The Linear Arrangement Library. A new tool for research on syntactic dependency structures.

Lluís Alemany-Puig lluis.alemany.puig@upc.edu
Juan Luis Esteban esteban@cs.upc.edu
Ramon Ferrer-i-Cancho ramon.ferrer@upc.edu
Universitat Politècnica de Catalunya
Jordi Girona 1-3
08034 Barcelona, Catalonia, Spain

Abstract

The new and growing field of Quantitative Dependency Syntax has emerged at the crossroads between Dependency Syntax and Quantitative Linguistics. One of the main concerns in this field is the statistical patterns of syntactic dependency structures. These structures, grouped in treebanks, are the source for statistical analyses in these and related areas; dozens of scores devised over the years are the tools of a new industry to search for patterns and perform other sorts of analyses. The plethora of such metrics and their increasing complexity require sharing the source code of the programs used to perform such analyses. However, such code is not often shared with the scientific community or is tested following unknown standards. Here we present a new open-source tool, the Linear Arrangement Library (LAL), which caters to the needs of, especially, inexperienced programmers. This tool enables the calculation of these metrics on single syntactic dependency structures, treebanks, and collection of treebanks, grounded on ease of use and yet with great flexibility. LAL has been designed to be efficient, easy to use (while satisfying the needs of all levels of programming expertise), reliable (thanks to thorough testing), and to unite research from different traditions, geographic areas, and research fields.

1 Introduction

Quantitative Linguistics is a discipline within Linguistics that aims to unveil linguistic laws and explain their origins (Köhler and Altmann, 2012; Best and Rottmann, 2017). Outstanding examples of these are Zipfian laws, e.g., Zipf's rank-frequency law, Zipf's law of abbreviation (Zipf, 1949), that are defined typically on one of languages' basic units: words. Another discipline in Linguistics is Dependency Syntax, a framework which primarily reduces the syntactic structure of a sentence to word pairwise dependencies. Each of these dependencies has a 'head' word and a 'dependent' word (in fields like Computer Science, these could be called 'parent' and 'child', respectively; the 'head' is also known as 'governor'). The collection of such dependencies in a sentence combined with the linear ordering of the words yields the so-called *syntactic dependency structure* (Mel'čuk, 1988; Kuhlmann and Nivre, 2006; Nivre, 2006; Gómez-Rodríguez et al., 2011) as in Figure 1. Therefore, the underlying structure of a syntactic dependency structure can be seen as a rooted tree (as in Figure 2(b)).

The combination of Quantitative Linguistics with Dependency Syntax has resulted into the emerging field of Quantitative Dependency Syntax `https://quasy-2019.webnode.com/`. The target of this field are syntactic dependency structures and aims to discover and understand statistical patterns in these structures. By linearizing the hierarchical structure "arises the concept of dependency distance or dependency length" (Liu et al., 2017), defined usually as the number of intervening words between the endpoints of the dependency plus one (Ferrer-i-Cancho, 2004) as in Figure 2(a). Another relevant concept is that of *syntactic dependency crossing* (Mel'čuk, 1988). Figure 1 shows two examples of syntactic crossings: two syntactic dependencies cross when the positions of their head and dependent words interleave. Said concept is used to define many formal constraints, such as projective and planar structures (Kuhlmann and Nivre, 2006) and 1-Endpoint-Crossing structures (Satta et al., 2013). See Gómez-Rodríguez et al. (2011) for a review.

Research in Cognitive Science has shown a tendency for languages to reduce dependency distances (Ferrer-i-Cancho, 2004; Liu, 2008; Futrell et al., 2015; Futrell et al., 2020; Ferrer-i-Cancho et al., 2021).

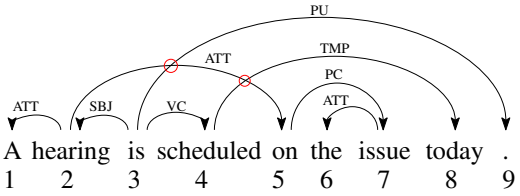

Figure 1: An example of a sentence and the syntactic dependencies among its words (adapted from Pighin (2012)). Relations are labeled with their grammatical category. Numbers below the sentence indicate the positions of the words. In this figure we see two syntactic crossings, marked with small red circles.

According to Liu et al. (2017), Hudson (1995) gave the first definition of dependency distance and presented a cognitive formulation of "the memory burden imposed by dependency distance on language processing". This tendency results from the action of a Dependency Distance Minimization (DDm) principle (Ferrer-i-Cancho, 2003; Ferrer-i-Cancho, 2004), supported by many models and theories (Liu et al., 2017; Temperley and Gildea, 2018) stemming from the more general Principle of Least Effort (Zipf, 1949), hence largely regarded as a linguistic universal. The statistical support for DDm comes from baselines that are used to perform statistical tests on the significance of dependency distances (Ferrer-i-Cancho, 2004; Gildea and Temperley, 2007; Liu, 2008; Park and Levy, 2009; Gildea and Temperley, 2010; Futrell et al., 2015; Yu et al., 2019; Ferrer-i-Cancho et al., 2021). Some of these baselines are defined on extreme conditions (e.g., maximum and minimum sum of dependency distances) which further motivates the study of extremal problems in Computer Science like the Minimum Linear Arrangement problem (Garey and Johnson, 1976; Goldberg and Klipker, 1976; Shiloach, 1979; Chung, 1984) and the Maximum Linear Arrangement Problem (Hassin and Rubinstein, 2000; DeVos and Nurse, 2018) and their variants under formal constraints. Other baselines are defined on 'uniformly random' conditions, typically in uniformly random permutations of the words of a sentence. However, formal constraints from Dependency Grammar, like projectivity and planarity (Sleator and Temperley, 1993; Kuhlmann and Nivre, 2006), have led to defining such random baselines conditioned to those formal constraints (Gildea and Temperley, 2007; Park and Levy, 2009; Futrell et al., 2015; Kramer, 2021; Alemany-Puig and Ferrer-i-Cancho, 2021) although these formal constraints have been argued to be epiphenomena of DDm (Gómez-Rodríguez and Ferrer-i-Cancho, 2017; Gómez-Rodríguez et al., 2020).

In this article we introduce a new tool to support research on the areas and the research problems reviewed above: the Linear Arrangement Library (LAL), which allows researchers to compute easily many of the metrics and algorithms that researchers have been proposing, while simplifying significantly the problem of calculating random or extremal baselines for them. In addition, LAL aims to simplify the process of working with collections of treebanks, one of the most successful recent examples being the Universal Dependencies collection (Zeman et al., 2020) and its variants (Gerdes et al., 2018). LAL is currently available from `https://cqllab.upc.edu/lal`.

In order to grasp the power of LAL we remind the reader that the syntactic dependency structure of a sentence can be defined as a triad composed of (1) a *directed graph structure* in which the vertices of the graph are the words of the sentence, (2) a *linear arrangement* of the vertices of the graph, and (3) *labels* of the edges of the graph which indicate the type of syntactic relationship between the words they relate. Two types of metrics (or scores) can be defined on such structures: word order-dependent, e.g., the sum of dependency distances (Gildea and Temperley, 2007) and word order-independent metrics, e.g., mean hierarchical distance (Jing and Liu, 2015). LAL allows one to compute many scores of each of these two sorts.

