How much of enhanced UD is contained in UD?

Adam Ek **Jean Philippe Bernardy**
Centre for Linguistic Theory and Studies in Probability
Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg
{adam.ek, jean-philippe.bernardy}@gu.se

Abstract

In this paper, we present the submission of team CLASP to the IWPT 2020 Shared Task on parsing enhanced universal dependencies (Bouma et al., 2020). We develop a tree-to-graph transformation algorithm based on dependency patterns. This algorithm can transform gold UD trees to EUD graphs with an ELAS score of 81.55 and a EULAS score of 96.70. These results show that much of the information needed to construct EUD graphs from UD trees are present in the UD trees. Coupled with a standard UD parser, the method applies to the official test data and yields and ELAS score of 67.85 and a EULAS score is 80.18.

1 Introduction

Universal Dependencies (UD) is a syntactic annotation schema focusing on representing shallow syntactic dependencies between *words*. One of the goals of UD has been to use it in semantic downstream tasks, such as event extraction (Fares et al., 2018; McClosky et al., 2011) or negation resolution (Fares et al., 2018) among others. In general, UD can be used as a shallow representation of argument structures, which are useful in a wide array of semantic tasks.

However, the UD format restricts the shape of dependencies to a tree structure. This can be limiting because semantics dependencies can in principle exhibit any graph structure. To remedy this situation, the Enhanced Universal Dependencies (EUD) schema was proposed (Schuster and Manning, 2016). The goal of the schema is to make certain implicit dependencies explicit, such as conjoined subjects and objects. The enhanced dependencies include additional edges between words, as well as augmented labels. For example in enhanced dependencies, the conjunction relation also include what type of conjunction is used, e.g. *and, or, but* and so on.

In this paper, we present our submission for the IWPT 2020 Shared Task on parsing enhanced universal dependencies from annotated UD treebanks. The task target treebanks from 17 different languages where the majority of the languages are Indo-European with five notable exceptions, Tamil (Dravidian), Arabic (Semitic), Finnish and Estonian (Uralic). The goal of the task to produce valid EUD graphs, given raw text as input.

In this context, this paper proposes to test the following hypothesis:

(H1) Most of the information provided by the EUD schema is contained in the basic UD schema.

That is, if (H1) holds, then it is possible to map, algorithmically, UD trees to EUD graphs.

Thus, we set out to implement such an algorithm. Concretely, we construct a tree-to-graph transformation, which recognizes patterns in the (basic) universal dependencies to derive enhanced dependencies.

This experiment provides a *lower bound* on the amount of EUD information which can be extracted from raw UD information, for representative inputs. In other words, we measure how much of EUD is contained in UD. This becomes a lower bound, because a better algorithm can always be conceived, and do better.

Conversely, additionally, by running our algorithm after a state-of-the-art basic UD parser, we will provide a baseline for the EUD reconstruction task.

The enhanced dependency parsing is evaluated using two metrics, ELAS and EULAS. ELAS calculate the F_1-score over both enhanced arcs and labels (for example: "and" in the label "conj:and"). EULAS on the other hand disregard label enhancements and calculate the F_1-score over the enhanced

Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task, pages 221–226

edges. To illustrate, given the gold label "conj:and" and a system prediction of "conj:or" (or "conj", i.e. if nothing is appended to the label), ELAS will count this as an error while EULAS won't.

Our results show that (H1) is vindicated: we achieve an ELAS score of 81.55 and EULAS of 96.70 on human-annotated dependency trees. As a baseline for the shared task, our method is also effective, achieving an ELAS score of 67.85 and a EULAS score of 80.18. (Thus losing about 15 percentage points when going through a machine-generation phase to obtain UD trees.)

2 Method

In essence, our method is to apply, as far as possible, the tree-to-graph recipes provided by Schuster and Manning (2016) to transform basic UD *trees* into EUD graphs.

2.1 Procedure

To obtain basic UD trees for our system we use the universal dependency treebanks provided by the shared task organizers. From these we apply our method on basic UD trees from two sources:

- for each language in the test data we use the Stanford Biaffine Dependency Parser (Dozat and Manning, 2016) provided in Stanza (Qi et al., 2020). We used the stanza model trained on the largest treebank of the language.
- the development and gold trees in the treebanks, *from which we have removed the enhanced dependencies*. Indeed, the gold data comes with plain UD trees as well.

