# Grafeno: Semantic Graph Extraction and Operation

Antonio F. G. Sevilla*, Alberto Fernández-Isabel† and Alberto Díaz‡
Department of Software Engineering and Artificial Intelligence
Universidad Complutense de Madrid, Madrid, Spain
Email: *afgs@ucm.es, †afernandezisabel@ucm.es, ‡albertodiaz@fdi.ucm.es

***Abstract*—Grafeno is a Natural Language Processing library for doing semantics. It represents semantic information with a graph structure, and is able to automatically extract this representation from the dependency analysis of a text. It aims to encompass the different possible approaches to doing graph semantics by being as modular and flexible as possible. It also provides functionality for operating on the graph and performing different experiments. In this article, we explain its design and use, and show its potential with two use cases.**

***Index Terms*—Natural Language Processing, Semantics, Concept Graphs, Information Extraction**

## I. Introduction

When dealing computationally with text written in any human language, one task still unresolved is to be able to extract a formal account of its meaning, the significance of it to a human reader. A possible approach is to use surface features, like token occurrence or n-grams, for its classification and management. However, the true meaning of the text lies in its deep analysis, beyond its surface form and even beyond the syntactic analysis.

While the issue of how to represent semantics is not fully resolved yet, one promising paradigm is that of concept graphs or networks [1]. Graphs are mathematical structures consistent of arbitrary objects, in this case concepts, and links between them, in this case the linguistic relation between concepts. The linked and network-like structure of graphs is useful for representing the complex and referential structure of semantic information.

*Grafeno*[1] is an under-development python library for the automatic extraction of these graphs from raw text. It is designed to be modular, so that different formalisms and theories can be accommodated. It relies on previous steps of linguistic analysis from external tools, and it is centered on the creation of the semantic graphs from their results.

In this document, we present the library, its general operation, and give two examples of real use cases that currently rely on it. To aid with the dissemination of *Grafeno*, the code is available as open source on *Github*[2]. Documentation is available on *Read The Docs*[3].

The document first presents some background to the semantic graph idea in section II. The actual graph design and construction is presented in section III, and its manipulation in section IV. Section V explains how to use the library in a real world setting, and presents two examples of current uses. Our conclusions are listed in section VI, and the future work delineated in section VII.

[1]grafeno means graphene in Spanish, the allotrope of carbon with a lattice-like structure.

[2]https://github.com/agarsev/grafeno

[3]http://grafeno.readthedocs.org/

## II. Related Work

Representing semantic information with a graph structure is not a new idea. [1] gives a very thorough theoretical description of semantic networks, including many complex phenomena. These networks connect concepts, the main entities ascribed to semantics, using edges and relations. The concepts and the network structure encode the meaning of utterances.

In the Functional Generative Description theory of linguistics [2], semantics are encoded in an "extended" tree, in correspondence to the dependency tree of syntax. In the most prominent implementation of the theory, the Prague Dependency Treebank (PDT) [3], these trees can have links that connect the nodes from different branches, different trees, or even with the "external" world –for example in the case of exophora. This use of additional connections in the tree makes it equivalent to a graph, and we follow (albeit somewhat loosely) the spirit of deep-grammar trees and some of the PDT conventions in the design of our semantic graphs.

However, automatically extracting these representations from natural language is not easy. In [4], the authors create the network from the nominal concepts in a text, and then use graph theory to find the most important topics in the network. These are finally used to extract an automatic summary of the original text. In [5], the graph structure is richer, created semi-automatically from a dependency parse of the text. In this work, we aim to combine the application of the method in [4] with the fuller representation in [5], and to do so automatically.

For this, existing tools and knowledge bases are required. FreeLing [6], [7] is a sophisticated open-source toolkit which provides many layers of linguistic analysis, including finding the dependency parse of a sentence.

Wordnet [8] is a lexical database of *Synsets*, counting over 140000 items. These items represent sets of lemmas[4] which share a meaning –are synonyms. But *Synsets* are also connected with each other through lexical relations such as hypernimy or hyponimy, creating a rich structure useful for lexical semantics.

