On the other hand, we expect that derivations including rules that operate directly on word forms should generally take precedence as more specific accounts of the same phenomenon. This expectation is in part based on the observation in Kovács et al. (2019) that choosing shorter derivations, i.e. those using fewer rules with higher arity to generate the same graph actually yields better results on the word order restoration task by virtue of being more specific. An example for this was the sentence *I really enjoyed reading it* in the `en_ewt-ud-dev` dataset. For the corresponding

[1] `https://github.com/coli-saar/alto`

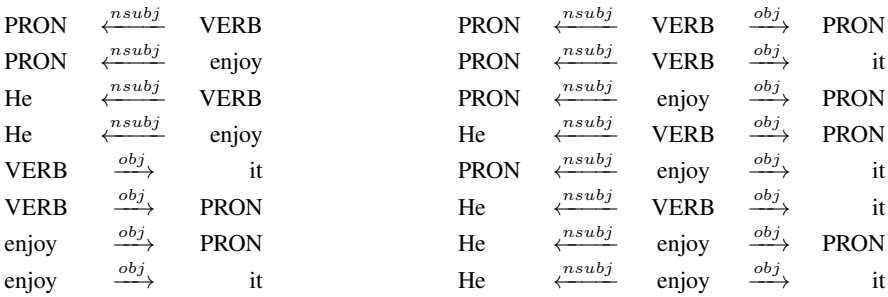

Figure 2: Subgraphs generated for the sentence *He enjoyed it*

UD graph the most likely (and correct) derivation included a rule connecting four dependents of the root verb simultaneously and constructing the sequence of POS-tags `PRP RB VBD VBG`, corresponding to the word order *I really enjoyed reading*. The next-best parse would have constructed the sentece *I enjoyed really reading it* using a separate rule to attach the edge `VBD` $\xrightarrow{advmod}$ `RB` that yields the word order *enjoy really*. When extending the grammar induction process to lemmata, we expect that such a rule could be overridden by a more specific one for adding *enjoy* $\xrightarrow{advmod}$ *really* that yields the correct word order based on what we observed in the training data. Indeed, not only does our final system yield the correct word order for this sentence, it does so using a rule that explicitly maps the subgraph *I* $\xleftarrow{nsubj}$ *enjoy* $\xrightarrow{advmod}$ *really* to the string *I really enjoy*, a construction observed five times in the training data.

3.2 Method

Our method for word order restoration involves a training step for counting all word- and POS-level subgraphs in the training data along with the corresponding word orders, a grammar construction step that for each input UD structure creates an IRTG with the most frequent rules corresponding to each of its subgraphs, and the parsing and decoding of the UD graphs using the constructed IRTG and the `alto` library. Our surface realization system performs the last two steps separately for each clause of each sentence, for details see Section 4. We now describe the three core steps in detail.

The training step involves traversing the UD graphs of all sentences in the training dataset for a given language. For each word in the graph with at least one dependent we take the subgraph containing this word and all its dependents, and for each of these we enumerate all its subgraphs that contain the head and at least one additional node. Finally for each of these subgraphs we generate all possible combinations such that some of the nodes are represented by their part-of-speech tags and the others by their lemma. For a head word with N dependents this means enumerating $2 \cdot (3^N - 1)$ subgraphs (since the head of the structure must be present in the subgraph and it must have at least one edge). Figure 2 shows an example of subgraphs generated for the simple 3-node graph *enjoy*/`VERB` $\xrightarrow{obj}$ *it*/`PRON`. The total number of unique subgraphs observed in the training data for each language is listed in Table 1. For each subgraph we then consider the relative order of the nodes in the surface form of the sentence and increment a counter corresponding to the particular ordering on the particular subgraph. Finally, for each pattern we sort all observed orderings by frequency.

