Aaron Schlutter, Andreas Vogelsang

Knowledge Extraction from Natural Language Requirements into a Semantic Relation Graph

Knowledge Extraction from Natural Language Requirements into a Semantic Relation Graph

Aaron Schlutter
Andreas Vogelsang
aaron.schlutter@tu-berlin.de
andreas.vogelsang@tu-berlin.de
Technische Universität Berlin
Berlin, Germany

ABSTRACT

Knowledge extraction and representation aims to identify information and to transform it into a machine-readable format. Knowledge representations support Information Retrieval tasks such as searching for single statements, documents, or metadata. Requirements specifications of complex systems such as automotive software systems are usually divided into different subsystem specifications. Nevertheless, there are semantic relations between individual documents of the separated subsystems, which have to be considered in further processes (e.g. dependencies). If requirements engineers or other developers are not aware of these relations, this can lead to inconsistencies or malfunctions of the overall system. Therefore, there is a strong need for tool support in order to detects semantic relations in a set of large natural language requirements specifications. In this work we present a knowledge extraction approach based on an explicit knowledge representation of the content of natural language requirements as a semantic relation graph. Our approach is fully automated and includes an NLP pipeline to transform unrestricted natural language requirements into a graph. We split the natural language into different parts and relate them to each other based on their semantic relation. In addition to semantic relations, other relationships can also be included in the graph. We envision to use a semantic search algorithm like spreading activation to allow users to search different semantic relations in the graph.

CCS CONCEPTS

• **Information systems** → **Document representation**; **Information extraction**; *Language models.*

KEYWORDS

knowledge extraction, requirement engineering, natural language processing, semantic relation graph, spreading activation

ACM Reference Format:
Aaron Schlutter and Andreas Vogelsang. 2020. Knowledge Extraction from Natural Language Requirements into a Semantic Relation Graph. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3387940.3392162

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3392162

1 INTRODUCTION

Complex systems are often developed by distributed teams or even across several companies (e.g., suppliers) that deliver subsystems, which are finally integrated into a product. One part of Requirements Engineering (RE) is to specify the requirements for the subsystems, i.e., write them down for further system development process. Due to the heterogeneity of the subsystems and their stakeholders, requirements are often spread across several specifications, which contain a set of natural language requirements [8]. A common approach to handle the amount of requirements specifications is to organize them in simple structures like documents, paragraphs, and folders. Since specifications are mostly written in natural language, they contain a lot of semantic content that is not formally stated but important for many engineering tasks.

Such informally noted information may be needed by different stakeholders, e.g., they need to know, how two subsystems interact with each other or if there are other specifications that relate to their current work-in-progress requirement. Several studies have shown that in such distributed development contexts, there is a high chance of developers being unaware of dependencies and important information from related subsystems [6, 27]. The corresponding task that may support developers is called Information Retrieval (IR). There are various information needs in RE, where typically a requirements engineer formulates a search query and a system suggests relevant results (called targets). A generic setup for such IR systems is to extract knowledge from the requirements, represent it in a knowledge base and provide a search algorithm to execute a query and show the results.

In order to obtain a comprehensive overview of the requirements, the statements of the semantic content must be related to each other. State-of-the-art Information Retrieval approaches use algebraic IR models (e.g., vector space models, Latent Semantic Indexing) or probabilistic models (e.g., Latent Dirichlet Allocation) [2]. More recently, machine-learning approaches have also been applied successfully [21]. While most approaches only consider single words or small parts of natural language, they rely on the distributional hypothesis [25] to compare semantic similarity and do not recognize semantic statements with their relations. Another disadvantage of such models is, that their knowledge base is often specific for certain kind of search queries and therefore not or hardly reusable for other IR tasks.

We follow a different approach on an explicit model of the knowledge represented in unrestricted NL requirements. We use a pipeline to translate NL requirements automatically into a semantic relation graph that encodes the terms as vertices and edges. We plan to use the graph with a spreading activation algorithm as a basis for various semantic searches (e.g. full text search, trace link recovery).

For analysis, we applied the pipeline to 7 datasets from different domains, with different sizes, and compared two of them with simpler knowledge graphs that we extracted in previous work. Compared to our previous study, we show, that we have fixed some shortcomings by adding more semantic content and considering more semantic relations.

2 BACKGROUND

2.1 Natural Language Requirements

Requirements specifications are often stated through unstructured natural language (NL) because it is equally available to all stakeholders. While there is usually no consistently applied pattern or template, requirements are expressed via technical terminology which is characterized by a reduced choice of words and the use of domain specific terms [9].