Thus when using the basic UD trees from the Stanford parser we obtain a baseline for the task, and when using the development/gold trees the *lower bound* on EUD information contained in UD.

Then, we apply a tree-matching procedure against the non-enhanced UD trees. The procedure locally inserts enhanced edges or deletes unwanted edges. Additionally (as a special case) the patterns also re-label some edges.

Our system contains several patterns, which are described in the next subsection. We first apply the patterns that modify the edge labels with case information. Then we apply all the other patterns, which add (and sometimes remove) edges from the basic tree. Here, we only apply the patterns on the (relabeled) input trees and not on the graph of EUD made by any other pattern. In this sense, one can say that patterns are applied in parallel. Once this is done, we convert the result back to the (enhanced) CONLLU format.

2.2 Patterns

Perhaps surprisingly, the patterns that we need to recognize are simple, involving only three nodes. The two patterns to recognize are shown in Fig. 5. Essentially, we need to match on three connected nodes. We need to identify two types of patterns. First, two arcs forming a two-step path (Figs. 5a, 5d and 5e). We refer to this style pattern as "Type 1". Second, with two arcs pointing away from a central node (Figs. 5b and 5c), referred to as "Type 2". In both cases, we have additional constraints on the (edge) labels. Together, the constraints on graph topology and labels form patterns that we can recognize and transform. The exhaustive list of patterns and transformations follows.

1. Type 1 pattern, with a relation label, which can be any of "nsubj","obj", "amod", "advcl","obl", "mark", "nmod", followed by a "conj" label. (Fig. 5a.) In this case we add an edge with the relation label to the other conjunct. A full dependency tree containing this pattern can be found in Fig. 1.

2. Type 2 pattern with a a relation label being either "nsubj" or "aux", and a "conj" label (Fig. 5b). We add a relation label to the other conjunct, but only if the conjunct is not itself "nsubj". Indeed, if it were, then we are conjoining two full sentences and then there is no need for an enhanced dependency. A full dependency tree containing this pattern can be found in Fig. 2.

3. Type 2 pattern with "xcomp" and "nsubj". Here we add an "nsubj:xsubj" edge (Fig. 5c).

4. Type 1 pattern, with "acl:relcl" followed by a relation label which can be either "nsubj","obj","obl", "advmod" (Fig. 5e). The target node should also be a *relative* pronoun, ie. it POS is "PRON" and its XPOS either "WP" (who, whom) or "WDT" (that, which). Indeed, this pattern is also found with other type of pronouns, but then it does not correspond to a relative clause. In this case we add a "ref" edge to the pronoun and a (reverse) relation edge between the first and second node.

The original relation edge is deleted. A full dependency tree containing this pattern can be seen in Fig. 4.

5. Type 1 pattern, with a conjunction followed by a case marking (Fig. 5d). Exhaustively, the type of labels are "case" followed by "obl" or "nmod"; "cc" followed by "conj", "mark" followed by "advcl" or "acl". In this case we enhance the label with the lemma of the target node. This pattern can be seen in Fig. 7.

