DepDist: Surface realization via regex and dependency distance tolerance

William Dyer
Oracle Corp
william.dyer@oracle.com

Abstract

This paper describes a method of inflecting and linearizing a lemmatized dependency tree by: (1) determining a regular expression and substitution to describe each productive word-form rule; (2) learning the dependency distance tolerance for each head-dependent pair, resulting in an edge-weighted directed acyclic graph (DAG); and (3) topologically sorting the DAG into a surface realization based on edge weight. The method's output for 11 languages across 18 treebanks is competitive with the other submissions to the Second Multilingual Surface Realization Shared Task (SR'19).

1 Introduction

The goal of the Second Multilingual Surface Realization Shared Task (SR'19) is to generate a morphologically inflected surface order from a lemmatized and unordered dependency tree (Mille et al., 2019). In track 1, all lemmas in the dependency tree are given, and the task is closed in the sense that only the provided training data may be used; outside data is not allowed.

Though conceptually straightforward, linearizing a dependency tree in an automated way is a relatively difficult task given issues such as projectivity, flexibility or variation in word-order preferences among humans, polysemy and homography, among others. Determining surface inflections is similarly difficult given the sometimes opaque relationship between spoken and written language, diversity among language varieties, usage preferences changing over time, and vestigial inflectional forms which may or may not be productive.

The approach outlined in this paper tackles the inflection part of the task by attempting to determine the productive rules for word forms, implemented as a series of regular expressions and substitutions. Given the closed nature of the task, these regular expressions are based on orthographic forms, rather than what would likely be more accurate phonological representations.

To linearize a dependency tree, the current study's approach is two-fold: first, learn the tolerance for how far apart a dependent and its head can be within the context of a given sentence; second, use this dependency distance tolerance to sort the tree into a surface order. The sorting can be accomplished such that only projective surface orders are generated, or without any baked-in notion of projectivity. Algorithms for both are presented here, but given the nature of the task—and based on empirical testing—only projective linearizations were submitted as part of the shared task.

2 Inflecting

The current model's approach to inflecting lemmas to arrive at wordforms is to first look up the lemma and target morphological form in the training data—if the form exists in the training data for the lemma, it is used in the test data. For example, the past participle of the lemma *do* is most likely present in the training data, so when the testing data prompts for it, *done* is simply supplied from the training set. More interestingly, lemmas unseen during training are handled with a series of regular expressions (regex) built up from the training data in an attempt to define a natural language's productive inflectional rules.

From a linguistic perspective, inflecting unseen test-set words is analogous to inflecting nonce words, a rather long-studied area. For example, the 'wug' test (Berko, 1958) shows that children possess knowledge about morphological rules. It is intuitive to conceive of these rules as regular or irregular—*box* → *boxes* illustrates the regular plural in English, *ox* → *oxen* an irregular form—and

Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019), pages 25–34
Hong Kong, China, November 3rd, 2019.
https://doi.org/10.18653/v1/P17

lemma *wordform*	*regex* *substitution*	*nonce lemma* *nonce wordform*
like- liked	^(.*)(e)$ \1\2d	chortle chortled
attach-- attached	^(.*)(h)$ \1\2ed	gallumph gallumphed
presentar presentó	^(.*)(t)ar$ \1\2ó	risotar risotó
triunfar triunfó	^(.*)(f)ar$ \1\2ó	galonfar galonfó

Table 1: Regular expressions and substitutions for simple past with nonces from *Jabberwocky* (Carroll, 1872) and Spanish translation *El Fablistanón* (Pascual, 1977).

to subsequently equate productive rules with regular forms only. However, a more accurate model is that speakers seem to inflect nonce words according to categories which span what we tend to think of as both regular and irregular classes. The college students studied by Bybee and Moder (1983) produce simple past forms for nonces such as *spling* ← *splung*, akin to 'strong' verbs such as *cling* ← *clung* and *string* ← *strung* (Wiese, 1996). Prasada and Pinker (1993) find that the production and acceptance of inflected nonces correlates with phonological distance from irregular clusters, with a bias towards regular forms (p. 48).

The current model approximates the phonological environment of word stems with regular expressions and morphological inflections with substitutions. There is no notion of regular or irregular classes; regexes and substitutions are built for all classes and sorted according to frequency. If a nonce word's lemma matches the regex of a morphological class from the training data, the associated substitution will provide an inflected form. Importantly, given the closed nature of SR '19—no outside data is allowed—the generated regexes and substitutions are defined and employed orthographically rather than phonetically or phonologically. As such, depending on the opacity of a language's orthographic system, information about allophones, syllables, and other phonetically important structures is lost. Interestingly, this loss does not seem to impact neural-network models of inflection (Wiemerslage et al., 2018), though the current model's rule-based approach likely suffers.