NL requirements specifications are typically arranged in documents as a hierarchy structure. A document consists of multi-level paragraphs, each having a headline and specifications. To distinguish each specification from each other, they have their own identifier. In practice, complex systems are often developed in terms of several loosely coupled subsystems, each of them stated in a single document. Requirements are spread over several specifications, so that a user can only understand them as a whole through the context.

2.2 Knowledge Representations

Knowledge representation focuses on the depiction of information that enables computers to solve complex problems. Borgida et al. [3] already noted in 1985 that knowledge representation is the basis for requirements engineering.

Dermeval et al. [8] report on the use of ontologies in requirements engineering in their systematic literature review. They reviewed 67 publications from academic and industrial application contexts dealing with different types of requirements. While only 34% reused existing ontologies, most of them specified their own ontology. The largest number of publications rely on textual requirements as the RE modeling style, especially in the specification phase.

Robeer et al. [24] automatically derive conceptual models from user stories. The models enable discussion between stakeholders and show promising accuracy results (precision and recall between 80 and 92%). They use heuristics to analyze the user stories due to semi-structured natural language.

In an earlier study [26], we presented an NLP pipeline that extracts knowledge from requirement documents and transforms it into a graph representing RDF[1] triples. We applied the approach to 2 datasets, an academic and an industrial one, and used the graph of the academic requirements specification to show the separation of two subsystems. The generated sample graphs were not well connected and yielded only a subset of fully connected vertices in a main graph.

3 APPROACH: GRAPH CONSTRUCTION

We use several techniques to extract information from the requirements and to build the semantic relation graph. Of course, a graph is not capable to store every aspect of any kind of information. Our graph does not meet the conditions of a valid ontology nor an RDF graph, but it tries to depict major, semantic parts of common NL (e.g., words and phrases within sentences, but also documents and corpora) and their semantic relation to each other. An ontology or an RDF graph would require at least some kind of typing of information, which is not always fully automated achievable from our point of view.

The main goal while building the graph is to store all available pieces of information in vertices as small as possible and connect those vertices with each other based on their relation. First, we use an NLP pipeline for the semantic content of the requirements. Second, we analyze the requirements for given structural characteristics which should be added to the graph, too. Finally, we combine all information to build the semantic graph as the knowledge base.

Currently, our approach only supports English, mainly because there exist a variety of different NLP techniques and tools that are not available for other languages. Furthermore, English is a relatively easy language, e.g., it consists only of a small set of part-of-speech tags, which we have to consider in the subsequent process.

3.1 Natural Language Processing Pipeline

Since we consider requirements as NL without any specific template or similar characteristics, the NLP pipeline consists of common components without special adjustments or optimizations for a certain kind of requirements specifications and is therefore able to process any kind of text. The core parts of our pipeline, Stanford CoreNLP [18] and DeepSRL [16] for semantic role labeling, are pipelines themselves and will be described in detail.

Stanford CoreNLP[2] is a collection of solutions for common NLP tasks, which are assembled and coordinated in a pipeline. We only use parts of it, particularly the tokenization, sentence splitting, part-of-speech (POS) tagging, lemmatizing (morphological analysis), dependency parser (grammatical structure), and coreference resolution (Coref). The first two tasks determine tokens (words, punctuation marks, etc.) and sentences in a NL text. Part-of-speech tagging categorizes these tokens by their grammatical role inside of a sentence, e.g., as noun, verb, or article.

Next, while lemmatizing, each word is annotated with its lemma, a base form which depends on the part-of-speech. For example, the lemma of "studying" (as verb) is "(to) study" (also a verb) and in comparison the lemma of "students" (noun) is "student" (also a noun). In contrast to this, stemming of these words would result in "stud" as the word stem. Therefore we use lemmatizing instead of stemming to keep the sense of each word.

Example 3.1. Barack Obama was born in the fabulous and tropical Hawaii, a small shallow isle in the big pacific ocean.

[1] https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[2] https://stanfordnlp.github.io/CoreNLP/

The dependency parser determines the grammatical structure of a sentence. While the result contains much more information, we use basically two features: the identification of noun phrases and the dependencies of coordinating conjunctions. In Example 3.1, amongst others, "Hawaii", "isle", and "ocean" are noun phrases (determiners and adjectives removed) and the coordinating conjunction "and" depend on the both adjectives "fabulous" and "tropical".

Example 3.2. Barack Obama was born in Hawaii. He is the president. Obama was elected in 2008.