The calculation of baselines is at the heart of Quantitative Dependency Syntax research as well as at the heart of LAL. For some random baselines, LAL offers exact algorithms or formulae to calculate the desired value under the null model, e.g., algorithms to calculate the expected sum of dependency distances (Ferrer-i-Cancho, 2004; Alemany-Puig and Ferrer-i-Cancho, 2021). The reality is, unfortunately, that algorithms and formulae to calculate exact expected values of certain scores might be difficult to derive.

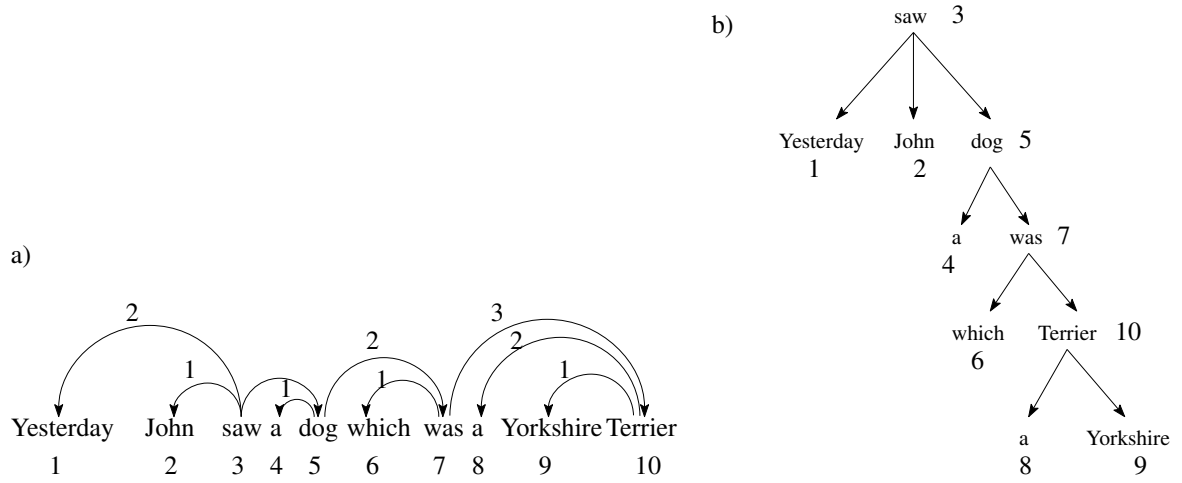

Figure 2: a) An example of syntactic dependency structure. Arc labels indicate edge lengths, each calculated as the absolute difference of the positions of the corresponding edge's endpoints. The numbers below the words indicate positions. b) The rooted tree underlying the sentence in a); the positions of the words are indicated below or to the right of each word. Adapted from (McDonald et al., 2005, Figure 2).

As an alternative, LAL allows researchers to resort to random sampling in order to calculate said values, which often involves tree and/or linear arrangement generation (Liu, 2008; Esteban and Ferrer-i-Cancho, 2017; Yadav et al., 2019). An example of an application of tree generation would be the calculation of the expected mean hierarchical distance (Jing and Liu, 2015) among n-vertex rooted trees[1]. Regarding linear arrangement generation, an example would be to calculate the expected flux weight (Kahane et al., 2017) over all arrangements of a tree.

Thanks to LAL the computation of random baselines can be restricted easily to two of the most frequently observed arrangements from a formal standpoint: planar orderings, where syntactic edges do not cross, and projective orders, namely planar orderings where the root is not covered (Sleator and Temperley, 1993; Kuhlmann and Nivre, 2006). For instance, the random baseline over the sum of dependency distances can be computed assuming unconstrained, projective and planar arrangements as shown in Table 1. In an unconstrained linear arrangement, edge crossings are allowed and the root may be covered.

The remainder of the article is organized as follows. Section 2 presents the design principles of LAL and its architecture. Section 3 gives further details about its functionalities and explains how to work with LAL following a standard research pipeline. We end with some suggestions for future development of LAL.

2 Design principles

2.1 Ease of use

Many measures/scores in Quantitative Dependency Syntax have been devised over the years (e.g., Jiang and Liu (2018)). Some of these are easy to calculate, e.g., the sum of dependency distances (or the sum of edge lengths) of an n-vertex syntactic dependency structure. This sum is easily computable in $O(n)$ time given the specification of the linear arrangement and that of the graph. However, the calculation of extremal values on linear arrangements (i.e. minimum or maximum values of a score), such as the solution to the Minimum Linear Arrangement (MLA) problem in unconstrained arrangements (Garey and Johnson, 1976; Shiloach, 1979; Chung, 1984) or of one of its constrained variants (Iordanskii, 1987; Hochberg and Stallmann, 2003; Gildea and Temperley, 2007; Bommasani, 2020; Alemany-Puig et al., 2022) are not straightforward because the algorithm is complex, it is hard to test or both. Likewise,

[1]This problem may be notoriously difficult to solve; take as a reference the work by Rényi and Szekeres (1967) where it is shown that the average labeled tree height H_n is such that $H_n \to \sqrt{2n\pi}$ as $n \to \infty$.

	D			C
	Unconstrained	Planar	Projective	Unconstrained
Minimum	Shiloach (1979), Chung (1984)	Hochberg and Stallmann (2003), Alemany-Puig et al. (2022)	Gildea and Temperley (2007), Alemany-Puig et al. (2022)	†
Complexity	$O(n^{2.2}), O(n^2)$	$O(n)$	$O(n)$	
Expected	Ferrer-i-Cancho (2004)	*	Alemany-Puig and Ferrer-i-Cancho (2021)	Verbitsky (2008)
Complexity	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Maximum Complexity	Under study	In progress $O(n)$	In progress $O(n)$	Under study

Table 1: The baselines on D, the sum of dependency distances, and C, the number of syntactic dependency crossings, that can be calculated on a given input tree using LAL. Columns 'Unconstrained', 'Planar' and 'Projective' are the different constraints under which the 'Minimum', 'Expected' and 'Maximum' values can be calculated with LAL. *: available in LAL but article not published yet; †: the minimum value of C is trivially 0 for every tree.

performing statistical tests and calculating expected values (random baselines) require random sampling methods when exact algorithms/formulae are not known; such sampling is typically done uniformly at random over all possible trees (Ferrer-i-Cancho et al., 2018; Gómez-Rodríguez et al., 2020; Yadav et al., 2019) or random arrangements (Ferrer-i-Cancho et al., 2018; Ferrer-i-Cancho et al., 2021). LAL's main design principle is to make these algorithms (and random sampling methods) easily accessible and, therefore, LAL has been designed to be an easy-to-use tool for Quantitative Dependency Syntax researchers focused in the analysis of syntactic dependency trees, and Computer Scientists and Mathematicians specializing in Discrete Mathematics. Moreover, advanced programming knowledge is not required to use LAL. For example, a researcher in Quantitative Dependency Syntax can process a treebank as easily as shown in Code 1.

```
import lal
err = lal.io.process_treebank("Cantonese.txt", "output_file.csv")
print(err)
```

Code 1: Python script for computing all scores on a single treebank with LAL. The input file is a series of head vectors, whose format is described in Section 3, and the output is a standard .csv file that can be loaded directly on a spreadsheet.