Defining the orthographic environment such that known lemma-to-wordform exemplars can be used to create a prototypical regex for a given class can be accomplished by (1) aligning the lemma and wordform; (2) recording the characters surrounding a replacement as atoms; (3) generalizing atoms not surrounding substitutions; and (4) determining the substitution(s).

Table 1 shows this process for a sample of simple past forms. For example, the English lemma *like* is aligned with the target wordform *liked*, the regex defining the environment is `^(.*)(e)$`, and the substitution with back-references is `\1\2d`. When applied to the nonce lemma *chortle*, the correct wordform *chortled* is produced. That is, the regex matches the lemma *chortle* ending in the character *e*, and the substitution maintains the atomic root *chortl*, maintains the final character *e*, and appends the character *d*.

Alignment of lemmas and wordforms is accomplished with the `pairwise2` module from Biopython (Cock et al., 2009). Regex and substitution generation is done with a deterministic algorithm which generalizes uninvolved atoms `(.*)`, records adjacent atoms in the lemma, and produces back-references and inflectional morphemes for the substitution. Morphological features are treated as full strings rather than as discrete features—something like `Mood=Ind|Person=3|Tense=Past|VerbForm=Fin|VERB`, depending on the corpus. Each of these feature sets is generally associated with multiple patterns, as in Table 1.

There are at least two intuitive approaches for choosing a regex pattern for an unknown lemma: the most detailed or the most frequent. The first approach relies on a principle going back to Pāṇini in which inflections obey specific conditions before general ones (cf. Embick and Marantz, 2005). However, during testing, this approach resulted in archaisms or typos in the generated text. Thus the most frequent pattern was chosen instead.

3 Linearizing

The task of linearizing a dependency tree can be informed by long-standing linguistic principles describing the placement of words in general—"what belongs together semantically is also placed close together" (Behaghel, 1932, p. 4), for example—as well as more recent work on dependency trees specifically, such as Dependency Distance minimization (DDM) (Hudson, 1995; Futrell et al., 2015; Liu et al., 2017). DDM is a general principle of tree ordering based on Head

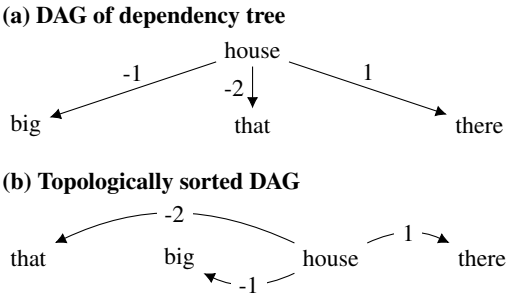


Figure 1: Target dependency distance tolerances for *that big house there*, represented as **(a)** a DAG showing dependency relations and **(b)** the topological sort.

Proximity (Rijkhoff, 1986), Early Immediate Constituents (Hawkins, 1994), Dependency Locality Theory (Gibson, 2000), and Minimize Domains (Hawkins, 2004), among others.

Submissions to SR '18, the first multilingual shared task, are generally based on sequence-to-sequence machine translation (Elder and Hokamp, 2018; Sobrevilla Cabezudo and Pardo, 2018), binary classification (Castro Ferreira et al., 2018; Puzikov and Gurevych, 2018; King and White, 2018; Madsack et al., 2018), or probabilistic n-gram language models (Singh et al., 2018).

3.1 Dependency distance tolerance

The DepDist approach to linearization relies on dependency distance tolerance, the idea that a dependent and head tolerate a certain contextual distance, measured as the number of intervening words, relative to other words in a sentence (Dyer, 2019). This dependency distance tolerance is learned from training data via a graph neural network (GNN) implemented within the Graph Nets framework (Battaglia et al., 2018) based on `word2vecf` syntactic embeddings (Levy and Goldberg, 2014). GNNs take advantage of message-passing neural networks (MPNN), in which nodes pass information and spatial-based convolutions and pooling are implemented (Gilmer et al., 2017; Wu et al., 2019).

Specifically, each word's 300-element syntactic embedding is included as a node attribute for a `networkx` graph constructed for each sentence in the training, dev, and testing sets. Input edge attributes are the average dependency distance between words from the training set. For example, if the determiner *the* precedes the noun *cat* by an average of 1.3 words in the training data, the input edge attribute for *the* $\leftarrow$ *cat* will be 1.3. After 5 training epochs of 100 iterations on a GNN with 64 neurons and 8 MPNN layers using an Adam optimizer in TensorFlow, output edge attributes reflect the learned dependency distance tolerances for each dependent-head pair in a given sentence.

