Aaron Schlutter, Andreas Vogelsang

Trace Link Recovery using Semantic Relation Graphs and Spreading Activation

Schlutter, A., & Vogelsang, A. (2020). Trace Link Recovery using Semantic Relation Graphs and Spreading Activation. 2020 IEEE 28th International Requirements Engineering Conference (RE). https://doi.org/10.1109/re48521.2020.00015

Trace Link Recovery using Semantic Relation Graphs and Spreading Activation

Aaron Schlutter
Technische Universität Berlin
Berlin, Germany
aaron.schlutter@tu-berlin.de

Andreas Vogelsang
Technische Universität Berlin
Berlin, Germany
andreas.vogelsang@tu-berlin.de

***Abstract*—Trace Link Recovery tries to identify and link related existing requirements with each other to support further engineering tasks. Existing approaches are mainly based on algebraic Information Retrieval or machine-learning. Machine-learning approaches usually demand reasonably large and labeled datasets to train. Algebraic Information Retrieval approaches like distance between tf-idf scores also work on smaller datasets without training but are limited in providing explanations for trace links. In this work, we present a Trace Link Recovery approach that is based on an explicit representation of the content of requirements as a semantic relation graph and uses Spreading Activation to answer trace queries over this graph. Our approach is fully automated including an NLP pipeline to transform unrestricted natural language requirements into a graph. We evaluate our approach on five common datasets. Depending on the selected configuration, the predictive power strongly varies. With the best tested configuration, the approach achieves a mean average precision of 40% and a Lag of 50%. Even though the predictive power of our approach does not outperform state-of-the-art approaches, we think that an explicit knowledge representation is an interesting artifact to explore in Trace Link Recovery approaches to generate explanations and refine results.**

I. Introduction

Trace Link Recovery (TLR) is a common problem in software engineering. While many engineering tasks profit from explicit links between related development artifacts [1], [2], these links are laborious to maintain manually and therefore rarely exist in projects [3]. Automatic TLR approaches aim for supporting engineers in finding related artifacts and creating trace links. Most approaches frame TLR as an Information Retrieval (IR) problem [4]. The IR approach builds upon the assumption that if engineers refer to the same aspects of the system, similar language is used across different software artifacts. Thus, tools suggest trace links based on Natural Language (NL) content [5].

State-of-the-art approaches use algebraic IR models (e.g., vector space models (VSM), Latent Semantic Indexing (LSI)), or probabilistic models (e.g., Latent Dirichlet Allocation (LDA)) [5]. More recently, machine-learning approaches have also been applied successfully [6]. It is hard to compare the performance of the approaches due to inconsistent use of evaluation metrics and severe threats to validity regarding the used datasets [5]. Algebraic and probabilistic as well as machine-learning approaches rely on implicit models of key terms in documents (e.g., as points in a vector space or as probability distribution). Trace links are recovered based on similarity notions defined over these models. Therefore, it is hard to analyze and explain *why* specific trace links are identified in the model. Another drawback of machine-learning approaches is the need to train the models on reasonably large datasets. However, TLR datasets usually consists of less than 500 artifacts (at least the ones used in scientific publications [5]).

We follow a different approach and base our TLR approach on an explicit model of the knowledge represented in unrestricted NL requirements. Our pipeline translates NL requirements automatically into a semantic relation graph that encodes terms and their relations as vertices and edges. We use Spreading Activation to identify related requirements (i.e., trace links). The semantic search algorithm spreads activation in pulses over the vertices starting from a query vertex. Vertices with higher activation indicate higher relevance for the query. To the best of our knowledge, this is the first work that uses semantic relation graphs and Spreading Activation for automated TLR.

We applied and evaluated the approach on 5 datasets, commonly used in TLR research, in terms of *mean average precision* and *Lag* for answer sets of 5, 10, and 30 trace link candidates. With the best tested configuration, our approach achieves an average precision around 40% and a Lag around 50%. While this performance does not outperform existing state-of-the-art approaches, the explicit representation of requirements content as a semantic relation graph allows to "follow" a trace link through a chain of statements that may serve as an explanation why the trace link exists. This may help engineers when examining and vetting the trace link candidates.

II. Background

A. Trace Link Recovery

Requirements traceability is defined as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction" [7], i.e., over several phases and periods of refinement while those phases. A trace link states a dependency, relation, or similarity between two artifacts, the source and the target. We do not distinguish the type of links in the following work as we interpret all of them as some kind of relation.

Borg et al. [5] present a mapping study of IR approaches for traceability. They focus on text retrieval and classify 79 publications including their approaches based on the used retrieval model. Borg et al. treat the IR process as essential, NLP techniques are interpreted as an optional prerequisite. They differentiate between algebraic, probabilistic, and statistical language models as well as miscellaneous aspects like weighting scheme, similarity measures/distance functions, and enhancement strategies. The majority of classified publications applied an algebraic model, while most were evaluated in experiments on benchmarks (without human intervention) and used precision and recall as metrics.

Our approach does not match any of the retrieval models or their categories as we do not transfer requirements into a mathematical (algebraic, probabilistic, or statistical) model nor do we primarily apply any mathematical operations. We use several NLP techniques to analyze the requirements, extract terms and their relations. Subsequently, we transfer the results into a semantic relation graph and use a semantic search algorithm to find related artifacts. This includes some of the miscellaneous aspects of Borg et al. like phrasing, term frequency, and optionally similarity measures.

B. Knowledge Representation

Knowledge representation focuses on the depiction of information that enables computers to solve complex problems. Borgida et al. [8] already noted in 1985 that knowledge representation is the basis for requirements engineering.

Dermeval et al. [9] 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. [10] automatically derive conceptual models from user stories. The models enable discussion between stakeholders and show promising accuracy results (precision and recall between 80-92%). They use heuristics to analyze the user stories due to semi-structured natural language.

In a former publication [11], we presented an NLP pipeline that extracts knowledge from requirement documents and transforms it into a graph representing RDF[1] triples (subjects and objects become vertices, predicates become edges). The two generated sample graphs were not well-connected and yielded only a subset of fully connected vertices in a main graph. They used one graph to show the separation of two subsystems in an exemplary requirement specification.

C. Basics of Spreading Activation

Spreading Activation has its origin in the field of psychology. It is a theoretical model of how our mind connects information and tries to find an appropriate context with associated terms for a new word. The basic assumption is that terms relevant to each other are strongly connected by short or many paths, while less relevant ones are connected less or not at all. This model is applied to various science areas like IR [12].

The graph algorithm consists basically of three phases. In the initial phase, the start vertices are activated, i.e., they will be assigned an initial activation value. While the spreading phase, this activation is step wise 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 final phase, a sorted candidate list is created using the activation values to sort all vertices by relevance.

III. Approach: Knowledge Base Construction

The main goal while building the semantic relation graph is to depict semantic parts of common NL (e.g., words and phrases within sentences, but also documents and corpora) in vertices and connect these with each other based on their relation. Up to a certain point the graph structure resembles a tree structure with multiple roots. The root vertices represent the specifications, the leafs small parts of NL like single words. The vertices in between represent combined words like phrases. This structure supports that single words have a greater distance (i.e. are less relevant) to a certain specification than phrases or whole statements. [13]

We use several techniques to extract information from the requirements and to build the 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. 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.

