Artificially Evolved Chunks for Morphosyntactic Analysis

Mark Anderson **David Vilares** **Carlos Gómez-Rodríguez**
Universidade da Coruña, CITIC
FASTPARSE Lab, LyS Research Group, Departamento de Computación
Campus Elviña, s/n, 15071
A Coruña, Spain
{m.anderson,david.vilares,carlos.gomez}@udc.es

Abstract

We introduce a language-agnostic evolutionary technique for automatically extracting chunks from dependency treebanks. We evaluate these chunks on a number of morphosyntactic tasks, namely POS[1] tagging, morphological feature tagging, and dependency parsing. We test the utility of these chunks in a host of different ways. We first learn chunking as one task in a shared multi-task framework together with POS and morphological feature tagging. The predictions from this network are then used as input to augment sequence-labelling dependency parsing. Finally, we investigate the impact chunks have on dependency parsing in a multi-task framework. Our results from these analyses show that these chunks improve performance at different levels of syntactic abstraction on English UD treebanks and a small, diverse subset of non-English UD treebanks.

1 Introduction

Shallow parsing, or chunking, consists of identifying constituent phrases (Abney, 1997). As such, it is fundamentally associated with constituency parsing, as it can be used as a first step for finding a full constituency tree (Ciravegna and Lavelli, 1999; Tsuruoka and Tsujii, 2005). However, chunking information can also be beneficial for dependency parsing (Attardi and DellOrletta, 2008; Tammewar et al., 2015), and vice versa (Kutlu and Cicekli, 2016). Latterly, Lacroix (2018) explored the efficacy of noun phrase (NP) chunking with respect to universal dependency (UD) parsing and POS tagging for English treebanks. As UD treebanks do not contain chunking annotation, they deduced chunks by adopting linguistic-based phrase rules. They observed improvements on POS and morphological feature tagging in a shared multi-task framework for the English treebanks in UD version 2.1 (Nivre et al., 2017). However, an increase in performance for parsing was only obtained for one treebank.

Contribution 1. We first relax the standard definition of chunks and present an evolutionary method to automatically deduce chunks for any language given a dependency treebank. 2. We show that chunking information can improve performances for POS tagging, morphological feature tagging, and dependency parsing, both in a multi-task and a single-task framework.

2 Chunks and chunking rules

While Lacroix (2018) described a method to obtain chunks from sentences with UD annotations, their approach is limited to NP chunks and requires hand-crafted linguistic rules, meaning that it cannot be transferred to other languages without language-specific knowledge. In contrast, we introduce a fully automatic approach to obtain chunks from UD-annotated sentences in a language-agnostic way. Figure 1 depicts our method of extracting candidate chunk types.

Chunk definition Here we loosen the definition of a chunk and consider any base-level subtree a possible chunk defined by the following criteria: (i) the components of a chunk are syntactically linked; (ii) there is only one level of dependency (one head and its dependents); (iii) the components are continuous; and (iv) no dependents within a chunk has a dependent outside the chunk.

[1]POS tagging is used throughout to refer to universal part-of-speech (UPOS) tagging.

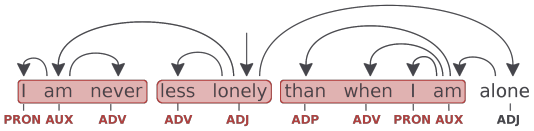


Figure 1: Candidate phrase rules are extracted by selecting subtrees with one level of dependency.

Describing chunks with rules For each subtree in the training set that meets the above criteria, the corresponding sequence of POS tags of its words is saved as a candidate rule. Each rule is collected for a given treebank to construct a ruleset of unique candidate chunk types. When more than one overlapping subtree meets these conditions the maximal substring is used, e.g. in Figure 1 PRON AUX ADV is chosen instead of PRON AUX or AUX ADV. We allow any chunk type with the exception of those containing the PUNC POS tag and we apply a mild frequency cut of 5 to make the problem more tractable. The English-EWT treebank, for example, results in a ruleset consisting of 512 candidates.