For example, given a simple tree of one head, *houses*, three dependents—*big*, *that*, and *there*—and a target linearization of *that big house there*, the learned directed dependency distances would be *that* $\overset{-2}{\leftarrow}$ *house*, *big* $\overset{-1}{\leftarrow}$ *house*, and *house* $\overset{1}{\rightarrow}$ *there*. In other words, the dependent *that* precedes its head *house* by two words, *big* precedes *house* by one word, and *there* follows *house* by one word. This example is shown in Figure 1(a).

The GNN framework allows for non-Euclidean data representations, such as graphs, to be explored from a deep learning perspective (Bronstein et al., 2017). Further, GNNs are invariant to permutations in the graph elements—ideal for this surface realization shared task—and can operate on inputs of varying sizes (Battaglia et al., 2018).

3.2 Topological sorting

A dependency tree can be represented by a directed acyclic graph (DAG) based on the [head $\rightarrow$ dependent] relation. Adding edge weights representing directed dependency distances—the number of words a dependent precedes or follows its head—allows the DAG to also represent the precedence relation. Thus an edge-weighted DAG is equivalent to a partially ordered set (poset).

The topological sort of a non-weighted DAG or poset is not guaranteed to be unique, but adding edge weights allows a single linear order to be generated. For example, Figure 1(b) shows the unique topological sort for *that big house there*, based on the precedence relations *house* $\overset{-2}{\rightarrow}$ *that*, *house* $\overset{-1}{\rightarrow}$ *big*, and *house* $\overset{1}{\rightarrow}$ *there*[1].

The linearization of a dependency tree can be projective (Marcus, 1965), in which there are no crossing arcs, or non-projective. More formally, a projective order is one in which every word w occurring between a dependent d and head h is dominated by h (Nivre, 2006), and as such is only defined for dependency-based DAGs[2].

[1]The notation of *house* $\overset{-2}{\rightarrow}$ *that* indicates the dependency relation by the direction of the edge and distance by edge weight. An equivalent notation would be *that* $\overset{2}{\prec}$ *house*.

[2]Posets can be classified according to their planarity, and while half-planarity corresponds to the 'no-crossing-arcs' sense of projectivity (Pitler et al., 2013), it does not capture the dominance definition.

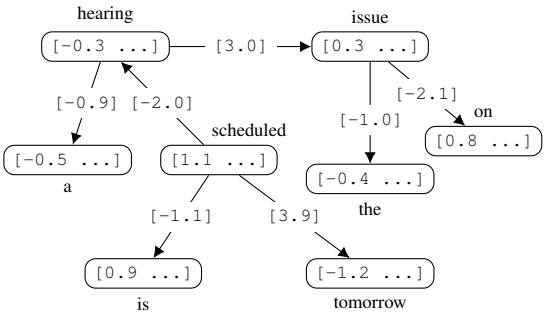
Algorithm 1: Topologically sort DAG (non-projective)

```
1:  function NonProjTopoSort(dag)
2:      tuples ← ∅
3:      tuples.add(0, dag.root)
4:      CalcI(root, dag, tuples)
5:      return tuples.sort()
6:  end function
7:  function CalcI(node, dag, tuples)
8:      if node.hasHead then
9:          head ← head(dag, node)
10:         edge = dag[node][node.head]
11:         tuples.add(head.i + edge, node)
12:     end if
13:     for child ∈ node.children do
14:         CalcI(child, dag, tuples)
15:     end for
16: end function
```

Algorithm 2: Topologically sort DAG (projective)

```
1:  function ProjTopoSort(dag)
2:      tuples ← ∅
3:      tuples.add(0, dag.root)
4:      CalcI(root, dag, tuples)
5:      return tuples.sort()
6:  end function
7:  function CalcI(node, dag, tuples)
8:      head ← head(dag, node)
9:      edge = dag[node][node.head]
10:     children ← 0
11:     for desc ∈ node.descendents do
12:         children+ ← abs(dag[desc][desc.head])
13:     end for
14:     if edge < 0 then
15:         children ← −children
16:     end if
17:     tuples.add(head.i + edge + children, node)
18:     for child ∈ node.children do
19:         CalcI(child, dag, tuples)
20:     end for
21: end function
```

(a) dependency structure used in GNN



(b) non-projective linearization

a	hearing	is	scheduled	on	the	issue	tomorrow
-2.9	-2.0	-1.1	0.0	0.9	2.0	3.0	3.9

(c) projective linearization

a	hearing	on	the	issue	is	scheduled	tomorrow
-9.9	-9.0	-5.0	-3.9	-2.9	-1.1	0.0	3.9

Figure 2: Linearizing a DAG. **(a)** A `networkx` graph provides the input and output of the GNN. Input node attributes are each word's syntactic embedding, and input edge attributes are the average directed dependency distances between connected words in the training data. Target edge attributes are the actual dependency distances between the words in the sentence. The GNN learns edge attributes for the test sentences. In practice, edge directions may be reversed to better take advantage of MPNN. **(b)** Non-projective linearization in which each word's index reflects its distance from the root. **(c)** Projective linearization, in which each word's index is set such that all descendents will be adjacent.