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.

A. 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 [14] and DeepSRL [15] for semantic role labeling, are pipelines themselves and will be described in detail.

We use the basic requirements in Example 1 to demonstrate the NLP pipelines and the extracted knowledge graph.

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

Example 1: If the driver pushes the start button, the engine of the vehicle is started. If he pushes the stop button, the engine is stopped. The start button is located inside the large and comfortable cockpit, the main place for the driver seat.

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.

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 1, amongst others, "button" and "cockpit" are noun phrases (determiners and adjectives removed) and the coordinating conjunction "and" depend on the both adjectives "large" and "comfortable".

Lastly, the coreference resolution is looking for mentions of the same entities. There are two different kinds of mentions. In Example 1, the word "he" refers to "the driver" 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, "engine" (in the second sentence from Example 1) also refers to "engine of the vehicle" but is a nominal reference which contains additional information, i.e., the entity *engine of the vehicle* is also just called *engine*.

Semantic role labeling (SRL) [16] 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.

The first sentence in Example 1 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 of the vehicle] and [$_{\text{AM-ADV}}$If the user pushes the button]. Because this is a passive clause, there is no first argument and "the engine of the vehicle" 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 driver] and [$_{\text{A1}}$the start button]. The numbered arguments would be the same even if the syntax of the sentence would change, e.g., "If [$_{\text{A1}}$the start button] is [$_{\text{V}}$pushed] [$_{\text{A0}}$by the driver], [...]".

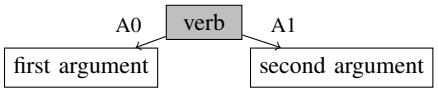


Fig. 1: Graph structure for single SRL predicate

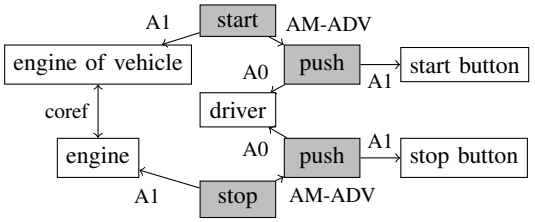


Fig. 2: SRL structure for first two sentences of Example 1

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 [17, 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 [17].

B. 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 relevant to each other than completely foreign ones.

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

C. Semantic 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 Fig. 1. If an argument contains a predicate itself,

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

[3]https://verbs.colorado.edu/~mpalmer/projects/ace.html

[4]https://github.com/luheng/deep_srl

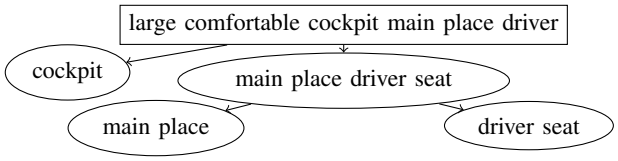


Fig. 3: Graph noun phrase structure for Example 1

the graph structure for that predicate is likewise added and connected via an edge between both verb vertices as shown in Fig. 2 for Example 1 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 Fig. 2.

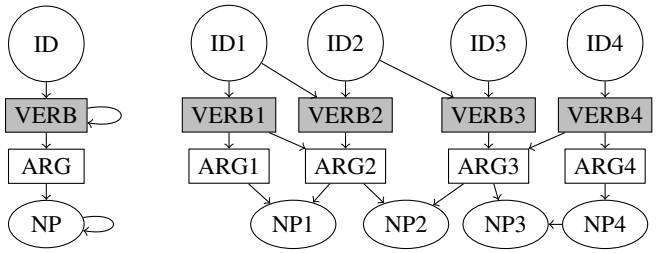
If an argument occurs more than once, the same argument vertex is used for both occurrences. This only applies to arguments that contain at least one noun, otherwise simple adjectives (e.g., "down") would lead as single arguments to a relation which is undesirable. To support this deduplication of arguments, 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 differentiates between verbs, nouns, and adjectives. For instance, "the engine of the vehicle" is transformed into "engine#n of#o vehicle#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. Because of this rule, there are two "push" verb vertices in Fig. 2 as they have different arguments.

While the arguments of the first two sentences in Example 1 only contain simple phrases, an argument may be a much more complex phrase. In the last sentence, the whole phrase "the large and comfortable cockpit, the main place for the driver seat" is the second argument. The resulting argument vertex will not lead to a deduplication if the argument of another predicate is just "cockpit". To circumvent this issue, we add additional vertices to the graph based on the given noun phrases and their dependencies as shown in Fig. 3.

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 1, "and" does not depend on a noun phrase and would lead to two vertices for the split arguments "inside the large" and "comfortable cockpit, the [...]".

In addition to the vertices and edges for NL, the additional information must also be added to the graph. This depends on the characteristics of the additional information. For each requirement an identifier 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 identifier vertices for each module and chapter which are connected to the requirement



(a) Model Level (b) Instance level

Fig. 4: Graph structure

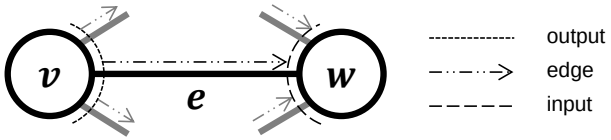


Fig. 5: Exemplary activation spreading from v to w over e

vertices. Trace links may be interpreted as edges between requirement identifier vertices as well.

The model level of the graph and an exemplary section of an instance level is shown in Fig. 4. In terms of path length between ID vertices, there is the strongest relation between ID1 and ID2 and between ID2 and ID3 with a path length of 2 and a very strong relation between ID3 and ID4 as their verb vertices share the same argument vertex ARG3. Without the edge between ID2 and ID3, there is only a weak relation between ID1/ID2 and ID3/ID4 because they would be only connected via the noun phrase vertex NP2.

IV. Approach: Semantic Search

We use Spreading Activation as semantic search algorithm to find, for a given query (trace link source), all related information (targets). Since the graph includes structural and semantic vertices, a link may be found by structural or semantic relations. If all vertices in a graph are connected through paths, each vertex has a (depictable) relationship to all others. Thus, we build a candidate list to sort all (reachable) targets for a query based on their relation.

