# Please Mind the Root: Decoding Arborescences for Dependency Parsing

**Ran Zmigrod**[🌲] **Tim Vieira**[🌴] **Ryan Cotterell**[🌲,🌳]

[🌲]University of Cambridge [🌴]Johns Hopkins University [🌳]ETH Zürich

rz279@cam.ac.uk tim.f.vieira@gmail.com
ryan.cotterell@inf.ethz.ch

## Abstract

The connection between dependency trees and spanning trees is exploited by the NLP community to train and to decode graph-based dependency parsers. However, the NLP literature has missed an important difference between the two structures: only *one* edge may emanate from the root in a dependency tree. We analyzed the output of state-of-the-art parsers on many languages from the Universal Dependency Treebank: although these parsers are often able to learn that trees which violate the constraint should be assigned lower probabilities, their ability to do so unsurprisingly degrades as the size of the training set decreases. In fact, the worst constraint-violation rate we observe is $24\%$. Prior work has proposed an inefficient algorithm to enforce the constraint, which adds a factor of $n$ to the decoding runtime. We adapt an algorithm due to Gabow and Tarjan (1984) to dependency parsing, which satisfies the constraint without compromising the original runtime.[1]

## 1 Introduction

Developing probabilistic models of dependency trees requires efficient exploration over a set of possible dependency trees, which grows exponentially with the length of the input sentence $n$.

Under an edge-factored model (McDonald et al., 2005; Ma and Hovy, 2017; Dozat and Manning, 2017), finding the maximum-a-posteriori dependency tree is equivalent to finding the maximum weight spanning tree in a weighted directed graph. More precisely, spanning trees in *directed* graphs are known as arborescences. The maximum-weight arborescence can be found in $\mathcal{O}(n^2)$ (Tarjan, 1977; Camerini et al., 1979).[2]

However, an oversight in the relationship between dependency trees and arborescences has gone largely unnoticed in the dependency parsing literature. Most dependency annotation standards enforce a **root constraint**: Exactly one edge may emanate from the root node.[3] For example, the Universal Dependency Treebank (UD; Nivre et al. (2018)), a large-scale multilingual syntactic annotation effort, states in their documentation (UD Contributors):

> There should be just one node with the root dependency relation in every tree.

This oversight implies that parsers may return *malformed* dependency trees. Indeed, we examined the output of a state-of-the-art parser (Qi et al., 2020) for 63 UD treebanks. We saw that decoding without a root constraint resulted in $1.80\%$ (on average) of the decoded dependency trees being malformed. This increased to $6.21\%$ on languages that contain less than one thousand training instances with the worst case of $24\%$ on Kurmanji.

The NLP literature has proposed two solutions to enforce the root constraint: (1) Allow invalid dependency trees—hoping that the model can learn to assign them low probabilities and decode singly rooted trees, or (2) return the best of $n$ runs of the CLE each with a fixed edge emanating from the root (Dozat et al., 2017).[4] The first solution is clearly problematic as it may allow parsers to predict malformed dependency trees. This issue is further swept under the rug with "forgiving" evaluation metrics, such as attachment scores, which give

[1] Our Python library is available at `https://github.com/rycolab/spanningtrees`.

[2] Several authors (e.g., Qi et al. (2020); McDonald et al. (2005)) opt for the simpler CLE algorithm (Chu and Liu, 1965; Bock, 1971; Edmonds, 1967), which has a worst-case bound of $\mathcal{O}(n^3)$, but is often fast in practice.

[3] A notable exception is the Prague Dependency Treebank (Bejček et al., 2013), which allows for multi-rooted trees.

[4] In practice, if constraint violations are infrequent, this strategy should be used as a fallback for when the *unconstrained* solution fails. However, this will not necessarily be the case, and is rarely the case during model training.

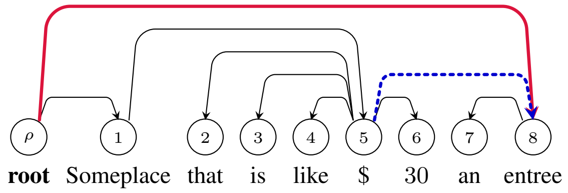

Figure 1: A malformed dependency tree from our experiment. Shown are the incorrect (highlighted) and correct (highlighted) dependency relations for token 8.

partial credit for malformed output.[5] The second solution, while correct, adds an unnecessary factor of $n$ to the runtime of root-constrained decoding.

In this paper, we identify a much more efficient solution than (2). We do so by unearthing an $\mathcal{O}(n^2)$ algorithm due to Gabow and Tarjan (1984) from the theoretical computer science literature. This algorithm appears to have gone unnoticed in NLP literature;[6] we adapt the algorithm to correctly and efficiently handle the root constraint during decoding in edge-factored non-projective dependency parsing.[7]

## 2 Approach

In this section, the marker ✍ indicates that a recently introduced concept is illustrated the worked example in Fig. 2. Let $G = (\rho, V, E)$ be a **rooted weighted directed graph** where $V$ is a set of nodes, $E$ is a set of weighted edges, $E \subseteq \{(i \xrightarrow{w} j) \mid i, j \in V,\ w \in \mathbb{R}\}$,[8] and $\rho \in V$ is a designated root node with no incoming edges. In terms of dependency parsing, each non-$\rho$ node corresponds to a token in the sentence, and $\rho$ represents the special root token that is not a token in the sentence. Edges represent possible dependency relations between tokens. The edge weights are scores from a model (e.g., linear (McDonald et al., 2005), or neural network (Dozat et al., 2017)). Fig. 1 shows an example. We allow $G$ to be a **multi-graph**, i.e., we allow multiple edges between pairs of nodes. Multi-graphs are a natural encoding of *labeled* dependency relations where possible labels between words are captured by multiple edges between nodes in the graph. Multi-graphs pose no difficulty as only the highest-weight edge between two nodes may be selected in the returned tree.