2.2 Connecting communities and traditions

LAL aims to unite research traditions and disciplines from all over the world as well as serving the distinct fields converging into Quantitative Dependency Syntax as much as possible. LAL integrates views, concepts and scores from the major research communities: Asia (China, Jing and Liu (2015); Japan, Komori et al. (2019)), North America (USA, Gildea and Temperley (2007) and Futrell et al. (2015)) and Europe (e.g., France, Kahane et al. (2017); Switzerland, Gulordava and Merlo (2016) and Gulordava and Merlo (2015)).

2.3 Openness and availability

After many years of research in Quantitative Dependency Syntax, the code used to calculate many of these metrics is not usually shared, thus forcing researchers in the Linguistics fields to repeat the same efforts as the original authors put into coding the algorithms. This repetition increases the probability of bugs in every researcher's code which lead to incorrect results used in their experiments to be published in scientific journals. We think that LAL is an answer to these challenges as it is an open-source project, licensed under the *GPL v3 Affero*[2]. The library's code is publicly available at `https://github.com/LAL-project/linear-arrangement-library.git`.

2.4 Robustness

Often, testing is not thorough or is not specified. Therefore, the increasing amount of research in Quantitative Dependency Syntax creates a need for a thoroughly-tested tool in which to find many if not most of the metrics devised so far. The continual testing that is applied to LAL solves this problem: tests are run periodically to ensure that all algorithms calculate correct values.

2.5 The architecture of LAL

The LAL project's architecture consists of a *core* and *extensions* (Figure 3). The core has two parts: the main branch and the testing branch (Figure 3). The latter is responsible for the robustness of the main branch. The test branch is not publicly available yet but it results from the transfer of knowledge and methods that enabled to test and correct classic algorithms with random and exhaustive methods (Esteban and Ferrer-i-Cancho, 2017; Alemany-Puig and Ferrer-i-Cancho, 2020; Alemany-Puig et al., 2022). The C++ language is used in LAL's core to implement its main functionalities and algorithms, some of which are parallelized internally to improve performance (a critical example is the function that computes (all) scores on a treebank collection). However, only the library branch is wrapped to Python (via SWIG (2020); Figure 3) given the fact that Python is usually the first choice of many scientists who are looking forward to automatizing their workflow as it is easy to use.

LAL extensions implement additional functionalities, e.g., dealing with the interface between existing treebanks and LAL. 'LAL extensions' is the place for contributing to the LAL ecosystem without knowledge on how LAL is implemented. A concrete LAL extension is introduced in the next section.

LAL's core is composed of 7 modules (Figure 3). Now follows a brief description of each of them.

- The *generate* module contains algorithms to generate trees uniformly at random or exhaustively, labeled or unlabeled, and free or rooted; algorithms to generate arrangements of trees under the projectivity or planarity constraint,

- In the *graphs* module users will find the implementations of different graph classes: undirected and directed graphs, free and rooted trees,

- The *io* (input/output) module contains algorithms to read data from a file in disk, but its current chief goal is to process input data, such as treebanks, to produce an output file with measures computable by the library,

- The *numeric* module contains wrappers of the GMP library (GMP, 2021) for arbitrary-precision integer and rational numbers to ensure exact precision in the calculations (mostly useful for the testing branch of the project or advanced research in mathematics),

- In the *linarr* module users will find many algorithms to compute measures of graphs that are defined on the linear ordering of the vertices, e.g., the sum of dependency distances (Gildea and Temperley, 2007),

[2]Many people think this license formally discourages commercial usage, but it certainly does not `https://www.gnu.org/licenses/agpl-3.0.en.html`.

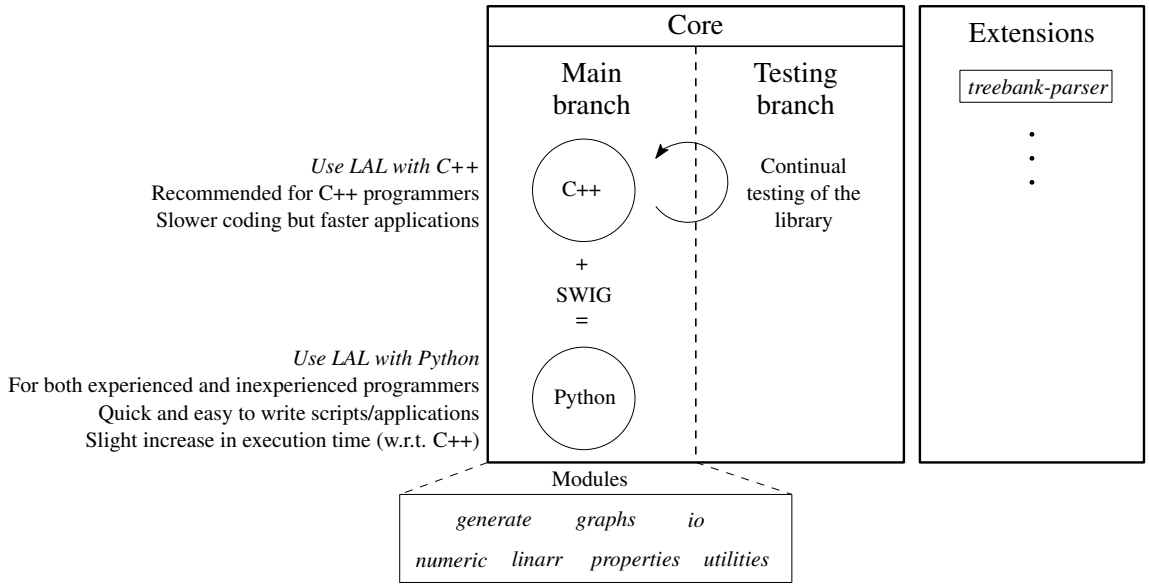

Figure 3: The architecture of LAL project: *core* and *extensions*. The core consists of the library and testing branch. The library is developed in C++, and wrapped into Python via SWIG (2020). Testing the library is a crucial part of its development. The library is composed of 7 modules (both C++ and Python options): *generate*, *graphs*, *io*, *numeric*, *linarr*, *properties*, and *utilities*, each of which is briefly described in Section 2.

- The *properties* module implements algorithms to compute measures of graphs that do not depend on the linear ordering of their vertices, only on their structure, e.g., the Mean Hierarchical Distance (Jing and Liu, 2015),

- Finally, the *utilities* module contains algorithms that are interesting to the general public but cannot be classified into the categories above, such as the tree isomorphism test that tells whether or not two trees are the same. This test only takes into account the structure of the tree (nodes and edges) and not possible labels on the edges (e.g. grammatical relations between words in a sentence) or possible tokens in the nodes (e.g. words from a sentence). The algorithm implemented in LAL (Aho et al., 1974) is an $O(n)$-time algorithm (where n is the number of vertices of the trees) adapted to the case when the two trees are free or rooted.

3 Working with LAL

LAL's capabilities range over three kinds of operations depending on the entity to which they are applied: operations on *individual* syntactic dependency structures (modules *graphs*, *linarr*, *properties*), operations on individual *treebanks* or on treebank *collections* (*io* module).

