On Syntactical Graphs-of-Words

Nabil Moncef Boukhatem, Davide Buscaldi, and Leo Liberti

Abstract A graph-of-words is a graph representation of natural language text based on proximity in the linear text reading order: the vertices are the words, and edges are induced by k left and right neighbours of the words. Vertices representing same or similar words are then contracted. We propose graphs-of-words where edges are instead induced on paths in the syntax trees (we investigate both dependency and constituency trees). We discuss some properties, advantages, and disadvantages of classic and new graphs-of-words on texts extracted from literature, as well as from a technical Q&A database.

1 Introduction

Natural language is human-specific, ambiguous, and often ungrammatical; its understanding is usually subjected to context knowledge. It is opposed to formal language, which is computer-specific, unambiguous, and must be grammatically perfect to be meaningful: its pragmatics are formally defined by the effect it has on an electronic or mechanical system. In this paper we use formal language constructs to instruct computers to deal with natural language text. More precisely, we focus on a very specific and well-known task in Natural Language Processing (NLP), i.e. that of *key-*

N. M. Boukhatem · L. Liberti (✉)
LIX CNRS, Ecole Polytechnique, 91128 Palaiseau, France
e-mail: leo.liberti@polytechnique.edu

N. M. Boukhatem
OneTeam, Paris, France

Present Address:
Lundimatin, Montpellier, France
e-mail: nboukhatem@oneteam.fr; moncef-nabil.boukhatem@lundimatin.fr

D. Buscaldi
LIPN CNRS, Université de Paris-Nord, Villetaneuse, France
e-mail: buscaldi@lipn.univ-paris13.fr

A. Brieden et al. (eds.), *Graphs and Combinatorial Optimization: from Theory to Applications*, AIRO Springer Series 13,
https://doi.org/10.1007/978-3-031-46826-1_14

word extraction: given a text in natural language, output K keywords that a human would find the most pertinent for the text. This obviously poses the issue of empirical verification: since different humans would have different preferences, how do we determine what keywords are "best"? In this paper we resort to *ground truths* put together by a restricted number of humans (see Sect. 3).

The most established method for extracting keywords from natural language text is probably based on ranking functions (see e.g. [14]) based on frequency of words in documents with respect to a set of documents called *corpus* [11]. The main application is automatic document indexing or summarization [12].

This paper replaces the concept of word frequency in documents with that of vertex degrees in graphs that represent the text. Methodologically speaking, the main contribution is a comparison between different graph representations of text. One of the graph representations we consider is derived from constituency syntax trees (see Sect. 1.3, which, to the best of our knowledge, has never been previously considered for this purpose).

1.1 Ranking Functions for Text

The earliest cornerstone of information retrieval in text is perhaps the TF-IDF ranking function. It consists of the product of two other functions: Term Frequency (TF) and Inverse Document Frequency (IDF) [15]. We shall limit our introduction to the functions we actually used in our computational experiments. In the following formulæ, we let C be a corpus (i.e., a set) of text documents, D be a document in C, and t be a term (i.e., a word) in D. Then:

$$\mathsf{tf}(t, D) = |(v \in D \mid v = t)| \tag{1}$$

$$\mathsf{TF}(t, D) = 1 + \ln(1 + \max(0, \ln(\mathsf{tf}(t, D)))) \tag{2}$$

$$\mathsf{IDF}(t, C) = \frac{\ln(|C| + 1)}{\sum_{D \in C} \mathsf{tf}(t, D)} \tag{3}$$

are the basic building blocks for two well-known ranking functions. These are:

$$\mathsf{TFIDF}(t, D, C) = \frac{\mathsf{TF}(t, D)\,\mathsf{IDF}(t, C)}{1 - b + \left(\frac{b|D|}{\sum_{P \in C} |P|/|C|}\right)} \tag{4}$$

$$\mathsf{BM25}(t, D, C) = \frac{(k_1 + 1)\mathsf{tf}(t, D)\,\mathsf{IDF}(t, C)}{k_1\left(1 - b + \left(\frac{b|D|}{\sum_{P \in C} |P|/|C|}\right)\right) + \mathsf{tf}(t, D)}, \tag{5}$$