An **arborescence** of $G$ is a subgraph $A = (\rho, V, E')$ where $E' \subseteq E$ such that:

(C1) Each non-root node has exactly one incoming edge (thus, $|E'| = |V|-1$);

(C2) $A$ has no cycles.

A **dependency tree** of $G$ is an arborescence that additionally satisfies

(C3) $|\{(\rho \rightarrow \_) \in E'\}| = 1$

In words, (C3) says $A$ contains exactly one out-edge from $\rho$. Let $\mathcal{A}(G)$ and $\mathcal{A}^\dagger(G)$ denote the sets of arborescences and dependency trees, respectively.

The weight of a graph or subgraph is defined as

$$\overline{w}(G) \overset{\text{def}}{=} \sum_{(i \xrightarrow{w} j) \in G} w \qquad (1)$$

In §2.1, we describe an efficient algorithm for finding the best (highest-weight) arborescence

$$G^* = \operatorname*{argmax}_{A \in \mathcal{A}(G)} \overline{w}(A) \qquad (2)$$

and, in §2.2, the best dependency tree.[9]

$$G^\dagger = \operatorname*{argmax}_{A \in \mathcal{A}^\dagger(G)} \overline{w}(A) \qquad (3)$$

### 2.1 Finding the best arborescence

A first stab at finding $G^*$ would be to select the best (non-self-loop) incoming edge for each node. Although, this satisfies (C1), it does not (necessarily) satisfy (C2). We call this subgraph the **greedy graph**, denoted $\overrightarrow{G}$.✍ Clearly, $\overline{w}(\overrightarrow{G}) \geq \overline{w}(G^*)$ since it is subject to fewer restrictions. Furthermore, if $\overrightarrow{G}$ happens to be acyclic, it is clearly equal to $G^*$. What are we to do in the event of a cycle? That answer has two parts.

*Part 1:* We call any cycle $C$ in $\overrightarrow{G}$ a **critical cycle**.✍ Naturally, (C2) implies that critical cycles can never be part of an arborescence. However, they help us identify optimal arborescences for certain *subproblems*. Specifically, if we were to "break" the cycle at any node $j \in C$ by removing its (unique) incoming edge, we would have an opti-

[5] We note exact match metrics, which consider the entire arborescence, do penalize root constraint violations

[6] There is one exception: Corro et al. (2016) mention Gabow and Tarjan (1984)'s algorithm in a footnote.

[7] Much like this paper, efficient root-constrained marginal inference is also possible without picking up an extra factor of $n$, but it requires some attention to detail (Koo et al., 2007; Zmigrod et al., 2020).

[8] When there is no ambiguity, we may abuse notation using $G$ to refer to either its node or edge set, e.g., we may write $(i \rightarrow j) \in G$ to mean $(i \rightarrow j) \in E$, and $i \in G$ to mean $i \in V$.

[9] Probabilistic models of arborescences (e.g., Koo et al. (2007); Dozat and Manning (2017)) typically seek the maximum a posteriori structure, $\operatorname{argmax}_A \prod_{e \in A} p_e = \operatorname{argmax}_A \sum_{e \in A} \log p_e$. This case can be solved as (1) by taking the weight of $e$ to be $\log p_e$ because $p_e \geq 0$.

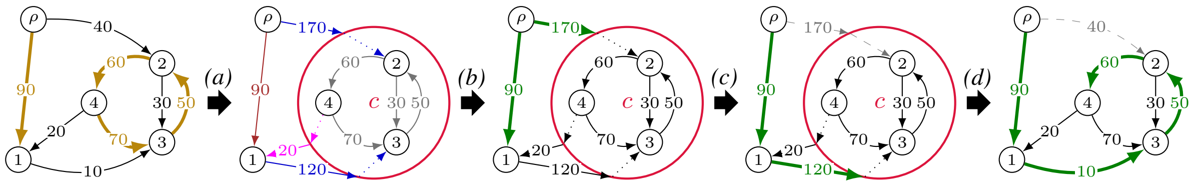

Figure 2: Worked example of finding the best dependency tree. Let $G$ be the graph in the left-most figure, the greedy graph $\overrightarrow{G}$ (highlighted) contains a critical cycle $C$, ②→④→③→②. Step *(a)* shows the contraction $G_{/C}$ where $C$ is replaced by ⓒ, and edges are cast as **enter**, **exit**, **external**, or **dead** edges in $G_{/C}$. We see the bookkeeping function $\pi$ (as ⋯▸), e.g., $\pi(c \xrightarrow{20} 1) = (4 \xrightarrow{20} 1)$ and $\pi(\rho \xrightarrow{170} c) = (\rho \xrightarrow{40} 2)$. Step *(b)* takes the greedy (sub)graph of $G_{/C}$ and since it contains no cycles, it is $(G_{/C})^*$ as (highlighted). Note that if we did not require a dependency tree, we could now use Theorem 1 to break $C$ at ②. Step *(c)* takes $(G_{/C})^*$, which has *two* root edges, $(\rho \xrightarrow{90} 1)$ and $(\rho \xrightarrow{170} c)$, and removes the edge with minimal consequence: removing $(\rho \xrightarrow{90} 1)$ leads to $\overline{w} = 190$, while removing $(\rho \xrightarrow{170} c)$ leads to $\overline{w} = 210$. We pick the latter. As deleting $(\rho \xrightarrow{170} c)$ does not lead to a critical cycle (*optimization case*), we remove it from the graph (shown as - - ▸) and so we get $(G_{/C})^\dagger$ (highlighted). Step *(d)* stitches $(G_{/C})^\dagger \looparrowright C^{(3)}$ yielding $G^\dagger$ (highlighted).

mal arborescence rooted at $j$ for the subgraph over the nodes in $C$. Let $C^{(j)}$ be a subgraph of $C$ rooted at $j$ that denotes the broken cycle at $j$. Let $G_C^{(j)}$ be the subgraph rooted at $j$ where $G_C$ contains all the nodes in $C$ and all edges between them from $G$. Since $C$ is a critical cycle, $C^{(j)}$ is the greedy graph of $G_C^{(j)}$. Moreover, as it is acyclic, we have that $C^{(j)} = (G_C^{(j)})^*$. The key to finding the best arborescence of the entire graph is, thus, determining where to break critical cycles.

