Aaron Schlutter, Andreas Vogelsang

Improving Trace Link Recovery using Semantic Relation Graphs and Spreading Activation

Schlutter A., Vogelsang A. (2021) Improving Trace Link Recovery Using Semantic Relation Graphs and Spreading Activation. In: Dalpiaz F., Spoletini P. (eds) Requirements Engineering: Foundation for Software Quality. REFSQ 2021. Lecture Notes in Computer Science, vol 12685. Springer, Cham. pp. 37-53. https://doi.org/10.1007/978-3-030-73128-1_3

Terms of Use
Copyright applies. A non-exclusive, non-transferable and limited right to use is granted. This document is intended solely for personal, non-commercial use.
Improving Trace Link Recovery using Semantic Relation Graphs and Spreading Activation

Aaron Schlutter\textsuperscript{1}\textsuperscript{[0000−0003−0537−3271]} and Andreas Vogelsang\textsuperscript{2}[0000−0003−1041−0815]

\textsuperscript{1} Technische Universität Berlin, Germany
aaron.schlutter@tu-berlin.de
https://www.aset.tu-berlin.de/menue/team/aaron_schlutter/

\textsuperscript{2} Software and Systems Engineering, Universität zu Köln, Germany
vogelsang@cs.uni-koeln.de
https://cs.uni-koeln.de/sse

Abstract. [Context & Motivation] 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. [Question/Problem] 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 considering the context of semantic statements. [Principal Ideas/Results] In this work, we revise our existing 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. The approach generates sorted candidate lists and is fully automated including an NLP pipeline to transform unrestricted natural language requirements into a graph and does not require any external knowledge bases or other resources. [Contribution] To improve the performance, we take a detailed look at five common datasets and adapt the graph structure and semantic search algorithm. Depending on the selected configuration, the predictive power strongly varies. With the best tested configuration, the approach achieves a mean average precision of 50\%, a Lag of 30\%, and a recall of 90\%.

1 Introduction

Trace Link Recovery (TLR) is a common problem in software engineering. While many engineering tasks profit from explicit links between related development artifacts \cite{26,1}, these links are laborious to maintain manually and therefore rarely exist in projects \cite{9}. 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 \cite{12}. 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 \cite{3}.
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)) [3]. More recently, machine-learning approaches have also been applied successfully [17]. It is hard to compare the performance of different approaches due to inconsistent use of evaluation metrics and severe threats to validity regarding the used datasets [3]. Algebraic and probabilistic as well as machine-learning approaches rely on implicit models of key terms in requirements (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 consist of less than 500 artifacts (at least the ones used in scientific publications [3]). Most are domain-specific, which means that additional care must be taken to ensure that the respective configuration (e.g., the neural net of a machine-learning approach) is not over fitted and thus less reusable for other datasets.

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 target requirements (i.e., trace links) for a given query requirement. The semantic search algorithm spreads activation in pulses over the vertices starting from the query vertex. Vertices with higher activation indicate higher relevance for the query. In our previous work [23], we introduced the basic concepts and evaluated the approach with bad to mediocre results. To improve the results, we take a detailed look at the used datasets with their pitfalls and how this influence the performance of our approach. Subsequently, we elaborate improvements for the graph building and structure as well as resulting adjustments to Spreading Activation. We applied and evaluated the approach on 5 datasets, commonly used in TLR research, in terms of mean average precision and Lag for a result list length of 5, 10, and 30 trace link candidates. With the best tested configuration, our approach achieves an average precision around 50%, Lag around 30% and recall around 90%.

2 Background

2.1 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” [6], i.e., over several phases and periods of refinement during 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. [3] 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 in general 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 textual requirements, extract terms and their relations \[21\]. 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. Our approach does not support any other kind of requirements than textual ones.

Likewise, we do not focus on automatic linking of related requirements without human intervention since we assume that trace links are usually used in sophisticated scenarios where every link is created manually based on given rules or guidelines. In our view, this cannot be achieved due to the ambiguity of natural language and the variety of guidelines. Instead, we want to support engineers who manually create trace links with a sorted candidate list.

2.2 Knowledge Representation

Knowledge representation focuses on the depiction of information that enables computers to solve complex problems. Borgida et al. \[4\] already noted in 1985 that knowledge representation is the basis for requirements engineering.