Lastly, the coreference resolution is looking for mentions of the same entities. There are two different kinds of mentions. In Example 3.2, the word "he" refers to "Barack Obama" and is a pronominal reference without any further meaning of the word "he". In another example, "he" might refers to a totally different person. In contrast, "Obama" (in the last sentence from Example 3.2) also refers to "Barack Obama" but is a nominal reference which contains additional information, i.e., the person Barack Obama is also just called Obama. Due to the interpretation of "is" as equation, the phrase "the president" is also a coreference for "Barack Obama".

Semantic role labeling (SRL) [4] is a sentence-based NLP task. At first, all predicates of a sentence are searched. Subsequent, all arguments for each predicate are associated with their roles within this sentence. The role of an argument is represented by its type, e.g., the verb gets *V*, main arguments are enumerated by *A0*, *A1* etc. and secondary arguments are prefixed by *AM-* and their concrete function, e.g., *AM-MNR* for *manner*. In general, the first argument is called the agent, the second the patient, all other roles depend on the verb.

Example 3.3. If the user pushes the button, the engine is started.

Example 3.3 contains two predicates "push" and "start". The verb of the predicate "start" (from the main clause) is [$_{\text{V}}$started], the arguments are [$_{\text{A1}}$the engine] and [$_{\text{AM-ADV}}$If the user pushes the button]. Because this is a passive clause, there is no first argument and "the engine" is just the receiver/patient of the start procedure. Also, the whole subordinate clause is labeled as an adverbial argument, as it describes the circumstances of the predicate. The verb and arguments of "push" are [$_{\text{V}}$pushes], [$_{\text{A0}}$the user] and [$_{\text{A1}}$the button]. The numbered arguments would be the same even if the syntax of the sentence would change, e.g., "If [$_{\text{A1}}$the button] is [$_{\text{V}}$pushed] [$_{\text{A0}}$by the user], [...]".

The semantic roles are predefined in the PropBank[3] database. Most of the identified propositions have at least one argument, about two-third also got a second argument [5, Table 1]. We use DeepSRL[4] as state-of-the-art implementation for SRL tagging. It uses a deep BiLSTM model to perform SRL and achieves an F1 score of 97.4% for CoNLL 2005 [5].

3.2 Structural Information

Besides the semantics in NL, requirements specifications often contain additional information. Very common is a hierarchical structure like single documents for modules, chapters within these documents and folders arranging the documents. It can also be concluded that two documents in the same folder are more related to each other than completely foreign ones.

Trace links also express some kind of relatedness between two or more requirements which should be taken into consideration.

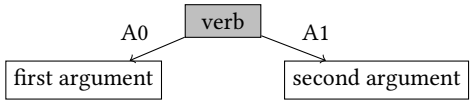


Figure 1: Graph structure for single SRL predicate

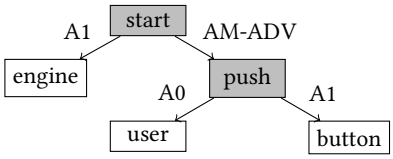


Figure 2: SRL structure for Example 3.3

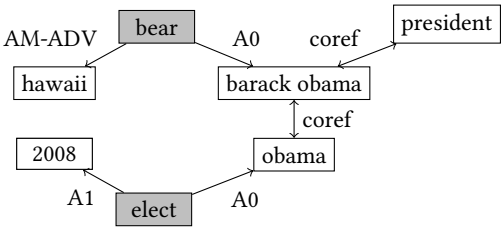


Figure 3: Graph structure for Example 3.2

3.3 Semantic Relation Graph

As last step, we build a graph which contains all information and relations, we have found in NL and the additional information. To support common graph-based algorithms like Spreading Activation, the graph must be a regular directed graph without hyperedges (more than 2 vertices connected to a single edge) nor hypervertices (a single vertex contains a graph within itself).

The leading structure for NL content is based on SRL predicates including their verb and arguments. A predicate is represented by the verb as a vertex in the graph (verb vertex) and additional vertices for each argument (argument vertex), as shown in Figure 1. If an argument contains a predicate itself, the graph structure for that predicate is likewise added and connected via an edge between both verb vertices as shown in Figure 2 for Example 3.3 instead of adding a single argument vertex containing that predicate. If there is a coreference between arguments, both argument vertices are either connected for a nominal one or only a single argument vertex is added for pronominal as shown in Figure 3.