where $b = 0.5$ and $k_1 = 1.2$. We aim at replacing TF with the weighted degree $\mathsf{tw}(t, D)$ of a word vertex t in a graph $G(D)$ representing a document, namely:

$$\mathsf{tw}(t, D) = \sum_{v \in N_{G(D)}(t)} d_{tv} \tag{6}$$

$$\mathsf{TWIDF}(t, D, C) = \frac{\mathsf{tw}(t, D)\,\mathsf{IDF}(t, C)}{1 - b + b\left(\frac{|D|}{\sum_{D \in C} |P|/|C|}\right)}, \tag{7}$$

where d_{uv} is the weight of the edge $\{u, v\}$ in the graph $G(D)$, and $b = 0.75$. The weights of the constants is taken from [16]. For unweighted graphs $G(D)$ we have $d_{uv} = 1$ for all edges $\{u, v\}$.

1.2 Graph-of-Words

Graphs can be used to summarize and extract keywords from a text in natural language [13]. In general, these graphs encode syntactical and sometimes semantic information on the edges, that represent relations on the words inferred from the text. Here we look at a purely syntactical construction proposed in Rousseau's Ph.D thesis [16] under the name *graph-of-words*.

In a graph-of-words (gow), the vertices are labeled by the words. The edges incident to each node are induced by the proximity of the words that are left and right of the node word in the linear text reading order. For example, in the sentence "Computers are close to understanding natural language", the words "are" and "to" are 1-proximal to "close", and the words "computers" and "understanding" are 2-proximal to "close". In a gow with proximity parameter 2, the node labelled by "close" would be adjacent to the vertices labelled by "computers", "are", "to", "understanding".

Note that, if a word occurs more than once in a text, this construction creates separate vertices referring to each occurrence. Moreover, the resulting graph would be a simple chain of embedded cliques, where almost every vertex has the same degree. This motivates a last contraction step in the construction of gows: if two or more vertices represent different occurrences of the same word, they are contracted to a single node. This last step is sometimes interpreted more broadly, for example by contracting vertices having same *lemmatized* word (i.e. the stem of the word without the desinences). An important pre-processing step to a useful gow is the removal of *stop-words*: words that are very frequent in most texts, but do not carry keyword-status information. Typically, stop-words are articles, auxiliary verbs or particles, prepositions, conjunctions, common adverbs, and so on. An example of a gow is given in Fig. 1. We note that gows of proximity k have at least $2k$ adjacencies.

1.3 Syntax Trees

In the framework of formal languages, *syntax trees* are the trace of a parsing algorithm for the sentences of the language. They also provide the mechanism by which

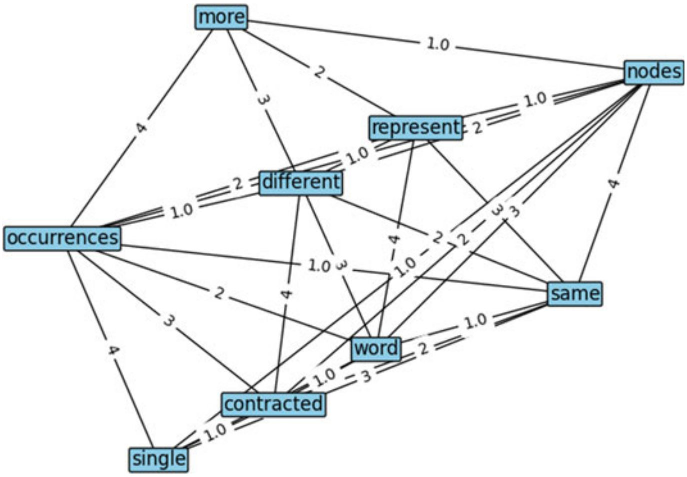

Fig. 1 A graph-of-word with proximity 2 of the sentence "if two or more nodes represent different occurrences of the same word, they are contracted to a single node". Edges are weighted by the text distance between the two word vertices, but this weighting is not essential. The node with largest degree is labelled by the word "nodes" (in the above graph, the two nodes corresponding to "nodes" and "node" were contracted)