LAL contains many functions with which one can calculate metrics/scores on a single tree. At present, the focus of LAL is on trees for simplicity and because of the high current interest on this kind of graphs. However, although fewer in number, LAL also contains functions that admit general graphs (e.g., graphs with cycles, and forests). Functions might depend on a given word ordering (*linarr* module) or not (*properties* module). Most functions in the *linarr* module typically require a linear arrangement as an input parameter. However, some functions return a linear arrangement rather than requiring one as an input parameter, as in the example given in Code 2. It is worth mentioning that LAL implements methods for a flexible construction of graphs (*graphs* module) by adding edges one by one, or in bulk, thus allowing complicated workflows, whenever those are needed. Nevertheless, LAL is also equipped with helper functions for simpler graph construction (e.g., construct a graph directly from its set of edges). Furthermore, it provides functions for reading graphs from a file in its *io* module.

```
import lal
t = lal.graphs.from_edge_list_to_free_tree([(0,1), ...])
Shiloachs_algorithm = lal.linarr.algorithms_Dmin.Shiloach
print(lal.linarr.min_sum_edge_lengths(t), Shiloachs_algorithm)
Chungs_algorithm = lal.linarr.algorithms_Dmin.Chung_2
print(lal.linarr.min_sum_edge_lengths(t), Chungs_algorithm)
```

Code 2: Python script to calculate the minimum baseline for the sum of dependency distances on a free tree with LAL applying two different algorithms: Shiloach's (Shiloach, 1979; Esteban and Ferrer-i-Cancho, 2017) and Chung's quadratic algorithm (Chung, 1984). LAL implements the correction of Shiloach's algorithm in (Esteban and Ferrer-i-Cancho, 2017). The free tree is specified as a list of pairs of edges. That baseline is the solution of the so-called minimum linear arrangement problem of computer science, hence the names of the algorithms mentioned above.

The library also implements treebank processing (*io* module). It provides its users with algorithms to generate data out of single treebank files and collections of treebanks. A collection of treebanks is simply a set of treebanks that are related to one another within some particular context. In the UD collection, texts from distinct languages annotated with the same formalism (Zeman et al., 2020), but there are many other possibilities: e.g., the novels of the same writer – where an individual treebank is a novel, all the articles in a given newspaper – where a single treebank is one of said articles.

In LAL a collection is represented by a plain text file listing all the treebank files in the collection. Treebanks are typically written in CoNLL-U format (Buchholz and Marsi, 2006), but LAL requires a simpler format with the essential information. In particular, LAL requires treebanks to be provided as a series of *head vectors* (or *head sequences*); a head vector of an n-word sentence is a sequence of n non-negative integer numbers in the positions from 1 to n where each number indicates the position of its parent word, and the number 0 indicates the root word of the sentence. The most important reason to use the intermediate format is the existence of many previous formats, and the possibility of many new formats coming to life; we believe it will help in reducing the library's usage complexity. This approach is similar to that taken by compiler developers and designers who use the so-called 'Three address code': it is an intermediate language into which many programming languages can be transformed, and is then compiled to the target machine. Nevertheless, we have also developed a LAL extension (Figure 3), the *treebank-parser* that applies core LAL functions to convert CoNLL-U-formatted files into the head vector format after applying optional preprocessing: (a) removal of punctuation marks, e.g., for crosslinguistic generality, (b) removal of functions words, e.g., to compare languages with many such words against languages where these are scarce and (c) removal of sentences shorter or longer than a given length (Ferrer-i-Cancho et al., 2021). This tool can be found online at `https://github.com/LAL-project/treebank-parser.git`.

The processing of treebanks (and collection of treebanks) can be done automatically by LAL, but it can also be customized by its users. In other words, the automatic processing of a treebank (or a collection of treebanks) is performed by the library applying the metrics/scores selected by the user, with optional internal parallelization, and other customizable options on the format of the output. One obvious drawback of automatic processing is that it is limited to the metrics/scores implemented in LAL. Nevertheless, should users look forward to calculating a new score on a treebank (or in a collection), they can still use LAL to carry out such a task since LAL implements algorithms to easily iterate over a file containing only head vectors, thus removing the aforementioned limitation. Therefore, LAL facilitates working on a single treebank or on a collection of treebanks.

Figure 4 illustrates one possible pipeline when working with a single treebank. That pipeline can easily be adapted to any treebank collection. A description now follows.

Phase 1 Firstly, one chooses a source of the input data. Such data may come from a handwritten text, or

a typeset text, which is to be analyzed by either a computer or by a human; the data may come from already-analyzed data such as the UD collection (Zeman et al., 2020; Gerdes et al., 2018), or treebanks annotated with the Stanford (de Marneffe et al., 2014) or Prague (Hajič et al., 2006) conventions.

Phase 2 Secondly, some preprocessing of the data might be required, e.g., removal of punctuation marks, removal of function words, or the exclusion of sentences that are too short or too long. See Ferrer-i-Cancho et al. (2021) for a complete example of such preprocessing. We call the resulting treebank 'Treebank (2)'.

Phase 3 Then, one transforms 'Treebank (2)' into 'Treebank (3)', a file containing only the so-called head vectors so that LAL can understand the data.

Phase 4 Here, LAL is used to generate a `.csv` file containing all the measures that are interesting for one's own research. Staying true to its efficiency design principle, LAL can evaluate all metrics it implements on every tree of the UD 2.6 treebank (Zeman et al., 2020) in $\sim$ 23 seconds, while producing only the primary data (on only the UD 2.6 treebank) of a recent article (Ferrer-i-Cancho et al., 2021) takes $\sim$ 6 seconds[3].

Phase 5 The last phase consists of using the data to perform statistical analyses on the treebank. These analyses comprise hypothesis and theoretical prediction testing (Ferrer-i-Cancho and Gómez-Rodríguez, 2021) as well as evaluation of the quality of a treebank (Alzetta et al., 2017; Heinecke, 2019). When the pipeline is adapted to a treebank collection, LAL supports research in typology (Croft et al., 2017; Alzetta et al., 2018) or on comparisons of annotation schemes (Osborne and Gerdes, 2019; Passarotti, 2016).

LAL extensions can be used for tasks in Phases 2 and 3. For the time being, the aforementioned extension *treebank-parser* (Figure 3) allows one to perform Phase 2 and Phase 3 in a row over treebanks in CoNLL-U format. In the future, we hope that LAL extensions grow in number with contributions that researchers wish to make to the LAL ecosystem for Phase 2 or Phase 3.

3.1 Capabilities

To begin with, LAL implements several algorithms to calculate relevant word order-dependent metrics (module *linarr*) in Quantitative Dependency Syntax. These include the sum of dependency distances (Gildea and Temperley, 2007); the number of edge crossings (Ferrer-i-Cancho et al., 2018), with the option to choose among several algorithms; the prediction of the number of crossings based on the length of the edges (Ferrer-i-Cancho, 2014); the class of syntactic dependency structure (such as WG_1 (Gómez-Rodríguez et al., 2011), 1-Endpoint Crossing (Satta et al., 2013), projective and planar (Sleator and Temperley, 1993; Kuhlmann and Nivre, 2006)); the solution to the MLA problem under several constraints (Table 1); the computation of dependency fluxes (Kahane et al., 2017); the proportion of head initial dependencies (Liu, 2010).