If an argument occurs more than once, the same argument vertex is used for both occurrences. This only applies to arguments that contain at least one noun, otherwise simple adjectives (e.g., "down") would lead as single arguments to a relation which is undesirable. To support this deduplication of arguments, the information text of arguments is transformed into a simplified presentation. For example, articles are removed, nouns and adjectives are replaced by their lemma and their simplified POS tag which only differs

[3] https://verbs.colorado.edu/~mpalmer/projects/ace.html
[4] https://github.com/luheng/deep_srl

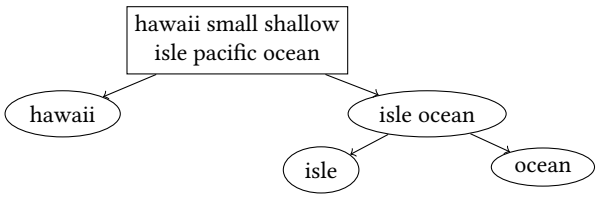


Figure 4: Graph noun phrase structure for Example 3.1

between verbs, noun, and adjectives. For instance, "the engine" is transformed into "engine#n".

As with arguments, duplicates may occur with verbs, too. Since a verb is on a par with its predicate, the arguments are also taken into account and a verb vertex is only reused, if the lemma of the verb and all argument vertices, except other verb vertices, are equal.

While the arguments in Example 3.2 and 3.3 only contain simple phrases, an argument may be a much more complex phrase. In Example 3.1, the whole phrase "in the fabulous and tropical Hawaii, a small shallow isle in the big pacific ocean" is the second argument. The resulting argument vertex will not lead to a deduplication, if the argument of another predicate is just "Hawaii". To circumvent this issue, we add additional vertices to the graph based on the given noun phrases and their dependencies as shown in Figure 4.

If an argument contains a coordinating conjunction that depends on noun phrases, the argument is split and for each part, the corresponding graph structure is added. Otherwise, a separation would lead to undesirables vertices. In Example 3.1, "and" does not depend on a noun phrase and would lead to two vertices for the splitted arguments "in the fabulous" and "tropical Hawaii, a […]".

Next to the vertices and edges for NL, the additional information have to be add to the graph, too. This depends on the characteristics of the additional information. For each requirement a vertex should be added which is connected to all verb vertices, whose predicates are contained in the requirement. A hierarchical structure may be added as a tree to the graph, e.g., including vertices for each module and chapter which are connected to the requirement vertices. Trace links can also be interpreted as edges between requirement vertices.

4 ANALYSIS: GRAPH CHARACTERISTICS

In our previous study [26] we introduced knowledge representation graphs which construction differs from the presented process. We built graphs for two requirements documents, namely Automotive System Cluster (ASC)[5] and "Charging system" (for electric vehicles) by our industry partner, both having an automotive background and containing real industrial data.

For our current approach, we reuse these datasets. Additional to the graph construction described in section 3.3, we add vertices for each requirement identifier which are connected to all predicates of all sentences in this requirement text. The graphs are shown in Figure 5. The colors in Figure 5a represent the two contained subsystems of ASC: ELC is blue, ACC green, and intersections of both are colored orange. Figure 5b has a different color coding for the single "Charging System": identifier vertices are blue, SRL predicate vertices orange, SRL argument vertices green, and noun phrase arguments are salmon.

[5]https://www.aset.tu-berlin.de/fileadmin/fg331/Docs/ASC-EN.pdf

Table 1: Datasets and corresponding graphs

Dataset	vertices	edges	no trace path
ASC	1,277	2,320	-
ASC [26]	345	440	-
Charging S.	40,453	107,862	-
Charging S. [26]	13,676	19,769	-
Infusion P.	1,731	3,622	0
CCHIT	9,239	22,669	0
GANNT	824	1,755	0
CM-1	7,616	18,375	0
WARC	1,188	1,975	63

For comparison, the graphs from [26] are shown in Figure 6. The main difference to the current process is that we used from SRL only the first two arguments of each predicate as vertices and connected them via an edge that represents the predicate. In addition, there were no vertices for identifiers or noun phrases.

Next to the 2 datasets, we analyze the tracing datasets Infusion Pump, CCHIT, GANNT, CM-1, and WARC from [15]. They have different domains, like health care, science, and business. Each of them contain source and target requirements that are linked to each other. In the graphs, source and target requirements are not directly linked (i.e. trace links are not considered, no edges are inserted).

Table 1 gives an overview of the semantic knowledge graphs for all datasets, including the number of vertices and edges. For the tracing datasets, we determine, how often there is no path between the source and target identifier vertices regardless of whether a trace link is actually defined.