Annotating with rulesets This ruleset (or any subset of it) can be applied to a UD treebank to obtain chunks, by using them as patterns that generate a chunk when they are matched by a sequence of POS tags and meet the criteria described above.[2] In particular, we can apply it to the training set to obtain a set of chunks on which to train a statistical chunker to process arbitrary texts and help morphosyntactic tasks. When annotating a treebank, the POS tag of the head is used as a suffix for the chunk type, e.g. DET ADJ NOUN would result in IOB tags of B-NOUN and I-NOUN, assuming the head of this phrase corresponds to the NOUN tag (Ramshaw and Marcus, 1999).

However, not all candidate rules are useful and can impact the ability of a chunker to make sensible predictions. For this reason, we will not use the whole candidate ruleset obtained from a training corpus, but instead try to find a subset of the ruleset whose resulting set of chunks strikes a good balance between the following criteria: (i) coverage (i.e. there should be enough chunks to maximize their informativeness for morphosyntactic tasks) and (ii) consistency and learnability (i.e. the chunks should follow patterns predictable enough to be easily learnable by a machine learning model, so that our approach is not undermined by low chunking accuracy). Our hypothesis is that these two characteristics (which we quantify with a fitness function in the next section) are reasonable proxies for the usefulness of a particular set of chunks for morphosyntactic tasks.

Note that to achieve this, it is not possible to merely remove error-prone rules from the ruleset because there is a complicated interplay between rules, i.e. if the 10% most error-prone rules are removed, the overall accuracy of the system is not guaranteed to improve. Furthermore, with so many candidate rules, it is not possible to try every combination as this results in an astronomical number (2^n). Therefore, we aim to use an evolutionary method to find optimal subsets of rules to be used when annotating treebanks.

3 Evolutionary search for chunk rules

Evolutionary algorithms aim to optimise an objective (fitness) function by evaluating a population of individuals and subsequently generating a new population based on the best performing individuals from the population (Back, 1996). This process is then repeated until a set number of generations is reached or until the fitness function converges. Each individual consists of a set of parameters and its corresponding objective function value, or fitness. The fitness of an individual is used to decide whether to use it as a parent for subsequent generations or to remove it from the population. We introduce the techniques used to select parents and how they are then used to generate offspring (Algorithm 1 in Appendix A).

K-best parent selection The selection operator makes the population converge. We used the simple k-best method where the top k individuals of a population are selected as the parents.

Mutation Mutation is a genetic operator which prevents a population becoming too genetically similar by randomly altering individuals. This ensures that at least some level of genetic diversity is maintained

[2]Rules are applied from longer (more specific) to shorter (more generic).

from generation to generation. Our individuals have binary genes, so our mutation operator flips each gene with a probability $P_{\text{mutate gene}}$.

Crossover Crossover is a genetic operator which also preserves genetic variety in a population. In single-point crossover, a random index κ is chosen and the substring 0-κ of parent$_x$ is replaced with the corresponding part of parent$_y$ and vice-versa. This results in two offspring. Single-point crossover can be extended to x-point crossover, where x points are used to cut individuals.

We used the DEAP framework for our implementation (Fortin et al., 2012), and the parameters in Table 6 (Appendix B). We represented our rulesets as a binary vector, where 1 meant a rule was used and 0 meant it was not. Our fitness function was obtained by combining the F1-score of a chunker implemented with the sequence-labelling framework NCRF++ (Yang and Zhang, 2018) and the proportion of the maximum compression rate, weighted 1.0 and 0.5 respectively. The compression rate, r, is defined as:

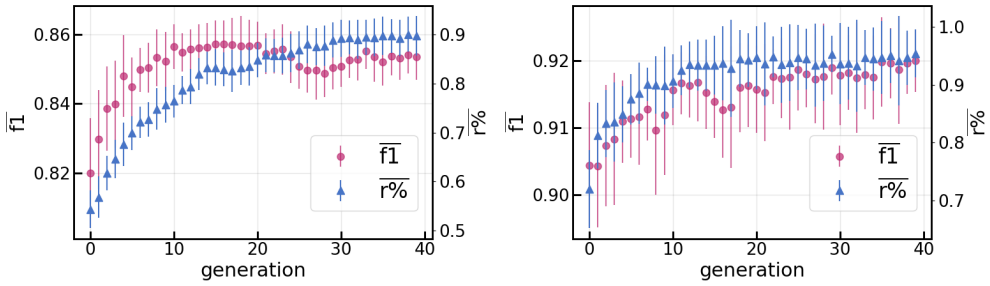
$$r = \frac{C_{\text{tokens}}}{C_{\text{chunks}} + C_{\text{out}}} \tag{1}$$

where C_{tokens} is the number of tokens in a treebank, C_{chunks} the number of chunks a ruleset creates, and C_{out} the number of tokens outside of chunks. And subsequently the proportion of the maximum compression rate, $r\%$ is defined as:

$$r\% = \frac{r_{\text{subset}} - 1}{r_{\text{all}} - 1} \tag{2}$$

where r_{subset} is the compression rate of the current rule subset and r_{all} is the compression rate of the full ruleset.

We used a small network for chunking due to the considerable computational costs of evolutionary algorithms. For each individual in each population, we trained a chunker for 5 epochs (see Table 7 in Appendix B for the parameters) and the corresponding model's best performance on the development set was taken as that individual's fitness along with the proportion of the maximum compression rate, $r\%$: the proportion of the maximum rate was used to prevent the algorithm from generating rulesets that generated few chunks and therefore minimising the potential impact. The convergence over 40 generations for English-EWT and Japanese-GSD can be seen in Figure 2.



(a) English-EWT (b) Japanese-GSD

Figure 2: Average F1-score and proportion of max compression for English-EWT (a) and Japanese-GSD (b) during evolutionary search for optimal chunk type candidates.

As a final step, we took the top 100 best rulesets from across generations and extracted the rules that appeared in at least 75% and 95% of these sets, as the evolutionary algorithm only managed to find a single set with a fairly low performance. Rulesets were obtained this way for each treebank, except the rulesets extracted from English-EWT were subsequently used on the other English UD treebanks. The statistics for the resulting chunks for the respective test data can be seen in Table 1.

4 Sequence-labelling framework

All the proposed tasks can be cast as sequence labelling, so in this work we have used a sequence-labelling framework to address them. In particular, we rely on bidirectional long short-term memory

	# rules		C/sent	
	75%	95%	75%	95%
en-ewt	230	134	3.11	2.71
en-gum	-	-	4.48	3.84
en-lines	-	-	4.47	4.18
en-partut	-	-	6.32	5.84
bg	152	108	3.94	3.65
de	135	106	4.05	3.90
ja	184	130	6.83	6.70

Table 1: Chunking statistics on test data for each treebank used where # rules is the number of rules in a ruleset for a given threshold and C/sent corresponds to the number of chunks per sentence found.

(BiLSTMs) networks (Hochreiter and Schmidhuber, 1997; Schuster and Paliwal, 1997). The input to the network are continuous word representations and character embeddings.

In this paper we used NCRF++ (Yang and Zhang, 2018), which uses stacked BiLSTMs, to generate contextualised hidden representations for every word ($\vec{h_i}$) in the input sentence. For decoding, it uses a feed-forward layer followed by a *softmax* activation:

$$P(y|\vec{h_i}) = softmax(\vec{W} \times \vec{h_i} + \vec{b}) \tag{3}$$

The single task models are optimised with cross-entropy loss, $\mathscr{L}$, defined as:

$$\mathscr{L} = -\sum log(P(y|h_i) \tag{4}$$

For the multi-task learning models, we implemented a hard-sharing architecture, where all the stacked BiLSTMs are shared across all tasks (Søgaard and Goldberg, 2016) . A separate feed-forward layer (as the one used in the single task setup) is used to decode the output for each task. With respect to the computation of the loss under the multi-task learning (MTL) setup, $\mathscr{L}_{MTL}$, is defined as:

$$\mathscr{L}_{MTL} = \sum_{t \in T} \beta_t \mathscr{L}_t \tag{5}$$

where t is a task from the set of all tasks, T; β_t is the corresponding weight for task t; and $\mathscr{L}_t$ is the cross-entropy loss for task t. A schematic of the network can be seen in Figure 3.

4.1 Dependency parsing as sequence labelling

In order to more readily utilise the multi-task framework for dependency parsing, we have cast dependency parsing as a sequence-labelling task. This was done by using the relative position encoding scheme introduced by Strzyz et al. (2019). We opted to use this encoding as it was the highest performing labelling scheme they evaluated. For each word in a sentence the dependency relation label is combined with the relative position of its head based on the POS tag of the head, e.g. a noun which is the subject of a verb (*son* in the input sentence in Figure 3) would have a label of +1,nsubj,VERB, where +1 indicates the head is the next VERB in the sentence and nsubj is the relation label.

5 Experiments

Data The analyses were undertaken using the English treebanks (EWT, GUM, LinES, and ParTUT) and also Bulgarian-BTB, German-GSD, and Japanese-GSD from UD v2.3 (Nivre et al., 2018). No results are given for Japanese-GSD for morphological feature tagging as it does not contain this information.

Network hyperparameters We used the framework as described above and hyperparameters from Vilares et al. (2019) which can be seen in Table 8 in the Appendix B. The standard input to the system consisted of word embeddings concatenated with character embeddings. All embeddings were randomly initialised.

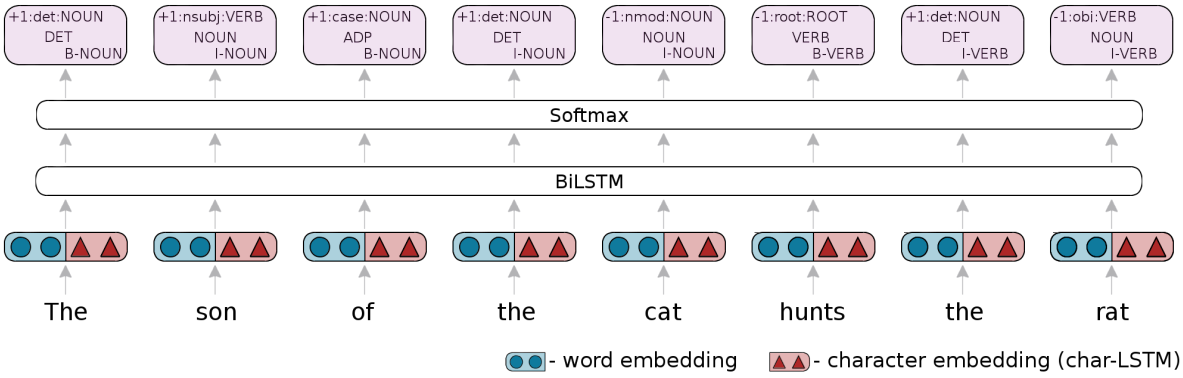


Figure 3: Multi-task architecture shown with sequence-labelling dependency parsing (as described in subsection 4.1), POS tagging, and chunking as shared tasks. Network input is a concatenation of word embeddings (circles) and character-level word embeddings (triangles) obtained from a character-based LSTM layer. The network is constructed of BiLSTM layers followed by a *softmax* layer for inference.

Experiment 1 We tested the impact of our chunks on POS and morphological feature tagging in a shared multi-task setting. This entails feeding word and character embeddings as input to the network with the output being some combination of POS tags, morphological feature tags, and chunk labels. These results were compared against the baseline taggers (single-task networks and POS and morphological features shared only). Tasks were equally weighted. As a further baseline we include results for POS and morphological feature tagging using UDPipe 2.2 (Straka and Straková, 2019).