*Part 2:* Breaking cycles is done with a recursive algorithm that solves the "outer problem" of fitting the (unbroken) cycle into an optimal arborescence. The algorithm treats the cycle as a single *contracted* node. Formally, a **cycle contraction** takes a graph $G$ and a (not necessarily critical) cycle $C$, and creates a new graph denoted $G_{/C}$ with the same root, nodes $(V \smallsetminus C \cup \{c\})$ where $c \notin V$ is a new node that represents the cycle, and contains the following set of edges: For any $(i \xrightarrow{w} j) \in G$

- **enter**: if $i \notin C, j \in C$, then $(i \xrightarrow{w'} c) \in G_{/C}$ where $w' = w + \overline{w}(C^{(j)})$. Akin to dynamic programming, this choice edge weight (due to Georgiadis (2003)) gives the best "cost-to-go" for breaking the cycle at $j$.
- **exit**: if $i \in C, j \notin C$, then $(c \xrightarrow{w} j) \in G_{/C}$
- **external**: if $i \notin C, j \notin C$, then $(i \xrightarrow{w} j) \in G_{/C}$
- **dead**: if $i \in C, j \in C$, then no edge related to $(i \xrightarrow{w} j)$ is in $G_{/C}$. This is because such an edge $(c \rightarrow c)$ would be a self-cycle, which can never be part of an arborescence.

Additionally, we define a **bookkeeping function**, $\pi$, which maps the nodes and edges of $G_{/C}$ to their counterparts in $G$. We overload $\pi(G)$ to apply point-wise to the constituent nodes and edges.

By (C1), we have that for any $A_C \in \mathcal{A}(G_{/C})$, there exists exactly one incoming edge $(i \rightarrow c)$ to the cycle node $c$. We can use $\pi$ to infer where the cycle was broken with $\pi(i \rightarrow c) = (i \rightarrow j)$. We call $j$ the **entrance site** of $A_C$. Consequently, we can stitch together an arborescence as $\pi(A_C) \cup C^{(j)}$. We use the shorthand $A_C \looparrowright C^{(j)}$ for this operation due to its visual similarity to unraveling a cycle.

$G_{/C}$ may also have a critical cycle, so we have to apply this reasoning recursively. This is captured by Karp (1971)'s Theorem 1.[10]

**Theorem 1.** *For any graph $G$, either $G^* = \overrightarrow{G}$ or $G$ contains a critical cycle $C$ and $G^* = (G_{/C})^* \looparrowright C^{(j)}$ where $j$ is the entrance site of $(G_{/C})^*$. Furthermore, $\overline{w}((G_{/C})^*) = \overline{w}(G^*)$.*

Theorem 1 suggests a recursive strategy for finding $G^*$, which is the basis of many efficient algorithms (Tarjan, 1977; Camerini et al., 1979; Georgiadis, 2003; Chu and Liu, 1965; Bock, 1971; Edmonds, 1967). We detail one such algorithm in Alg 1. Alg 1 can be made to run in $\mathcal{O}(n^2)$ time for dense with the appropriate implementation choices, such as Union-Find (Hopcroft and Ullman, 1973) to maintain membership of nodes to contracted nodes, as well as radix sort (Knuth, 1973) to sort incoming edges to contracted nodes; using a regular sort would add a factor of $\log n$ to the runtime.

[10] We have lightly modified the original theorem. For completeness, App. A provides a proof in our notation.

**Algorithm 1**

```
1:  def opt$(G)$ :                                  ▷ Find $G^* \in \mathcal{A}(G)$ or $G^\dagger \in \mathcal{A}^\dagger(G)$
2:     if $\overrightarrow{G}$ has a cycle $C$ :                       ▷ Recursive case
3:        return opt$(G_{/C}) \looparrowright C^{(j)}$
4:     else                                                  ▷ Base case
5:        if we require a dependency tree (§2.2) :
6:           return constrain$(G)$
7:        else
8:           return $\overrightarrow{G}$
9:  def constrain$(G)$ : ▷ Find $G^\dagger \in \mathcal{A}^\dagger(G)$; $\overrightarrow{G} \in \mathcal{A}(G)$.
10:    $\sigma \leftarrow$ set of $\rho$'s outgoing edges in $\overrightarrow{G}$
11:    if $|\sigma| = 1$ : return $\overrightarrow{G}$              ▷ Root constraint satisfied
12:    $G' \leftarrow \underset{e\in\sigma:G''=G\setminus\!\!\setminus e}{\text{argmax}}\ \overline{w}(\overrightarrow{G''})$          ▷ Find best edge removal
13:    if $\overrightarrow{G'}$ has cycle $C$ :                       ▷ Reduction case
14:       return constrain$(G_{/C}) \looparrowright C^{(j)}$
15:    else                                              ▷ Optimization case
16:       return constrain$(G')$
```

### 2.2 Finding the best dependency tree

Gabow and Tarjan (1984) propose an algorithm that does additional recursion at the base case of opt$(G)$ (the additional if-statement at Line 5) to recover $G^\dagger$ instead of $G^*$.