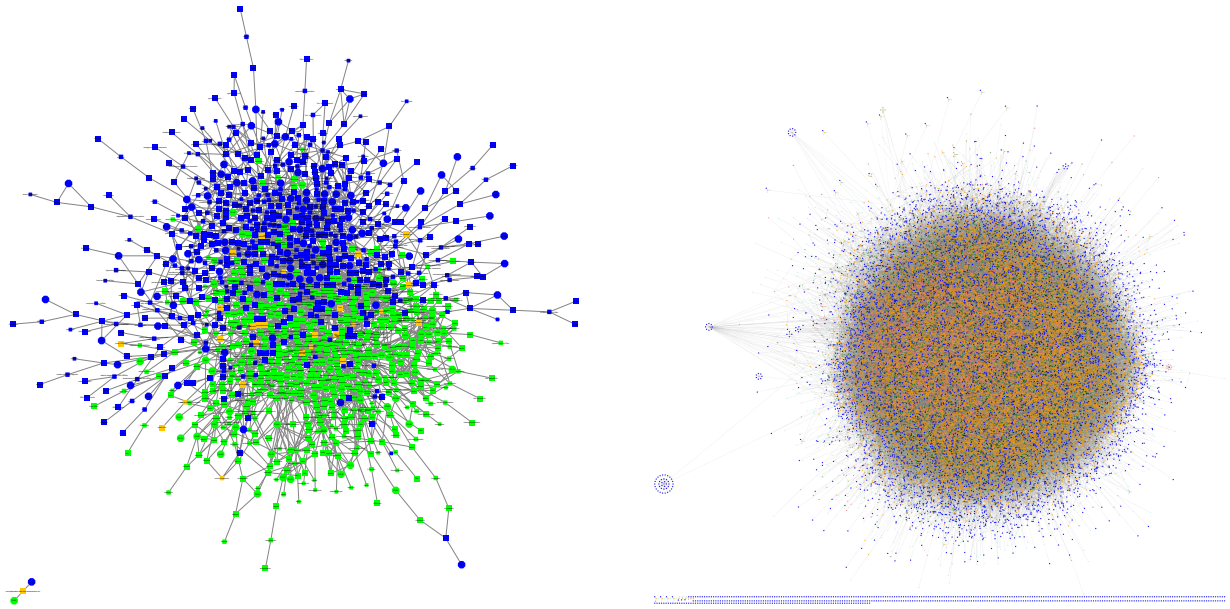
Figure 7 shows the degree centrality for each dataset, i.e., how many incoming and outgoing edges a type has. Due to some extreme outliers up to 2,130, the plot is cut off above 50 to make the boxes clearly visible. We do not differentiate between indegree and outdegree, because the direction of the edges is independent from the statement, it represents, due to the fact that in most cases, the semantics of an edge may be restated the other way around. While the datasets and graphs vary in size, the average degree centrality is nearly the same.

Figure 8 shows the distribution of existing shortest path lengths between each source and target vertex of the tracing datasets. Again, despite the different dataset and graph sizes, the average shortest distance between a source and a target is nearly the same.

5 DISCUSSION

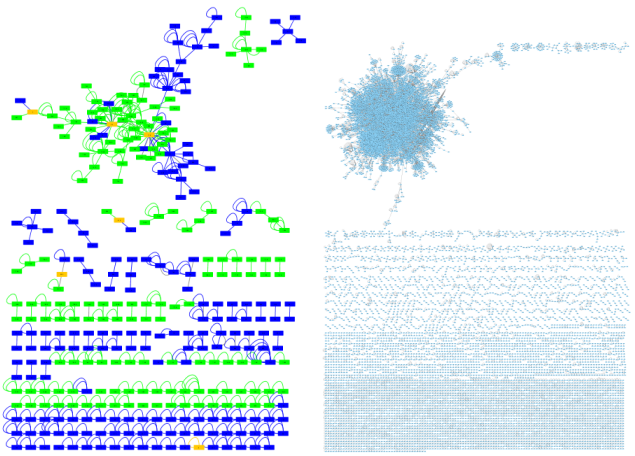
There are mainly two parts of the considered semantic search that will affect the performance results, the graph and the search algorithm, which both also depend on each other. In absence of a concrete search algorithm, we will take a look at the graph characteristics including its semantic relations and try to argue which parts might not be optimal.