At the time of performing word order restoration on an UD graph we enumerate all patterns in the input graph and query the model for the most frequent word order, allowing us to create an IRTG rule whose two interpretations correspond to the graph operations building the particular pattern and the string operations concatenating the corresponding words in the order that is most typical of that pattern based on the training data. The test dataset also specifies the relative order of some elements with respect to their heads using the `lin` feature. When such information is present, we use it to skip contradictory orderings regardless of their observed frequency. Patterns containing more than 4 dependents in addition to the head are discarded to reduce parsing complexity. We expect this to have negligible effect on parsing quality because patterns this large are likely the result of superposing subpatterns already inferable from the data. It is possible, for our method to yield a grammar that does not cover the input UD graph, this

Lang	N_{patt}	D_{max}	$\|V\|$	D_{words}	N_{tok}
ar	8.6M	4.8	14K	36.9	224K
en	29.8M	5.0	25K	17.6	352K
es	50.2M	5.5	48K	29.0	827K
fr	37.1M	5.7	34K	24.6	429K
hi	17.2M	5.5	15K	21.1	281K
id	7.0M	5.2	19K	21.8	98K
ja	14.5M	5.6	24K	22.5	160K
ko	8.6M	3.9	119K	12.9	353K
pt	27.2M	5.2	32K	25.7	462K
ru	41.6M	4.7	51K	18.0	946K
zh	14.8M	6.8	20K	24.7	99K

Table 1: Basic figures describing the training datasets: number of unique patterns (N_{patt}), average number of nodes in the largest pattern in a sentence (D_{max}), number of unique lemmata ($|V|$), average number of words in a sentence (D_{words}) and total tokens in the training data (N_{tok})

would mean that there is a graph edge that hasn't been observed in the training data even on the POS level. Since this affects less than 0.2% of all generated grammars across languages, we did not implement any fallback mechanism and in these few cases our system outputs the original randomized word order for the given sentence or clause.

Finally, we use the resulting IRTG grammar to parse the UD graph and decode the string constructed by the most likely derivation. We assign identical probabilities to all rules so that the search for the most likely parse will yield the derivation with the smallest number of rules. This means that we prefer rules that connect a larger number of dependents to their head and rules that directly refer to lemmata as opposed to POS-tags. The presence of frequency counts in our model would allow us to create multiple, weighted IRTG rules for a single pattern, but exploring the effects of weighting schemes that could in some cases counteract the general preference for shorter derivations containing more specific rules is left for future work.

4 Hierarchical surface realization

Our approach to rule induction presented in the previous section yields a large number of rules. The grammar for an average sentence in the `en_ewt` dataset would contain over 1000 IRTG rules, making the parsing process prohibitively slow. We reduce the average complexity of the task by cutting UD graphs along edges that connect clauses of a sentence, thereby splitting them up into smaller graphs for which our grammar-based approach can be performed separately. The resulting word sequences can then be arranged relative to each other by performing word order restoration on the UD graph of the main clause, this time including dependents of the predicate from other clauses and interpreting them as representing their clauses. We shall now explain this method in detail and illustrate it with an example.

A simple example of an UD graph of multiple clauses, taken from the `en_ewt-ud` test set, is shown in Figure 3 and corresponds to the sentence *Click here to view it*. We begin by cutting the graph on any of the edges that may encode dependency relationships between clauses: `acl, advcl, ccomp, xcomp, conj`. In this case, the edge *click* $\xrightarrow{advcl}$ *view* is found and the graph is split into two new UD graphs that correspond to the fragments *Click here* and *view it*. What follows are three separate runs of the word order restoration algorithm described in the previous section. The first two receive as their input the UD graphs *click* $\xrightarrow{advmod}$ *here* and *view* $\xrightarrow{obj}$ *it* to determine the word order of each fragment. To arrange them with respect to each other, we run our algorithm on the UD graph *click* $\xrightarrow{advcl}$ *view*, the edge connecting the clauses, and interpret the outcome not as an order of the two words but as that of their respective clauses. This approach also enforces that non-clausal dependents of the root of a clause group together with respect to each clausal dependent, which we considered an advantage based on initial experiments with the one-step approach: arguments of a predicate separated by an intervening clause were the errors that we observed would make longer sentences entirely incomprehensible. Our algorithm is defined recursively, since a dependent clause may itself have clausal dependents. On a