computers assign semantics to high-level programs, or, in other words, execute code [6, 10]. Parsing algorithms use a *formal grammar* in order to drive a recursive analysis of a formal language sentence. The grammar consists of a set of rules of the form

$$\mathsf{tag} \longrightarrow \mathsf{comp}_1^1 \ldots \mathsf{comp}_{n_1}^1 \mid \ldots \mid \mathsf{comp}_1^h \ldots \mathsf{comp}_{n_k}^h,$$

which requires that a phrase tag be decomposed in one of h ways, each of which consists of a certain number of components, which can themselves be phrase tags or words. The grammar includes rules for each of the component tags down to the words, which are part of a given vocabulary. Each sentence input is assigned an initial tag, e.g. S for "sentence". The parser resolves tags recursively in terms of the component tags prescribed by the grammar rules, for as long as there are relevant rules that apply. In so doing, the parser produces a syntax tree. If the parser stops before all tags are resolved into constant words, the sentence does not conform to the grammar rules (this is how interpreters and compilers flag syntax errors). Otherwise, the recursive parsing process can also assign executable machine code to each of the constant words (which may be loops, tests, assignments), and then compose the code into an executable program (this is how interpreters and compilers turn a high-level language program into a set of actions performed by the CPU).

Noam Chomsky is credited with the popularization of syntax trees applied to natural languages [1], where the sentence tag S is usually mapped to the decomposition $\mathsf{NP}\ \mathsf{VP}$: i.e., a sentence corresponds to a noun phrase and a verb phrase. These two tags are then recursively decomposed until the words are reached. Since natural lan-

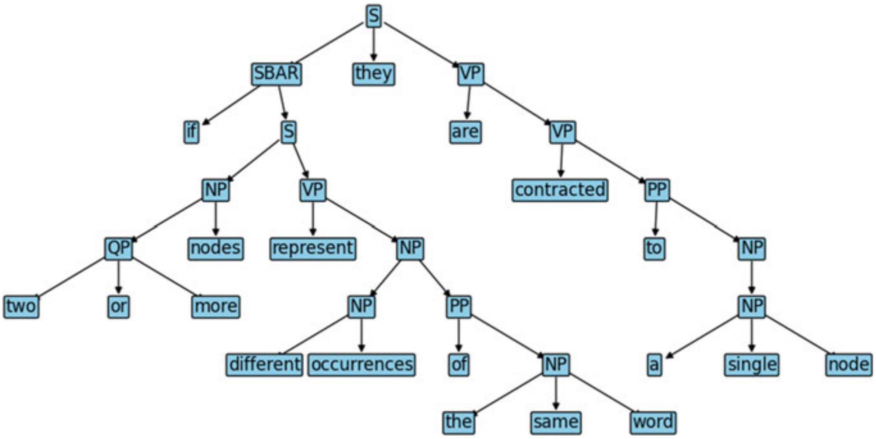

Fig. 2 The constituency tree for the same sentence as in Fig. 1. The tags are: S (sentence), SBAR (subordinate sentence), NP (noun phrase), VP (verb phrase), QP (quantificational phrase), PP (propositional phrase)

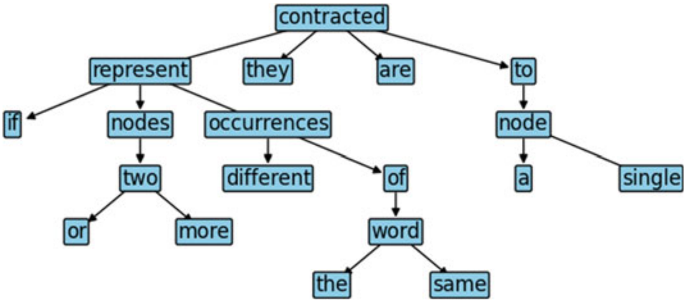

Fig. 3 The dependency tree for the same sentence as in Fig. 1. The arcs are usually labeled by the dependency tag of a child node to its parent node, not shown here because they are not used in this paper

guage is not formal, in general there may be many possible recursive decompositions, all leading to a different meaning, without an obvious way to choose between them. Chomsky's trees are called *constituency trees* (see Fig. 2 for an example).