Suppose that the set of edges emanating from the root in $\overrightarrow{G}$ is given by $\sigma$ and $|\sigma| > 1$. We consider removing each edge in $(\rho \rightarrow j) \in \sigma$ from $G$. Since $G$ may have *multiple* edges from $\rho$ to $j$, we write $G\setminus\!\!\setminus e$ to mean deleting *all* edges with the same edge points as $e$. Let $G'$ be the graph $G\setminus\!\!\setminus e'$ where $e' \in \sigma$ is chosen greedily to maximize $\overline{w}(\overrightarrow{G'})$. Consider the two possible cases:

*Optimization case.* If $G'$ has no critical cycles, then $\overrightarrow{G'}$ must be the best arborescence with one fewer edges emanating from the root than $\overrightarrow{G}$ by our greedy choice of $e'$.✍

*Reduction case.* If $G'$ has a critical cycle $C$, then all edges in $C$ that do not point to $j$ are in $\overrightarrow{G}$. If $e' \notin G^\dagger$, then $C$ is critical cycle in the context of constrained problem and so we can apply Theorem 1 to recover $G^\dagger$. Otherwise, $e \in G^\dagger$ and we can break $C$ at $j$ to get $C^{(j)}$, which is comprised of edges in $\overrightarrow{G}$. Therefore, we can find $(G_{/C})^\dagger$ to retrieve $G^\dagger$. This notion is formalized in the following theorem.[11]

**Theorem 2.** *For any graph G with $G^* = \overrightarrow{G}$, let $\sigma$ be the set of outgoing edges from $\rho$ in $G^*$. If $|\sigma| = 1$, then $G^\dagger = G^*$. Otherwise, let $G' = G\setminus\!\!\setminus e'$ for $e' \in \sigma$ that maximizes $\overline{w}(\overrightarrow{G'})$, then either $G^\dagger = G'^\dagger$ or there exists a critical cycle $C$ in $G'$ such that $G^\dagger = (G_{/C})^\dagger \looparrowright C^{(j)}$ where $j$ is the entrance site of $(G_{/C})^\dagger$.*

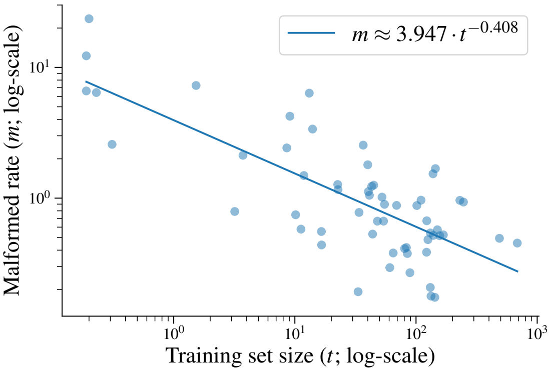

Figure 3: Proportion of malformed trees when decoding pre-trained models (Qi et al., 2020) for languages with varying training set sizes.

Theorem 2 suggests a recursive strategy constrain (Alg 1) for finding $G^\dagger$ given $G^*$. Gabow and Tarjan (1984, Theorem 7.1) prove that such a strategy will execute in $\mathcal{O}(n^2)$ and so when combined with opt$(G)$ (Alg 1) leads to a $\mathcal{O}(n^2)$ runtime for finding $G^\dagger$ given a graph $G$. The efficiency of the algorithm amounts to requiring a bound of $\mathcal{O}(n)$ calls to constrain that will lead to the *reduction case* in order to obtain any number *optimization cases*. Each recursive call does a linear amount of work to search for the edge to remove and to stitch together the results of recursion. Rather than computing the greedy graph from scratch, implementations should exploit that each edge removal will only change one element of the greedy graph. Thus, we can find $\overline{w}(\overrightarrow{G\setminus\!\!\setminus e'})$ in constant time.

## 3 Experiment

How often do state-of-the-art parsers generate malformed dependency trees? We examined 63 Universal Dependency Treebanks (Nivre et al., 2018) and computed the rate of malformed trees when decoding using edge weights generated by pre-trained models supplied by Qi et al. (2020). On average, we observed that 1.80% of trees are malformed. We were surprised to see that—although the edge-factored model used is not expressive enough to capture the root constraint *exactly*—there are useful correlates of the root constraint in the surface form of the sentence, which the model appears to use to workaround this limitation. This becomes further evident when we examine the relative change[12] in UAS (0.0083%) and exact match scores (0.60%)

[11] For completeness, App. B provides a proof of Theorem 2.

[12] The relative difference is computed with respect to the unconstrained algorithm's scores.

| Setting | # Languages | Malformed rate | Rel. Δ UAS | Rel. Δ Exact Match |
|---|---|---|---|---|
| High | 20 | $0.63\%$ | $0.0041\%$ | $0.15\%$ |
| Medium | 32 | $1.02\%$ | $0.0012\%$ | $0.22\%$ |
| Low | 11 | $6.21\%$ | $0.0368\%$ | $2.91\%$ |

Table 1: Average malformed rate, relative UAS change, and relative exact match score change for different data settings. The 63 languages are split by their training set size $|\mathrm{train}|$ into high ($|\mathrm{train}| \geq 10,000$), medium ($1,000 \leq |\mathrm{train}| < 10,000$), and low ($|\mathrm{train}| < 1,000$).