Algorithm 1 sorts an edge-weighted DAG without regard to projectivity. Each node's distance from the root is calculated by adding the weight of the node's edge with its head to its head's index (lines 9-11). This distance becomes an index i for each word; sorting these indexes from smallest to largest (line 5) creates a linearization for the dependency tree which may or may not be projective. The calculation of root distance in Algorithm 1 runs in $\mathcal{O}(n)$ time, since each node is only visited once and is able to calculate its distance based on the index of its parent node. The sorting algorithm is not specified, but assuming something like merge sort (Knuth, 1998) with a time complexity of $\mathcal{O}(n \log n)$, the overall complexity of Algorithm 1 would be $\mathcal{O}(n \log n)$.

Algorithm 2 sorts an edge-weighted DAG into a projective linearization based on the idea that each dependent d should be placed *vis-à-vis* its head h such that all descendents of d could be placed between d and h. The index i in a linearization for dependent d is the sum of (1) the index of its head h (line 8); (2) the edge weight between d and h (line 9); and (3) the summed absolute value of the edges of all descendents of d whose polarity matches that of d (lines 10-16). The calculation of i in Algorithm 2 runs in $\mathcal{O}(n \log n)$ time, since each node is visited once by CalcI, and then in lines 11-13 each descendent node is visited. Coupled with merge sort, Algorithm 2 overall runs in $\mathcal{O}(n \log n)$ time.

Algorithms 1 and 2 are exemplified in Figure 2, in which a dependency tree (a), is sorted into a valid non-projective linearization (b) and a projective linearization (c). Due to the nature of GNNs, the size of the graph need not be standardized—the graph is simply a series of connected nodes and edges. Input node attributes are passed along directed edges, and output edges reflect learned distance tolerances. These tolerances are then used to topologically sort the DAG.

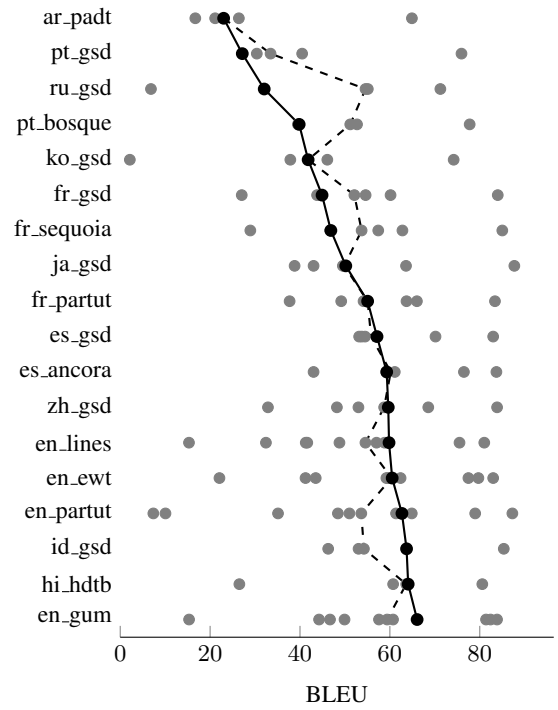


Figure 3: BLEU scores per corpus for all track-1 submissions. DepDist shown by black connected points. Median indicated by dashed line.

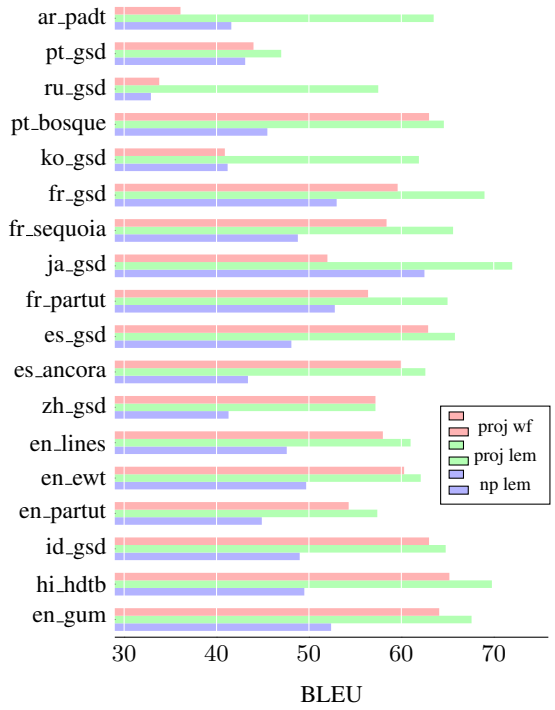


Figure 4: BLEU scores from dev sets realized as projective wordforms (red), projective lemmas (green), and non-projective lemmas (blue).