Researchers will find in LAL many word order-independent metrics (metrics that do not depend on the ordering of the vertices) in the *properties* module, including the Mean Hierarchical Distance (Jing and Liu, 2015); the variance and expected number of crossings in unconstrained arrangements of trees, useful for variable standardization (Alemany-Puig and Ferrer-i-Cancho, 2020); the expected sum of edge lengths under three different formal constraints (Table 1) to cover different views about the nature of formal constraints (Yadav et al., 2021), and the variance of said sum in unconstrained linear arrangements (Ferrer-i-Cancho, 2019); the calculation of the m-th moment of degrees about zero used to calculate the well-known hubiness coefficient (Ferrer-i-Cancho et al., 2018), extended to the out- and in-degrees[4]; the number of pairs of independent edges (two edges are independent when they share no vertices); the

[3]Running times are estimated on a brand-new PC with a 3.30 GHz, 6-core (2 threads/core) *i5-10600* CPU (16 GB); the program uses 6 threads and runs on Ubuntu 20.04.

[4]Trivially, the m-th moment of in-degree about zero of any n-vertex tree is $\langle k_{in}^m \rangle = (n-1)/n$. Nevertheless, a function to calculate it is provided.

Sampling method	Type of tree		References
Exhaustive	Labeled	Free	Prüfer (1918)
		Rooted	Based on E-L-F*
	Unlabeled	Free	Wright et al. (1986)
		Rooted	Beyer and Hedetniemi (1980)
Random	Labeled	Free	Prüfer (1918)
		Rooted	Based on Rn-L-F**
	Unlabeled	Free	Wilf (1981)***
		Rooted	Nijenhuis and Wilf (1978)

Table 2: The eight different possibilities of tree generation in LAL. By 'random' we mean 'uniformly random over the complete set of the respective kind of trees'. * Based on Exhaustive-Labeled-Free tree generation. ** Based on Random-Labeled-Free tree generation. *** The algorithm includes the correction pointed out in (Marohnić, 2018).

central and centroidal vertices of trees (Harary, 1969). Furthermore, one can classify trees into classes according to their structure. These classes are: *linear*, *star*, *quasistar* and *bistar* (San Diego and Gella, 2014), *caterpillar* (Harary and Schwenk, 1973), and *spider* (Bennett et al., 2019) trees.

In order to overcome the research limitations arising from the lack of research on every possible expected value that can be elicited, LAL is equipped with several algorithms for the generation of trees (Table 2) and of linear arrangements of trees under several formal constraints (projectivity and planarity[5] (Kuhlmann and Nivre, 2006)), and, for the sake of ease of use, it also provides exhaustive and random generation of unconstrained arrangements, i.e., exhaustive and random generation of permutations. Random generation allows one to estimate, via random sampling of the appropriate structure, the expected value of a certain existing or new metric. For example, one could easily approximate the expected Mean Hierarchical Distance (MHD) over the set of uniformly random unlabeled rooted trees by sampling said trees and averaging their MHD. The same can be said about those metrics dependent on word order. It goes without saying that said approximation is not limited to what LAL can calculate: researchers with sufficient programming skills can implement their own metrics either in C++ or in Python and approximate that metric's expectation and other aspects of its distribution under null models; the online documentation at `https://cqllab.upc.edu/lal/guides` explains how these estimates can be calculated and provide examples that inexperienced programmers can easily adapt to the metric of their choice. In previous research, the methods of generation of random trees do not warrant uniform sampling of the space of possible trees (Liu, 2007; Courtin and Yan, 2019). LAL simplifies research where uniformity is required.

3.2 Applications of LAL

LAL's domain of application revolves primarily around studies on Quantitative Dependency Syntax. We have dealt with various applications of LAL when describing Phase 5 of the pipeline (Section 3, Figure 4). LAL is also a convenient tool for reproducing results from previous research. For instance, LAL allows one to reproduce the results of the analyses to predict the actual number of dependency crossings (Gómez-Rodríguez and Ferrer-i-Cancho, 2017), the results on the scaling of sum of dependency distances in minimum linear arrangements (Esteban et al., 2016), or recent findings on the degree of optimality of languages (Ferrer-i-Cancho et al., 2021). This research eventually converged into the formal development of LAL.

LAL offers many possibilities for research on the similarity among trees. For instance, LAL can be used to find how many isomorphic tree structures are present in two treebanks. Also, given any treebank, one can also find the amount of unique trees (up to graph isomorphism). Furthermore, rooted trees are

[5]Random and exhaustive generation of planar arrangements is available in LAL but not published at the time of submission.