The graphs provide connections between almost all possible queries and targets, i.e., there are paths between each pair of vertices. The 3 vertices in the lower left corner of Figure 5a are an



(a) Automotive System Cluster (ASC) (b) Charging System

Figure 5: Semantic relation graphs



(a) Automotive System Cluster (b) Charging System

Figure 6: Knowledge representation graphs from [26]

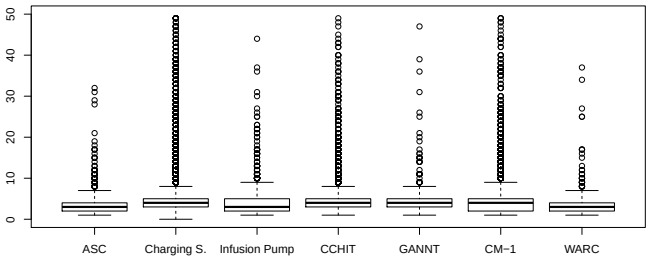


Figure 7: Degree centrality for all datasets (cut above 50, greatest outlier at 2,130)

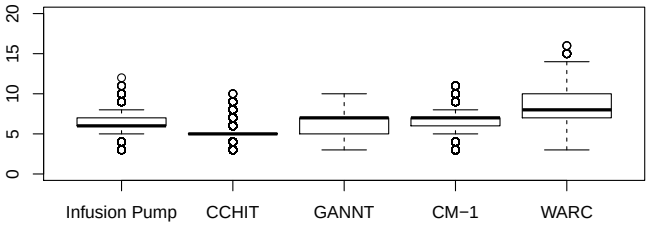


Figure 8: Shortest Path Lengths for Trace Links

exception, they represent the identifiers AL-141 and FA-55, and their content "<not elaborated within the demonstrator>". In Figure 5b at the bottom, most of the single vertices represent identifiers without any text content like a sentence or a noun phrase. Our previous approach [26] in comparison yielded only a subset of fully connected vertices in a main graph. This is also achieved, because we include more information in more graph elements (see Table 1).

Figure 7 indicate, that there are only a few strongly connected vertices while the majority is only connected to certain other, relevant information. Excluding those strongly connected nodes, the graph is able to depict a meaningful relationship path between single vertices. Figure 8 shows that the length of the shortest relationship paths stays constant, i.e., even in larger datasets such relationships are still comprehensible.

However, there is still room for improvement respectively other interpretations how information should be parted and connected to each other. For example, we do merge noun phrase vertices without considering their adjectives or coreference. But in some cases, the

adjective may crucially impact the meaning, e.g., "pacific ocean" vs. (any) "ocean" in Example 3.1.

Furthermore, we do not merge or connect semantic similar vertices. Two or more words/phrases are semantic similar, if they have the same meaning but different spelling. There are different approaches to identify such semantic similarities. A very common approach are word embeddings like *word2vec* [20] or *GloVe* [23]. They rely on the distributional hypothesis that similar words or phrases are used in similar contexts. Such word embeddings are build as mathematical vectors with many dimensions to calculate the distance between words respectively their context. This also causes words of opposite meaning to be related to each other because they occur in the same context [22]. Another approach is a database which contain known similarities, e.g., WordNet[6]. They usually contain only common similarities and are not aware of technical terms.

Also there is no identification of common phrases. We plan to use tf-idf to downgrade certain vertices which phrases are commonly used. While a graph algorithm may also downgrade such vertices (e.g. based on its edge count), it have only a local but not a global scope (based on general corpora, e.g., newspaper), where these phrases are commonly used but not in our datasets.

6 APPLICATION: SEMANTIC SEARCH

Since the graph includes structural and semantic vertices, a relationship may be found by structural or semantic artifacts. If all vertices in a graph are connected through paths, each vertex has a (depictable) relationship to all others. Different search applications are conceivable, depending on the query, relationships between artifacts (i.e. graph characteristics), and the targets.

6.1 Spreading Activation

We plan to use Spreading Activation as semantic search algorithm to find for a given query all related information (targets). We'll use this to create a list of candidates to sort all (reachable) targets for a query based on their relatedness.

Spreading Activation has its origin in the field of psychology. It is a theoretical model, how our mind connects information and tries to find an appropriate context with associated terms for a new word. The basic assumption is that more relevant terms are highly interconnected while less relevant are less or not connected at all. This model is applied to various science areas like IR [7].

Spreading Activation works on graphs and consists of three phases. In the initial phase, start vertices representing the query are activated, i.e., they will be assigned an initial activation value. While the spreading phase, this activation is stepwise distributed over the graph, i.e., the activation of a vertex is transferred to related (connected) vertices. These steps are called pulses and at the end of each pulse, a termination condition is checked to stop the pulsation. In the end phase the activation values are used to order all vertices by relevance. Depending on the type of elements searched, the results list can also be filtered, but the values are not used for classification (e.g., by a threshold), since they are not limited to an upper or lower bound and not directly comparable.