3.3 Projective linearizations

Imposing a projective limitation on generated outputs is a theoretically dubious action when describing natural language (Ferrer-i Cancho and Gómez-Rodríguez, 2016; Yadav et al., 2019). However, given the strong tendency towards projectivity (Mambrini and Passarotti, 2013; Gómez-Rodríguez, 2016), the nature of SR'19 as a fundamentally Natural Language Generation (NLG) rather than descriptive task, as well as empirical observation of the projective and non-projective outputs of the current model (§4.1), it was decided to submit only projective linearizations.

4 Results

DepDist was run on 18 corpora across 11 languages provided by the organizers of SR'19, based on Universal Dependencies corpora. Due to time constraints, only the largest corpus for each language was used for training, though linearizations were generated for nearly all test corpora[3].

Results for the DepDist submission measured by BLEU score (Papineni et al., 2002) compared to other track-1 submissions for the 18 corpora are shown in Figure 3. DepDist performed below the median for six corpora: Portuguese (GSD & Bosque), Russian (GSD), French (GSD & Sequoia), and Spanish (Ancora). DepDist was the median for three corpora: Arabic (GSD), Korean (GSD), and French (ParTUT). DepDist performed better than the median for nine corpora: Japanese, (GSD), Spanish (GSD), Chinese (GSD), English (LinES, EWT, ParTUT, & GUM), Indonesian (GSD), and Hindi (HDTB).

While the performance on some corpora was significantly below the median—especially Russian and both Portuguese—DepDist generally performed close to or slightly better than the median. Thus DepDist seems to be fairly average in terms of BLEU score compared to the other submissions, suggesting that it is a competitive solution.

4.1 Error analysis

Figure 4 plots BLEU scores for the dev set of each corpus differentiated based on projectivity and inflections. The first, red bar shows the projective linearization of wordforms. The second, green bars show the scores based on linearization only, without inflecting. The third, blue bars show non-projective linearizations of lemmas.

[3]Though ko_kaist was the largest supplied Korean corpus, parsing errors prevented training and generation of output; thus only ko_gsd was used. Similarly, the large size of ru_syntagrus inhibited training, so only ru_gsd was used.

		morph features		*wordforms*	
corpus	*lemmas*	*given*	*%*	*generated*	*%*
ar_padt	28264	21901	77.5	2138	9.8
en_ewt	25096	16653	66.4	2324	14.0
en_gum	13326	8880	66.6	1691	19.0
en_lines	15623	9796	62.7	1007	10.3
en_partut	3408	2179	63.9	179	8.2
es_ancora	52617	43676	83.0	8196	18.8
es_gsd	12000	6854	57.1	646	9.4
fr_gsd	10021	5770	57.6	400	6.9
fr_partut	2604	1752	67.3	147	8.4
fr_sequoia	10050	6306	62.7	384	6.1
hi_hdtb	35430	30137	85.1	2605	8.6
id_gsd	11780	6036	51.2	661	11.0
ja_gsd	12438	219	1.8	21	9.6
ko_gsd	11677	82	0.7	35	42.7
pt_bosque	10199	6553	64.3	1062	16.2
pt_gsd	31496	3065	9.7	294	9.6
ru_gsd	11548	7268	62.9	3236	44.5
zh_gsd	12012	1349	11.2	112	8.3

Table 2: The number of lemmas in each test corpus showing (1) the number and percentage for which morphological features were given and (2) the number and percentage of wordforms generated via regex.

Across all corpora, projective linearizations of lemmas in the dev set generate the highest BLEU scores. The difference between the first two bars for each corpus indicates how well the inflection subtask performed, and the difference between the second and third bars indicates the performance of the linearization subtask.

In all languages other than Chinese, poor inflections hurt the scores, and in Arabic, Japanese, Korean, and Russian, the inflections were quite detrimental. The regex methodology used in the current study depends on a set of morphological features to use as a key for finding an appropriate pattern, but corpora vary as to what proportion of lemmas have this morphological listing. Table 2 shows the number and rate of lemmas at which morphological features are listed, as well as how many of those were generated by the regex pattern-substitution methodology. For example, of the 28,264 lemmas in the Arabic (PADT) test corpus, 21,901 (77.5%) had associated morphological information; of those 21,901, only 2,138 (9.8%) were generated via regex substitution—the other 90.2% were found in the training data.

Both Japanese (GSD) and Korean (GSD) provide exceedingly low rates of morphological data—1.8% and 0.7%, respectively. Thus the differential between the BLEU scores of projective wordforms (first, red bars) and projective lemmas (second, green bars) in Figure 4 for these two languages is likely due to lack of morphological feature sets in the corpora.