Dependency trees are different types of trees originally introduced to linguistics by Louis Tesnière. The root of the tree is the main verb of the sentence, which has subject and main complement as child nodes. Each noun node has articles, adjectives, adverbs as child nodes (see Fig. 3 for an example).

Our interest in syntax trees is that they provide a binary relation on words alternative to linear text order proximity. For dependency trees, this order is natural. For constituency trees, the words in a sentence appear as leaf nodes. In both cases, since (undirected) trees are connected, each word is adjacent to any other word by means of the shortest path between the corresponding nodes. This allows us to define a natural edge weight equal to the length of the shortest path.

Contributions of this paper. In this paper we present gows based on different syntactical relations: k-proximity, dependency, constituency. While k-proximity [17] and dependency-based gows [3] are not new, to the best of our knowledge, constituency trees were never used to construct gows so far. We computationally evaluate gows of these different types on several counts.

2 Graph-of-Words Construction Algorithms

By a *sentence* we mean a string that a human could correctly transform into a valid syntax tree. A *phrase* is a sub-string of a sentence, which appears as a sub-tree of the sentence's syntax tree. Sentences are also assumed to be equivalent to lists of *tokens*, where each token can be either a word or a punctuation symbol. Notation-wise, for a sentence s we let s_i be the ith token of s for every $i \leq |s|$, which is the number of tokens of s.

All our gow construction algorithms have three main phases:

1. **generation** of a binary relationship on words;
2. **projection** over important words (and removal of non-important ones: typically these includes punctuation and stop-words);
3. **contraction** of like words (typically words with the same lemmatization, or with a similar meaning according to an existing vocabulary or encyclopedia [2]).

2.1 Proximity Gows

In proximity gows the two phases (generation, projection) may be carried out in either order, but changing the order yields different weights (usefulness of edge weights in proximity gow is doubtful, though [16]). For a string of n tokens, the **generation** phase is as follows. Initially, $V = \{s_1, \ldots, s_n\}$ and E is empty. Then we add edges $\{s_i, s_{i-h}\}$ and $\{s_i, s_{i+h}\}$ for all $1 \leq h \leq k$ and for all $h < i < n - h$.

The **projection** phase, if carried out before generation, simply removes the tokens deemed unimportant from the sentence s. The new list of tokens s' is then subjected to the generation phase. Otherwise projection re-arranges edges incident to removed token vertices: we iteratively replace pairs of edges $(\{v, u\}, \{u, w\})$ incident to a removed vertex u by means of an edge $\{v, w\}$ with weight $d_{vw} = d_{vu} + d_{uw}$. Note that the removal process may add an edge $\{v, w\}$ involving a removed vertex: this edge will be part of a replaced pairs later in the iteration.

Proposition 1 *Let $G = (V, E)$ be the the k-proximity graph obtained from the sentence $s = (s_1, \ldots, s_n)$ by performing generation first, then projection; and $H = (U, F)$ be obtained by projection then generation. We have $G = H$.*

Proof We have $V = U$ because projection removes the same vertices whether carried out before or after generation. Let us now consider an edge $\{u, v\} \in E$, where $u = s_i$ and $v = s_j$ for some $i < j$. If $j - i \leq k$ in the original sentence s then projection either leaves $j - i$ invariant or makes it smaller, so $\{u, v\} \in F$. Assume now that $j - i = k + 1$. This means that there is an index h with $i < h < j$ such that s_h is a removed node. Then, after generation, there must be an edge pair $(\{s_i, s_h\}, \{s_h, s_j\})$ in the graph that is replaced by a single edge $\{s_i, s_j\}$: obviously, since s_h is removed first in H, this edge is also in F. By induction, the same holds for any value of $j - i > k$. The argument showing that edges in F must also be in E is similar. □