Dermeval et al. \[5\] 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. \[19\] 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 \[21\], we presented an NLP pipeline that extracts knowledge from requirement documents and transforms it into a graph representing RDF\(^3\) 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.

\(^3\) \text{https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/}
2.3 Existing approach

In our previous work [23], we used NLP techniques to extract semantic information from requirements and build a semantic relation graph as a knowledge base. We used Spreading Activation as semantic search algorithm to find, for a given query (trace link source), all related information (targets). The approach is fully automated to enable engineers to apply TLR to NL requirements without any further effort (e.g., maintain external resources like dictionaries). Comparable to machine-learning, an appropriate configuration must be initially determined for the approach. Such a configuration includes the general structure of the semantic relation graph as well as the configuration of Spreading Activation.

We built an NLP pipeline consisting of Stanford CoreNLP [15] and DeepSRL [8] to extract information from the requirements and to build the graph. The main goal while building the semantic relation graph is to depict semantic parts of common NL in vertices and connect these with each other based on their relation. The graph structure resembled a tree structure with multiple roots (identifier vertices) and vertices arranged in levels for verbs, arguments, and noun phrases. This structure supported that short phrases have a greater distance (i.e., are less relevant) to a certain specification than complex phrases or whole (verb) statements [22]. We used the given structure of semantic role labeling (SRL), which associates arguments with a semantic role to their predicate within a sentence. In addition, we used several basics like Lemmatizing or part-of-speech tagging to split and compare natural language phrases or filter insignificant parts.

To answer trace queries with the help of the semantic relation graph, we used Spreading Activation. We utilized the state-of-the-art options and algorithm by Hartig [7] to configure the spreading of activation. There are several modes and parameter to adjust attenuation, sending or branching to calculate the activation values during the pulsation. To find an appropriate configuration, we used an exploratory approach and discovered 500 random configurations. The final activation values of the identifier vertices are used to build a sorted candidate list with all (reachable) targets for a query based on their relation.

We evaluated the approach on five datasets of Huffman Hayes et al. [13]. We achieved bad to mediocre results, i.e., the top configurations have a mean average precision of 40% and a Lag of 50%, while most configurations have much worse results. Certain discrete modes are clearly favored and the chosen values for the numerical parameters seem to fit as there are (local) maxima identifiable. To compare the performance of configurations, we ranked them and found out that a configuration has a constant quality for different datasets. This is probably the case because the basic (syntactic and semantic) structure of requirements or natural language per se is usually similar, even if the concrete content differs.

We concluded that an almost optimal configuration, once determined, may be reused on other datasets with different requirements, key terms and relations. Even if there are specific configurations for individual datasets that perform better in individual case, the evaluation has shown that a uniform configuration

\[\text{Our Implementation: } \text{https://github.com/tub-aset/spreadingactivation}\]
Table 1: Vocabulary of all Datasets