Corpora with especially high rates of wordforms being generated via regex rather than found in the training data include Portuguese (Bosque) at 16.2%, Spanish (Ancora) at 18.8%, English (GUM) at 19%, Korean (GSD) at 42.7%[4], and Russian (GSD) at 44.5%. While the performance of the inflection systems for the first three of those corpora is relatively good, the very poor performance of Russian is surprising. The cause of the exceedingly poor performance of Arabic inflection is also unclear, given the high rate of provided morphological features (77.5%) and fairly normal rate of wordform generation (9.8%); perhaps the methodology is poorly suited to Arabic and/or Russian inflectional patterns.

One possible source of error is the use of the most frequent regex pattern when generating wordforms, rather than the most detailed or specific. This likely creates a bias towards overly 'regular' forms whereby the phonetic environment is not able to properly trigger substitutions. This effect may be more strongly felt by languages with richer morphologies, such as Arabic and Russian.

In general, the reliance on orthography for defining phonetic environments for regexes and substitutions almost certainly contributes to error. This could probably be improved by using IPA transcriptions or distinctive phonetic features rather than standard orthography, as well as a more flexible regex patterning which could better handle allophones. A relatively straightforward way to implement a bit of phonetic flexibility would be to utilize substitution matrices when aligning lemmas to their target wordforms (Smith and Waterman, 1981). This approach would allow, for example, a [b] and [v] to be seen as more similar than other phones, and could therefore be combined into a single regex atom for a given language.

The second and third bars in Figure 4 for each corpus differentiate based on projectivity: in all cases the non-projective linearizations (third, blue bars) have lower scores than the projective ones (second, red bars). This is not too surprising, since a single misplaced word can drastically reduce BLEU scores. However, if the GNN were able to better learn dependency distance tolerances, the non-projective sorting algorithm should produce

[4]The exceedingly low rate of provided morphological features for Korean (GSD) renders the percentage of generated wordforms a rather uninformative number for this corpus.

results similar to the projective algorithm, if not better, given the existence of non-projective sentences in possibly all natural languages (Ferrer-i Cancho and Gómez-Rodríguez, 2016) and preference for certain linearizations such as Figure 2(b).

Because the same GNN is used to learn distances, and projectivity is only realized during linearization, the difference in performance between projective and non-projective linearizations suggests that the GNN is learning tendencies for dependents to precede or follow their heads, as well as the relative tolerances among sibling dependents, to a certain degree. However, the accuracy of those tolerances with respect to all other words in a sentence leaves room for improvement, probably via an enhanced GNN architecture.

5 Discussion

The regex-based approach to inflection employed in the current study is linguistically motivated. Regex patterns would seem to be an adequate method for modeling exemplars and grouping them into templates, and substitutions allow for productive inflectional patterns to be applied to uninflected lemmas. The choice of which regex pattern to employ for a given lemma may be more complex than outlined here—a choice between the most frequent or the most detailed, and given the error rates around inflections in Figure 4, perhaps the most detailed would perform better. Still, the notion is plausible. A trade-off between frequency and level of regex detail might go some way towards modeling the loss of increasingly obscure inflectional patterns in favor of those which are more frequent.

DepDist tackles the problem of linearization entirely within a dependency framework. Words are represented by their syntactic embeddings, and the neural network is a graph built from a dependency tree. The learning of dependency distance tolerances is accomplished via these embeddings and graphs. The only point at which the notion of linearity comes into play is after all learning has completed, when distance tolerances are fed into a deterministic algorithm for topological sorting.

This approach is quite different from an n-gram language model or one based on machine translation. With DepDist, if adjacent words are not connected by a dependency relation, their linear adjacency is in a sense emergent, a necessary by-product of converting a two-dimensional tree into a one-dimensional linearization. Thus the order of sibling dependents is not directly modeled, but is instead implicitly reflected in the relative distance tolerances. However, due to message passing, siblings can be made indirectly aware of each other—since dependents pass their embedding node attributes to the head, the calculation of edge attributes between the head and each dependent reflects the presence of other siblings.

Further, DepDist is not an end-to-end machine-learning model. The actual linearized strings are not the target; rather, individual dependency distance tolerances are the target of learning. The data structure which results from weighting the edges of a directed graph and its subsequent topological sort generate a linearization based on dependency distance tolerance.

Although a projectivity constraint was artificially employed in the implementation of DepDist outlined here, if the GNN were to better learn dependency distance tolerances, that constraint would not be needed. Instead, observed rates of projectivity among languages should arise as a result of topologically sorting based on distance tolerance. Crucially, the rate of projectivity is not directly learned. A GNN—or human—is exposed to language in which head-dependent pairs have certain distance tolerances, tolerances which can be learned. Assembling the pairs such that these tolerances are obeyed results in largely projective linearizations, though not exclusively so, thereby reflecting a tendency towards projectivity.