A. Spreading Activation

There are different ways to configure the spreading of the activation values. We use the state-of-the-art configuration options and algorithm by Hartig [18, p. 90 ff.][5], which are based on several modes. Fig. 5 shows the basic spreading of activation from one vertex to another including the interim values like the output activation of sending vertices, the edge activation of transporting edges, and the input activation of receiving vertices. At first, a vertex activation is initially assigned to a query vertex. Algorithm 1 shows how the vertex activation is spread across the graph. During each pulse, the output activation (line 5), the edge activation (line 7), the input activation (line 11) and the vertex activation (line 12) are

[5]Our Implementation: https://github.com/tub-aset/spreadingactivation

Algorithm 1: Spreading Activation

Data: $G = (V, E), p_{max}, \tau, d$
Result: vertex$[v, p_{max}]$ for $v \in V$

1 $p \leftarrow 0$;
2 **while** $p < p_{max}$ **do**
3 $p \leftarrow p + 1$;
4 **for** $v \in V$: output$[v, p-1] \geq \tau$ **do**
5 output$[v, p] \leftarrow$ vertex$[v, p-1] * \texttt{att}(d, p) * \texttt{br}(v)$;
6 **for** $e \in \texttt{se}(v, p)$ **do**
7 edge$[v, e, p] \leftarrow$ output$[v, p] * \texttt{ew}(v, e)$;
8 **end**
9 **end**
10 **for** $v \in V$ **do**
11 input$[v, p] \leftarrow \sum_{\substack{w \in V : \texttt{adj}(v, w) \\ e \in E : \texttt{inz}(v, e)}}$ edge$[w, e, p]$;
12 vertex$[v, p] \leftarrow \texttt{act}($input$[v, p] +$ vertex$[v, p-1])$;
13 **end**
14 **end**

TABLE I: Activation Modes

Mode	Function
IDENTITY (ID)	$\texttt{act}(x) := x$
SIG (S)	$\texttt{act}(x) := 2 * (\frac{1}{1+e^{-x}} - 0.5)$
LOG2 (L2)	$\texttt{act}(x) := log_2(x+1)$
LOG10 (L10)	$\texttt{act}(x) := log_{10}(x+1)$
LOG2SIG (L2S)	$\texttt{act}(x) := 2 * (\frac{1}{1+e^{-log_2(x+1)}} - 0.5)$
LOG10SIG (L10S)	$\texttt{act}(x) := 2 * (\frac{1}{1+e^{-log_{10}(x+1)}} - 0.5)$

calculated for all relevant vertices and edges. There are modes for activation (TABLE I), attenuation (TABLE II), branching (TABLE III), and sending (TABLE V, line 6), an edge weight (TABLE IV) as well as two factors, the minimum activation τ (line 4) and the attenuation factor d (line 5), which are included in the calculation.

Hartig uses different edge weights based on specific edge types. We are using only a constant edge weight (see TABLE IV) because we assume all edges are equal.

The direction of the spreading is independent of the direction of the edge due to the fact that in most cases, the semantics of an edge may be restated the other way around, e.g., "X has function Y" becomes "Y is a function of X".

The final vertex activation values are used for sorting the candidate list but not for classification (e.g., by a threshold), since they are not limited to an upper or lower bound and not directly comparable (500 is not twice as important as 250).

B. Result Explanation via Spread Graphs

We use *spread graphs* and their *minimization* as proposed by Michalke et al. [19] to comprehend which parts of the graph primarily contribute to the activation of the results. While they use the semantics of RDF graphs to generate an explanation

TABLE II: Attenuation Modes

Mode	Function
IGNORE (IG)	$\texttt{att}(d, p) := 1$
FIXED (FX)	$\texttt{att}(d, p) := d$
INCREASING (IN)	$\texttt{att}(d, p) := 0.99^p * d$

TABLE III: Branching Modes

Mode	Function
NONE (NO)	$\texttt{br}(v) := 1$
DEGREE (DG)	$\texttt{br}(v) := \frac{1}{\lvert e \in E : \texttt{inz}(v, e) \rvert}$
FANOUT (FA)	$\texttt{br}(v) := \frac{\lvert m \in V : \texttt{adj}(v, m) \rvert}{\lvert V \rvert}$
BRANCH (BR)	$\texttt{br}(v) := \frac{1}{max(1, \lvert \texttt{se}(v, p) \rvert)}$

based on different patterns, we present excerpts of contributing requirements specifications to the user as our graph contains phrases and words that are directly traceable to their origin.

A *spread graph* is a directed graph illustrating the Spreading Activation process on the *original graph*. They utilize the activation flow and values during Spreading Activation. Each vertex in a spread graph represents the state of a vertex of the original graph after a certain pulse. Each edge of a spread graph represents a corresponding edge in the original graph, which transports activation from a vertex to another during a pulse, and therefore connects the representatives of its vertices in the spread graph.

A *minimized spread graph* removes unnecessary vertices from a *spread graphs* that do not contribute to the activation of a given result vertex. These vertices can be regarded as irrelevant to the final activation of the given result vertex and therefore do not contribute to the result explanation. A minimized spread graph tracks back the edges of a spread graph in opposite direction, starting at the given result vertex.

Fig. 6 shows a minimized spread graph for the exemplary graph of Fig. 4 after a Spreading Activation with 4 pulses. The given result vertex is ID2 and the starting vertex is ID1. The transporting vertices in the middle are VERB2, ARG2, and VERB1. Each of the verb, argument, and noun phrase vertices can be traced back to their occurrences in the NL requirements. To generate an explanation for the user, we use these text occurrences and show a side-by-side view of the source and

TABLE IV: Edge Weight Modes

Mode	Function
CONSTANT	$\texttt{ew}(v, e) := 1$

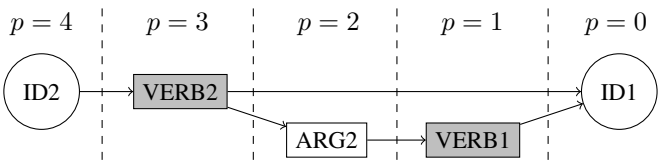


Fig. 6: Minimized Spread Graph between ID2 and ID1

TABLE V: Send Modes

Mode	Function
BASIC (B)	$\mathtt{se}(v,p) := \{e \in E : \mathtt{inz}(v,e)\}$
RECENT_RECEIVER (RR)	$\mathtt{se}(v,p) := \{e \in E : \mathtt{inz}(v,e), \mathsf{input}[v,p-1] > 0\}$
FORWARD (FW)	$\mathtt{se}(v,p) := \{e \in E : \mathtt{inz}(v,e), \mathsf{edge}[v,e,p-1] = 0\}$
FORWARD_LOOP (FWL)	$\mathtt{se}(v,p) := \{e \in E : \mathtt{inz}(v,e), (\mathsf{edge}[v,e,p-1] = 0) \vee (\exists e_1 \in E, e \neq e_1, \mathsf{edge}[v,e_1,p-1] > 0)\}$
FW-RR	$\mathtt{se}(v,p) :=$ FORWARD $\cap$ RECENT_RECEIVER
FWL-RR	$\mathtt{se}(v,p) :=$ FORWARD_LOOP $\cap$ RECENT_RECEIVER

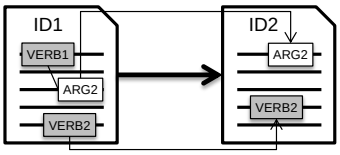


Fig. 7: Explanation for Trace Link Candidates

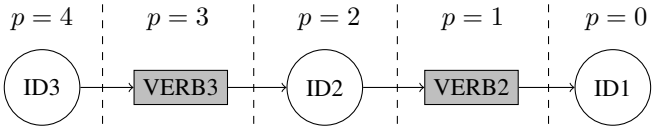


Fig. 8: Minimized Spread Graph between ID3 and ID1

target requirement and highlight the relevant text passages in both views as shown in Fig. 7. To indicate a higher relevance (i.e. a higher activation value) of certain text passages to the user, the highlighting intensity can vary.