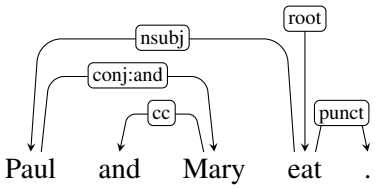


Figure 1: Example sentence for pattern shown in Fig. 5a

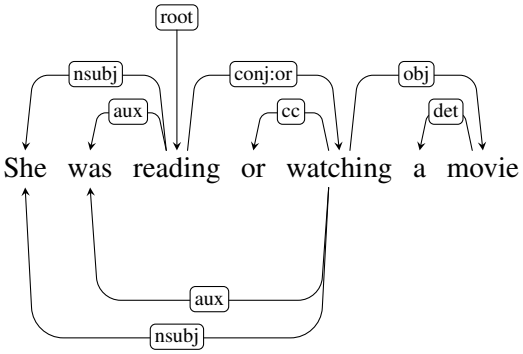


Figure 2: Example sentence for pattern shown in Fig. 5b

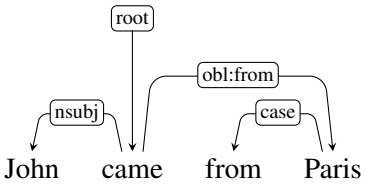


Figure 3: Example sentence for pattern shown in Fig. 5d

2.2.1 Other patterns

For patterns Items 1 and 2, the direction of conjunction dependency could conceptually be inverted, yielding two other patterns (shown in Figs. 6a and 6b). However, we have not implemented these patterns in our system. For the first pattern, the reason is simple: it is not representable as a UD tree (a node cannot have two heads). For the second pattern, applying it results in a small loss in performance across the board, and thus it is best left inactive. Additionally, we have not implemented any pattern for dealing with ellipsis in the current approach. We plan on addressing ellipsis in future work.

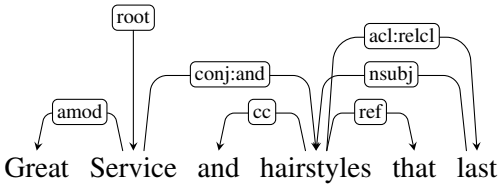
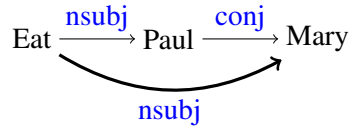
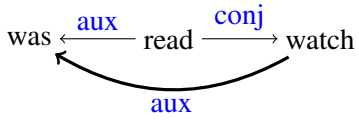


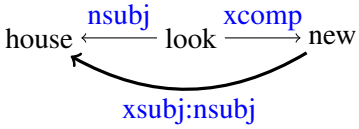
Figure 4: Example sentence for pattern shown in Fig. 5e



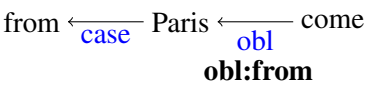
(a) Relation pointing to the conjuncts



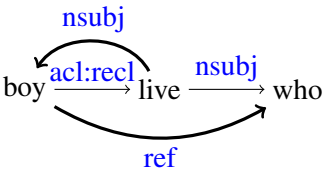
(b) Relation pointing away from the conjuncts.



(c) Xcomp special case



(d) Label taken from other word (lemma)



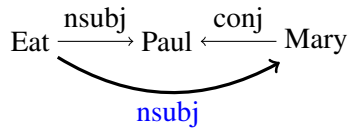
(e) Relative clause

Figure 5: Implemented transformation patterns. Added elements are shown in bold.

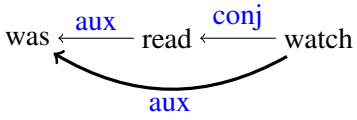
3 Results

We present our official results for the shared task in Table 1. The scores are obtained by applying our tree-to-graph transformation to basic dependency trees generated by the Stanford Dependency Parser.

The scoring for unlabeled edges is typically



(a) Source pattern not representable in UD format



(b) Transformation pattern leading to a loss in performance

Figure 6: Transformation patterns **not** implemented.

LANG	ELAS	EULAS
ar	51.26	75.62
bg	84.90	87.72
cs	67.13	83.44
en	82.87	83.86
et	60.44	79.43
fi	65.96	83.30
fr	72.76	84.39
it	87.14	88.74
lv	66.01	80.60
nl	78.93	80.20
lt	52.56	67.37
pl	71.22	86.71
ru	70.37	88.02
sk	65.16	83.31
sv	71.35	73.84
ta	42.15	55.32
uk	63.24	81.24
Avg.	**67.85**	**80.18**
Std.	**11.69**	**8.19**

Table 1: Coarse ELAS and EULAS on the languages in the shared task test data.

around 80%. The biggest outlier being Tamil, at 55.32%. The scoring for labeled edges is around 12 points lower, with more variation in scores.

As explained in the introduction, to isolate the accuracy of the tree-to-graph transformation from the performance of the underlying UD parser, we also apply it to the human-annotated dependency trees. We test our approach on both the development and test data. The results from this experiment are shown in Table 2.