There are different ways to configure the spreading of the activation values. Berthold et al. [1] proved that restriction is necessary. Otherwise, approaches are equivalently converging to a query-independent state in which the same central vertices are displayed as result for each query.

7 RELATED WORK

Other semantic approaches often deal with semantic distance models between single parts (e.g., words or phrases). Mahmoud et al. [17] uses various semantically enabled IR methods like VSM including thesaurus or Part-of-Speech, LSI, LDA, explicit semantic analysis, and normalized Google distance. They calculate vectors for each word and try to find related documents based on small distances. Their results revealed that explicit semantic methods tend to do a better job than latent methods.

Guo et al. [10] also focus on single words, using word embeddings as semantic enhancement to train several RNN to accomplish a deep learning approach. They focus on automatically generated trace links and achieve a higher MAP than the existing approaches VSM and LSI on their unpublished dataset.

Guo et al. [11] present a technique to build an ontology semi-automatically. They focus on term mismatches between related documents. Compared to the classification technique, which performs best when sufficient training data is available, the ontology shows improvements because it aims to add semantics which enables higher levels of reasoning. The big disadvantage is the effort to create an ontology. Another benefit of the ontology is that it forms the basis for textual explanations of trace links [12].

Hartig et al. [13, 14] use Spreading Activation to find related hazard analysis and risk assessments (HARA). Initially they map a given class diagram including information such as classes, properties, instances, and data values to a Web Ontology Language[7] model. Using this model, an ontology is automatically created from existing HARA. During the search phase, they first apply Spreading Activation to the ontology to find the relevant subnetwork. In a second step, they filter the results within this subnetwork by the sought-after type sorted by their assigned activation. Finally, they generate a textual explanation for the user derived from spread graphs by Michalke et al. [19].

8 CONCLUSION

In this paper, we present a novel approach for knowledge extraction using semantic relations between parts of natural language that are stored in a knowledge graph. The approach is fully automated, does not have any prerequisites to the natural language requirements (except English language) and is scalable to various sizes of corpora. The graph and its semantic relations are independent from the syntax of the natural language and able to depict user-comprehensible paths between distant linked statements. We propose to use spreading activation as semantic search algorithm and support the user by providing an explanation for each result.

REFERENCES

[1] Michael R. Berthold, Ulrik Brandes, Tobias Kötter, Martin Mader, Uwe Nagel, and Kilian Thiel. 2009. Pure spreading activation is pointless. In *Conference*

[6] https://wordnet.princeton.edu/

[7] https://www.w3.org/TR/owl2-overview/

on information and knowledge management (CIKM). Association for Computing Machinery (ACM), Beijing, China, 1915–1918. https://doi.org/10.1145/1645953.1646264

[2] Markus Borg, Per Runeson, and Anders Ardö. 2014. Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering (EMSE)* 19, 6 (2014), 1565–1616. https://doi.org/10.1007/s10664-013-9255-y

[3] Alexander Borgida, Sol Greenspan, and John Mylopoulos. 1985. Knowledge Representation as the Basis for Requirements Specifications. , 152–169 pages. https://doi.org/10.1007/978-3-642-70840-4_13

[4] Xavier Carreras and Lluís Màrques. 2004. Introduction to the CoNLL-2004 Shared Task: Semantic Role Labeling. In *Computational Natural Language Learning (CoNLL)*. Association for Computational Linguistics (ACL), Boston, MA, USA, 89–97.

[5] Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Computational Natural Language Learning (CoNLL)*. Association for Computational Linguistics (ACL), Ann Arbor, MI, USA, 152–164.

[6] Marcelo Cataldo and James D. Herbsleb. 2011. Factors Leading to Integration Failures in Global Feature-oriented Development: An Empirical Analysis. In *International Conference on Software Engineering (ICSE)*. Association for Computing Machinery (ACM), Waikiki, Honolulu, Hawaii, 161–170. https://doi.org/10.1145/1985793.1985816

[7] Fabio Crestani. 1997. Application of Spreading Activation Techniques in Information Retrieval. *Artificial Intelligence Review* 11, 6 (1997), 453–482. https://doi.org/10.1023/A:1006569829653

[8] Diego Dermeval, Jéssyka Vilela, Ig Ibert Bittencourt, Jaelson Castro, Seiji Isotani, Patrick Brito, and Alan Silva. 2016. Applications of ontologies in requirements engineering: A systematic review of the literature. In *Requirements Engineering (RE)*. Springer, Beijing, China, 405–437. https://doi.org/10.1007/s00766-015-0222-6

[9] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. 2017. PURE: A Dataset of Public Requirements Documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, Lisbon, Portugal, 502–505. https://doi.org/10.1109/RE.2017.29