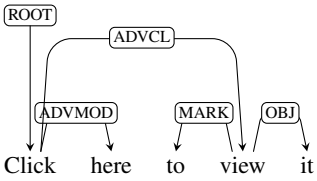

Figure 3: Sample sentence with multiple clauses

sample of 500 English sentences held out from the training dataset, we invoked the core word order restoration method 1794 times, or about 3.6 times per sentence (a single dependent clause would warrant 3 invocations). Even in such a small sample, we see one example each for recursions of depth 4, 5, and even 6, the latter caused by the following sentence (roots of clauses are marked by indices corresponding to depth): *Perhaps had we not gone$_1$ into this restaurant believing$_2$ Zahav was going$_3$ to be golden$_4$ as its name suggests$_5$ (and as the many golden reviews seem$_5$ to attest$_6$), we would have enjoyed$_0$ a decent little expensive experience.*

The reduction in average complexity of the grammars can also be illustrated using the same sample of 500 English sentences: before introducing the hierarchical approach, the 500 IRTG grammars contained more than 400 000 rules, or over 800 rules per grammar on average, resulting in an average parse time of over 4 seconds per sentence (excluding the 42 sentences for which parsing did not finish in 60 seconds, these can be parsed using a simplified grammar, see Section 5 for details). Under the clause-based approach, 1794 IRTG grammars are built to handle the 500 sentences, these contain less than 200 rules on average, and parsing time for individual clauses is just above 700 ms, which brings the per sentence parse time to 2.7 seconds. Note that we did not analyze the runtime of our entire system, these statistics are based on net parse times reported by the alto library, run on a system with 56 Intel Xeon E5-2697 v3 2.6Ghz CPUs. We expect that the greatest bottleneck in overall performance is the per-sentence generation of grammars and invocations of `alto`, but for now we chose to keep this architecture as one that makes both debugging and introspection relatively straightforward.

5 Overall architecture

Our full solution for the Surface Realization task is a pipeline of the grammar-based word order restoration system presented in Sections 3-4 and the DL-based reinflection model used in Kovács et al. (2019). For each language we begin by processing all training and test data to build a shared vocabulary for the given language, mapping lemmata to integers so that they can be represented in alto input and grammar files without the need to transform special characters. After performing the counting of subgraphs that is the training step of the grammar induction system, the model is kept in memory to efficiently serve lookup requests through a REST API to one or several instances of our tool performing word order restoration for datasets of the same language. When processing the test UD graphs, the core word order restoration algorithm is invoked for each subclause, as described in Section 4. The set of dependency edges used to split clauses is configurable and in all our experiments contained the relations `acl`, `advcl`, `ccomp`, `xcomp`, and `conj`. The IRTG rules generated are written to an alto grammar file, the UD graph is written to a file using the PENMAN notation[2], one of the formats handled by `alto`. Next, the parser is invoked as a subprocess with a configurable timeout per sentence, set to 120 seconds in our experiments. In case of a timeout, a new IRTG is generated keeping only those rules that generate subgraphs with a single edge, i.e. we retry with a much simpler grammar that can only add edges to the UD graph one-by-one. Under the clause-based approach this affected only 1% of all clauses in the test datasets. Finally, the linearized sentences are printed in the CoNLLU format and processed by the LSTM-based reinflection model trained using the parameters in Kovács et al. (2019), and a simple postprocessing step which uses simple pattern matching to detect URLs in lemmata and replaces their 'inflected' forms in the final output with the original URL. We do not perform detokenization, our final

[2] `https://penman.readthedocs.io/en/latest/notation.html`

output is the sequence of reinflected tokens separated by spaces.