Experiment 2 We used the best predictions (when using chunking) from experiment 1 as additional features for a sequence-labelling dependency parser (Strzyz et al., 2019). Therefore, network input consisted of word and character embedding and then some combination of POS tags, morphological feature tags, or chunk labels with the sole output being a dependency parser tag. We used gold tags and labels as input during training, but at runtime we used predicted tags and labels. For baselines we train a model with no features which is decoded with predicted POS tags using UDPipe 2.2 (as the sequence-labelling encoding we are using requires POS tags to resolve dependency heads) and also a model trained with POS tags as features but also using UDPipe 2.2 predicted POS tags at runtime.

Experiment 3 We tested the impact of our chunks on a sequence-labelling dependency parser in a multi-task framework with and without the other tasks. POS tagging was treated as a secondary main task with a weight of 0.5 (as POS tags are needed to decode the sequence-labelling scheme for the dependency parser) and chunks and morphological features were considered auxiliary tasks with a weight of 0.25 when used. The input during this experiment were only word and character embeddings. An example is shown in Figure 3 where the shared tasks are chunking, POS tagging, and dependency parsing. The baseline used here is a model trained solely to predict dependency parsing tags which are then decoded using predicted POS tags from UDPipe 2.2.

6 Results and discussion

As seen in Table 2 the multi-task framework with chunks improves the performance of both POS and morphological tagging for all English treebanks. In the same table, it is clear that they do not aid Bulgarian, but they do improve POS tagging performance for German and Japanese. Table 3 shows that chunking performance consistently improves in the multi-task setting. Parsing performance is improved across all treebanks when the predictions from experiment 1 are used as features (Table 4), but only for English-EWT (the largest treebank) and ParTUT (the smallest) do the predicted chunks explicitly improve performance and for the other treebanks only the other predicted features help. This is in contrast to the findings of Nguyen and Verspoor (2018), who obtained higher performance for larger treebanks. In the multi-task setting for the dependency parser (Table 5), the chunking information consistently aids

	ewt		gum		lines		partut	
	pos	feats	pos	feats	pos	feats	pos	feats
udpipe	94.44	95.37	93.88	94.21	94.73	94.83	94.10	94.01
single	95.08	96.09	94.61	94.92	95.64	95.57	94.69	94.54
pos+feats	95.23	96.21	94.60	95.26	95.59	95.71	94.63	94.16
pos+feats+chunks$_{75}$	**95.89**	**96.72**	**95.58**	**96.31**	**96.38**	**96.45**	96.04	95.51
pos+feats+chunks$_{95}$	95.86	96.52	95.52	96.21	96.35	96.33	**96.21**	**95.60**

	bg		de		ja	
	pos	feats	pos	feats	pos	feats
udpipe	**97.78**	**95.55**	92.03	70.18	96.39	-
single	97.41	95.06	93.07	87.14	96.97	-
pos+feats	97.69	94.84	92.90	**87.28**	-	-
pos+feats+chunks$_{75}$	97.49	94.58	**93.34**	87.03	96.98	-
pos+feats+chunks$_{95}$	97.44	94.45	92.90	87.11	**97.09**	-

Table 2: Multi-task tagging performance on English UD treebanks (en-ewt, en-gum, en-lines, and en-partut), Bulgarian-BTB (bg), German-GSD (de), and Japanese-GSD (ja) UD treebanks: single, single-task training; pos, with POS tagging; feats, with morphological feature tagging (except Japanese (ja) which has no morphological features); and chunks$_x$, with chunks with threshold x.

	baseline		multi	
	75%	95%	75%	95%
en-ewt	89.99	91.59	91.84	92.98
en-gum	85.76	88.11	88.08	89.98
en-lines	86.01	88.38	88.45	90.67
en-partut	88.36	90.78	91.79	93.30
bg	92.27	92.60	93.79	94.45
de	88.74	88.97	89.35	89.62
ja	93.35	92.73	94.39	94.02