Fig. 8 shows another minimized spread graph for the exemplary graph of Fig. 4 with the given result vertex ID3. Next to the verb vertices, the identifier vertex of ID2 is part of the transporting vertices in the middle. In this case, the documents have to be chained for comparison to generate an explanation between ID1 and ID3. This also allows users to find relations between documents without a direct connection.

V. Evaluation

We evaluate the approach for TLR on five common datasets (Infusion Pump, CCHIT-2-WorldVista, GANNT, CM-1, and WARC of [20]). They come from different domains like health care, science, or business. For comparison, we use a syntactical VSM approach with tf-idf values for lemmas and compare requirements using cosine similarity [21], [22].

TABLE VI gives an overview of the datasets, including the number of sources and targets, how many of them are linked, the number of actually defined trace links in the answer set, and how many would be possible at maximum. We build semantic relations graphs for each dataset, containing both high and low level requirement specifications. Next to the described vertices and edges for verbs, arguments, and noun phrases, the graphs contain additional identifier vertices for each requirement specification that are connected to the verb vertices, because they indicate the strongest semantics. These identifier vertices are used as query vertex for the initial activation and to identify target requirements. TABLE VI also contains the

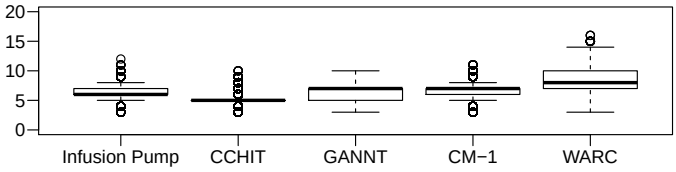


Fig. 9: Shortest Path Lengths for Graphs

number of vertices and edges and how often there is no path between a source and a target.

Fig. 9 shows the distribution of existing shortest path lengths between identifier vertex of each source and target. While the datasets and graphs vary in size, the average shortest distance between a source and a target is nearly the same.

To find an appropriate configuration, we used an exploratory approach and discovered 500 random but valid configurations based on the configuration parameter ranges in TABLE VII (see [18, Table 4.13]). A configuration is valid if Spreading Activation does not terminate before the given number of pulses for all queries of all datasets. Otherwise, this indicates too strong restriction to spread activation and would lead to no results.

A. Metrics

There are several evaluation metrics that depend on different goals when evaluating TLR. Shin et al. [23] performed a systematic literature review and defined three different goals. Goal 1 is to find trace links with high accuracy, e.g., to support tasks like coverage analysis. Goal 2 is to find relevant documents excluding irrelevant documents to reduce unnecessary effort for human analysts. Goal 3 is to rank all documents so that the relevant ones are near the top of the retrieved list, also to reduce human effort. Our approach supports goal 3, as we build a ranked list of all documents and generate an explanation for the user, who is manually creating or checking trace links.

To evaluate the achievement of goal 3, Shin et al. mention three different metrics, i.e., average precision (AP), Lag, and AUC (area under the ROC curve). Each metric focuses on different weighting schemes for the position in the ranked list. AP assigns a non-proportionally higher weight to a correct link ranked at the top and thereby rewards correct links at the top. Lag assigns a non-proportionally higher weight to a correct link ranked at the bottom of the list, which penalizes those links. AUC uses the same weight for all correct links but is

TABLE VI: Datasets and corresponding semantic relation graphs

	Sources		Targets		Links		Graph		
Dataset	overall	linked	overall	linked	actual	possible	vertices	edges	no path
Infusion Pump	126	104	21	20	131	2,646	1,731	3,622	0
CCHIT	116	72	1064	415	587	123,424	9,239	22,669	0
GANNT	17	17	68	68	68	1,156	824	1,755	0
CM-1	235	155	220	150	361	51,700	7,616	18,375	0
WARC	63	60	89	79	136	5,607	1,188	1,975	63

TABLE VII: Configuration Parameter

Mode	Values
Activation	{ID, S, L2, L10, L2S, L10S}
Attenuation	{IG, FX, IN}
Branching	{NO, DG, FA, BR}
Sending	{B, RR, FW, FWL, FW-RR, FWL-RR}
d	$[0.8 - 1]$
τ	$[0 - 0.15]$
p_{max}	$[10 - 50]$

TABLE VIII: Best configurations

		Per metric		Overall		Syntactical VSM	
Dataset	ND	MAP	Lag	MAP	Lag	MAP	Lag
Infusion Pump	5	0.430	0.504	0.402	0.516	0.530	0.418
	10	0.453	0.515	0.429	0.527	0.537	0.420
	30	0.469	0.517	0.440	0.552	0.548	0.447
CCHIT	5	0.111	0.661	0.077	0.709	0.251	0.452
	10	0.092	0.603	0.065	0.613	0.218	0.393
	30	0.097	0.540	0.070	0.574	0.223	0.384
GANNT	5	0.349	0.294	0.304	0.433	0.412	0.238
	10	0.388	0.283	0.350	0.421	0.454	0.222
	30	0.421	0.338	0.382	0.415	0.492	0.329
CM-1	5	0.259	0.537	0.259	0.537	0.392	0.432
	10	0.282	0.508	0.282	0.508	0.422	0.438
	30	0.306	0.554	0.301	0.554	0.447	0.484
WARC	5	0.353	0.489	0.299	0.543	0.470	0.372
	10	0.365	0.479	0.333	0.525	0.490	0.396
	30	0.388	0.500	0.355	0.548	0.502	0.421

not applicable as it is a classification accuracy metric and not a rank accuracy metric [24]. While a high value is desired for AP, Lag indicates how many incorrect links are proposed before a correct one and should be as low as possible.

Shin et al. show five different types of thresholds for the ranked list. Despite the fact that they recommend relative thresholds rather than absolute ones, we use ND (number of retrieved documents) with the values 5, 10, and 30, which cuts the list after a fixed number of retrieved documents. We justify this decision by assuming that the approach should be used in setups including a lot of requirements documents, but a user is not capable to check through thousands of potential candidates and understand the explanations in detail. A relative threshold such as 10% of the list would yield only 2 results for Infusion Pump but more than 40 for CCHIT.

The metric values are summarized on *average* across all traces, i.e., mean average precision (MAP) for AP. We calculated the metrics only for source and target artifacts that are linked to any artifact because our approach does no classification and provides results for all queries (even for queries without any correct answer).