The scoring for enhanced edges and unenhanced labels (EULAS) is typically above 95%, with little variation. Tamil is no longer an outlier. The scoring for enhanced edges and labels can be classified into two categories depending on the language. In one

	DEV		TEST	
LANG	ELAS	EULAS	ELAS	EULAS
ar	66.44	96.38	64.14	96.55
bg	94.32	96.95	94.55	97.01
cs	75.65	94.63	76.52	95.12
en	97.57	98.53	97.64	98.65
et	73.35	93.96	72.30	95.53
fi	74.22	95.29	74.40	95.12
fr	83.59	97.93	88.15	99.05
it	96.29	97.61	96.15	97.77
lt	77.50	93.98	72.17	95.49
lv	69.21	96.10	77.47	93.91
nl	96.71	97.55	96.27	97.57
pl	87.03	97.33	80.95	96.49
ru	77.22	96.31	76.72	96.59
sk	72.24	95.82	75.37	96.42
sv	93.85	96.66	94.19	96.67
ta	72.49	99.58	75.04	99.48
uk	75.95	96.04	74.42	96.49
Avg.	**82.97**	**96.53**	**81.55**	**96.70**
Std.	**10.42**	**1.52**	**10.24**	**1.43**

Table 2: Coarse ELAS and EULAS on the gold trees in the development and test data.

category the ELAS is nearly as good as the EULAS case (bg, en, it, nl, sv). Another category scores about 15 points lower (ar, cs, et, fi, lt, lv, ru, sk, ta, uk). French and Polish scores are somewhere in-between.

Comparing Table 1 and Table 2 suggests that the performance of the full pipeline is highly dependent on the underlying parser. In addition to the ELAS and EULAS score being higher, the standard deviation is also much higher for EULAS, 8.19 points in the test versus 1.43 points on the gold data.

To get a further sense on how much our algorithm is sensitive to the quality of the input raw UD trees, we can compare the score that it obtains when fed the gold (raw) UD trees and the trees provided by Stanza for the same text. The results are shown in Table 3.

	Official submission		Stanza trees		Gold trees	
	ELAS	EULAS	ELAS	EULAS	ELAS	EULAS
Avg.	67.85	80.18	66.54	79.91	81.55	96.70

Table 3: Coarse ELAS and EULAS on the test data using our tree-to-graph algorithm on Stanza trees, Stanza trees only and with gold trees.

We find that using our algorithm with stanza trees only marginally increase the performance. Given that we know that our trees are effective for gold trees, this corroborates the suspicion that while having a state-of-the-art performance in dependency parsing, Stanza produces trees that have a significant number of errors for the paths rele-

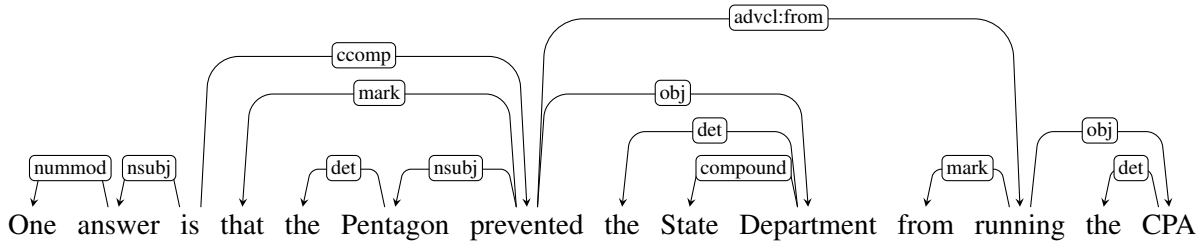


Figure 7: Example sentence for pattern shown in Fig. 5d

vant to EUD. If the UD trees were of good quality for the parts that are relevant for EUD, we would see a larger increase in performance based on the potential of our algorithm. To some extent, this indicates that when developing a EUD parser, the best approach may be to parse UD and EUD in parallel, and not sequentially. This modus operandi may help the parser to produce fewer UD errors for the parts relevant to EUD.

4 Discussion/Analysis

We can draw the following conclusions from our experiments:

- Our pattern recognizer fails to annotate many enhanced labels for languages where case is expressed by morphological features.

  This is not surprising: we simply did not implement any label rewriting based on such features. (Indeed, the pattern Fig. 5d recognizes case based on a preposition rather than a morphological feature.) This is a shortcoming which we plan to eliminate in future work.

  This shortcoming explains much of the discrepancy between the EULAS and the ELAS scores.

- The performance of the system is heavily dependent on the quality of the UD parser which is used. In other words, our experiments show that, even when using good UD parser, much of the errors are imputable to the UD parser rather than to the algorithmic rewriting step.