|                  | Infusion P. | CCHIT | GANNT | CM-1 | WARC | WARCf |
|------------------|------------|-------|-------|------|------|-------|
|                  | high       | low   | high  | low  | high | low   |
| Stopwords        | 1,090      | 486   | 875   | 9,621| 228  | 635   |
|                  |            |       |       |      | 1,954| 6,874 |
|                  |            |       |       |      | 355  | 512   |
|                  |            |       |       |      | 404  | 654   |
| Token            | 2,325      | 948   | 1,560 | 16,311| 327 | 940   |
|                  |            |       |       |      | 3,159| 10,310|
|                  |            |       |       |      | 634  | 987   |
|                  |            |       |       |      | 745  | 1,239 |
| Token (SRL)      | 2,036      | 813   | 1,552 | 16,245| 275 | 936   |
|                  |            |       |       |      | 3,030| 9,502 |
|                  |            |       |       |      | 616  | 947   |
|                  |            |       |       |      | 741  | 1,230 |
| Sentences        | 296        | 52    | 121   | 1,167| 29   | 107   |
|                  |            |       |       |      | 276  | 1,019 |
|                  |            |       |       |      | 66   | 98    |
|                  |            |       |       |      | 66   | 99    |
| Sent. (SRL)      | 186        | 49    | 120   | 1,163| 29   | 106   |
|                  |            |       |       |      | 274  | 811   |
|                  |            |       |       |      | 60   | 85    |
|                  |            |       |       |      | 66   | 99    |
| Noun phrases     | 1,251      | 527   | 908   | 9,921| 215  | 595   |
|                  |            |       |       |      | 1,865| 6,150 |
|                  |            |       |       |      | 369  | 538   |
|                  |            |       |       |      | 417  | 668   |
| NP (SRL)         | 961        | 382   | 878   | 9,706| 170  | 578   |
|                  |            |       |       |      | 1,715| 5,139 |
|                  |            |       |       |      | 343  | 479   |
|                  |            |       |       |      | 408  | 640   |
| Lemma            | 2,325      | 948   | 1,560 | 16,311| 327 | 940   |
|                  |            |       |       |      | 3,159| 10,310|
|                  |            |       |       |      | 634  | 987   |
|                  |            |       |       |      | 745  | 1,239 |
| (unique)         | 646        | 225   | 563   | 1,978| 125  | 196   |
|                  |            |       |       |      | 696  | 1,718 |
|                  |            |       |       |      | 262  | 426   |
|                  |            |       |       |      | 287  | 462   |
| Representatives  | 46         | 45    | 15    | 157  | 6    | 29    |
|                  |            |       |       |      | 42   | 360   |
|                  |            |       |       |      | 8    | 15    |
|                  |            |       |       |      | 8    | 16    |
| Nominal          | 30         | 42    | 12    | 105  | 2    | 21    |
|                  |            |       |       |      | 20   | 283   |
|                  |            |       |       |      | 1    | 8     |
|                  |            |       |       |      | 1    | 8     |
| Pronominal       | 22         | 10    | 5     | 73   | 4    | 17    |
|                  |            |       |       |      | 17   | 87    |
|                  |            |       |       |      | 4    | 7     |
|                  |            |       |       |      | 4    | 8     |
| Proper           | 0          | 11    | 0     | 7    | 0    | 1     |
|                  |            |       |       |      | 9    | 202   |
|                  |            |       |       |      | 3    | 3     |
|                  |            |       |       |      | 3    | 3     |
|                  |            |       |       |      | 3    | 3     |

for all datasets provides almost equivalent results. That is probably because the basic characteristics of natural language (like the relation between semantic arguments) and trace links (e.g., related requirements use equivalent terms) are similar in different domains. Furthermore, we concluded that improvements for the approach will mainly be found in adjustments of the graph.

3 Datasets: Characteristics of Requirements

We use five common datasets of Huffman Hayes et al. [13] to evaluate our approach: Infusion Pump, CCHIT-2-WorldVista, GANNT, CM-1, and WARC. They come from different domains like health care, science, or business, and are different according to their size and scope of the requirements. We also used them in our previous work [23]. In the meantime we found out that the download link\(^5\) of Huffman Hayes et al. contains a faulty WARC dataset where several requirements missed parts of their content (i.e., the original dataset contains the identifier of a requirement as prefix followed by a dash but Huffman Hayes et al. incorrectly removed text before the last rather than the first occurrence of a dash). To circumvent this issue, we use a fixed version\(^6\), hereby called WARCf.

Table 1 contains a statistical overview of the vocabulary of all datasets, separated into high-level (query) and low-level (target) requirements. At first, we

\(^5\) [https://selab.netlab.uky.edu/AIRE-2019-hayes-payne-leppelmeier-meta-data.zip](https://selab.netlab.uky.edu/AIRE-2019-hayes-payne-leppelmeier-meta-data.zip)

\(^6\) [http://sarec.nd.edu/coest/datasets/WARC.zip](http://sarec.nd.edu/coest/datasets/WARC.zip)
removed all stop words using the NLTK\textsuperscript{7} stop word list. The suffix (SRL) for tokens means that there is an SRL argument that covers this token in terms of indices. For Lemma of tokens we also determined how many unique occurrences there are in order to draw conclusions about the reuse of words. To parse the natural language text, we use Stanford CoreNLP \cite{manning2014stanford} and DeepSRL \cite{amano2017deep}.