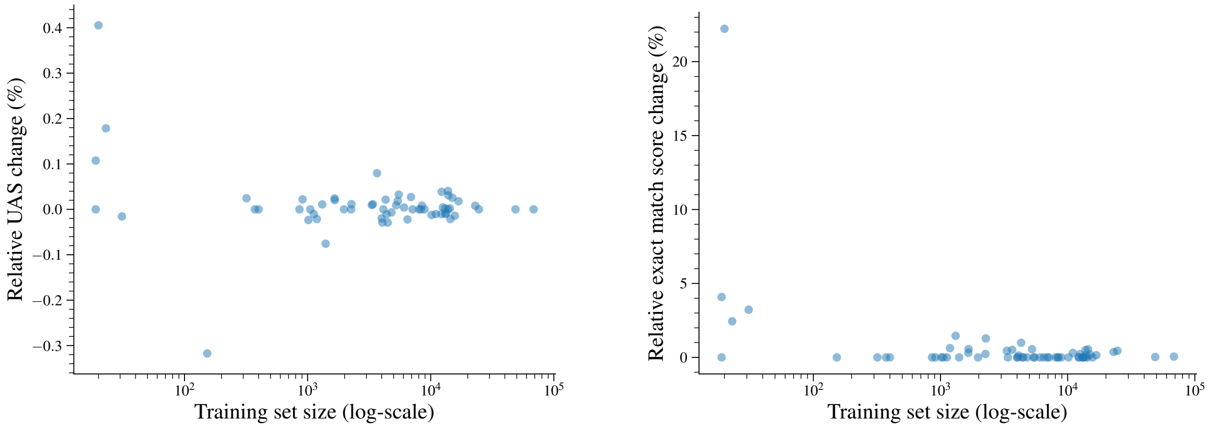

Figure 4: Relative change in UAS and exact match score when using the unconstrained and constrained algorithms for languages with varying training set sizes.

when using the constrained algorithm as opposed to the unconstrained algorithm.

Nevertheless, given less data, it is harder to learn to exploit the surface correlates; thus, we see an increasing average rate of violation, $6.21\%$, when examining languages with training set sizes of less than $1,000$ sentences. Similarly, the relative change in UAS and exact match score increases to $0.0368\%$ and $2.91\%$ respectively. Indeed, the worst violation rate was $24\%$ was seen for Kurmanji which only contains 20 sentences in the training set. Kurmanji consequently had the largest relative changes to both UAS and exact match scores of $0.41\%$ and $22.22\%$. We break down the malformed rate and accuracy changes by training size in Tab. 1. Furthermore, the correlation between training size and malformed tree rate can be seen in Fig. 3 while the correlation between training size and relative accuracy change can be seen in Fig. 4. We provide a full table of the results in App. C.

## 4 Conclusion

In this paper, we have bridged the gap between the graph-theory and dependency parsing literature. We presented an efficient $\mathcal{O}(n^2)$ for finding the maximum arborescence of a graph. Furthermore, we highlighted an important distinction between dependency trees and arborescences, namely that dependency trees are arborescences subject to a *root constraint*. Previous work uses inefficient algorithms to enforce this constraint. We provide a solution which runs in $\mathcal{O}(n^2)$. Our hope is that this paper will remind future research in dependency parsing to please mind the root.

## Acknowledgments

We would like to thank all reviewers for their valuable feedback and suggestions. The first author is supported by the University of Cambridge School of Technology Vice-Chancellor's Scholarship as well as by the University of Cambridge Department of Computer Science and Technology's EPSRC.

## References

Eduard Bejček, Eva Hajičová, Jan Hajič, Pavlína Jínová, Václava Kettnerová, Veronika Kolářová, Marie Mikulová, Jiří Mírovský, Anna Nedoluzhko, Jarmila Panevová, Lucie Poláková, Magda Ševčíková, Jan Štěpánek, and Šárka Zikánová. 2013. Prague dependency treebank 3.0.

F. C. Bock. 1971. An algorithm to construct a minimum directed spanning tree in a directed network. *Developments in Operations Research*.

Paolo M. Camerini, Luigi Fratta, and Francesco Maffioli. 1979. A note on finding optimum branchings. *Networks*, 9(4).

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14.

Caio Corro, Joseph Le Roux, Mathieu Lacroix, Antoine Rozenknop, and Roberto Wolfler Calvo. 2016. Dependency parsing with bounded block degree and well-nestedness via Lagrangian relaxation and branch-and-bound. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 355–366, Berlin, Germany. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of the International Conference on Learning Representations*.

Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, Vancouver, Canada. Association for Computational Linguistics.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards, Section B: Mathematics and Mathematical Physics*, 71(4).

Harold N. Gabow and Robert Endre Tarjan. 1984. Efficient algorithms for a family of matroid intersection problems. *Journal of Algorithms*, 5(1).

Leonidas Georgiadis. 2003. Arborescence optimization problems solvable by Edmonds' algorithm. *Theoretical Computer Science*, 301(1-3).

John E. Hopcroft and Jeffrey D. Ullman. 1973. Set merging algorithms. *SIAM J. Comput.*, 2(4).

Richard M. Karp. 1971. A simple derivation of Edmonds' algorithm for optimum branchings. *Networks*, 1(3).

Donald E. Knuth. 1973. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley.

Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. 2007. Structured prediction models via the matrix-tree theorem. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Xuezhe Ma and Eduard Hovy. 2017. Neural probabilistic model for non-projective MST parsing. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Rogier Blokland, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Kamil Kopacewicz, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phuong Lê H`ông, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan

Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Shinsuke Mori, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Luong Nguy˜ên Thị, Huy`ên Nguy˜ên Thị Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adédayọ Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino-Passos, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roșca, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Isabela Soares-Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Yuta Takahashi, Takaaki Tanaka, Isabelle Tellier, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jing Xian Wang, Jonathan North Washington, Seyi Williams, Mats Wirén, Tsegay Woldemariam, Tak-sum Wong, Chunxiao Yan, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. 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.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the Association for Computational Linguistics: System Demonstrations*.