- Our tree-to-graph transformation works well on the gold trees, missing less than 4% of the edges, when given gold UD trees as input. Therefore, Broadly speaking, (H1) is vindicated: UD contains most of the information which is necessary to reconstruct EUD graphs.

We note however that (not enhanced) UD is incapable of expressing the difference between the structure of the following two sentences:

(1) She was reading or watching a movie. (Fig. 2)
(2) She was cleaning and eating fruits. (Fig. 8)

In (1) "a movie" is an object of a verb ("watching") which is conjoined with another verb ("reading"), but it applies only to a single verb. In (2), we also have a conjunction between verbs, but "fruits" is the object of both verbs. Yet, the UD structure is the same for both sentences. And thus, our algorithm cannot recognize the difference between the two trees. One solution is to use EUD, but another solution would be to use parse trees, which can make the grouping explicit.

Figuring out which case applies depends on the semantics and pragmatics.

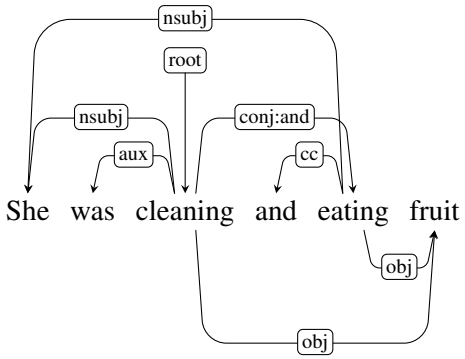


Figure 8: "She was cleaning and eating fruit"

Even though we have shown that UD contains most information to recover EUD, EUD annotations do have some additional value in certain cases. However, because they have a more rich structure (graph vs. tree), it may be that EUD is more suited as inputs to machine-learning systems. Our systems can help to test this hypothesis by inserting

it (or not) in the training pipeline of a state-of-the-art EUD parser. If the EUD parser performs just as well with reconstructed EUD data compared to human-constructed EUD data, then we would know that manual EUD annotations are not necessary. It is likely that we would observe a middle ground situation, where reconstructed EUD data helps, but does not supplant human-constructed EUD. In such a situation, our system can be useful as a bootstrapping tool. (We note however that excellent EUD parsers are not available just yet; in fact they are the purpose of the IWPT task which our system is entering.)

In addition to bootstrapping, a run of our system can provide a baseline for such systems. Indeed, while performing well on gold data, our approach is transparent (only 5 patterns are applied) and efficient, requiring a few seconds to generate enhanced graphs from a treebank.

Future work In addition to the shortcoming about case labeling mentioned above, our system struggles with labels that include multi-word tokens. For example, a valid adverbial clause modifier is *so_that*. Currently our system is not able to identify these. To incorporate this our model would either need to utilize statistical or deep learning methods, or look for children with the "fixed" relation.

Another case that our system is not capable of handling is ellipsis. The problem of ellipsis is difficult, especially in a tree-rewriting approach. We decided not to tackle this problem and instead plan on using a deep learning parser for this task in future work.

Yet our main venue for future work is combining our simple, yet effective method, with deep learning for the problematic cases that are out of reach. These cases include ellipsis, multi-word tokens, objects of conjoined verbs.

Acknowledgments

We would like to thank the reviewers for their helpful comments.The research reported in this paper was supportedby a grant from the Swedish Research Council (VR project 2014-39) for the establishment of the Centre for Linguistic Theory and Studies in Probability (CLASP) at the University of Gothenburg.

References

Gosse Bouma, Djamé Seddah, and Daniel Zeman. 2020. Overview of the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, Seattle, US. Association for Computational Linguistics.

Timothy Dozat and Christopher D Manning. 2016. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*.

Murhaf Fares, Stephan Oepen, Lilja Øvrelid, Jari Björne, and Richard Johansson. 2018. The 2018 shared task on extrinsic parser evaluation: on the downstream utility of english universal dependency parsers. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 22–33.

David McClosky, Mihai Surdeanu, and Christopher D Manning. 2011. Event extraction as dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1626–1635. Association for Computational Linguistics.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D Manning. 2020. Stanza: A python natural language processing toolkit for many human languages. *arXiv preprint arXiv:2003.07082*.

Sebastian Schuster and Christopher D Manning. 2016. Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 2371–2378.