Table 1 reveals that DeepSRL does not consider all token and even miss several sentences (especially in Infusion Pump high and CM-1 low). These tokens and sentences are not parsable, e.g., the sentence does not contain an (identified) verb. The same observation also applies to noun phrases. On average, only 90\% of all noun phrases are covered by an SRL argument, with a minimum of almost 72\%. It is also evident that the lemma of tokens is often reused with an average of about 4 times to a maximum of more than 8 times.

Fig. 1 and 2 show the intersection of unique lemma vocabulary for all datasets. While the lemma of tokens is much more reused between high-level and low-level requirements, this is not the case for noun phrases.

4 Approach Revisions

Based on the insights in Section 3, we revise our approach to improve the performance on TLR. The analysis of the datasets shows that we need to focus more on single words than on whole phrases and also take the frequency into account. Furthermore, we need to consider parts that are not parsed by SRL.

4.1 Knowledge Base Construction

We reuse the NLP pipeline \cite{schlutter2020task} with the same techniques such as part-of-speech tagging, lemmatizing (morphological analysis), dependency parser (grammatical structure), coreference resolution and SRL. In contrast to our previous work, we changed significantly the building and the resulting structure of the graph.

\textsuperscript{7} https://www.nltk.org
Unlike before we choose a bottom-up approach, starting with single words. While we continue to believe that words like adjectives, verbs or adverbs without their context or role should not be used, all nouns (identified by their POS tag) are added as a vertex to the graph. On the next level, nouns are assembled into phrases. Therefore, all noun and person phrases (NP and PP within the dependency tree) that do not contain a verb phrase (VP) are also added as vertices to the graph and connected via edges to their contained nouns. If nouns or phrases occur multiple times, the same vertices are reused. To support this reuse, the lemma of words is used. In addition, phrases are trimmed based on the POS tag, e.g., determiners or prepositions at the beginning are removed.

Coreferences are also applied to the graph. If a phrase is pronominal, no vertex is added but instead the vertex of the representative is used. Other coreferences are depicted as edges between the reference and representative vertices.

Last but not least verb vertices are added to the graph. As before, we assume that a predicate describe a larger context between their arguments. This time, we do not add separate argument vertices but instead use directly the noun and phrase as argument vertices. Arguments are assigned to their verb in two ways. If there is an SRL predicate with an argument, that covers a phrase or a single noun, we add the verb as vertex and connect it via an argument edge labeled with the semantic role to the corresponding vertex of the noun or phrase. But if SRL was not able to find a predicate for an argument, the dependency tree is used to find a superordinated verb phrase (VP). In this case, the verb of the verb phrase is added as verb vertex and linked to the argument vertex without a certain semantic role label, i.e., we just use arg as any argument.

Verb vertices are only reused, if the lemma of the verb and its arguments are equal. The arguments are differentiated according to their type. If the verb has any noun or phrase argument, only they are used to check the equality of verbs. If no such argument is present, all other verb arguments are taken into account.

Adjectives and adverbs are also represented in the graph. While adjectives are implicit contained in phrases, adverbs are explicitly added as parts to the label of verb vertices. Again, if they are covered by an SRL argument, we use additional to the lemma itself the semantic role as prefix. If an adverb is not covered by SRL, we use the same process as for phrase arguments to search within the dependency tree for the corresponding verb.

To omit the distance between identifier and short phrase vertices, we eliminated the identifier vertices. Instead, the requirement identifiers are added as metadata to all vertices of a requirement. If a noun or phrase is used several times within a requirement, the identifier is also indicated several times. Thus, the frequency can be considered in the further course.

Fig. 3 shows the model and an abstract instance of the graph structure. While most parts are used by both requirements $req1$ and $req2$, therefore indicating a high relevance to each other, there are some vertices to which this does not apply. Vertex $noun1$ is only used once in requirement $req1$, likewise $phrase3$ is used twice only in $req2$. Although $noun1$ has no relation to the other words, its vertex
can be attributed to the corresponding requirement and may contribute to TLR, if in the further course more requirements are added that will also use it.

In contrast to our previous graphs, even if there is no verb within a sentence, other (relevant) parts like phrases or nouns are assigned to requirements and therefore taken into account for TLR. Adjectives and adverbs are closer the words they refer to. If SRL is not able to parse the sentence or single phrases, we use dependency relations to find the corresponding verb relation.