Given a weighted graph $G = (V, E, d)$ where V is a set of tokens of a string s, the **contraction** in G of a subset $R \subset V$ s.t. $|R| \geq 2$ is as follows: (i) a representative $r \in R$ is chosen; (ii) in all edges $\{v, u\} \in E$ with $v \notin R$ and $u \in R$ the symbol u is replaced by r, with $d_{vr} = d_{vu} + d_{ur}$; (iii) all edges in the induced subgraph $G[R]$ are removed from E; (iv) all vertices in R except from r are removed from V.

Corollary 1 *Before contraction, the token graph $G = (V, E)$ constructed by generation and projection has $|V| - 2k$ vertices (from the $(k + 1)$-st to the $(n - k)$th) having the same node degree $2k$.*

Proof By Proposition 1, the graph $G = (V, E)$ can be constructed by projection first and then generation. Therefore this graph is a k-proximity graph, where the ith vertex has degree $2k$ for all $k < i \leq n - k$. □

Corollary 1 shows that the contraction step is essential to yielding proximity gows with range of different vertex degrees. This feature is important insofar as our aim is to look at word ranking functions based on vertex degrees in gows rather than word frequencies in documents.

2.2 Dependency

A dependency tree is by definition a tree graph over the sentence tokens. The **generation** of dependency trees from sentences is carried out by either Probabilistic Context-Free Grammar (PCFG) parsers [9] or appropriately trained neural networks [5].

Projection and **contraction** are the same as for proximity-based gows. We note that the projection step on dependency trees has a weak impact on connectivity, since most of the important tokens are naturally set at nodes closer to the root than non-important ones.

2.3 Constituency

A constituency tree is a tree graph over sentence tokens as well as syntax tags. In this sense, constituency trees can be seen as “liftings” from dependency trees. To a given constituency tree, one can retrieve the corresponding dependency tree[1] [7]. Vice-versa, there may be more than one constituency tree corresponding to a given dependency tree [18]. Existing algorithms aim at finding the smallest corresponding constituency tree.

The **generation** of constituency trees from sentences is carried out by either PCFG parsers (see https://nlp.stanford.edu/software/srparser.html) or appropriately trained neural networks (see https://pypi.org/project/benepar/).

Because constituency trees have more nodes than just tokens from the given sentence, a preliminary projection step is necessary to remove all of the non-token nodes. This is different from the projection step in proximity and dependency gows, because the impact on connectivity when removing grammatical tag nodes is considerable. We therefore defined a more connectivity-aware variant of projection: (i) for any pair (u, v) of leaf nodes (word tokens) in the constituency tree T of the sentence s, compute the shortest path $u \to v$ in T having length ℓ, and add the edge $\{u, v\}$ with weight $d_{uv} = \ell$ to the graph; (ii) remove all arcs adjacent to at least one non-leaf node; (iii) remove all non-leaf nodes. This preliminary projection step transforms the constituency tree into a graph on the word tokens from the sentence s.

We note that the most efficient algorithm for computing shortest paths in trees is by means of the Lowest Common Ancestor (LCA) of the origin and destination nodes. This yields a linear-time shortest path algorithm.

Projection and **contraction** are the same as for proximity-based gows.

3 Computational Experiments

Our benchmark aims at establishing advantages and disadvantages of different types of gows in keyword extraction tasks. We consider two corpora: a literary one, and a technical one. We extract keywords from documents in these corpora using the following rank functions: TFIDF and BM25 using term frequency, and TWIDF on k-proximity, constituency tree, and dependency tree based gows (see Sect. 1.1).

Our code is written in Python 3.10. For dependency and constituency syntax trees we made use of `spaCy` 3.4.4 [5] and `benepar` 0.2.0 [8]. Graphs were encoded and handled in NetworkX [4] 2.8.6. Experiments were obtained on an Apple M1 Max CPU with 64GB RAM and MacOS 12.6.3. See http://www.github.com/leoliberti/syntaxGraphOfWords to access the code and the corpora.