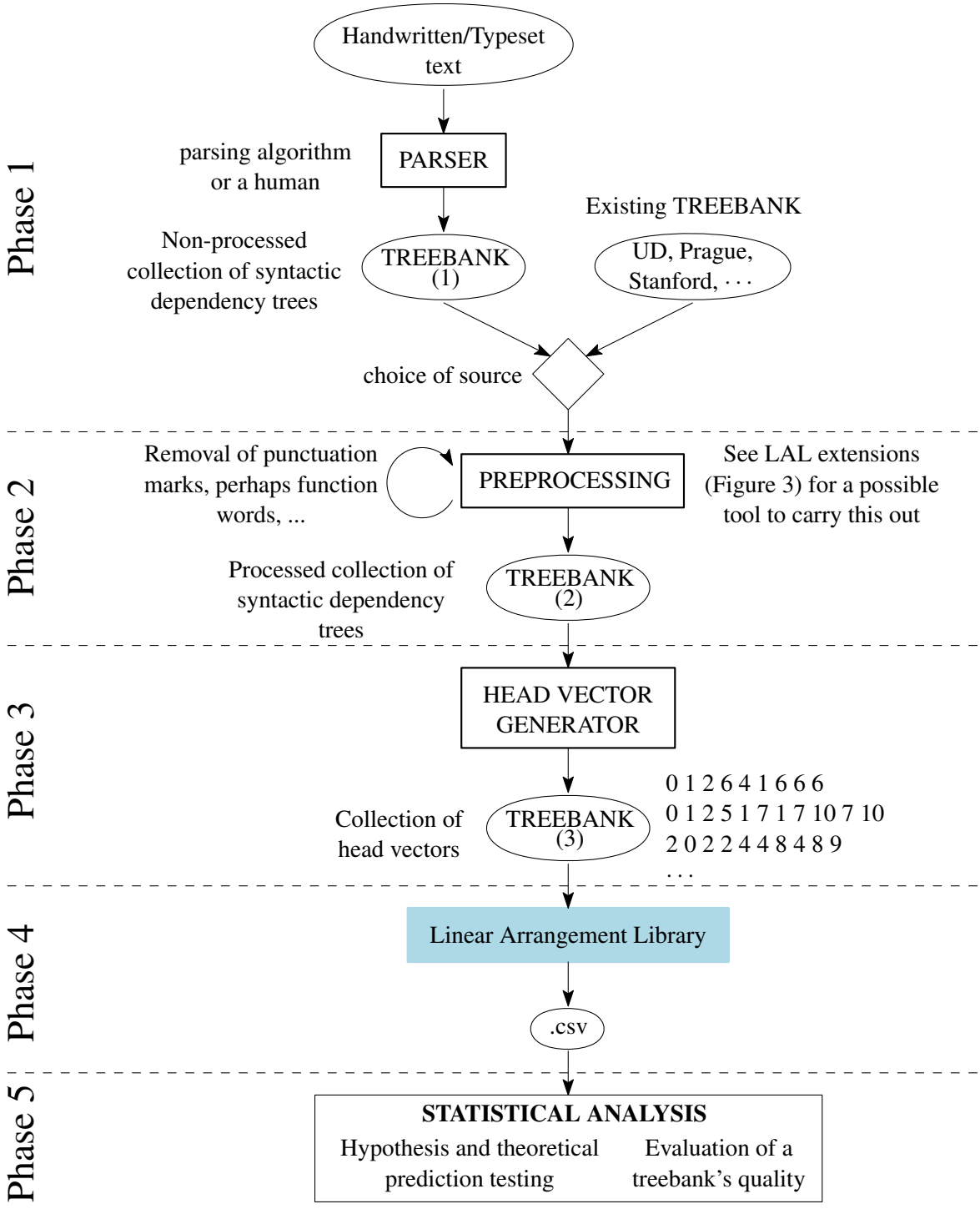

Figure 4: A standard pipeline for research on a single treebank. The pipeline comprises five phases that start with the choice of an existing treebank or producing it from raw text (Phase 1) and ends with analyses of the output produced by LAL (Phase 5). In Phase 4, LAL receives a treebank that has been preprocessed and transformed into a format that LAL can digest.

implemented so that users can extract rooted subtrees and perform similarity tests based on subgraph isomorphism. These have already been tackled for general graphs (Cordella et al., 2004; Jüttner and Madarasi, 2018).

Beyond the realm of measurements on treebanks, LAL contains tree-generation (or linear-arrangement generation) methods that can help one to test new algorithms. With this one can easily implement the testing protocol to assert the correctness of the implementation of the algorithm to compute the minimum sum of dependency distances (Esteban et al., 2016; Esteban and Ferrer-i-Cancho, 2017).

4 Future work

LAL is a growing project to support current and future research. We plan to extend the library with algorithms to calculate extreme or random baselines for scores for which a fast exact algorithm is not available yet. For instance, we plan to add efficient algorithms for the calculation of the maximum sum of edge lengths in unconstrained arrangements and also on said maximum under the projectivity and planarity constraint (Table 1). We will also work on the classification of linear arrangements into different classes of formal constraints and also extend the library with functionalities that ease common tasks in the analysis of syntactic dependency structures or that reflect consolidated results from the distinct disciplines involved. We are also open to update the library based on demands of engaged users.

Acknowledgements

We thank Aleksandra Petrova for helpful comments. LAP is supported by Secretaria d'Universitats i Recerca de la Generalitat de Catalunya and the Social European Fund. RFC and LAP are supported by the grant TIN2017-89244-R from MINECO (Ministerio de Economía, Industria y Competitividad). RFC is also supported by the recognition 2017SGR-856 (MACDA) from AGAUR (Generalitat de Catalunya). JLE is funded by the grant PID2019-109137GB-C22 from MINECO.

References

Alfred V. Aho, Jeffrey E. Hopcroft, and John D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley series in computer science and information processing. Addison-Wesley Publishing Company, Michigan University, 1st edition.

Lluís Alemany-Puig and Ramon Ferrer-i-Cancho. 2020. Edge crossings in random linear arrangements. *Journal of Statistichal Mechanics*, 2020:023403.

Lluís Alemany-Puig and Ramon Ferrer-i-Cancho. 2021. Linear-time calculation of the expected sum of edge lengths in projective linearizations of trees. *arXiv*.

Lluís Alemany-Puig, Juan Luis Esteban, and Ramon Ferrer-i-Cancho. 2022. Minimum projective linearizations of trees in linear time. *Information Processing Letters*, 174:106204.

Chiara Alzetta, Felice Dell'Orletta, Simonetta Montemagni, and Giulia Venturi. 2017. Dangerous relations in dependency treebanks. In *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories*, pages 201–210, Prague, Czech Republic.

Chiara Alzetta, Felice Dell'Orletta, Simonetta Montemagni, and Giulia Venturi. 2018. Universal Dependencies and quantitative typological trends. a case study on word order. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May. European Language Resources Association (ELRA).

Patrick Bennett, Sean English, and Maria Talanda-Fisher. 2019. Weighted Turán problems with applications. *Discrete Mathematics*, 342:2165–2172, 8.

Karl-Heinz Best and Otto Rottmann. 2017. *Quantitative Linguistics, an Invitation*. RAM-Verlag, Lüdenscheid, Germany, 1 edition.

Terry Beyer and Sandra Mitchell Hedetniemi. 1980. Constant time generation of rooted trees. *SIAM Journal on Computing*, 9(4):706–712.

Rishi Bommasani. 2020. Generalized optimal linear orders. Master's thesis, Cornell University.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, 06. Association for Computational Linguistics.

Fan R. K. Chung. 1984. On optimal linear arrangements of trees. *Computers & Mathematics with Applications*, 10(1):43–60.

Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372.

Marine Courtin and Chunxiao Yan. 2019. What can we learn from natural and artificial dependency trees. In *Proceedings of the First Workshop on Quantitative Syntax (Quasy, SyntaxFest 2019)*, pages 125–135, Paris, France, 08. Association for Computational Linguistics.

William Croft, Dawn Nordquist, Katherine Looney, and Michael Regan. 2017. Linguistic typology meets universal dependencies. In *TLT*.

Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. 2014. Universal stanford dependencies: A cross-linguistic typology. In *LREC*.

Matt DeVos and Kathryn Nurse. 2018. A maximum linear arrangement problem on directed graphs. *arXiv*.

Juan Luis Esteban and Ramon Ferrer-i-Cancho. 2017. A correction on shiloach's algorithm for minimum linear arrangement of trees. *SIAM Journal on Computing*, 46(3):1146–1151.

Juan Luis Esteban, Ramon Ferrer-i-Cancho, and Carlos Gómez-Rodríguez. 2016. The scaling of the minimum sum of edge lengths in uniformly random trees. *Journal of Statistical Mechanics: Theory and Experiment*, 2016(6):063401, jun.

Ramon Ferrer-i-Cancho and Carlos Gómez-Rodríguez. 2021. Anti dependency distance minimization in short sequences. a graph theoretic approach. *Journal of Quantitative Linguistics*, 28(1):50–76.

Ramon Ferrer-i-Cancho, Carlos Gómez-Rodríguez, and Juan Luis Esteban. 2018. Are crossing dependencies really scarce? *Physica A: Statistical Mechanics and its Applications*, 493:311–329.

Ramon Ferrer-i-Cancho, Carlos Gómez-Rodríguez, Juan Luis Esteban, and Lluís Alemany-Puig. 2021. The optimality of syntactic dependency distances. *Physical Review E*, page in press.

Ramon Ferrer-i-Cancho. 2003. *Language Universals: Principles and origins*. Ph.D. thesis, Universitat Politécnica de Catalunya - BarcelonaTech. (unpublished).

Ramon Ferrer-i-Cancho. 2004. Euclidean distance between syntactically linked words. *Physical Review E*, 70(5):5.