4.2 Semantic Search

The graph algorithm Spreading Activation consists of three phases. In the initial phase, the start vertices are activated, i.e., they will be assigned an initial activation value. During 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.

Due to the fact that there are no identifier vertices for requirements anymore, we need to change phase 1 and 3 of Spreading Activation. For the initial activation, we use the metadata by selecting the vertices that are included in the query requirement. To consider the frequency, vertices with multiple query occurrences are higher activated, i.e., we use tf-idf values [14] as initial activation values. The tf-idf values are also used in phase 3, where the target requirements are determined by adding the final activation values with tf-idf as factor.

In addition, we also adjust phase 2, the spreading process. We now consider the level/type of vertices and edges, e.g., a noun vertex might need a different treatment than a verb one or an A1 edge might be more relevant than an adjunct one. Therefore, we use different modes and parameter based on the type, except for $p_{max}$ as a global parameter. In case of the edge weight, we introduce a new parameter $W$ that replaces the CONSTANT factor of 1 (cf. [23, Table IV]) to
Improving TLR using Semantic Relation Graphs and Spreading Activation

5 Evaluation

We evaluate the approach for TLR on the five given datasets (Infusion Pump, CCHIT-2-WorldVista, GANNT and CM-1 of [13] as well as WAR Cf).

We build semantic relations graphs for each dataset, containing both high and low level requirement specifications. Table 2 gives an overview of the number of vertices and edges that have only high-level or low-level requirements as origin and intersecting elements (comparable to the Venn diagrams in Fig. 1 and 2).

To find an appropriate Spreading Activation configuration, we used an exploratory approach and discovered 500 random but valid configurations. While the modes are randomly chosen, the numerical parameter ranges are $d \in [0 - 1]$, $\tau \in [0 - 1]$, $p_{max} \in [1 - 50]$, and $W \in [0 - 1]$. A configuration is valid if spreading stops before the given number of pulses for all queries of all datasets. Otherwise, this indicates too strong restriction to spread activation and lead to no results.

As we use tf-idf values for initial activation, we need to adapt $\tau$, the minimum activation needed to spread in current pulse, for each dataset. Fig. 4 shows the distribution of tf-idf values for all datasets. While the basic shape is equal for all datasets, they all have their mean at a different level. If we would choose the same $\tau$ for all datasets, this would result in different limitations of the tf-idf activation values for the same configurations. Instead, we add per dataset a factor of three times the mean value to $\tau$, indicated as red lines in Fig. 4.

### Table 2: Semantic relation graphs

| Dataset    | Vertices |          |          |          | Edges         |          |          |
|------------|----------|----------|----------|----------|---------------|----------|----------|
|            | high     | both     | low      | high     | both         | low      |          |
| Infusion P. | 1,130    | 80       | 306      | 1,249    | 1,059        | 352      |          |
| CCHIT      | 655      | 238      | 5,552    | 582      | 5,137        | 6,894    |          |
| GANNT      | 156      | 34       | 431      | 141      | 378          | 564      |          |
| CM-1       | 1,096    | 378      | 4,514    | 1,094    | 5,765        | 5,120    |          |
| WAR Cf     | 294      | 149      | 591      | 252      | 803          | 582      |          |

Fig. 4: Distribution of tf-idf values

give various semantic roles different weight. To optimally map the relevance of semantic roles to $W$, we follow the formula in the PropBank Guidelines: $A0 > A1 > A2-A6 > AM$ [2, p. 8]. For the argument type $arg$, we use the $AI$ weight.
5.1 Metrics

There are several evaluation metrics that depend on different goals when evaluating TLR. Shin et al. [24] 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 requirements excluding irrelevant requirements to reduce unnecessary effort for human analysts. Goal 3 is to rank all requirements 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 requirements.

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 not applicable as it is a classification accuracy metric and not a rank accuracy metric [10]. 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 requirements) with the values 5, 10, and 30, which cuts the list after a fixed number of retrieved requirements. We justify this decision by assuming that the approach should be used in setups including a lot of requirements, but a user is not capable to check through thousands of potential candidates. A relative threshold such as 10% of the list would yield only 2 results for Infusion Pump but more than 40 for CCHIT.