[4]Used in the linguistic sense of the representative word form of a lexeme.

Apart from constructing the graph itself, it is important to have a task to perform, where measurements and comparisons can be drawn. An example is automatic summarization of text like in [4] and [5]. Another possibility is to transform our semantic graphs into a conceptual map, a very similar representation which can then be used for creative conceptual blending, like is done in [9]. But even if similar, these concept maps present important differences with other kind of semantic networks. Being able to accommodate and process these differences is one of the driving factors for collecting the code into a library.

## III. Building the Semantic Graph

The main goal of *Grafeno* is to automatically extract the semantic representation of a text in graph form. In this section this process is described. First, the design of the graph is explained in section III-A. Then, an overview of the creation process is given in section III-B. An important part of this are the *transformer* modules, explained in section III-C, of which some of them are presented in section III-D.

### A. Design

The main construct of the *Grafeno* library is the semantic graph. Mathematically, a graph is a pair formed by a set of nodes, in this case representing concepts, and the edges between them. Formally, $G = \langle N, E \rangle$, where $E \subset N \times N$. Concepts are defined axiomatically, symbolically, just an arbitrary string, but in reality they are intended to evoke in the human reader the appropriate idea. These concept names can come from the word lemmas in the text, or different stages in the processing can modify or create them. Ultimately, they are just labels, arbitrary identifiers.

Apart from the concept name or representation, nodes can have further attributes and features, which help extend the information the graph has on the particular concept. A prominent example of these possible features is the WordNet synset object [8], obtained at one point during processing and stored in the node.

The other component of the graph is the set of edges. These edges are directed, so they have a source or parent node, and a target or child node. The relation between concepts that these edges represent is encoded under the label “functor”, a string of free-form text attached to the edge.

While *Grafeno* does not enforce any restriction on the functor names, many parts of the library use a common convention for them, loosely based on existent linguistic and semantic practices. There are deep grammar functors, such as “AGENT” or “THEME”, lexical ones such as “HYP” (for hypernym, a “is a” relation) or discourse ones such as “SEQ” (sequence, for linear ordering of sentences). And alongside the “functor” label, edges can store additional attributes, as nodes do. This allows the storage of structured or complex semantic information in the graph edges. For example, an “ATTR” relation that describes an attributive relation between two concepts (often nominal and adjectival ones) can further

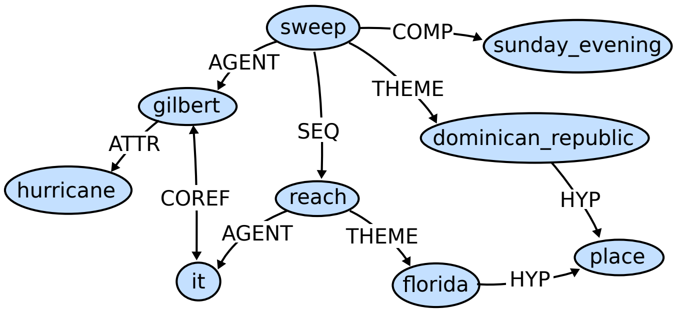

Fig. 1. An illustration of the semantic graph for the pair of sentences “Hurricane Gilbert swept the Dominican Republic Sunday evening. It then reached Florida.”

specify that the property being described is that of color, or size.

Figure 1 shows a fully processed semantic graph, including some of the mentioned relations. This graph represents the meaning behind the utterance “Hurricane Gilbert swept the Dominican Republic Sunday evening. It then reached Florida.” Parts of the graph are directly extracted from the syntactic analysis of the sentences, some are added from external knowledge bases (like “HYP”) and some are based on heuristics, like “COREF”, for co-reference detection.

### B. Graph creation

For the creation of the concept graph, the necessary input is raw text. If the original documents have markup or other formatting, the text must first be pre-processed and cleaned by the user. For now, the library only understands English, but it is designed to be (surface) language-independent. Of course, how language independent can semantics be is a matter of discussion, and in the particular case of *Grafeno* it will completely depend on the specificity of the modules and rules used.