6 Evaluation

The Surface Realization Shared Task uses multiple metrics for both automatic and human evaluation of system outputs. A complete presentation of all evaluations is available in Mille et al. (2020), here we present a few figures that are especially relevant to our submission. We participate in the track T1a, the closed track of the shallow surface realization task (T1). Since our method is entirely language-independent, we submitted results for all 11 languages and 35 datasets. Besides us, only the IMS team submitted outputs for languages other than English, Table 2 compare our results with theirs and with the output of the system we have extended (Kovács et al., 2019) using automatic evaluation metrics. A direct comparison with last year's system is not possible because it was trained separately for each dataset while our setup trained a single model for each language. Still, it is clear that our changes had very different effects on output quality across languages, which underlines the need for a more systematic analysis of our performance (see Section 7).

Dataset	BME-TUW 2020			BME-TUW 2019			IMS 2020		
	BLEU	NIST	DIST	BLEU	NIST	DIST	BLEU	NIST	DIST
T1_en_filtered_lines-fix	60.80	12.76	62.73	63.37	12.94	60.94	88.34	14.00	86.51
T1_es_filtered_lines-fix	61.13	13.03	57.66	16.34	10.50	35.26	87.85	14.09	87.35
T1_fr_filtered_lines-fix	46.33	11.25	58.82	51.14	11.42	66.53	89.22	13.98	91.64
T1_ko_filtered_lines-fix	53.61	11.12	62.55	40.90	10.98	49.04	76.57	11.50	80.20
T1_pt_filtered_lines-fix	42.22	10.64	59.08	11.78	9.26	34.89	83.57	13.57	84.58
T1_ru_filtered_lines-fix	51.89	12.04	56.67	13.54	10.87	33.07	77.91	12.86	82.37
T1_ar_padt-ud-test	26.00	8.29	44.78	26.40	8.29	43.06	69.56	12.63	75.80
T1_en_ewt-ud-test	57.25	12.52	65.23	59.22	12.62	62.69	86.16	13.78	88.48
T1_en_gum-ud-test	60.77	12.10	62.86	57.57	11.99	56.07	88.89	12.96	91.41
T1_en_lines-ud-test	55.98	11.78	61.44	48.78	11.54	52.77	85.05	12.98	85.89
T1_en_partut-ud-test	57.96	10.22	58.39	61.37	10.34	61.22	89.72	11.07	90.38
T1_es_ancora-ud-test	59.32	13.57	55.66	61.09	13.52	58.15	87.42	14.90	85.66
T1_es_gsd-ud-test	54.60	11.6	55.50	53.74	11.44	59.03	84.61	12.86	82.60
T1_fr_gsd-ud-test	43.21	10.41	55.48	43.80	10.33	59.35	86.08	12.58	84.64
T1_fr_partut-ud-test	52.46	9.05	62.26	49.17	8.99	56.87	87.09	10.56	85.84
T1_fr_sequoia-ud-test	45.25	10.56	57.61	46.72	10.55	59.28	87.25	12.65	85.65
T1_hi_hdtb-ud-test	57.20	11.98	57.55	63.63	12.26	64.04	84.53	13.32	83.03
T1_id_gsd-ud-test	59.16	12.14	59.62	54.22	11.82	55.57	87.53	12.89	86.41
T1_ja_gsd-ud-test	50.89	10.08	60.57	49.53	9.99	57.03	89.36	12.53	87.00
T1_ko_gsd-ud-test	58.37	12.12	66.14	46.08	11.98	52.10	81.14	12.44	85.49
T1_ko_kaist-ud-test	57.05	12.77	62.88	47.23	12.65	50.90	79.96	13.18	84.52
T1_pt_bosque-ud-test	39.89	9.89	55.63	39.53	9.77	58.72	82.92	12.40	84.59
T1_pt_gsd-ud-test	30.68	8.97	53.49	30.39	8.85	54.93	80.59	13.33	87.86
T1_ru_gsd-ud-test	54.28	11.90	53.78	54.58	11.91	52.67	77.43	12.40	78.89
T1_ru_syntagrus-ud-test	54.79	14.17	56.72	50.91	13.80	55.6	81.82	15.45	83.02
T1_zh_gsd-ud-test	50.58	11.62	54.56	58.72	11.85	59.29	86.36	12.86	83.89
T1_en_pud-ud-test	58.67	12.48	61.85	60.42	12.60	59.84	85.37	13.41	85.75
T1_ja_pud-ud-test	51.08	10.34	55.77	53.65	10.56	56.72	88.88	13.20	85.67
T1_ru_pud-ud-test	46.07	11.26	56.17	10.15	9.64	32.08	69.07	11.85	82.40
T1_en_ewt-Pred-HIT-edit	55.50	12.36	62.82	58.07	12.49	60.36	84.72	13.61	86.65
T1_en_pud-Pred-LATTICE	54.76	12.27	59.93	53.46	12.29	56.13	79.74	13.16	82.19
T1_es_ancora-Pred-HIT	59.70	13.57	56.14	61.26	13.51	58.38	86.97	14.81	86.96
T1_hi_hdtb-Pred-HIT	56.83	11.97	57.43	64.27	12.29	64.58	84.42	13.31	83.52
T1_ko_kaist-Pred-HIT	56.74	12.77	62.31	46.72	12.63	50.16	81.01	13.22	85.36
T1_pt_bosque-Pred-Stanford	41.86	9.89	56.60	40.42	9.73	59.72	84.35	12.40	87.35