[10] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. 2017. Semantically Enhanced Software Traceability Using Deep Learning Techniques. In *International Conference on Software Engineering (ICSE)*. IEEE, Buenos Aires, Argentina, 3–14. https://doi.org/10.1109/ICSE.2017.9

[11] Jin Guo, Marek Gibiec, and Jane Cleland-Huang. 2017. Tackling the term-mismatch problem in automated trace retrieval. *Empirical Software Engineering (EMSE)* 22, 3 (2017), 1103–1142. https://doi.org/10.1007/s10664-016-9479-8

[12] Jin Guo, Natawut Monaikul, and Jane Cleland-Huang. 2015. Trace links explained: An automated approach for generating rationales. In *Requirements Engineering (RE)*. IEEE, Ottawa, Canada, 202–207. https://doi.org/10.1109/RE.2015.7320423

[13] Kerstin Hartig. 2019. *Entwicklung eines Information-Retrieval-Systems zur Unterstützung von Gefährdungs- und Risikoanalysen.* Ph.D. Dissertation. Technische Universität Berlin. https://doi.org/10.14279/depositonce-8408

[14] Kerstin Hartig and Thomas Karbe. 2016. Recommendation-Based Decision Support for Hazard Analysis and Risk Assessment. In *Conference on Information, Process, and Knowledge Management (eKNOW)*. International Academy, Research and Industry Association (IARIA), Venice, Italy, 108–111. https://doi.org/10.14279/depositonce-6974

[15] Jane Huffman Hayes, Jared Payne, and Mallory Leppelmeier. 2019. Toward Improved Artificial Intelligence in Requirements Engineering: Metadata for Tracing Datasets. In *International Requirements Engineering Conference Workshops (REW)*. IEEE, Jeju Island, South Korea, 256–262. https://doi.org/10.1109/REW.2019.00052

[16] Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep Semantic Role Labeling: What Works and What's Next. In *Association for Computational Linguistics*. Association for Computational Linguistics (ACL), Vancouver, Canada, 473–483. https://doi.org/10.18653/v1/p17-1044

[17] Anas Mahmoud and Nan Niu. 2015. On the role of semantics in automated requirements tracing. In *Requirements Engineering (RE)*. Springer, Ottawa, Canada, 281–300. https://doi.org/10.1007/s00766-013-0199-y

[18] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *System Demonstrations*. Association for Computational Linguistics (ACL), Baltimore, MD, USA, 55–60.

[19] Vanessa N. Michalke and Kerstin Hartig. 2016. Explanation Retrieval in Semantic Networks. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*. SciTePress, Porto, Portugal, 291–298. https://doi.org/10.14279/depositonce-7136

[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *International Conference on Learning Representations (ICLR)*. arXiv, Scottsdale, AZ, USA. https://arxiv.org/abs/1301.3781

[21] Chris Mills. 2017. Towards the automatic classification of traceability links. In *Automated Software Engineering (ASE)*. Urbana-Champaign, IL, USA, 1018–1021. https://doi.org/10.1109/ASE.2017.8115723

[22] Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Lina Maria Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. 2016. Counter-fitting Word Vectors to Linguistic Constraints. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics (ACL), San Diego, CA, USA, 142–148. https://doi.org/10.18653/v1/N16-1018

[23] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics (ACL), Doha, Qatar, 1532–1543. https://doi.org/10.3115/v1/d14-1162

[24] Marcel Robeer, Garm Lucassen, Jan Martijn E. M. van der Werf, Fabiano Dalpiaz, and Sjaak Brinkkemper. 2016. Automated Extraction of Conceptual Models from User Stories via NLP. In *Requirements Engineering (RE)*. IEEE, Beijing, China, 196–205. https://doi.org/10.1109/RE.2016.40

[25] Magnus Sahlgren. 2008. The distributional hypothesis. *Italian Journal of Linguistics* 20, 1 (2008), 33–54.

[26] Aaron Schlutter and Andreas Vogelsang. 2018. Knowledge Representation of Requirements Documents Using Natural Language Processing. In *Natural Language Processing for Requirements Engineering (NLP4RE) (CEUR Workshop Proceedings)*, Vol. 2075. RWTH Aachen, Utrecht, Netherlands. https://doi.org/10.14279/depositonce-7776

[27] Andreas Vogelsang. 2020. Feature dependencies in automotive software systems: Extent, awareness, and refactoring. *Journal of Systems and Software (JSS)* 160 (2020). https://doi.org/10.1016/j.jss.2019.110458