3.1 The Literary Dataset

The literary corpus contains 18 short documents extracted from various literary work, each consisting of a single paragraph. The lexical and grammatical quality of these

[1] See https://github.com/wenkokke/dep2con.

Table 1 Comparative results on a set of paragraphs from various literary sources, from which we extracted the three highest-rank keywords with various methods. We report the number of keywords given by each method that is in the list of three keywords in the ground truth

Instance		TermFreq		Graphs-of-words			
Source	*kw*	TFIDF	BM25	1-*prox*	4-*prox*	*con*	*dep*
1177 b.C.	3	1	1	1	2	0	1
Crossings	3	1	1	1	1	0	0
The golden bough	3	1	1	0	0	0	0
Illuminating Eco	3	0	0	0	0	0	0
The island of the day before	3	1	1	1	2	1	1
The library of Babel	3	0	0	0	0	0	1
Media stories: Malvinas	3	1	1	1	1	1	0
Neverwhere	3	2	2	1	1	0	1
Nothing	3	0	0	0	0	0	0
Paine	3	0	0	0	0	0	0
Foucault's Pendulum	3	0	0	0	0	0	0
The perks of being a wallflower	3	1	1	1	1	0	0
Quantum computing since Democritus	3	0	0	0	0	0	0
Richard III	3	1	1	1	1	0	1
The seventh function of language	3	0	0	0	0	0	0
Walden	3	1	1	0	0	0	0
When the sleeper wakes	3	1	1	1	1	0	0
Wisdom	3	0	0	0	0	0	1
Total	54	***11***	***11***	*8*	*10*	*2*	*6*

excerpts is perfect. The ground truth is a set of three keywords per document. These keywords were established by the authors of this paper before obtaining the computational results (we admit nonetheless to a considerable risk of personal bias in our ground truth).

The keywords extracted automatically from the literary corpus are the 3 topmost ranking ones according to the values of term frequency and graph degree rank functions. In Table 1 we report the number of keywords guessed by the automatic methods that are part of the set of keywords in the ground truth.

We see from Table 1 that term frequency based ranking methods are better than gow-based methods. Amongst the latter, 4-proximity gows yield the best performance. We also note that the two term frequency based rankings have exactly the same performance.

3.2 The Technical Dataset

The technical corpus consists of 449 documents, each of which is a client question to technical support. The corresponding ground truth was collected by one of the authors of this paper (NB) in the course of his work at OneTeam. The questions are "as asked", with the normal amount of lexical quirks and ungrammatical phrases. These documents are short (8.6 words on average). We therefore restricted k-proximity to $k = 1$, otherwise the central word in the sentence would have ended up having an abnormally high vertex degree in the k-proximity gow. The average number of keywords per document in the ground truth is 2.4, but the maximum is 5: we therefore allowed the extraction of up to 5 keywords (the gows often had fewer than five vertices, however).

In Table 2 we present comparative statistical distributions on the success scores of each method on documents with a certain number of ground truth keywords. Each entry has the format $x@y$ to mean that a given method was able to find y correct keywords x times, when ranking the *docs* documents having |GT| keywords in their ground truth. The total $9 + 238 + 143 + 36 + 3 = 428$ falls short of the total of 449 documents since 21 documents had no keywords. Moreover, the marginal sums do not match *docs* because we did not print the number of times methods found zero correct keywords (it suffices to subtract the marginal sums from *docs*).