Dependency distance tolerance seems to be a psychologically real measure. In the current study, the tolerances are learned via GNN, but they might be operationalized in other ways, especially by psycholinguistic or information-theoretic measures (cf. Scontras et al., 2017; Dyer, 2018; Hahn et al., 2018). That is, a dependent which tolerates a large linear distance from its head, such as the adverb *tomorrow* in the example in Figure 2, may have a lower pointwise mutual information (Church and Hanks, 1989) or surprisal (Futrell and Levy, 2017) with its head, or may have higher or lower subjectivity than the auxiliary *is*. As such, because *tomorrow* and *scheduled* belong together semantically less than *is* and *scheduled*, or they depend on each other less, the adverb is allowed to be placed farther away. This is a sort of conceptual inversion of Behaghel (1932)—what does not necessarily belong together can be placed far apart.

5.1 Future directions

Given that the performance of DepDist is competitive with many of the other submissions to SR '19, the approach seems promising. The error analysis indicates deficiencies in the rule-based approach to inflections, possibly due to reliance on orthography to approximate phonetic environments, as well as a reliance on morphological-feature listings which may not always be present in Universal Dependencies corpora. The GNN's ability to learn accurate dependency distance tolerances at the sentence level is promising, but leaves significant room for improvement. For example, the GNN's architecture may be too small, the syntactic embedding framework may be too old to properly generalize from training data, the training data may be too limited, and the training of only 5 epochs may be too few to properly learn distance tolerances. All of these areas can be explored in future study.

Finally, training was confined to a single training corpus per language—future study should at least take advantage of all available corpora for a given language. More promisingly, transfer learning could be employed to take advantage of cross-linguistic tendencies regarding dependency distance tolerance.

6 Summary

This paper describes the DepDist submission to SR '19. The approach to inflecting uses regular expressions and substitutions to learn morphological prototypes from training exemplars, which can be applied to words unseen during training. Linearizing a tree is accomplished by first learning dependency distance tolerances via syntactic word embeddings and a graph neural network (GNN), then sorting the resulting edge-weighted directed acyclic graph (DAG) according to either projective or non-projective algorithms, only the former of which were submitted. The results of DepDist are competitive, the approach is linguistically grounded, and there is ample room for improvement to both the inflectional module and GNN architecture.

Acknowledgments

Special thanks to Ariel Gutman for insight into tree-sorting algorithms and to the anonymous reviewers for their comments and suggestions.

References

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261 [cs, stat]*.

Otto Behaghel. 1932. *Deutsche Syntax eine geschichtliche Darstellung*. Carl Winters Universitätsbuchhandlung, Heidelberg.

Jean Berko. 1958. The Child's Learning of English Morphology. *WORD*, 14(2-3):150–177.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42. ArXiv: 1611.08097.

Joan L. Bybee and Carol Lynn Moder. 1983. Morphological Classes as Natural Categories. *Language*, 59(2):251–270.

Ramon Ferrer-i Cancho and Carlos Gómez-Rodríguez. 2016. Crossings as a side effect of dependency lengths. *Complexity*, 21(S2):320–328.

Lewis Carroll. 1872. *Through the Looking-Glass, and What Alice Found There*. MacMillan and Co, London.

Thiago Castro Ferreira, Sander Wubben, and Emiel Krahmer. 2018. Surface Realization Shared Task 2018 (SR18): The Tilburg University Approach. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 35–38, Melbourne, Australia. Association for Computational Linguistics.

Kenneth Ward Church and Patrick Hanks. 1989. Word association norms, mutual information, and lexicography. In *Proceedings of the 27th annual meeting on Association for Computational Linguistics* -, pages 76–83, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. 2009. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423.

William Dyer. 2018. Integration complexity and the order of cosisters. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 55–65, Brussels, Belgium. Association for Computational Linguistics.

William Dyer. 2019. Weighted posets: Learning surface order from dependency trees. In *Proceedings of the 18th International Workshop on Treebanks and Linguistic Theories*, Paris, France. Association for Computational Linguistics.

Henry Elder and Chris Hokamp. 2018. Generating High-Quality Surface Realizations Using Data Augmentation and Factored Sequence Models. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 49–53, Melbourne, Australia. Association for Computational Linguistics.

David Embick and Alec Marantz. 2005. Cognitive neuroscience and the English past tense: Comments on the paper by Ullman et al.*. *Brain and Language*, 93(2):243–247.

Richard Futrell and Roger Levy. 2017. Noisy-context surprisal as a human sentence processing cost model. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 688–698.