Table 3: Chunker F1 scores in multi task setting where the baseline presented is from training the chunker for a given ruleset with threshold 75% or 95% as a single task and multi is from training with pos and morphological feature tagging except for Japanese (ja) which has no morphological features.

	en-ewt		en-gum		en-lines		en-partut	
	uas	las	uas	las	uas	las	uas	las
no features^{udpipe}	80.97	77.87	76.70	72.71	76.43	71.87	81.63	78.67
pos^{udpipe}	84.88	81.79	81.09	76.87	79.06	74.08	84.01	80.63
pos	86.15	83.29	**83.03**	**79.31**	80.76	76.12	85.83	82.69
pos-feats	86.32	83.37	82.83	79.13	**81.15**	**76.48**	86.71	83.60
pos-chunks$_{75}$	85.84	82.87	82.49	78.83	80.86	76.04	87.03	83.86
pos-chunks$_{95}$	85.80	82.86	81.95	78.19	80.32	75.55	86.65	83.36
pos-feats-chunks$_{75}$	**86.43**	**83.41**	82.61	78.86	81.13	76.21	87.09	83.86
pos-feats-chunks$_{95}$	85.99	83.04	82.15	78.50	80.82	76.09	**87.35**	**84.04**

	bg		de		ja	
	uas	las	uas	las	uas	las
no features^{udpipe}	86.49	82.43	63.20	58.86	89.96	88.43
pos^{udpipe}	89.48	85.30	79.39	74.04	92.49	90.42
pos	89.47	85.11	81.77	76.69	**93.68**	**91.70**
pos-feats	**89.74**	**85.48**	**82.05**	**77.12**	-	-
pos-chunks$_{75}$	89.23	84.67	81.49	76.54	93.28	91.41
pos-chunks$_{95}$	89.06	84.77	81.55	76.40	92.95	91.20
pos-feats-chunks$_{75}$	89.11	84.83	81.77	76.71	-	-
pos-feats-chunks$_{95}$	89.24	85.07	81.41	76.38	-	-

Table 4: Feature input ablation for dependency parser with English UD treebanks (en-ewt, en-gum, en-lines, and en-partut), Bulgarian-BTB (bg), German-GSD (de), and Japanese-GSD (ja) UD treebanks: no features^{udpipe}, no features but UDPipe predicted POS tags used to decode; pos, gold POS tags for training and predicted POS tags for runtime (pos^{udpipe} UDPipe predicted POS tags used); feats, gold morphological feature tags for training and predicted feature tags for runtime; and chunks$_x$, gold chunks with threshold x at training time and predicted chunks for runtime.

	en-ewt		en-gum		en-lines		en-partut	
	uas	las	uas	las	uas	las	uas	las
single^{udpipe}	80.97	77.87	76.70	72.71	76.43	71.87	81.63	78.67
pos	84.52	81.30	78.94	74.96	78.75	74.13	83.66	80.25
pos-feats	84.21	81.14	79.51	75.42	78.56	73.87	84.10	81.31
pos-chunks$_{75}$	**84.55**	**81.51**	79.54	75.48	78.17	73.55	83.86	81.13
pos-chunks$_{95}$	84.42	81.34	79.60	75.54	78.72	**74.20**	83.57	80.16
pos-feats-chunks$_{75}$	84.25	81.24	**79.81**	**75.84**	78.75	73.95	84.01	80.90
pos-feats-chunks$_{95}$	84.24	81.18	79.48	75.36	**78.84**	74.15	**84.98**	**81.92**

	bg		de		ja	
	uas	las	uas	las	uas	las
single^{udpipe}	86.49	82.43	63.20	58.86	89.96	88.43
pos	88.00	83.89	80.75	75.59	**93.25**	91.45
pos-feats	88.07	83.89	80.46	75.50	-	-
pos-chunks$_{75}$	87.90	83.66	**81.29**	**75.96**	**93.25**	**91.61**
pos-chunks-$_{95}$	88.07	83.93	80.98	75.71	93.04	91.28
pos-feats-chunks$_{75}$	**88.26**	**84.00**	80.77	75.52	-	-
pos-feats-chunks$_{95}$	88.09	83.67	80.69	75.63	-	-