Robert Endre Tarjan. 1977. Finding optimum branchings. *Networks*, 7(1).

UD Contributors. Root relation in universal dependencies. `https://universaldependencies.org/u/dep/root.html`. Accessed: 2020-05-30.

Ran Zmigrod, Tim Vieira, and Ryan Cotterell. 2020. Efficient computation of expectations under spanning tree distributions. *Transactions of the Association for Computational Linguistics*.

## A Proof of Theorem 1

To prove Theorem 1, we note a correspondence between graphs and contracted graphs.

**Proposition 1.** *Given a rooted graph $G$ and a (not necessarily critical) cycle $C$ in $G$. For any $A \in \mathcal{A}(G)$ that has a single edge $e = (i \xrightarrow{w} j) \in A$ such that $i \notin C$ and $j \in C$, there exists $A_C \in \mathcal{A}(G_{/C})$ and $A' \in \mathcal{A}(G_C^{(j)})$ such that $A = A_C \looparrowright A'$. Furthermore,*

$$\overline{w}(A) = \overline{w}(A_C) - \overline{w}(C^{(j)}) + \overline{w}(A') \tag{4}$$

*Proof.* Since $e$ is the only edge in $A$ from a non-cycle node to a cycle node (**enter**), every edge $e' \in G_{/C}$ such that $\pi(e') \in A$ forms an arborescence $A_C \in \mathcal{A}(G_{/C})$. Note that the set of edges in $A$ for which there is no corresponding edge in $G_{/C}$ are **dead** edges. In fact, as $A$ satisfies (C1), these edges form an arborescence $A' \in \mathcal{A}(G_C^{(j)})$. Therefore, $A = A_C \looparrowright A'$.
Furthermore, consider the weight of $A$:

$$\begin{aligned}
\overline{w}(A) &= \sum_{(i' \xrightarrow{w'} j') \in \pi(A_C)} w' + \overline{w}(A') && (5)\\
&= \sum_{(i' \xrightarrow{w'} j') \in \pi(A_C \smallsetminus \{e\})} w' + w + \overline{w}(A') && (6)\\
&= \sum_{(i' \xrightarrow{w'} j') \in A_C \smallsetminus \{e\}} w' + w + \overline{w}(A') && (7)\\
&= \sum_{(i' \xrightarrow{w'} j') \in A_C} w' - \overline{w}(C^{(j)}) + \overline{w}(A') && (8)\\
&= \overline{w}(A_C) - \overline{w}(C^{(j)}) + \overline{w}(A') && (9)
\end{aligned}$$

Note that (7) follows because $e$ is the only edge in $A$ from a non-cycle node to a cycle node, and (8) follows by the construction of **enter** edges in $G_{/C}$. □

As a corollary, we also have that every arborescence in the contracted graph $G_{/C}$ can be expanded into an arborescence in $G$.

**Corollary 1** (Expansion lemma)**.** *Given a rooted graph $G$ with a cycle $C$, every arborescence $A_C \in \mathcal{A}(G_{/C})$ is related to an arborescence $A \in \mathcal{A}(G)$ by $A = A_C \looparrowright C^{(j)}$ where $j$ is the entrance site of $A_C$. Furthermore $\overline{w}(A) = \overline{w}(A_C)$.*

*Proof.* Let $j$ be the entrance site of $A_C$ into $C$. As $A_C \in \mathcal{A}(G_{/C})$ and $C^{(j)} \in \mathcal{A}(G_C^{(j)})$, Proposition 1 constructs $A \in \mathcal{A}_\rho(G)$ as desired. Furthermore, $\overline{w}(A) = \overline{w}(A_C) - \overline{w}(C^{(j)}) + \overline{w}(C^{(j)}) = \overline{w}(A_C)$. □

Note that Proposition 1 does not account for all arborescences in $\mathcal{A}(G)$. We next show that such arborescences which cannot be constructed using Proposition 1 will never be $G^*$.

**Lemma 1.** *Given a rooted graph $G$ with a critical cycle $C$. We have that for all $j \in C$*

$$G_C^{(j)*} = C^{(j)} \tag{10}$$

*Proof.* Since $G_C^{(j)}$ is a subgraph of $G$ it must be that $\overrightarrow{G_C^{(j)}}$ is also a subgraph of $\overrightarrow{G}$. Since $C$ is a critical cycle, $C^{(j)}$ does not have cycles and equals $\overrightarrow{G_C^{(j)}}$. Therefore $C^{(j)} = G_C^{(j)*}$. □

**Lemma 2.** *Given a rooted graph $G$ with a critical cycle $C$ and $A \in \mathcal{A}(G)$. If $e = (i \rightarrow j) \in A$ and $e' = (i' \rightarrow j')$ such that $i, i' \notin C$ and $j, j' \in C$, then there exists a $A' \in \mathcal{A}(G)$ with $e \in A'$ and $e' \notin A'$ such that $\overline{w}(A) \leq \overline{w}(A')$.*

*Proof.* Construct $A'$ such that for every edge $e'' = (i'' \rightarrow j'') \in G_{/C}$, if $j'' \neq c$ and $\pi(e'') \in A$, then $\pi(e'') \in A'$. Additionally, let $e$ be in $A'$ as well as the edges in $C^{(j)}$. Then $A'$ has no cycles and each non-root node contains a single incoming edge, so $A' \in \mathcal{A}(G)$. Since $A$ and $A'$ contain identical edges except for those pointing to nodes in $C \smallsetminus \{j\}$, by Lemma 1, $\overline{w}(A) \leq \overline{w}(A')$. □

**Theorem 1.** For any graph $G$, either $G^* = \overrightarrow{G}$ or $G$ contains a critical cycle $C$ and $G^* = (G_{/C})^* \looparrowright C^{(j)}$ where $j$ is the entrance site of $(G_{/C})^*$. Furthermore, $\overline{w}(G_{/C}{}^*) = \overline{w}(G^*)$.