The clean text in a suitable language is then run through a linguistic analyzer pipeline, which morphologically analyzes the words and extracts a dependency tree from each sentence. In the current version, FreeLing [7] is used for the dependency analysis, but other tools can be used for this stage of processing. FreeLing is a fast and reliable library, with configurable pipelines for many different languages. However, also in the works are modules that allow *Grafeno* to use other syntactic analyzers (e.g., MaltParser [10]).

After the linguistic tool-chain has finished, *Grafeno*’s own processing starts. First, the morphologically analyzed words are transformed into semantic nodes, with the relevant “concept” identified. Then, dependency relations are processed, some of them being transformed into semantic edges with a corresponding “functor”. Some words might not turn into any semantic node (mainly function words), and not every dependency relation has its equivalent semantic one.

When the nodes and edges have been created, further processing can be done. Some nodes can be merged, edges deleted, or even new ones created. What specific operations

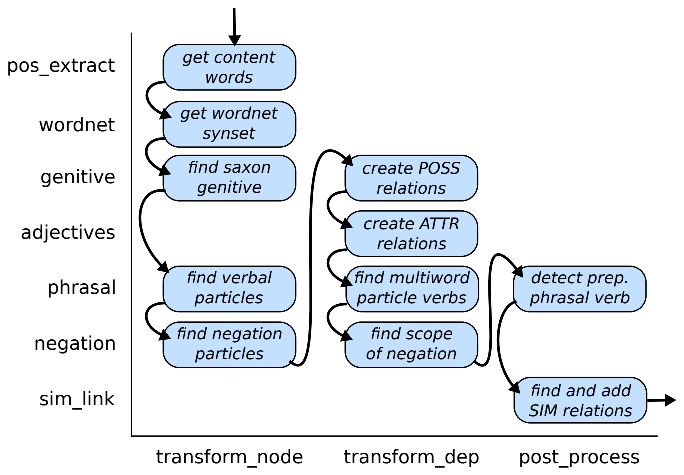

Fig. 2. Execution flow between modules and processing steps in a composite Transformer. Boxes in the same row belong to the same module.

are done in each of the steps is defined by the pipeline of modules chosen by the user. Each module can do work in one or more of the steps, cooperating with the others to create the final semantic nodes and edges. The final result is a function of the input text, what modules have been used and in what order.

### *C. Transformers*

These modules that create the graph step by step are called "transformers" in the *Grafeno* library. Their function is to *transform* (hence the name) a dependency tree into a semantic graph. There are currently twenty seven transformers, some performing general tasks, some performing very specific ones. They are designed as an incremental procedure, in which later steps can rely on information obtained by previous ones. This modularity happens along two dimensions.

The first dimension is the already mentioned sequence of phases, in which first nodes and then edges are created. But there also is the second dimension created by the composability of the transformers. Any number of transformers can be tied together in some order in a transformation pipeline. For each of the steps, all the different transformer modules are called in sequence. And just as the dependency transformation step can rely on information extracted by the node transformation step, any module can rely on information already processed by a previous module.

### *D. Example modules*

In Figure 2 a diagram of the execution flow in an example composite transformer (based on the one used for the use case in section V-B) can be seen. In the horizontal dimension the different phases of processing advance from left to right, and the modules are composed in a pipeline from top to bottom. Note how the flow passes through some modules more than once. Some interesting modules are explained in this section, and others just enumerated.

*a) `pos_extract`:* This is a general module which can start very different transformation pipelines, the name coming from Part-of-speech extraction. It takes some words in the dependency tree and transforms them into semantic nodes, using the lemma as the concept label. Which words are extracted is defined by their *pos* tag. By default, only content words are used (nouns, verbs, adjectives and adverbs). However, this is configurable, and other parts of speech can also be defined as semantically relevant. Apart from the "concept" label, the `pos_extract` module also adds the `sempos` (semantic equivalent to part-of-speech) attribute, useful for later processing stages to distinguish verbs from nouns, for example.