Ramon Ferrer-i-Cancho. 2014. A stronger null hypothesis for crossing dependencies. *EPL (Europhysics Letters)*, 108(5):58003, 12.

Ramon Ferrer-i-Cancho. 2019. The sum of edge lengths in random linear arrangements. *Journal of Statistical Mechanics: Theory and Experiment*, 2019:053401, 05.

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.

Richard Futrell, Roger Park Levy, and Edward Gibson. 2020. Dependency locality as an explanatory principle for word order. *Language*, 96(2):371–412.

Michael R. Garey and David Stifler Johnson. 1976. Some simplified NP-complete graph problems. *Theoretical Computer Science*, pages 237–267.

Kim Gerdes, Bruno Guillaume, Sylvain Kahane, and Guy Perrier. 2018. SUD or surface-syntactic universal dependencies: An annotation scheme near-isomorphic to UD. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 66–74, Brussels, Belgium, 11. Association for Computational Linguistics.

Daniel Gildea and David Temperley. 2007. Optimizing grammars for minimum dependency length. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 184–191, Prague, Czech Republic, 06. Association for Computational Linguistics.

David Gildea and David Temperley. 2010. Do Grammars Minimize Dependency Length? *Cognitive Science*, 34(2):286–310.

GMP. 2021. The GNU multiple precision arithmetic library. `https://gmplib.org/`. Accessed: 2021-09-16.

Mark K. Goldberg and Israel A. Klipker. 1976. A Minimal Placement of a Tree on the Line. Technical report, Physico-Technical Institute of Low Temperatures. Academy of Sciences of Ukranian SSR, USSR. in Russian.

Carlos Gómez-Rodríguez and Ramon Ferrer-i-Cancho. 2017. Scarcity of crossing dependencies: A direct outcome of a specific constraint? *Physical Review E*, 96:062304.

Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2011. Dependency Parsing Schemata and Mildly Non-Projective Dependency Parsing. *Computational Linguistics*, 37(3):541–586.

Carlos Gómez-Rodríguez, Morten H. Christiansen, and Ramon Ferrer-i-Cancho. 2020. Memory limitations are hidden in grammar. *Arxiv*, page under review.

Kristina Gulordava and Paola Merlo. 2015. Diachronic trends in word order freedom and dependency length in dependency-annotated corpora of Latin and ancient Greek. In *Proceedings of the Third International Conference on Dependency Linguistics (Depling 2015)*, pages 121–130, Uppsala, Sweden, 08. Uppsala University.

Kristina Gulordava and Paola Merlo. 2016. Multi-lingual dependency parsing evaluation: a large-scale analysis of word order properties using artificial data. *Transactions of the Association for Computational Linguistics*, 4:343–356.

Jan Hajič, Jarmila Panevová, Eva Hajičová, Jarmila Panevová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, and Marie Mikulová. 2006. Prague Dependency Treebank 2.0. CDROM CAT: LDC2006T01, ISBN 1-58563-370-4. Linguistic Data Consortium.

Frank Harary and Allen J. Schwenk. 1973. The number of caterpillars. *Discrete Mathematics*, 6:359–365.

Frank Harary. 1969. *Graph Theory*. Addison-Wesley, Reading, MA.

Refael Hassin and Shlomi Rubinstein. 2000. Approximation algorithms for maximum linear arrangement. In *Scandinavian Workshop on Algorithm Theory - Algorithm Theory - SWAT 2000*, volume 1851, pages 231–236.

Johannes Heinecke. 2019. ConlluEditor: a fully graphical editor for universal dependencies treebank files. In *Proceedings of the Third Workshop on Universal Dependencies (UDW, SyntaxFest 2019)*, pages 87–93, Paris, France, 08. Association for Computational Linguistics.

Robert A. Hochberg and Matthias F. Stallmann. 2003. Optimal one-page tree embeddings in linear time. *Information Processing Letters*, 87(2):59–66.

Richard Hudson. 1995. Measuring syntactic difficulty. *Unpublished paper*.

Mikhail Anatolievich Iordanskii. 1987. Minimal numberings of the vertices of trees — approximate approach. In Lothar Budach, Rais Gatič Bukharajev, and Oleg Borisovič Lupanov, editors, *Fundamentals of Computation Theory*, pages 214–217, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jingyang Jiang and Haitao Liu, editors. 2018. *Quantitative Analysis of Dependency Structures*. De Gruyter Mouton, Berlin, 1 edition.

Yingqi Jing and Haitao Liu. 2015. Mean hierarchical distance. Augmenting mean dependency distance. In *Proceedings of the Third International Conference on Dependency Linguistics*, pages 161–170.

Alpár Jüttner and Péter Madarasi. 2018. VF2++ – An improved subgraph isomorphism algorithm. *Discrete Applied Mathematics*, 242:69–81. Computational Advances in Combinatorial Optimization.

Sylvain Kahane, Chunxiao Yan, and Marie-Amélie Botalla. 2017. What are the limitations on the flux of syntactic dependencies? evidence from ud treebanks. In *Proceedings of the Fourth International Conference on Dependency Linguistics*, pages 73–82, 9.

Reinhard Köhler and Gabriel Altmann. 2012. *Quantitative Syntax Analysis*. Quantitative linguistics. De Gruyter Mouton.

Saeko Komori, Masatoshi Sugiura, and Wenping Li. 2019. Examining MDD and MHD as syntactic complexity measures with intermediate Japanese learner corpus data. In *Proceedings of the Fifth International Conference on Dependency Linguistics (Depling, SyntaxFest 2019)*, pages 130–135, Paris, France, 08. Association for Computational Linguistics.

Alex Kramer. 2021. Dependency lengths in speech and writing: A cross-linguistic comparison via YouDePP, a pipeline for scraping and parsing YouTube captions. In *Proceedings of the Society for Computation in Linguistics*, volume 4, pages 359–365.

Marco Kuhlmann and Joakim Nivre. 2006. Mildly non-projective dependency structures. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, COLING-ACL '06, pages 507–514, 07.

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.

Haitao Liu. 2007. Probability distribution of dependency distance. *Glottometrics*, 15:1–12, 06.

Haitao Liu. 2008. Dependency distance as a metric of language comprehension difficulty. *Journal of Cognitive Science*, 9(2):159–191.

Haitao Liu. 2010. Dependency direction as a means of word-order typology: a method based on dependency treebanks. *Lingua*, 120(6):1567–1578.

Luka Marohnić. 2018. Graph theory package for giac/xcas - reference manual. `https://usermanual.wiki/Document/graphtheoryusermanual.346702481/view`. Accessed: 2020-01-13.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530.

Igor Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany, NY, USA.

Albert Nijenhuis and Herbert S. Wilf. 1978. *Combinatorial Algorithms: For Computers and Hard Calculators*. Academic Press, Inc., Orlando, FL, USA, 2nd edition.

Joakim Nivre. 2006. Constraints on non-projective dependency parsing. In *EACL 2006 - 11th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 73–80.

T. Osborne and K. Gerdes. 2019. The status of function words in dependency grammar: A critique of Universal Dependencies (UD). *Glossa: A Journal of General Linguistics*, 4(1):17.