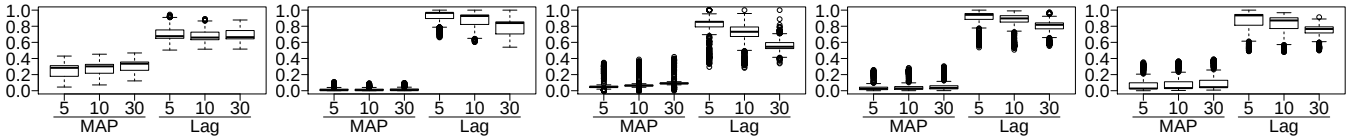
B. Results for Datasets

The MAP and Lag for ND 5, 10, and 30 for each dataset are shown in Fig. 10. While the majority of configurations show bad to mediocre results, the best ones achieve a MAP around 40% and a Lag around 50% on all ND thresholds, except for MAP of CCHIT. While Lag of CCHIT is comparable to the other datasets, the low MAP indicates that the correct results are not the top results but are shown at a lower position.

An explanation for the low MAP scores on some datasets is that for some queries, the approach did not list any correct result within the list of top-N results. For example, all search results for the CM-1 dataset are shown as heat maps in Fig. 11. The x-axis lists the 500 configurations, the y-axis contains 155 queries for each linked high-level requirement. The color scheme indicates the ratio of correct links in the answer list from light gray for no correct links at all to dark gray for 100% of all correct links, independent of the position in the result list, i.e., the darker a column, the better a configuration, and the darker a row, the easier it is to answer a query correctly. There are a few queries where, regardless of the configuration, valid links are found often, e.g., around query 125–140 and 30–40. This pattern becomes clearer as the ND threshold increases. In addition, there are some queries where results almost never contain a valid link, e.g., between 150–155. Comparable to this vertical patterns, there are comparable horizontal ones with some of the best configurations around 310 and some of the worst around 400.

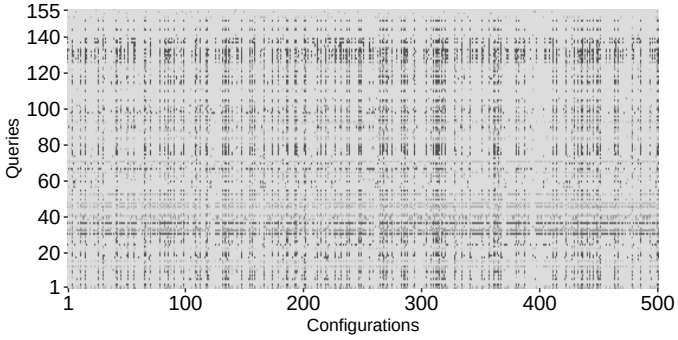
The corresponding heat maps of the other datasets (Fig. 12) have similar vertical patterns at the same places, especially for the configurations around 310. This leads us to the assumption that a configuration has a constant quality for different datasets. To compare and rank configurations, we calculated a harmonic mean between MAP and Lag. Each configuration shows comparable results for all datasets based on its rank of all configurations. This is evident from the fact that the average difference between 4 of 5 ranks of a single configuration is much smaller (around 100) than the total number of all configurations (500) as shown in Fig. 13.

TABLE VIII shows the best metric values of all configurations for each dataset and threshold, compared per metric, for the overall best configuration, and for the syntactical

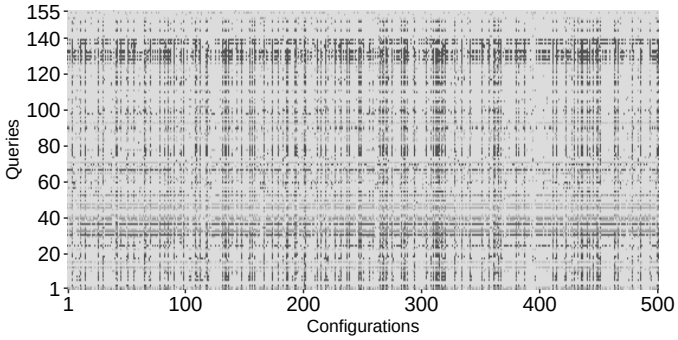


(a) Infusion Pump (b) CCHIT (c) GANNT (d) CM-1 (e) WARC

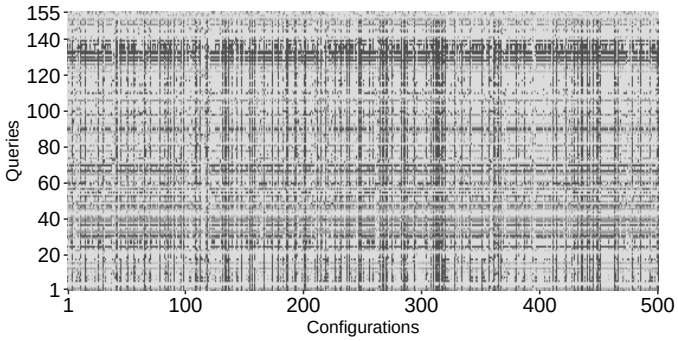
Fig. 10: MAP and Lag at ND 5, 10, and 30



(a) Results at ND 5

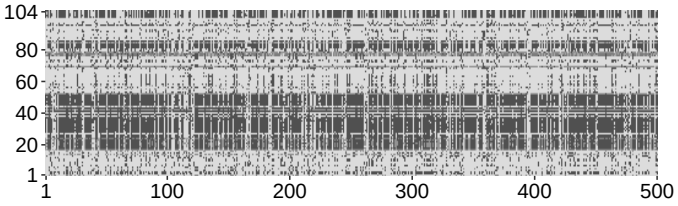


(b) Results at ND 10

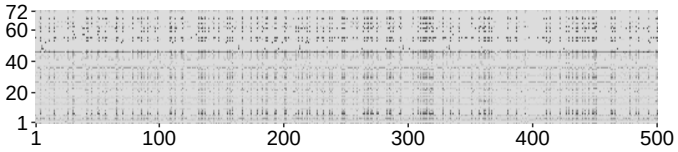


(c) Results at ND 30

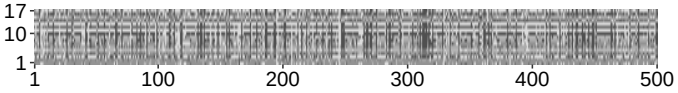
Fig. 11: Search results for CM-1 dataset