*b) `phrasal`:* This is a complex and specific module, which tries to deal with the phenomenon of phrasal verbs in English. Its processing takes place in all of the three main steps: first, it has to detect any particles that accompany the verb. Later, when the auxiliary dependency is found between verb and particle, the correct concept is determined. However, the syntactic parser does not always produce the desirable analysis, for example with prepositional phrasal verbs. Using the extra available semantic information, heuristics are implemented to promote an apparent prepositional complement into the actual theme of the verb, and analyze the verb+preposition combination as a multi-word concept.

*c) `sim_link`:* This module adds a new type of relations to the graph. It works after the nodes and edges have been processed, and after the final ones have been added to the graph. It then tries to relate the new nodes to the already existing ones, using lexical similarity measures [11]. When it finds two concepts with a similarity above a configurable threshold, it adds a new edge between them, with functor `SIM` for similarity. These extra paths add a new way to navigate the graph, and link similar concepts together from across different parts of the structure.

*d) some other transformers:* The `wordnet` module adds to concept nodes their `SynSet` object. The `genitive` module finds and adds genitive semantic relations, searching for the saxon genitive and specific prepositional phrases. `adjectives` processes these words and relates them to the modified noun, and the `negation` module finds the scope of the negative particles up the dependency tree, and modifies the affected concepts appropriately.

## IV. Processing the Graph

In the previous section the construction of the graph has been explained, both its general design and particular transformations that extract it from the dependency analysis. Finding this representation of the meaning of a text might be enough in some cases, but often further processing will be needed. *Grafeno* tries to help with this, and offers some operations useful for the processing of the graph.

The implementation of the concept graphs in *Grafeno* is but a thin wrapper on top of the graph object in the python library NetworkX [12]. These graphs are implemented sparsely, as a dictionary of dictionaries, with convenient properties and methods. Since the underlying graph structure can be directly accessed and worked with, many graph-theoretical operations can be performed with little to no adaptation.

Among the currently implemented operations in *Grafeno* there are those taken straight from graph theory, like a few clustering methods; those in charge of plumbing and general processing, like edge-filtering and domain subgraph, and some based more on semantics, like generalization. A final procedure is that of linearizing the graph into a sequential structure.

*1) Clustering:* Finding clusters in a semantic graph can be useful, as they serve to identify themes and topics of the text, an important part of the discourse structure. *Grafeno* provides three different possibilities. The first one is a degree-based method [13], which uses connectivity to find centroids (nodes with the highest salience) and their neighbourhoods. A second option is Markov clustering (MCL) [14], also based on connectivity. It simulates random walks on the graph to find connected neighbourhoods. Finally, the Louvain community detection method [15] uses heuristics to efficiently optimize the modularity of the clustering arrangement. This modularity is defined as a value between -1 and 1 that measures the density of links inside communities as compared to links between communities.

*2) Filtering:* Two different filtering modules have been developed. The first one takes into account functor information for removing or renaming semantic edges. A second more sophisticated filter tries to extract the domain subgraph: the set of nodes and edges which represent the most relevant information about the central topic of the text. Different heuristics are under research, but for now the main topic is defined as the concept with highest salience in the semantic graph, and the relevant information is the connected subgraph that it spawns.

*3) Semantic-based:* Based on the work in [5] the `generalization` operation has been implemented. It is a bit different in that it takes as input two different concept graphs and produces a new one, instead of processing only one input graph. It works by finding a generalization of the nodes in equivalent semantic relations. For example, if the input graphs correspond to the sentences *"A man picked up an apple."* and *"The woman lifted some pears."* the end result will be *"Adults raise edible fruits."*. It uses WordNet for finding the generalization[5] of the concepts involved, and it has the potential to be used in both text simplification and automatic summary generation.