*Proof.* There are two cases to consider.

*Case 1*: $G$ does not contain a critical cycle. Trivially, $G^* = \overrightarrow{G}$.

*Case 2*: $G$ contains a critical cycle $C$. By Corollary 1, we can construct an arborescence $A = (G_{/C})^* \looparrowright C^{(j)} \in \mathcal{A}(G)$, we now prove that no other $A' \in \mathcal{A}(G)$ can have a higher weight. Firstly, by Lemma 2, we only need to consider $A'$ that satisfy Proposition 1. Therefore, $A'$ must be decomposable into an arborescence $A_C \in \mathcal{A}(G_{/C})$ and an arborescence in $\mathcal{A}(G_C^{(j')})$ where $j'$ is the entrance site of $A_C$. Then since $(G_{/C})^*$ is optimal, we have that $A_C = (G_{/C})^*$ and $j' = j$. As $C^{(j)}$ is optimal (by Lemma 1), $A$ must also be optimal and so $G^* = (G_{/C})^* \looparrowright C^{(j)}$. □

## B Proof of Theorem 2

We prove Theorem 2 by showing that both the *optimization* and *reduction* cases described in the main text lead to progress towards finding $G^\dagger$.

**Lemma 3.** *For any graph $G$ with $G^* = \overrightarrow{G}$, let $\sigma$ be the set of outgoing edges from $\rho$ in $\overrightarrow{G}$. If $|\sigma| > 1$, let $G' = G \backslash\!\backslash e'$ for $e' \in \sigma$ that maximizes $\overline{w}(\overrightarrow{G'})$. If there exists a critical cycle $C$ in $G'$, then $G^\dagger = (G_{/C})^\dagger \looparrowright C^{(j)}$ where $j$ is the entrance site of $(G_{/C})^\dagger$.*

*Proof.* Let $e' = (\rho \rightarrow i)$ and $e \in G_{/C}$ such that $\pi(e) = e'$. We know that $e$ always exists as $e'$ emanates from the root. By Corollary 1, we know that $A = (G_{/C})^\dagger \looparrowright C^{(j)} \in \mathcal{A}(G)$ where $j$ is the entrance site of $(G_{/C})^\dagger$. Furthermore, As $C$ has no edges emanating from the root, $A \in \mathcal{A}^\dagger(G)$. There are two cases to consider:

*Case 1* ($e \in (G_{/C})^\dagger$): As $C^{(j)}$ is a subgraph of $\overrightarrow{G}$, $A$ must have the highest weight in $\mathcal{A}^\dagger(G)$, so $G^\dagger = A$.

*Case 2* ($e \notin (G_{/C})^\dagger$): Then $e'$ cannot be in $G^\dagger$, and the edge pointing to $i$ in $C$ is the next best possible edge incoming to $j$. Therefore, whichever way we break $C$ in $A$, we will get a set of edges with maximal weight and so $G^\dagger = A$. □

**Lemma 4.** *For any graph $G$ with $G^* = \overrightarrow{G}$, let $\sigma$ be the set of outgoing edges from $\rho$ in $\overrightarrow{G}$. If $|\sigma| > 1$, let $G' = G \backslash\!\backslash e'$ for $e' \in \sigma$ that maximizes $\overline{w}(\overrightarrow{G'})$. Either $G^\dagger = G'^\dagger$ or there exists a critical cycle $C$ in $G'$ such that $G^\dagger = (G_{/C})^\dagger \looparrowright C^{(j)}$ where $j$ is the entrance site of $(G_{/C})^\dagger$.*

*Proof.* Let $j$ be the entrance site of $(G_{/C})^\dagger$. Proof by induction on $r = |\sigma|$.

*Base case* ($r = 2$): If $G'$ does not contain a critical cycle, then clearly $G'^\dagger = G'^*$. Since we choose $e'$ to maximize $\overrightarrow{G'}$ and $G'$ is a subgraph of $G$, $G^\dagger = G'^\dagger$. Otherwise, $G'$ has a critical cycle $C$. Then by Lemma 3, $G^\dagger = (G_{/C})^\dagger \looparrowright C^{(j)}$ .

*Inductive case* ($r > 2$): Let $\sigma'$ be the set of outgoing edge from $\rho$ in $\overrightarrow{G'}$. Then clearly $|\sigma'| = r - 1 > 1$. If $G'$ does not contain a critical cycle, then $G'^* = \overrightarrow{G'}$ and we satisfy the induction hypothesis. Otherwise, $G'$ has a critical cycle $C$. Then by Lemma 3, $G^\dagger = (G_{/C})^\dagger \looparrowright C^{(j)}$. □

**Theorem 2.** For any graph $G$ with $G^* = \overrightarrow{G}$, let $\sigma$ be the set of outgoing edges from $\rho$ in $G^*$. If $|\sigma| = 1$, then $G^\dagger = G^*$, otherwise if $G' = G \backslash\!\backslash e'$ for $e' \in \sigma$ that maximizes $\overline{w}(\overrightarrow{G'})$, then either $G^\dagger = G'^\dagger$ or there exists a critical cycle $C$ in $G'$ such that $G^\dagger = (G_{/C})^\dagger \looparrowright C^{(j)}$ where $j$ is the entrance site of $(G_{/C})^\dagger$.

*Proof.* There are two cases to consider.

*Case 1* ($|\sigma| = 1$): Then $G^*$ has one edge emanating from the root so clearly $G^\dagger = G^*$.