Table 5: Multi-task parsing results for English (en-ewt, en-gum, en-lines, and en-partut), Bulgarian-BTB (bg), German-GSD (de), and Japanese-GSD (ja) UD treebanks: single^{udpipe}, parsing as single task with UDPipe predicted POS tags used to decode parser output; pos, with POS tagging as aux. task; feats, with morphological feature tagging as aux. task; and chunks$_x$, with chunking as aux. task for threshold x.

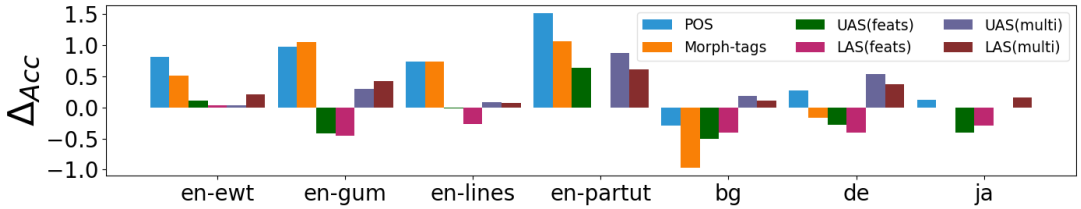


Figure 4: Difference in accuracy for each task between the best model with chunks and the best without.

performance with a meaningful increase in accuracy observed over baseline models for each treebank.

As can be seen in Figure 4, the change in performance when using the predicted chunks as a feature for parsing is less profound than in the multi-task experiments. Only two English treebanks explicitly benefit from predicted chunks, whereas all treebanks benefit from at least one feature. So the performance is at least implicitly improved by using our chunks, except for the more morphologically-rich (especially with respect to verbal inflection) Bulgarian. The treebank used for Japanese, generally an agglutinative language, does not contain morphological features, so perhaps it too would not improve with chunks if they could have been used. Therefore, it would be interesting to evaluate whether the impact of chunking information is predicated by certain linguistic features. Furthermore, the increase in performance for each treebank for the multi-task experiments suggests that the performance when using the chunks as input would improve with better predicted chunks, which corroborates the findings of Lacroix (2018).

7 Conclusion

We have introduced a language-agnostic method for extracting chunks from dependency treebanks. We have also shown the efficacy of these chunks with respect to improving POS tagging, morphological feature tagging, and dependency parsing for a number of UD treebanks.

Acknowledgments

This work has received funding from the European Research Council (ERC), under the European Unions Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from the ANSWER-ASAP project (TIN2017-85160-C2-1-R) from MINECO, and from Xunta de Galicia (ED431B 2017/01). We thank one anonymous reviewer for in-depth comments and suggestions.

References

S. Abney, 1997. *Part-of-Speech Tagging and Partial Parsing*, pages 118–136. Springer Netherlands, Dordrecht.

Gluseppe Attardi and Felice DellOrletta. 2008. Chunking and dependency parsing. In *Proceedings of LREC Workshop on Partial Parsing: Between Chunking and Deep Parsing*.

Thomas Back. 1996. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press.

Fabio Ciravegna and Alberto Lavelli. 1999. Full text parsing using cascades of rules: an information extraction perspective. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics*.

Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, jul.

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

Mucahit Kutlu and Ilyas Cicekli. 2016. Noun phrase chunking for turkish using a dependency parser. In *Information Sciences and Systems 2015*, pages 381–391. Springer.

Ophélie Lacroix. 2018. Investigating NP-Chunking with Universal Dependencies for English. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 85–90.