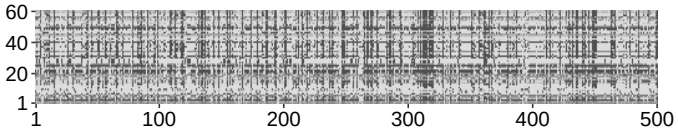
(a) Infusion Pump: Results at ND 5



(b) CCHIT: Results at ND 30



(c) GANNT: Results at ND 30



(d) WARC: Results at ND 30

Fig. 12: Search results for other datasets at certain ND

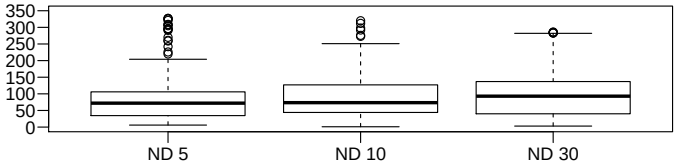


Fig. 13: Rank differences of all configurations

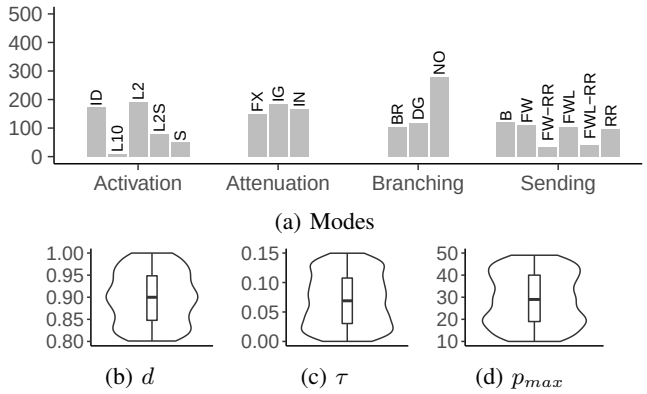
VSM approach. In most cases, the quality of the overall best configuration is close to that of the individual best configurations. In contrast, the syntactical approach outperforms the semantic approach.

C. Limitations

There are mainly two parts of the approach that affect the performance results, the graph and Spreading Activation, which both also depend on each other. As Fig. 11 illustrates, certain queries never lead to a correct result, which indicates that there are other targets activated stronger. None of the 500 random configurations was able to overcome this circumstance.

On one hand, the graph structure might not be optimal. For example, we merge noun phrase vertices without considering their adjectives or coreference. But in some cases, the adjective may impact the meaning, e.g., “large and comfortable cockpit” vs. (any) “cockpit” in Example 1.

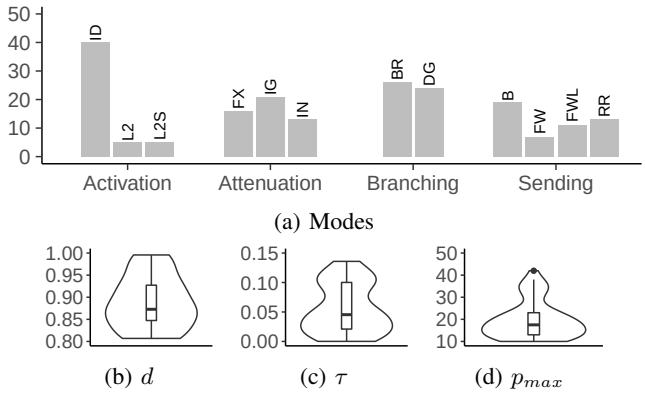
Furthermore, we do not merge or connect semantically similar vertices. Two or more words/phrases are semantically similar if they have the same meaning but different syntax.



(a) Modes

(b) d (c) τ (d) p_{max}

Fig. 14: Parameter distributions of all valid configurations



(a) Modes

(b) d (c) τ (d) p_{max}

Fig. 15: Parameter distributions of top 50 configurations

There are different approaches to identify such semantic similarities. Word embeddings like *word2vec* [25] or *GloVe* [26] rely on the distributional hypothesis [27] 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 with respect to their context. This also causes words of opposite meaning to be related to each other because they occur in the same context [28]. Another approach is a database that contains known similarities, e.g., WordNet[6]. They usually contain only common similarities and are not aware of technical terms.

On the other hand, the search for relevant information is challenging, too. There is no easy way to derive an optimal configuration for a generic Spreading Activation that provides the best results for all search queries. At least any kind of restriction within the configuration is needed [29]. A good configuration depends on the one hand on all parameters that strongly depend on each other, and on the other hand, on the graph and its characteristics. Fig. 14 shows the distributions of all parameter values within the 500 random but valid configurations. In comparison, Fig. 15 shows the distributions of parameter values within the top-50 configurations. While the numerical values d, τ, and p_{max} are almost equally distributed within the 500, the discrete modes already show a filtering in all configurations, e.g., LOG10SIG and FANOUT are not present. In the top-50, there is a clear trend for each numerical value. For the discrete modes, only IDENTITY is clearly favored while LOG10, SIG, NONE, and the combined FW-RR and FWL-RR are completely excluded.

D. Comparison with other TLR Approaches

An objective comparison with other TLR approaches is hardly possible. Shin et al. [30] analyzed 24 publications on TLR and observed that some of the used evaluation metrics are not appropriate for evaluating the task. Only 4 approaches were evaluated using MAP and only 1 using Lag. 10 out of 24 did not report their summarization method, only one used *average* while 4 (or more, as assumed by Shin et al.) used *aggregation*. In addition, they observed 7 different threshold types like PR (recall levels at which precision was measured), PD (percentage of retrieved documents), and ND, which was used only once.

We found 10 publications that used at least one of our datasets and reported AP, MAP, or Lag. The results are shown in TABLE IX. We have averaged our MAP and Lag values for each dataset of the overall best configuration (right column in TABLE VIII) over all ND thresholds of 5, 10, and 30. If the other authors reported more than one metric to compare several approaches or configurations, we used only the best one for comparison with our approach. To summarize the reported AP by Gibiec et al. [32, Figure 5a], we used *average* at ND10. Tian et al. [37] used only a subset of CM-1 including 22 high-level and 53 low-level requirements. [37] and [38] reported their metrics only as figures, so we had to estimate the values. Cleland-Huang et al. [39] used only HIPAA-related requirements of the CCHIT dataset (78 of 1064 requirements), traced them to 11 (instead of 116) regulatory codes and summarized their metrics over 10 datasets (78 CCHIT requirements of 244 total). Dietrich et al. [40] also used the CCHIT dataset but considered only 383 requirements.

Since none used the same threshold type or summarization method, the results are only partially comparable.

VI. Discussion

We achieve moderate results in the evaluation without any explicit assumptions on the requirements or particular optimization for TLR. The graph and the algorithm scales for different sizes of datasets and is (almost) immediately applicable (i.e., no training needed, only one pass through the NLP pipeline). Also, the graph adapts immediately to changes in the data as new requirements are parsed and vertices/edges are inserted directly into the graph as well as existing vertices/ edges are removed if requirements are removed. Finding an optimal configuration for the semantic search algorithm is a non-trivial task. Certain discrete modes are clearly favored and the chosen values for the numerical parameters in TABLE VII seem to fit as there are (local) maxima identifiable in Fig. 15. Therefore, we assume that improvements in the approach will mainly be found in the adjustment of the graph. Due to the

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

TABLE IX: Comparison with other publications

	Our overall		VSM		[31]	[32]	[33]		[34]	[35]		[36]	[37]	[38]	[39]	[40]
threshold:	ND5-30		ND5-30		PR	ND10	PD100		–	PR		PR	–	PR	–	PD100
sum:	avg		avg		avg	–	–	avg	–	avg		avg	avg	–	–	avg
metric:	MAP	Lag	MAP	Lag	MAP	AP	AP	MAP	MAP	MAP	Lag	MAP	MAP	AP	AP	MAP
Infusion P.	0.42	0.53	0.54	0.43	–	–	–	–	–	–	–	–	–	–	–	–
CCHIT	0.07	0.63	0.23	0.41	0.25	0.20	–	–	–	–	–	–	–	–	0.36	0.45
GANNT	0.35	0.42	0.45	0.26	0.48	–	0.56	0.65	–	–	–	–	–	–	–	–
CM-1	0.28	0.53	0.42	0.45	0.40	–	–	–	0.50	0.02	0.39	0.23	0.36	0.07	–	–
WARC	0.33	0.54	0.49	0.40	0.63	–	–	–	0.56	–	–	–	–	–	–	–

constant length of shortest paths between sources and targets (Fig. 9) and the fact that the configurations perform comparable on different datasets (Fig. 13), we assume that an almost optimal configuration, once determined, may be reused.