In addition to AP and Lag, we evaluate the recall [14] at certain ND, i.e., the fraction of relevant links that are found until ND limit is reached. In case there are more valid links than ND, we use ND as maximum of relevant links.

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 artifacts that are linked because our approach does no classification and provides results for all queries (even for queries without any correct answer).

5.2 Results for Datasets

Fig. 5 shows the MAP, Lag and recall values at ND 5, 10, and 30 for each dataset. The numbers above and below are the respective top values. While the majority of configurations already show improved results, the best ones achieve a MAP roughly around 50% and a Lag around 30% on all ND thresholds, depending on the dataset. While Lag of CCHIT is comparable to the other datasets, the MAP indicates that correct results are not in top results but shown at a lower position.

Fig. 6 shows all search results for the CM-1 dataset as heat maps. The x-axis lists the 500 configurations, the y-axis contains 155 queries for each linked
Fig. 5: MAP and Lag at ND 5, 10, and 30

Fig. 6: Results of the CM-1 dataset

Fig. 7: Other results at certain ND

Fig. 8: Parameter distributions of top-50
high-level requirement. The color scheme indicates the recall, i.e., 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. The darker a column is, the better a configuration, and the darker a row it is easier to answer a query correctly. There are a few queries where, regardless of the configuration, valid links are found often, e.g., around query 135 and 75. 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., around 140. Comparable to this horizontal patterns, there are vertical ones with some of the best configurations around 300 and some of the worst around 160.

The corresponding heat maps of the other datasets (Fig. 7) have similar vertical patterns at the same places, especially for the configurations around 300. This leads us to the assumption that a configuration has a constant quality for different datasets. We already showed that the quality of configurations is quite constant for different datasets [23].

Table 3 shows the best metric values of all configurations for each dataset and threshold of our new approach compared to our previous approach [23, cf. TABLE VIII], with the deviation by how much the metric value have improved. The overall best configuration is determined by the best harmonic mean between MAP and Lag. In all cases the new approach has better results in metric values
Table 3: Best configurations

| Per metric     | Inf. P. | CCHIT | GANNT | CM-1 | WARCf |
|----------------|---------|-------|-------|------|-------|
| ND             | 5       | 10    | 30    | 5    | 10    | 30    |
| New approach   | MAP .51 | .53   | .54   | .24  | .20   | .20   |
|                | Lag .43 | .45   | .50   | .41  | .18   | .22   |
|                | Recall .74 | .90   | 1.0   | .32  | .32   | .43   |
| Improvement to [23] | MAP .08 | .08    | .07   | .13  | .10   | .06   |
|                | Lag .07 | .09   | .16   | .10  | .10   | .11   |
| Overall        | MAP .52 | .53   | .54   | .22  | .18   | .19   |
|                | Lag .43 | .43   | .45   | .54  | .47   | .46   |
|                | Recall .73 | .84   | .92   | .29  | .29   | .41   |
| Improvement to [23] | MAP .12 | .10   | .10   | .14  | .11   | .12   |
|                | Lag .09 | .10   | .10   | .17  | .14   | .13   |

with an average increase of almost 10%. The WARC dataset has improved even more, but this is due to the fact that the old evaluation used the unfixed version. Except for CCHIT, the recall is above 90% at ND30.

5.3 Limitations

In our previous work, we stated that there are mainly two parts of the approach that affect the performance results, the graph and Spreading Activation. As we have already seen in Section 3, also the datasets with their characteristics need to be taken into account to have an influence on the performance. Even though we have tried to take a closer look at them and to address or avoid certain problems, there will still be some that have a negative impact.

For example, the datasets reveal at a brief overview that some of the sentences or statements are formulated in a very complex way or contain a list of key points. This may lead to heavily branched (verb) vertices, which might be handled differently by Spreading Activation than other (simpler or more common) sentence structures, e.g., the initial activation might be broadly distributed and lead to super-spreading of those vertices or to no spreading at all due to a high $\tau$.

In addition, we rely on the quality of the NLP tools, especially the complex tasks SRL and dependency trees. If they fail to capture the natural language, these errors will also appear in the graph and probably result in a lower performance.