Table 2 Comparative statistics on the technical corpus. Under "Input" we report the number (*docs*) of documents having |GT| keywords in the ground truth. Each data entry $x@y$ in row (|GT|, *docs*) and method-indexed column means that the corresponding method found y out of |GT| ground truth keywords in x documents

Input		Ranking method				
|GT|	*docs*	TermFreq		Graphs-of-words		
		TFIDF	BM25	1-*proximity*	*Constituency*	*Dependency*
1	9	6@1	6@1	6@1	6@1	6@1
2	238	111@1	111@1	**115@1**	**115@1**	113@1
		14@2	14@2	14@2	14@2	**15@2**
3	143	41@1	41@1	41@1	**43@1**	42@1
		76@2	76@2	**77@2**	76@2	**77@2**
		1@3	1@3	1@3	1@3	1@3
4	36	17@1	17@1	17@1	17@1	17@1
		6@2	6@2	6@2	6@2	6@2
		0@3	0@3	0@3	**1@3**	0@3
		1@4	**1@4**	**1@4**	0@4	**1@4**
5	3	3@1	3@1	3@1	3@1	3@1
Total	*428*	*276*	*276*	*281*	***282***	*281*

In this experiment we find that gows are more effective at keyword extraction than term frequency. Constituency tree based gows are marginally better than other gows. We also note, again, that the two term frequency based methods attain equal performance levels.

4 Conclusion

We looked at graphs-of-words constructed using syntax trees, and their performance in extracting keywords from text. There is no clear dominance of term frequency versus graph-of-words rankinds. Graph-of-words scored better with short ungrammatical sentences, term frequency in literary texts. In the future, we may apply this technique to structures such as "knowledge graphs", which can be obtained by mapping the words in the text into structured knowledge sources.

References

1. Chomsky, N.: Syntactic Structures. Mouton, The Hague (1956)
2. Eco, U.: Semiotics and the Philosophy of Language. Indiana University Press, Bloomington, IN (1984)
3. Franciscus, N., Ren, X., Stantic, B.: Dependency graph for short text extraction and summarization. J. Inf. Telecommun. **3**(4), 413–429 (2019)
4. Hagberg, A., Schult, D., Swart, P.: Exploring network structure, dynamics, and function using NetworkX. In: Varoquaux, G., Vaught, T., Millman, J. (eds.) Proceedings of the 7th Python in Science Conference (SciPy2008), pp. 11–15. Pasadena, CA (2008)
5. Honnibal, M., Montani, I.: Industrial-Strength Natural Language Processing in Python. spaCy (2023)
6. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, MA (1979)
7. Johansson, R., Nugues, P.: Extended Constituent-to-Dependency conversion for English. In: Proceedings of the 16th Nordic Conference of Computational Linguistics, NODALIDA, pp. 105–112 (2007)
8. Kitaev, N., Klein, D.: Constituency parsing with a self-attentive encoder. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, vol. 1 (Long Papers), pp. 2676–2686. ACL (2018)
9. Klein, D., Manning, C.: Accurate unlexicalized parsing. In: Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423–430 (2003)
10. Levine, R., Mason, T., Brown, D.: Lex and Yacc, 2nd edn. O'Reilly, Cambridge (1995)
11. Meyer, F.: English Corpus Linguistics. CUP, Cambridge (2004)
12. Mihalcea, R.: Graph-based ranking algorithms for sentence extraction, applied to text summarization. In: Proceedings of the 42nd Annual Meeting of the Association for Computational Lingusitics, volume Companion Volume of *ACL* (2004)
13. Mihalcea, R., Tarau, P.: Textrank: bringing order into text. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 404–411 (2004)
14. Pérez-Agüera, J., Arroyo, J., Greenberg, J., Perez Iglesias, J., Fresno, V.: Using BM25F for semantic search. In: Proceedings of 3rd International Semantic Search Workshop, pp. 1–8 (2010)

15. Roelleke, T., Wang, J.: TF-IDF uncovered: a study of theories and probabilities. In: Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 435–442 (2008)
16. Rousseau, F.: Graph-of-words: Mining and retrieving text with networks of features. Ph.D. thesis, LIX, Ecole Polytechnique, France (2015)
17. Rousseau, F., Vazirgiannis, M.: Graph-of-word and TW-IDF: new approach to ad hoc IR. In: Proceedings of CIKM, New York, 2013. ACM
18. Xia, F., Palmer, M.: Converting dependency structures to phrase structures. In: Allan, J. (ed.) Proceedings of the First International Conference on Human Language Technology Research, HLT, San Francisco, 2001. Morgan Kaufman