Our approach does not outperform the syntactical VSM. The reason may be the characteristics of the graph or Spreading Activation. While VSM only compares single words with each other, which seems to be adequate, our approach weight coherent phrases higher due to their shorter path length, which may lead to more activation for more distant results. We think that improvements should be sought in the graph in particular, e.g., by not using the hierarchical structure of nouns, but directly connecting them to arguments.

Another option to improve the quality may be to combine both approaches, the syntactical VSM and our Spreading Activation. For example, use a VSM approach to identify targets and then rank them by Spreading Activation and generate explanations based on the filtered sub-graph (cf. [41]).

In addition to Spreading Activation configuration, we plan to use tf-idf to downgrade vertices with common phrases. While Spreading Activation is also able to downgrade such vertices, it has 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.

While our experiments focus only on small datasets with a manageable number of possible results, in practice, there are much larger datasets, so that human interaction (goal 3 of Shin et al. [23]) may no longer be achieved. In this case, we have to change the metrics (e.g., not ND 30 but PR 100) and would therefore possibly choose other configurations (e.g., with more restrictions to accomplish a discrimination for automation instead of ranking).

Other semantic approaches often deal with semantic distance models between single parts (e.g., words or phrases). Mahmoud et al. [42] 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. Amongst others, the CM-1 dataset was evaluated and achieved a MAP of 20–40% and a Lag of 10–60% (overall, no ND). Their results show that explicit semantic methods may outperform latent methods.

Guo et al. [43] also focus on single words, using 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 existing VSM and LSI approaches on their unpublished dataset.

Guo et al. [44] 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 [45].

Hartig et al. [18], [46] 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. To improve their configuration optimization, they used semantic network skeletons [47]. They also tried to find optimal configurations using evolutionary algorithms, but the maximal fitness seems to quickly convert against an upper limit [18, p. 145 ff.]. There is only evidence that certain modes or combinations of them are better than others. Finally, they generate a textual explanation for the user derived from spread graphs by Michalke et al. [19].

VII. Conclusion

In this paper, we present a novel approach for Trace Link Recovery using semantic relations between parts of natural language, stored in a semantic relation graph, and searched by a semantic search algorithm. While the approach is fully automated, it does not have any prerequisites with regard to the format or the content of the natural language (except for English language) and is scalable to various sizes of corpora. We achieve moderate results on several datasets without any specific optimization. To improve the user confidence, we are able to generate an explanation between each query and result requirement by identifying and highlighting the contributing text passages.

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

References

[1] S. Winkler and J. von Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Software & Systems Modeling (SoSyM)*, 2010, https://doi.org/10.1007/s10270-009-0145-0.

[2] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *Transactions on Software Engineering (TSE)*, 2002, https://doi.org/10.1109/TSE.2002.1041053.

[3] M. Heindl and S. Biffl, "A case study on value-based requirements tracing," in *European Software Engineering Conference (ESEC/FSE)*, 2005, https://doi.org/10.1145/1081706.1081717.

[4] J. Huffman Hayes, A. Dekhtyar, and J. Osborne, "Improving requirements tracing via information retrieval," in *Requirements Engineering (RE)*, 2003, https://doi.org/10.1109/ICRE.2003.1232745.

[5] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability," *Empirical Software Engineering (EMSE)*, vol. 19, no. 6, pp. 1565–1616, 2014, https://doi.org/10.1007/s10664-013-9255-y.

[6] C. Mills, "Towards the automatic classification of traceability links," in *Automated Software Engineering (ASE)*, 2017, https://doi.org/10.1109/ASE.2017.8115723.

[7] O. Gotel and C. W. Finkelstein, "An analysis of the requirements traceability problem," in *Requirements Engineering (RE)*, 1994, https://doi.org/10.1109/ICRE.1994.292398.

[8] A. Borgida, S. Greenspan, and J. Mylopoulos, "Knowledge Representation as the Basis for Requirements Specifications," pp. 152–169, 1985, https://doi.org/10.1007/978-3-642-70840-4_13.

[9] D. Dermeval, J. Vilela, I. I. Bittencourt, J. Castro, S. Isotani, P. Brito, and A. Silva, "Applications of ontologies in requirements engineering: A systematic review of the literature," in *Requirements Engineering (RE)*. Springer, 2016, pp. 405–437, https://doi.org/10.1007/s00766-015-0222-6.

[10] M. Robeer, G. Lucassen, J. M. E. M. van der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated Extraction of Conceptual Models from User Stories via NLP," in *Requirements Engineering (RE)*. IEEE, 2016, pp. 196–205, https://doi.org/10.1109/RE.2016.40.

[11] A. Schlutter and A. Vogelsang, "Knowledge Representation of Requirements Documents Using Natural Language Processing," in *Natural Language Processing for Requirements Engineering (NLP4RE)*. RWTH Aachen, 2018, https://doi.org/10.14279/depositonce-7776.

[12] F. Crestani, "Application of spreading activation techniques in information retrieval," *Artificial Intelligence Review*, vol. 11, no. 6, pp. 453–482, 1997, https://doi.org/10.1023/A:1006569829653.

[13] A. Schlutter and A. Vogelsang, "Knowledge Extraction from Natural Language Requirements into a Semantic Relation Graph," in *Knowledge Graph for Software Engineering (KG4SE)*. Association for Computing Machinery (ACM), 2020, https://doi.org/10.14279/depositonce-9772.2.

[14] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The stanford CoreNLP natural language processing toolkit," in *System Demonstrations*. Association for Computational Linguistics (ACL), 2014, pp. 55–60.

[15] L. He, K. Lee, M. Lewis, and L. Zettlemoyer, "Deep Semantic Role Labeling: What Works and What's Next," in *Association for Computational Linguistics*. Association for Computational Linguistics (ACL), 2017, pp. 473–483, https://doi.org/10.18653/v1/p17-1044.

[16] X. Carreras and L. Màrques, "Introduction to the CoNLL-2004 shared task: Semantic role labeling," in *Computational Natural Language Learning (CoNLL)*. Association for Computational Linguistics (ACL), 2004, pp. 89–97.

[17] X. Carreras and L. Màrquez, "Introduction to the CoNLL-2005 shared task: Semantic role labeling," in *Computational Natural Language Learning (CoNLL)*. Association for Computational Linguistics (ACL), 2005, pp. 152–164.