Dat Quoc Nguyen and Karin Verspoor. 2018. An improved neural network model for joint POS tagging and dependency parsing. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 81–91, Brussels, Belgium, October. Association for Computational Linguistics.

Joakim Nivre, Željko Agić, Lars Ahrenberg, et al. 2017. Universal Dependencies 2.1. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, et al. 2018. Universal Dependencies 2.3. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Lance A Ramshaw and Mitchell P Marcus. 1999. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

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

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 231–235, Berlin, Germany, August. Association for Computational Linguistics.

Milan Straka and Jana Straková. 2019. Universal dependencies 2.4 models for UDPipe (2019-05-31). LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez. 2019. Viable dependency parsing as sequence labeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 717–723, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Aniruddha Tammewar, Karan Singla, Bhasha Agrawal, Riyaz Bhat, and Dipti Misra Sharma. 2015. Can distributed word embeddings be an alternative to costly linguistic features: A study on parsing hindi. In *Proceedings of the 6th Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2015)*, pages 21–30.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Chunk parsing revisited. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 133–140, Vancouver, British Columbia, October. Association for Computational Linguistics.

David Vilares, Mostafa Abdou, and Anders Søgaard. 2019. Better, faster, stronger sequence tagging constituent parsers. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3372–3383, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Jie Yang and Yue Zhang. 2018. NCRF++: An Open-source Neural Sequence Labeling Toolkit. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Appendix A Evolutionary algorithm

Algorithm 1 Evolutionary algorithm

```
for gen ← max_gen do
    for ind in population do
        ind.fit ← GETFITNESS(ind)
    end for
    offspring ← SELECT(population)
    offspring ← CLONE(offspring)
    for pair in offspring_2i, offspring_2i+1 do
        if random < P_crossover then
            pair ← CROSSOVER(pair)
        end if
    end for
    for ind in offspring do
        if random < P_mutate then
            ind ← MUTATE(ind)
        end if
    end for
    population ← offspring
end for

function GETFITNESS(ind)
    rules ← CONVERT(ind)
    train, dev ← CHUNKTREEBANKS(rules)
    TRAINCHUNKER(train)
    F1 ← EVALULATECHUNKER(dev)
    Rp ← GETMAXRPROPORTION(dev)
    return F1 + 0.5·Rp
end function
```

Appendix B Hyperparameters

hyperparameter	**value**
population size	100
number of generations	4
k-best	5
P_{mutate}	0.5
$P_{\text{mutate gene}}$	0.05
$P_{\text{crossover}}$	0.5
decay (linear)	0.1

Table 6: Hyperparameters for the evolutionary algorithm: k-best, the number of best parents chosen to seed next generation; P_{mutate}, the probability an individual will mutate; $P_{\text{mutate gene}}$, the probability a given gene will mutate; $P_{\text{crossover}}$, the probability a pair of individuals will crossover; and decay is how much P_{mutate} and $P_{\text{crossover}}$ decrease after each generation.

hyperparameter	**value**
BiLSTM dimensions	200
BiLSTM layers	1
word embedding dimensions	50
character embedding dimensions	30
character hidden dimensions	50
character CNN layers	4
CNN window size	3
optimiser	SGD
loss function	cross entropy
learning rate	0.015
decay (linear)	0.05
momentum	0.9
dropout	0.5
L_2 regularisation	$1x10^{-8}$
epochs	5
training batch size	10
runtime batch size	128

Table 7: Hyperparameters for the neural-net chunker used during the evolutionary algorithm.

hyperparameter	**value**
BiLSTM dimensions	800
BiLSTM layers	2
word embedding dimensions	100
character embedding dimensions	30
character hidden dimensions	50
feature dimensions	20
optimiser	SGD
loss function	cross entropy
learning rate	0.2
decay (linear)	0.05
momentum	0.9
dropout	0.5
epochs	100
training batch size	8
runtime batch size	128

Table 8: Hyperparameters for the network used in all experiments.