Richard Futrell, Kyle Mahowald, and Edward Gibson. 2015. Large-scale evidence of dependency length minimization in 37 languages. *Proceedings of the National Academy of Sciences*, 112(33):10336–10341.

Edward Gibson. 2000. The dependency locality theory: A distance-based theory of linguistic complexity. *Image, language, brain*, pages 95–126.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. *arXiv:1704.01212 [cs]*.

Carlos Gómez-Rodríguez. 2016. Restricted Non-Projectivity: Coverage vs. Efficiency. *Computational Linguistics*, 42(4):809–817.

Michael Hahn, Judith Degen, Noah Goodman, Dan Jurafsky, and Richard Futrell. 2018. An Information-Theoretic Explanation of Adjective Ordering Preferences. In *Proceedings of the 40th annual conference of the Cognitive Science Society.*, London. Cognitive Science Society.

John A. Hawkins. 1994. *A Performance Theory of Order and Constituency*. Cambridge University Press, Cambridge.

John A. Hawkins. 2004. *Efficiency and Complexity in Grammars*. Oxford University Press, Oxford.

Richard Hudson. 1995. Measuring syntactic difficulty.

David King and Michael White. 2018. The OSU Realizer for SRST '18: Neural Sequence-to-Sequence Inflection and Incremental Locality-Based Linearization. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 39–48, Melbourne, Australia. Association for Computational Linguistics.

Donald Knuth. 1998. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, Boston.

Omer Levy and Yoav Goldberg. 2014. Dependency-Based Word Embeddings. In *ACL (2)*, pages 302–208. Citeseer.

Haitao Liu, Chunshan Xu, and Junying Liang. 2017. Dependency distance: A new perspective on syntactic patterns in natural languages. *Physics of Life Reviews*, 21:171–193.

Andreas Madsack, Johanna Heininger, Nyamsuren Davaasambuu, Vitaliia Voronik, Michael Käufl, and Robert Weißgraeber. 2018. AX Semantics' Submission to the Surface Realization Shared Task 2018. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 54–57, Melbourne, Australia. Association for Computational Linguistics.

Francesco Mambrini and Marco Passarotti. 2013. Non-Projectivity in the Ancient Greek Dependency Treebank. In *DepLing*, pages 177–186.

Solomon Marcus. 1965. Sur la notion de projectivité. *Mathematical Logic Quarterly*, 11(2):181–192.

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.

Joakim Nivre. 2006. *Inductive Dependency Parsing*. Springer, Dordrecht, The Netherlands.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Emilio Pascual. 1977. *El Fablistanón*. Edival-Ortells.

Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2013. Finding Optimal 1-Endpoint-Crossing Trees. *Transactions of the Association for Computational Linguistics*, 1:13–24.

Prasada and S. Pinker. 1993. Generalizations of regular and irregular morphology. *Language and Cognitive Processes*, 8(1):1–56.

Yevgeniy Puzikov and Iryna Gurevych. 2018. BinLin: A Simple Method of Dependency Tree Linearization. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 13–28, Melbourne, Australia. Association for Computational Linguistics.

Jan Rijkhoff. 1986. Word Order Universals Revisited: The Principle of Head Proximity. *Belgian Journal of Linguistics*, 1:95–125.

Gregory Scontras, Judith Degen, and Noah D. Goodman. 2017. Subjectivity Predicts Adjective Ordering Preferences. *Open Mind*, pages 1–14.

Shreyansh Singh, Ayush Sharma, Avi Chawla, and A.K. Singh. 2018. IIT (BHU) Varanasi at MSR-SRST 2018: A Language Model Based Approach for Natural Language Generation. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 29–34, Melbourne, Australia. Association for Computational Linguistics.

T.F. Smith and M.S. Waterman. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197.

Marco Antonio Sobrevilla Cabezudo and Thiago Pardo. 2018. NILC-SWORNEMO at the Surface Realization Shared Task: Exploring Syntax-Based Word Ordering using Neural Models. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 58–64, Melbourne, Australia. Association for Computational Linguistics.

Adam Wiemerslage, Miikka Silfverberg, and Mans Hulden. 2018. Phonological Features for Morphological Inflection. In *Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 161–166, Brussels, Belgium. Association for Computational Linguistics.

Richard Wiese. 1996. Phonological versus morphological rules: on German Umlaut and Ablaut. *Journal of Linguistics*, 32(1):113–135.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2019. A Comprehensive Survey on Graph Neural Networks. *arXiv:1901.00596 [cs, stat]*.

Himanshu Yadav, Samar Husain, and Richard Futrell. 2019. Are formal restrictions on crossing dependencies epiphenomenal? In *Proceedings of the 18th International Workshop on Treebanks and Linguistic Theories*, Paris, France. Association for Computational Linguistics.