Fig. 8 to 11 show the parameter distribution of top-50 configurations for all levels and types. Fig. 8 reveals that the first argument is most important, while adjunct arguments seem to almost be ignored. This insight does not take place in the graph structure, e.g., arguments are considered uniformly to find identical verbs. The parameter $\tau$ seems to have a low limit below 0.5 (at least for nouns and verbs), this may be due to the fact that three times the mean tf-idf value as upper limit is too restrictive in combination with a high $p_{max}$. 
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. There are different approaches to identify such semantic similarities like word embeddings (word2vec [16], GloVe [18]) or databases (WordNet\(^8\)). While word embeddings demand additional computation and a reasonably large dataset, databases often contain only general but not domain-specific terms.

Overall, it is difficult to determine the influence of individual factors or their combination, since determining a good and valid configuration for Spreading Activation by random selection requires a lot of computation. It is possible that some of the improvements presented are also counterproductive, but we are not able to verify this in detail. Borg et al. [3] stated that TLR as an IR-problem is often based on empirical software engineering research with small datasets and therefore may conduct in “the cave of IR evaluation”.

6 Discussion

We achieve improved results in the evaluation without any explicit assumptions on the requirements (e.g., certain patterns like user stories). This time we included a common IR-solution, the tf-idf values into our approach to improve the quality. 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 seem to fit as there are (local) maxima identifiable in Fig. 8 to 11. Table 3 shows that the performance of the overall best configuration is comparable to configurations with an individual best metric result. We assume that an almost optimal configuration, once determined, may be reused for other datasets as well.

Unfortunately, as we have already shown in our previous work [23], there are currently few publications for comparison of our approach. Most of them do not use the datasets or only report precision, recall, or F1-score. Some approaches also require manual effort, which may be very specific to the dataset in question. For comparison, we used a syntactical VSM approach with tf-idf values for lemmas and compare requirements using cosine similarity [20,25].

Compared to our previous work [23], the results have improved. While the top values of MAP and Lag only slightly changed, the average performance of a random configuration is greatly increased (cf. Fig. 5d and 6 and [23, Fig. 10d and 11]). This is due to the fact that MAP and Lag have non-proportional weights, which causes strong variation due to changes at the top of the result list, but only small in the further part. In contrast, the syntactical approach still outperforms the semantic approach in terms of MAP and Lag but this time our approach is

\(^{8}\) https://wordnet.princeton.edu/
quite close to baseline performance. From a practical point of view, the recall of about 90% means that an engineer will have seen almost all links in the sorted candidate list after only 30 items, independent of the position within the list. Huffman Hayes et al. [11] rate a recall above 80% as excellent, which we achieve for every dataset except CCHIT with ND30.

7 Conclusion

In this paper, we improve our 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 better results than before by incorporating some general characteristics of the datasets and their requirements. In addition, we improved Spreading Activation by using a modified graph structure, tf-idf as activation values and consider even more semantics of natural language.