Table 2: Automatic evaluation on all languages and datasets. See Mille et al. (2020) for a more detailed presentation and methodology.

Other participants submitted solutions only for a subset of the 8 English datasets, Table 3 shows results of human evaluation on the two datasets for which outputs were available from all 2020 participants. It is clear that according to these figures our system is well behind all other participants. Finally, Table 4 compares our system with the 2019 system that it extends, based on human evaluation on 6 datasets.

For each dataset, system outputs were scored on both their readability and their semantic similarity to the reference sentences. On the Russian and Spanish out-of-domain datasets derived from Wikipedia, our system significantly outperforms the original submission of Kovács et al. (2019) according to both metrics.

Team	Meaning				Readability			
	ewt		wiki		ewt		wiki	
	Ave.	Ave. z	Ave.	Ave. z	Ave.	Ave. z	Ave.	Ave. z
HUMAN					75.7	0.417	87.4	0.592
IMS	92.7	0.534	92.3	0.475	73.9	0.374	82.1	0.383
ADAPT	90.7	0.476	91.6	0.441	72.5	0.320	81.5	0.373
Concordia	87.0	0.332	88.7	0.275	70.2	0.270	79.6	0.401
BME 2020	79.3	0.086	81.8	-0.050	58.2	-0.152	60.8	-0.299
BME 2019	77.4	0.024	82.4	-0.074	56.7	-0.208	64.4	-0.181

Table 3: Human evaluation of 2020 submissions on two English datasets of the T1a track. We list average scores (Ave) and average standardized scores (Ave. z) for both meaning similarity and readability evaluations, see Mille et al. (2020) for more evaluation details.

Data	Meaning				Readability			
	BME 2020		BME 2019		BME 2020		BME 2019	
	Ave.	Ave. z	Ave.	Ave. z	Ave.	Ave. z	Ave.	Ave. z
en_ewt	79.3	0.086	77.4	0.024	58.2	-0.152	56.7	-0.208
en_wiki	81.8	-0.050	82.4	-0.074	60.8	-0.299	64.4	**-0.181**
ru_syn	81.2	-0.166	81.3	-0.177	69.7	-0.166	67.3	-0.230
ru_wiki	78.2	**-0.079**	68.2	-0.493	63.2	**0.050**	37.7	-0.781
es_ancora	70.2	-0.276	70.6	-0.271	66.4	-0.401	67.1	-0.378
es_wiki	69.8	**-0.170**	55.5	-0.726	77.2	**0.015**	62.2	-0.628