*4) Linearization:* A different procedure is that of linearization: the process of converting the complex, recurrent and possibly cyclical structure of the graph into a linear sequence, usually of characters or bytes. This is useful for human consumption, but also for communication between different tools with different formats for input and output. It can also be a way to generate natural language from a semantic structure.

There are currently seven linearizer modules in the library. The simplest linearizer outputs all the concepts found in

[5]Wordnet provides a hierarchical hypernym structure, where it is possible to find the lowest common hypernym of two concepts.

```
%YAML 1.2
---
# Summarizes a text by extracting the most relevant
 ↪  sentences.
transformers:
    - pos_extract
    - unique
    - extend
    - sim_link
    - sentences
transformer_args:
    sempos:
        noun: n
    extended_sentence_edges: [ HYP ]
operations:
    - op: cluster
      hubratio: 0.2
linearizers:
    - cluster_extract
linearizer_args:
    summary_length: 100
    summary_margin: 10
    normalize_sentence_scores: True
```

Listing 1: The summarization pipeline

the graph. Another one generates a text by extracting sentences from the original text. A third one outputs the graph structure as a sequence of triples, expressions of the form `relation(node1, node2)` which represent all edges and non-isolated nodes. An experimental linearizer module has been started, which walks the semantic structure of the graph, generating natural language along the way.

The remaining modules are specializations of the ones mentioned, since the architecture of the linearizers is equivalent to that of the transformers, where different modules can be composed and each takes charge of different steps in the process. Basic modules are in charge of extractive generation, abstractive text production, or node and edge mapping, and more specific modules direct the process by selecting the appropriate substructure, ordering it, and generating the textual representation.

## V. Case Studies

As we have seen, *Grafeno* offers a variety of possible operations and modules. In previous sections an overview of different steps of processing has been given, but to better understand the use of the library, in this section two use cases are presented, proofs of concept that outline the operation and results of the library.

*Grafeno* comes with an interpreter which can read configuration scripts and execute them. These scripts can describe full pipelines of experiments, including all the modules and parameters used for extracting the semantic graph, and any further operation to be applied. The scripts are written as configuration files in `YAML`[6], helping with reproducibility and fast development of new experiments. These files also serve as pre-built configurations that can be use without or with little modification.

For example, a user who does not want to experiment, but rather utilize *Grafeno* for extracting summaries, can directly load the pipeline presented in listing 1, and explained in

[6]http://yaml.org/

```
%YAML 1.2
---
# Extracts a concept map from a text.
transformers: [ pos_extract, wordnet, adjectives,
 ↪  negation, genitive, prep_rise, unique,
 ↪  attr_class, verb_collapse, specific_edges ]
transformer_args:
    sempos:
        noun: n
        adjective: j
    attach_adjectives: True
    keep_attached_adj: True
operations:
    - op: filter_edges
      remove: [ isa ]
      rename: { be: is }
      frequency: { max: 15, min: 0 }
    - op: domain_subgraph
linearizers:
    - prolog
```

Listing 2: The concept map pipeline

section V-A. In section V-B, the pipeline in listing 2 is explained, showing how it can be used to extract a concept map from the original text.

## A. Extractive summarization

One of the seminal ideas for the development of this library was doing text summarization based on concept graphs. The approaches in [4], [5] lay the ground work for doing this, the first one using an extractive approach, and the latter focusing on a more semantic description of the text, leaving as an open question the issue of the actual summary generation.

With *Grafeno*, the ideas of [4] can be reproduced, but the graphs can be enriched with more information, like that of [5]. The full experiment consists of a series of steps which can be seen in listing 1.

The first step is that of extracting the graph. It is configured in the values `transformers` and `transformer_args`, specifying the modules and parameters which will be used. In this case, nouns will be extracted (module `pos_extract` and parameter `sempos`). Module `unique` ensures that there is only one node per concept, even if it appears several times in the text. `extend` adds WordNet information in the form of the hypernym structure, and `sim_link` relates the concepts according to lexical similarity measures [11]. `sentences` records which nodes represent each sentence, which is used when generating the final summary. After creating the graph, a `clustering` operation is performed. Finally, the `linearizer` used is `cluster_extract`, an extractive summary generator which selects the sentences most aligned with the main topic (cluster) of the original text.