References

1. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering Trace-ability Links between Code and Documentation. Transactions on Software Engineering 28(10), 970–983 (2002). https://doi.org/10.1109/TSE.2002.1041053
2. Bonial, C., Bonn, J., Conger, K., Hwang, J., Palmer, M., Reese, N.: English PropBank Annotation Guidelines (2015), https://raw.githubusercontent.com/propbank/propbank-documentation/master/annotation-guidelines/Propbank-Annotation-Guidelines.pdf
3. Borg, M., Runeson, P., Ardö, A.: Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering 19(6), 1565–1616 (2014). https://doi.org/10.1007/s10664-013-9255-y
4. Borgida, A., Greenspan, S., Mylopoulos, J.: Knowledge Representation as the Basis for Requirements Specifications (1985). https://doi.org/10.1007/978-3-642-70840-4_13
5. Dermeval, D., Vilela, J., Bittencourt, I.I., Castro, J., Isotani, S., Brito, P., Silva, A.: Applications of ontologies in requirements engineering: a systematic review of the literature. In: Requirements Engineering. pp. 405–437. Springer (2016). https://doi.org/10.1007/s00766-015-0222-6
6. Gotel, O., Finkelstein, C.W.: An Analysis of the Requirements Traceability Problem. In: Requirements Engineering (1994). https://doi.org/10.1109/ICRE.1994.292398
7. Hartig, K.: Entwicklung eines Information-Retrieval-Systems zur Unterstützung von Gefährdungs- und Risikoanalysen. Ph.D. thesis, Technische Universität Berlin (2019). https://doi.org/10.14279/depositonce-8408
8. He, L., Lee, K., Lewis, M., Zettlemoyer, L.: Deep Semantic Role Labeling: What Works and What’s Next. In: Association for Computational Linguistics. pp. 473–483. ACL (2017). https://doi.org/10.18653/v1/p17-1044
9. Heindl, M., Biffl, S.: A Case Study on Value-based Requirements Tracing. In: European Software Engineering Conference (2005). https://doi.org/10.1145/1081706.1081717
10. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating Collaborative Filtering Recommender Systems. Transactions on Information Systems 22(1), 529–565 (2004). https://doi.org/10.1145/963770.963772
11. Huffman Hayes, J., Dekhtyar, A., Sundaram, S.K.: Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. Transactions on Software Engineering 32(1), 4–19 (2006). https://doi.org/10.1109/TSE.2006.3
12. Huffman Hayes, J., Dekhtyar, A., Osborne, J.: Improving Requirements Tracing via Information Retrieval. In: Requirements Engineering (2003). https://doi.org/10.1109/ICRE.2003.1232745
13. Huffman Hayes, J., Payne, J., Leppelmeier, M.: Toward Improved Artificial Intelligence in Requirements Engineering: Metadata for Tracing Datasets. In: Artificial Intelligence for Requirements Engineering, pp. 256–262. IEEE (2019). https://doi.org/10.1109/REW.2019.00052
14. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press (2008). https://doi.org/10.1017/CBO9780511809071
15. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The Stanford CoreNLP Natural Language Processing Toolkit. In: System Demonstrations, pp. 55–60. ACL (2014)
16. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient Estimation of Word Representations in Vector Space. Computing Research Repository (2013), https://arxiv.org/abs/1301.3781
17. Mills, C.: Towards the Automatic Classification of Traceability Links. In: Automated Software Engineering (2017). https://doi.org/10.1109/ASE.2017.8115723
18. Pennington, J., Socher, R., Manning, C.D.: GloVe: Global Vectors for Word Representation. In: Empirical Methods in Natural Language Processing, pp. 1532–1543. ACL (2014). https://doi.org/10.3115/v1/d14-116
19. Robeer, M., Lucassen, G., van der Werf, J.M.E.M., Dalpiaz, F., Brinkkemper, S.: Automated Extraction of Conceptual Models from User Stories via NLP. In: Requirements Engineering, pp. 196–205. IEEE (2016). https://doi.org/10.1109/RE.2016.40
20. Salton, G.: Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley Longman Publishing Co., Inc. (1989)
21. Schlutter, A., Vogelsang, A.: Knowledge Representation of Requirements Documents Using Natural Language Processing. In: Natural Language Processing for Requirements Engineering. RWTH Aachen (2018). https://doi.org/10.14279/depositonce-7776
22. Schlutter, A., Vogelsang, A.: Knowledge Extraction from Natural Language Requirements into a Semantic Relation Graph. In: Knowledge Graph for Software Engineering. ACM (2020). https://doi.org/10.14279/depositonce-9772.2
23. Schlutter, A., Vogelsang, A.: Trace Link Recovery using Semantic Relation Graphs and Spreading Activation. In: Requirements Engineering, pp. 20–31. IEEE (2020). https://doi.org/10.1109/RE48521.2020.00015
24. Shin, Y., Huffman Hayes, J., Cleland-Huang, J.: Guidelines for Benchmarking Automated Software Traceability Techniques. In: Symposium on Software and Systems Traceability (2015). https://doi.org/10.1109/SST.2015.13
25. Singhal, A.: Modern Information Retrieval: A Brief Overview. Computer Society Technical Committee on Data Engineering 24(4), 35–43 (2001), http://singhal.info/ieee2001.pdf
26. Winkler, S., von Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. Software Systems Modeling 9, 529–565 (2010). https://doi.org/10.1007/s10270-009-0145-0