Table 4: Human evaluation of our system and the 2019 submission of Kovács et al. (2019) for all datasets. We list average scores (Ave) and average standardized scores (Ave. z) for both meaning similarity and readability evaluations. Bold figures indicate a system that significantly outperforms the other on the given metric. See Mille et al. (2020) for more evaluation details.

7 Conclusion and future work

We have presented a modification of a language-independent Surface Realization system with a fully rule-based component for word order restoration. Our main contribution is a new mechanism for generating mappings between graph and string operations using IRTG grammars that choose for a word and its dependencies the most specific patterns observed in the training data. Unlike the original system, our solution makes little use of rule weights, their sole function is currently to ensure that shorter derivations for a UD graph are preferred over longer ones. We believe that our method may be improved by introducing non-determinism in the generated grammars via some weighting scheme that allows seemingly non-optimal patterns to be chosen over those that were observed most frequently. Given the large number of parameters of our system, we also suggest the use of additional, silver-standard training data, which would be available in virtually unlimited quantities for all languages for which a high quality UD parser is available. Finally, the most important next step should be the thorough quantitative and qualitative analysis of the performance gap between our method for word order restoration and that of the state-of-the-art. Observing and quantifying correct and incorrect decisions of the grammar and updating the induction mechanism accordingly is what we consider the greatest potential of our rule-based and thus fully explainable solution.

Acknowledgements

We thank our three anynomous reviewers for the many helpful comments and suggestions. Project partly supported by BRISE-Vienna (UIA04-081), a European Union Urban Innovative Actions project. Á.K. and A.K. were partially supported by 2018-1.2.1-NKP-00008: Exploring the Mathematical Foundations of Artificial Intelligence. J.Á. was partially supported by the Hungarian Scientific Research Fund (OTKA), contract number 120145 and by the Hungarian Artificial Intelligence National Laboratory.

References

Bruno Courcelle and Joost Engelfriet. 2012. *Graph structure and monadic second-order logic*. Cambridge University Press.

Wenchao Du and Alan W Black. 2019. Learning to order graph elements with application to multilingual surface realization. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*, pages 18–24, Hong Kong, China, November. Association for Computational Linguistics.

Johannes Gontrum, Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2017. Alto: Rapid prototyping for parsing and translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 29–32.

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*, Dublin.

Alexander Koller. 2015. Semantic construction with graph grammars. In *Proceedings of the 14th International Conference on Computational Semantics (IWCS)*, London.

Ádám Kovács, Evelin Ács, Judit Ács, Andras Kornai, and Gábor Recski. 2019. BME-UW at SRST-2019: Surface realization with interpreted regular tree grammars. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*, pages 35–40, Hong Kong, China. Association for Computational Linguistics.

Simon Mille, Anja Belz, Bernd Bohnet, Yvette Graham, and Leo Wanner. 2019. The Second Multilingual Surface Realisation Shared Task (SR'19): Overview and Evaluation Results. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR), 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Hong Kong, China.

Simon Mille, Anya Belz, Bernd Bohnet, Thiago Castro Ferreira, Yvette Graham, and Leo Wanner. 2020. The third multilingual surface realisation shared task (SR'20): Overview and evaluation results. In *Proceedings of the 3rd Workshop on Multilingual Surface Realisation (MSR 2020)*, Dublin, Ireland. Association for Computational Linguistics.

Xiang Yu, Agnieszka Falenska, Marina Haid, Ngoc Thang Vu, and Jonas Kuhn. 2019. IMSurReal: IMS at the surface realization shared task 2019. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*, pages 50–58, Hong Kong, China, November. Association for Computational Linguistics.