The pipeline described fully reproduces the experiments in [4]. However, changing the modules and parameters in the configuration file allows the experiment to be repeated on the same data but with slightly different procedures. [16] describes and evaluates some of these experiments. For example, using also verbs as concept nodes was tried (the original approach uses only nouns), and using richer links or not between the edges.

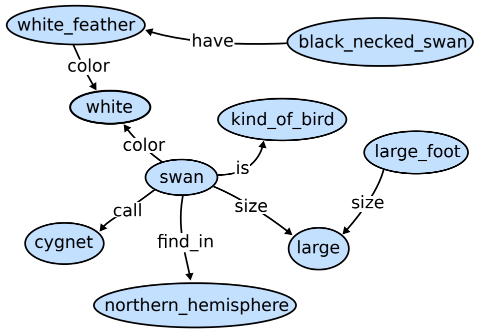

Fig. 3. A concept map extracted from the Simple Wikipedia entry for “Swan”. Note that while `large_foot -size-> large` might seem obvious to the human reader, to the computer these are just arbitrary identifiers, so the information needs to be encoded.

## B. Concept map creation

A completely different experiment, shown as a configuration file in listing 2, is that of concept map creation. Within the ConCreTe ERC project[7], as a collaboration with the University of Coimbra we are automatizing the extraction of concept maps from text. This is part of the Conceptual Blending workflow [17], where two different input domains are blended into a creative new concept domain. However, the input domains are yet to be fully automatically extracted from text, presenting an opportunity.

Concept domains are also semantic networks, but what constitutes a node and an edge is slightly different. Nodes comprise only nominal concepts, entities either concrete or abstract, and the edges between them represent the linguistic relations between them.

In listing 2 the configuration for *Grafeno* to create these concept maps can be seen. For the Simple Wikipedia entry for “Swan”[8], the resulting concept map can be seen in figure 3.

For this experiment, a more complex series of transformers is needed. More modules are used, in order to interpret the thematic relations in the dependency analysis of the original text. Subtler nuances of meaning, such as negation and prepositional phrases, also need to be processed. And after the graph is extracted, some utility modules are used. `verb_collapse` adapts the graph structure to that required for concept maps, collapsing structures of the type `Entity1 <-AGENT- Verb -THEME-> Entity2` into `Entity1 -Verb-> Entity2`. `specific_edges` renames general functors such as “ATTR” or “COMP” with the actual properties (such as color, or size), or complements (in, above, behind...) being described. The last step is to apply some filters which remove noise from the graph and retain only the domain subgraph. In this case the graph is optionally linearized into a set of *Prolog* predicates, of the form `Relation(Node1,Node2)`, for further consumption.

[7]http://conceptcreationtechnology.eu/
[8]https://simple.wikipedia.org/wiki/Swan

## VI. Conclusions

In this document, we have described *Grafeno*, a general-purpose semantic graph library. The library is formed by a number of modules, which can be combined either in code or in a scripted configuration file to build a complete experimentation pipeline. The main functionality of the library is to automatically extract a representation of the meaning of text, and has components to tackle thematic, discourse and lexical semantics phenomena.

These modules are used to extract the different parts of text meaning from its previous syntactic dependency analysis. This meaning is encoded as conceptual nodes, roughly the content words, with their representing "concept" and other relevant features attached. The nodes are connected by edges, labelled with a "functor", the semantic equivalent of syntactic functions. The method for concept and functor definition and extraction is flexible, and can accommodate very different use cases and needs.

On one hand, it can be used for exploring different semantic designs, allowing the researcher to implement an automatic procedure for extracting these designs from text. An example semantic graph has been shown, an actual result of the library from real English text.

On the other hand, *Grafeno* can be used as a complete tool that performs full experiments, from the extraction and analysis of the meaning, to its further processing and output. Two different use cases have been explained, real examples with very different goals, which demonstrate the potential of the library for the development of complete and useful semantic experiments.