*Case 2* ($|\sigma| > 1$). This is immediate from Lemma 4. □

## C Decoding UD Treebanks

| **Language** | **\|Train\|** | **\|Test\|** | **Malformed Rate** | **Rel. Δ UAS** | **Rel. Δ Exact Match** |
|---|---|---|---|---|---|
| Czech | 68495 | 10148 | 0.45% | 0.000% | 0.052% |
| Russian | 48814 | 6491 | 0.49% | 0.000% | 0.027% |
| Estonian | 24633 | 3214 | 0.93% | 0.000% | 0.448% |
| Korean | 23010 | 2287 | 0.96% | 0.008% | 0.366% |
| Latin | 16809 | 2101 | 0.52% | 0.018% | 0.151% |
| Norwegian | 15696 | 1939 | 0.52% | -0.014% | 0.000% |
| Ancient Greek | 15014 | 1047 | 0.57% | 0.026% | 0.186% |
| French | 14450 | 416 | 1.68% | -0.021% | 0.546% |
| Spanish | 14305 | 1721 | 0.17% | 0.002% | 0.000% |
| Old French | 13909 | 1927 | 0.52% | 0.031% | 0.145% |
| German | 13814 | 977 | 1.54% | 0.040% | 0.495% |
| Polish | 13774 | 1727 | 0.00% | 0.000% | 0.000% |
| Hindi | 13304 | 1684 | 0.18% | -0.009% | 0.000% |
| Catalan | 13123 | 1846 | 0.54% | 0.002% | 0.000% |
| Italian | 13121 | 482 | 0.21% | -0.010% | 0.000% |
| English | 12543 | 2077 | 0.48% | 0.004% | 0.217% |
| Dutch | 12264 | 596 | 0.67% | 0.039% | 0.000% |
| Finnish | 12217 | 1555 | 0.39% | -0.010% | 0.000% |
| Classical Chinese | 11004 | 2073 | 0.96% | -0.010% | 0.304% |
| Latvian | 10156 | 1823 | 0.88% | -0.012% | 0.000% |
| Bulgarian | 8907 | 1116 | 0.27% | 0.000% | 0.000% |
| Slovak | 8483 | 1061 | 0.38% | 0.008% | 0.000% |
| Portuguese | 8328 | 477 | 0.42% | 0.000% | 0.000% |
| Romanian | 8043 | 729 | 0.41% | 0.000% | 0.000% |
| Japanese | 7125 | 550 | 0.00% | 0.000% | 0.000% |
| Croatian | 6914 | 1136 | 0.88% | 0.027% | 0.000% |
| Slovenian | 6478 | 788 | 0.38% | -0.022% | 0.000% |
| Arabic | 6075 | 680 | 0.29% | 0.004% | 0.000% |
| Ukrainian | 5496 | 892 | 0.90% | 0.032% | 0.000% |
| Basque | 5396 | 1799 | 0.67% | 0.018% | 0.000% |
| Hebrew | 5241 | 491 | 1.02% | 0.009% | 0.556% |
| Persian | 4798 | 600 | 0.67% | -0.007% | 0.000% |
| Indonesian | 4477 | 557 | 1.26% | -0.029% | 0.000% |
| Danish | 4383 | 565 | 0.53% | -0.011% | 0.000% |
| Swedish | 4303 | 1219 | 1.23% | 0.021% | 0.988% |
| Old Church Slavonic | 4124 | 1141 | 1.05% | 0.000% | 0.128% |
| Urdu | 4043 | 535 | 1.12% | -0.029% | 0.000% |
| Chinese | 3997 | 500 | 1.80% | -0.020% | 0.000% |
| Turkish | 3664 | 983 | 2.54% | 0.080% | 0.513% |
| Gothic | 3387 | 1029 | 0.78% | 0.011% | 0.000% |
| Serbian | 3328 | 520 | 0.19% | 0.009% | 0.446% |
| Galician | 2272 | 861 | 1.16% | 0.011% | 1.282% |
| North Sami | 2257 | 865 | 1.27% | 0.000% | 0.230% |
| Armenian | 1975 | 278 | 0.00% | 0.000% | 0.000% |
| Greek | 1662 | 456 | 0.44% | 0.020% | 0.565% |
| Uyghur | 1656 | 900 | 0.56% | 0.024% | 0.309% |
| Vietnamese | 1400 | 800 | 3.38% | -0.076% | 0.000% |
| Afrikaans | 1315 | 425 | 6.35% | 0.011% | 1.460% |
| Wolof | 1188 | 470 | 1.49% | -0.021% | 0.625% |
| Maltese | 1123 | 518 | 0.58% | -0.010% | 0.000% |
| Telugu | 1051 | 146 | 0.00% | 0.000% | 0.000% |
| Scottish Gaelic | 1015 | 536 | 0.75% | -0.024% | 0.000% |
| Hungarian | 910 | 449 | 4.23% | 0.022% | 0.000% |
| Irish | 858 | 454 | 2.42% | 0.000% | 0.000% |
| Tamil | 400 | 120 | 0.00% | 0.000% | 0.000% |
| Marathi | 373 | 47 | 2.13% | 0.000% | 0.000% |
| Belarusian | 319 | 253 | 0.79% | 0.024% | 0.000% |
| Lithuanian | 153 | 55 | 7.27% | -0.317% | 0.000% |
| Kazakh | 31 | 1047 | 2.58% | -0.016% | 3.226% |
| Upper Sorbian | 23 | 623 | 6.42% | 0.178% | 2.439% |
| Kurmanji | 20 | 734 | 23.57% | 0.405% | 22.222% |
| Buryat | 19 | 908 | 6.61% | 0.107% | 4.082% |
| Livvi | 19 | 106 | 12.26% | 0.000% | 0.000% |

Table 2: Accompanying table for §3