[18] K. Hartig, "Entwicklung eines Information-Retrieval-Systems zur Unterstützung von Gefährdungs- und Risikoanalysen," Ph.D. dissertation, Technische Universität Berlin, 2019, https://doi.org/10.14279/depositonce-8408.

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

[20] J. Huffman Hayes, J. Payne, and M. Leppelmeier, "Toward Improved Artificial Intelligence in Requirements Engineering: Metadata for Tracing Datasets," in *Artificial Intelligence for Requirements Engineering (AIRE)*. IEEE, 2019, pp. 256–262, https://doi.org/10.1109/REW.2019.00052.

[21] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[22] A. Singhal, "Modern Information Retrieval: A Brief Overview," *IEEE Computer Society Technical Committee on Data Engineering (TCDE)*, vol. 24, no. 4, pp. 35–43, 2001, http://singhal.info/ieee2001.pdf.

[23] Y. Shin, J. Huffman Hayes, and J. Cleland-Huang, "Guidelines for Benchmarking Automated Software Traceability Techniques," in *Symposium on Software and Systems Traceability (SST)*, 2015, https://doi.org/10.1109/SST.2015.13.

[24] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *Transactions on Information Systems*, 2004, https://doi.org/10.1145/963770.963772.

[25] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *Computing Research Repository (CoRR)*, 2013, https://arxiv.org/abs/1301.3781.

[26] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Empirical Methods in Natural Language Processing*. Association for Computational Linguistics (ACL), 2014, pp. 1532–1543, https://doi.org/10.3115/v1/d14-1162.

[27] M. Sahlgren, "The distributional hypothesis," *Italian Journal of Linguistics*, vol. 20, no. 1, pp. 33–54, 2008.

[28] N. Mrkšić, D. Ó Séaghdha, B. Thomson, M. Gašić, L. M. Rojas-Barahona, P.-H. Su, D. Vandyke, T.-H. Wen, and S. J. Young, "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), 2016, pp. 142–148, https://doi.org/10.18653/v1/N16-1018.

[29] M. R. Berthold, U. Brandes, T. Kötter, M. Mader, U. Nagel, and K. Thiel, "Pure spreading activation is pointless," in *Conference on information and knowledge management (CIKM)*. Association for Computing Machinery (ACM), 2009, pp. 1915–1918, https://doi.org/10.1145/1645953.1646264.

[30] Y. Shin, J. Huffman Hayes, and J. Cleland-Huang, "A Framework for Evaluating Traceability Benchmark Metrics," *Technical Reports*, no. 21, 2012, https://via.library.depaul.edu/tr/21/.

[31] D. Farrar and J. Huffman Hayes, "A Comparison of Stemming Techniques in Tracing," in *Symposium on Software and Systems Traceability (SST)*. IEEE, 2019, pp. 37–44, https://doi.org/10.1109/SST.2019.00017.

[32] M. Gibiec, A. Czauderna, and J. Cleland-Huang, "Towards mining replacement queries for hard-to-retrieve traces," in *Automated Software Engineering (ASE)*. ACM Press, 2010, p. 245, https://doi.org/10.1145/1858996.1859046.

[33] H. Kuang, H. Gao, H. Hu, X. Ma, J. Lu, P. Mader, and A. Egyed, "Using Frugal User Feedback with Closeness Analysis on Code to Improve IR-Based Traceability Recovery," in *International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 369–379, https://doi.org/10.1109/ICPC.2019.00055.

[34] W. Li and J. H. Hayes, "Traceability Challenge 2013: Query+ enhancement for semantic tracing (QuEST): Software verification and validation research laboratory (SVVRL) of the University of Kentucky," in *Traceability in Emerging Forms of Software Engineering (TEFSE)*. IEEE, 2013, pp. 95–99, https://doi.org/10.1109/TEFSE.2013.6620162.

[35] A. Mahmoud, N. Niu, and S. Xu, "A semantic relatedness approach for traceability link recovery," in *International Conference on Program Comprehension (ICPC)*. IEEE, 2012, pp. 183–192, https://doi.org/10.1109/ICPC.2012.6240487.

[36] H. Sultanov, J. H. Hayes, and W.-K. Kong, "Application of swarm techniques to requirements tracing," in *Requirements Engineering (RE)*, 2011, pp. 209–226, https://doi.org/10.1007/s00766-011-0121-4.

[37] Q. Tian, Q. Cao, and Q. Sun, "Adapting Word Embeddings to Traceability Recovery," in *Information Systems and Computer Aided Education (ICISCAE)*. IEEE, 2018, pp. 255–261, https://doi.org/10.1109/ICISCAE.2018.8666883.

[38] X. Zou, R. Settimi, and J. Cleland-Huang, "Improving automated requirements trace retrieval: a study of term-based enhancement methods," *Empirical Software Engineering (EMSE)*, vol. 15, no. 2, pp. 119–146, 2010, https://doi.org/10.1007/s10664-009-9114-z.

[39] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *International Conference on Software Engineering (ICSE)*. ACM Press, 2010, p. 155, https://doi.org/10.1145/1806799.1806825.

[40] T. Dietrich, J. Cleland-Huang, and Y. Shin, "Learning effective query transformations for enhanced requirements trace retrieval," in *Automated Software Engineering (ASE)*. IEEE, 2013, pp. 586–591, https://doi.org/10.1109/ASE.2013.6693117.

[41] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the Wild: Automatically Augmenting Incomplete Trace Links," in *International Conference on Software Engineering (ICSE)*. New York, New York, USA: ACM Press, 2018, pp. 834–845, https://doi.org/10.1145/3180155.3180207.

[42] A. Mahmoud and N. Niu, "On the role of semantics in automated requirements tracing," in *Requirements Engineering (RE)*. Springer, 2015, pp. 281–300, https://doi.org/10.1007/s00766-013-0199-y.

[43] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically Enhanced Software Traceability Using Deep Learning Techniques," in *International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 3–14, https://doi.org/10.1109/ICSE.2017.9.

[44] J. Guo, M. Gibiec, and J. Cleland-Huang, "Tackling the term-mismatch problem in automated trace retrieval," *Empirical Software Engineering (EMSE)*, vol. 22, no. 3, pp. 1103–1142, 2017, https://doi.org/10.1007/s10664-016-9479-8.

[45] J. Guo, N. Monaikul, and J. Cleland-Huang, "Trace links explained: An automated approach for generating rationales," in *Requirements Engineering (RE)*. IEEE, 2015, pp. 202–207, https://doi.org/10.1109/RE.2015.7320423.

[46] K. Hartig and T. Karbe, "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), 2016, pp. 108–111, https://doi.org/10.14279/depositonce-6974.

[47] ——, "Semantic Network Skeleton - A Tool to Analyze Spreading Activation Effects," in *International Conference on Information, Process, and Knowledge Management (eKNOW)*. IARIA, 2016, pp. 126–131, https://doi.org/10.14279/depositonce-6973.