## VII. Future Work

Many future and current lines of work in our group involve semantic graphs, and this library can serve as base and tooling for them. Therefore, a clear line of future work resides on finding and developing new experiments and pipelines in which to apply the work in *Grafeno*. Collaborations with other groups who might need such a semantic tool have been started and look to be quite promising.

While that is being done, there is still work that can be done on the internals themselves. Some of it is work which the current linguistic practice already knows how to deal with, and some is unexplored, ready for experimentation and new discoveries.

In the graph creation step, semantic relations close to syntax can always be fine-tuned and made more precise. However, the real interesting part lies in discourse analysis: improving co-reference and anaphora resolution, and description of the discourse structure. We believe the graph structure is ideal for this task.

In the operations step, more ways to cluster the graph and analyze its substructure are under research, and combining these with other graph-theoretical methods might lead to improvements on the summarization experiments.

And as a final step, the linearization of the graph is a very interesting problem to tackle. Improving on our extractive approaches, we would like to take our summarization pipeline to the next level: generating abstractive summaries. This requires solid natural language generation machinery, which gives wide opportunity for research and development.

## Acknowledgments

This work is funded by ConCreTe. The project ConCreTe acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET grant number 611733.

This research is also funded by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund TIN2015-66655-R (MINECO/FEDER).

## References

[1] J. F. Sowa, "Conceptual structures: information processing in mind and machine," 1983.

[2] P. Sgall, E. Hajičová, and J. Panevová, *The meaning of the sentence in its semantic and pragmatic aspects*. Springer Science & Business Media, 1986.

[3] E. Bejček *et al.*, "Prague dependency treebank 3.0," Prague, Czech republic, 2013.

[4] L. P. Morales, A. D. Esteban, and P. Gervás, "Concept-graph based biomedical automatic summarization using ontologies," in *Proceedings of the 3rd textgraphs workshop on graph-based algorithms for natural language processing*. Association for Computational Linguistics, 2008, pp. 53–56.

[5] S. Miranda, A. Gelbukh, and G. Sidorov, "Generación de resúmenes por medio de síntesis de grafos conceptuales," *Revista signos*, vol. 47, no. 86, pp. 463–485, 2014.

[6] X. Carreras, I. Chao, L. Padró, and M. Padró, "Freeling: An open-source suite of language analyzers," in *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*, 2004.

[7] L. Padró and E. Stanilovsky, "Freeling 3.0: Towards wider multilinguality," in *Proceedings of the Language Resources and Evaluation Conference (LREC 2012)*. Istanbul, Turkey: ELRA, May 2012.

[8] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[9] F. C. Pereira and A. Cardoso, "Experiments with free concept generation in divago," *Knowledge-Based Systems*, vol. 19, no. 7, pp. 459–470, 2006.

[10] J. Nivre *et al.*, "Maltparser: A language-independent system for data-driven dependency parsing," *Natural Language Engineering*, vol. 13, no. 02, pp. 95–135, 2007.

[11] J. Jiang and D. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," in *Proceedings on International Conference on Research in Computational Linguistics*, 1997, pp. pp. 19–33.

[12] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, Aug. 2008, pp. 11–15.

[13] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, pp. 457–479, 2004.

[14] J. Vlasblom and S. J. Wodak, "Markov clustering versus affinity propagation for the partitioning of protein interaction graphs," *BMC bioinformatics*, vol. 10, no. 1, p. 1, 2009.

[15] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[16] A. F. G. Sevilla, A. Fernández-Isabel, and A. Díaz, "Enriched semantic graphs for extractive text summarization," in *The Conf. of the Spanish Association for AI (CAEPIA 2016), LNAI*, vol. 9868, 2016.

[17] M. Žnidaršič *et al.*, "Computational creativity infrastructure for online software composition: A conceptual blending use case," in *Proceedings of the Seventh International Conference on Computational Creativity*, 2016, to be published.