Y. Albert Park and Roger Park Levy. 2009. Minimal-length linearizations for mildly context-sensitive dependency trees. In *Proceedings of the 10th Annual Meeting of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT) conference*, pages 335–343, Stroudsburg, PA, USA. Association for Computational Linguistics.

Marco Carlo Passarotti. 2016. How far is stanford from prague (and vice versa)? comparing two dependency-based annotation schemes by network analysis. *L'ANALISI LINGUISTICA E LETTERARIA*, pages 21–46.

Daniele Pighin. 2012. The Ti*k*Z-`dependency` package. `https://osl.ugr.es/CTAN/graphics/pgf/contrib/tikz-dependency/tikz-dependency-doc.pdf`. Accessed: 2021-06-17.

Heinz Prüfer. 1918. Neuer Beweis eines Satzes über Permutationen. *Arch. Math. Phys*, 27:742–744.

Alfréd Rényi and George Szekeres. 1967. On the height of trees. *Journal of the Australian Mathematical Society*, 7(4):497–507.

Immanuel T. San Diego and Frederick S. Gella. 2014. The b-chromatic number of bistar graph. *Applied Mathematical Sciences*, 8(116):5795–5800.

Giorgio Satta, Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2013. Finding optimal 1-Endpoint-Crossing trees. In *Transactions of the Association for Computational Linguistics*, pages 13–24.

Yossi Shiloach. 1979. A minimum linear arrangement algorithm for undirected trees. *SIAM Journal on Computing*, 8(1):15–32.

Daniel Sleator and Davy Temperley. 1993. Parsing English with a link grammar. In *Proceedings of the Third International Workshop on Parsing Technologies (IWPT'93)*, pages 277–292. ACL/SIGPARSE.

SWIG. 2020. Swig 4.0.2. `http://www.swig.org/`. Accessed: 2021-06-09.

David Temperley and Daniel Gildea. 2018. Minimizing syntactic dependency lengths: Typological/cognitive universal? *Annual Review of Linguistics*, 4(1):67–80.

Oleg Verbitsky. 2008. On the obfuscation complexity of planar graphs. *Theoretical Computer Science*, 396(1):294–300.

Herbert S. Wilf. 1981. The uniform selection of free trees. *Journal of Algorithms*, 2:204–207.

Robert Alan Wright, Bruce Richmond, Andrew Odlyzko, and Brendan D. McKay. 1986. Constant time generation of free trees. *SIAM Journal on Computing*, 15:540–548, 05.

Himanshu Yadav, Samar Husain, and Richard Futrell. 2019. Are formal restrictions on crossing dependencies epiphenominal? In *Proceedings of the 18th International Workshop on Treebanks and Linguistic Theories (TLT, SyntaxFest 2019)*, pages 2–12, Paris, France, 08. Association for Computational Linguistics.

Himanshu Yadav, Samar Husain, and Richard Futrell. 2021. Do dependency lengths explain constraints on crossing dependencies? *Linguistics Vanguard*, 7(s3):20190070.

Xiang Yu, Agnieszka Falenska, and Jonas Kuhn. 2019. Dependency length minimization vs. word order constraints: An empirical study on 55 treebanks. In *Proceedings of the First Workshop on Quantitative Syntax (Quasy, SyntaxFest 2019)*, pages 89–97, Paris, France, 08. Association for Computational Linguistics.

Daniel Zeman, Joakim Nivre, Mitchell Abrams, Elia Ackermann, Noëmi Aepli, Željko Agić, Lars Ahrenberg, Chika Kennedy Ajede, Gabrielė Aleksandravičiūtė, Lene Antonsen, Katya Aplonova, Angelina Aquino, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Furkan Atmaca, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, Colin Batchelor, John Bauer, Kepa Bengoetxea, Yevgeni

Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Agnė Bielinskienė, Rogier Blokland, Victoria Bobicev, Loïc Boizou, Emanuel Borges Völker, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Kristina Brokaitė, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Tatiana Cavalcanti, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Ethan Chi, Jinho Choi, Yongseok Cho, Jayeol Chun, Alessandra T. Cignarella, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Elvis de Souza, Arantza Diaz de Ilarraza, Carly Dickerson, Bamba Dione, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Hanne Eckhoff, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Olga Erina, Tomaž Erjavec, Aline Etienne, Wograine Evelyn, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Kazunori Fujita, 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 González Saavedra, Bernadeta Griciūtė, Matias Grioni, Loïc Grobol, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Tunga Güngör, Nizar Habash, Jan Hajič, Jan Hajič jr., Mika Hämäläinen, Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Johannes Heinecke, Oliver Hellwig, Felix Hennig, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Takumi Ikeda, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Hildur Jónsdóttir, Fredrik Jørgensen, Markus Juutinen, Hüner Kaşıkara, Andre Kaasen, Nadezhda Kabaeva, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Elena Klementieva, Arne Köhn, Abdullatif Köksal, Kamil Kopacewicz, Timo Korkiakangas, Natalia Kotsyba, Jolanta Kovalevskaitė, 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, Maria Levina, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Yuan Li, 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, Hiroshi Matsuda, Yuji Matsumoto, Ryan McDonald, Sarah McGuinness, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Maria Mitrofan, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Tomohiko Morioka, Shinsuke Mori, Shigeki Moro, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Robert Munro, 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, Yoshihiro Nikaido, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Atul Kr. Ojha, Adédayọ̀ Olúòkun, Mai Omura, Emeka Onwuegbuzia, Petya Osenova, Robert Östling, Lilja Øvrelid, Şaziye Betül Özateş, Arzucan Özgür, Balkız Öztürk Başaran, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino-Passos, Angelika Peljak-Łapińska, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Daria Petrova, Slav Petrov, Jason Phelan, Jussi Piitulainen, Tommi A Pirinen, Emily Pitler, Barbara Plank, Thierry Poibeau, Larisa Ponomareva, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Peng Qi, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Petru Rebeja, Siva Reddy, Georg Rehm, Ivan Riabov, Michael Rießler, Erika Rimkutė, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Valentin Roșca, Davide Rovati, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Alessio Salomoni, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Dage Särg, Baiba Saulīte, Yanin Sawanakunanon, Salvatore Scarlata, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Hiroyuki Shirasu, Muh Shohibussirri, Dmitry Sichinava, Aline Silveira, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Maria Skachedubova, Aaron Smith, Isabela Soares-Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Shingo Suzuki, Zsolt Szántó, Dima Taji, Yuta Takahashi, Fabio Tamburini, Takaaki Tanaka, Samson Tella, Isabelle Tellier, Guillaume Thomas, Liisi Torga, Marsida Toska, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Utku Türk, Francis Tyers, Sumire Uematsu, Roman Untilov, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Andrius Utka, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Aya Wakasa, Lars Wallin, Abigail Walsh, Jing Xian Wang, Jonathan North Washington, Maximilan Wendt, Paul Widmer, Seyi Williams, Mats Wirén, Christian Wittern, Tsegay Woldemariam, Tak-sum Wong, Alina Wróblewska, Mary Yako, Kayo Yamashita, Naoki Yamazaki, Chunxiao Yan, Koichi Yasuoka, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Hanzhi Zhu, and Anna Zhuravleva. 2020. Universal dependencies 2.6. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

George Kingsley Zipf. 1949. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley Press, Oxford, England.