Traceability Network Analysis: A Case Study of Links In Issue Tracking Systems

Alexander Nicholson, Deeksha M. Arya, Jin L.C. Guo
School of Computer Science, McGill University, Montreal, Canada
{*alexander.nicholson, deeksha.arya*}*@mail.mcgill.ca*, *jguo@cs.mcgill.ca*

***Abstract*—Traceability links between software artifacts serve as an invaluable resource for reasoning about software products and their development process. Most conventional methods for capturing traceability are based on pair-wise artifact relations such as trace matrices or navigable links between two directly related artifacts. However, this limited view of trace links ignores the propagating effect of artifact connections as well as the trace link properties at a project level. In this work, we propose the use of network structures to provide another perspective from which reasoning on a collective of trace events is possible. We explore various network analysis techniques in the issue tracking system of sixty-six open source projects. Our observation reveals two salient properties of the traceability network, i.e. scale free and triadic closure. These properties provide a strong indication of the applicability of network analysis tools and can be used to identify and examine important "hub" issues. As a stepping stone, these properties can further support project status analysis and link type prediction. As a proof-of-concept, we demonstrate the effectiveness of applying the triadic closure property to link type prediction.**

***Index Terms*—software traceability, network science, graphs, link prediction**

I. Introduction

Traceability is the ability to construct, maintain, and use software artifact connections during software development. In highly regulated domains, such as aircraft [30] and medical device software [6], traceability is strictly required by regulatory authorities to demonstrate that the function and quality of the software products have met prevailing standards. In other domains, traceability has also been adopted to support a variety of tasks including code comprehension, impact analysis and test coverage analysis [13]. Semantically-typed trace links, in particular, provide rich descriptions of artifact relations. These link types help users to both reason about trace link validity and use trace links effectively. Based on the relevance to the task, users can examine specific types of links while ignoring others.

One of the main barriers to seeing the advantages of traceability is that creating and maintaining a complete set of trace links throughout the lifetime of a project is an expensive undertaking [23]. For this reason, automated trace-link evaluation solutions are particularly attractive [7]. In the case of projects with historical data, data-driven traceability methods have been proposed to recover and predict trace links [29]. Most of those methods extract features from individual artifact pairs, either from artifact texts or meta data, and then apply supervised learning techniques to classify whether valid trace links exist between those pairs. However, there are two major limitations to those approaches. Firstly, trace link types are normally ignored during the automated trace link evaluation process. Secondly, those methods are based on a limited view of trace links, i.e. only the pair-wise relations between artifacts are used. The rich structure of artifact connections across projects is not considered. In this work, we aim to fill these gaps by analyzing the entire trace link network structure including semantic link types.

Network structure has provided insight into the principles that govern the entity relations and interactions in many problem domains, such as transportation systems [9], neural activity [33] and research collaborations [10], to name a few. From a visualization perspective, the topology of a network can make recurring phenomena more readily apparent than summary statistics or a strictly tabular view, and hence many tools have been created to better visualize network structures [1, 3, 4]. Additionally in certain types of networks, the existing connection structure is frequently indicative of the future structure. These network types have been found to recur in a number of areas of inquiry, independent of the attributes or complexity of the entities within the network. A network structure view of software projects enables us to examine whether the implications of these network types typically hold in software projects as well. We can adopt tools and terminology from network science for describing and inspecting the trace link relations between software artifacts in more detail.

For many open-source projects, issue tracking systems such as JIRA and Github are the primary project management tools. Consequently, much of the recorded data stored about these projects can be found in the form of issues and their associated metadata and events. Issues can be used to report bugs, request new features, accumulate feedback for early design solutions [5]. Many issue trackers also provide configurable trace link solutions including customized link types. Developers can use these link types to define relationships between issues that arise from their own work process or from the complexity of the software itself. This built-in trace link capability in addition to the existing data about interactions between various stakeholders and software artifacts are two examples of relational data in issue trackers that can be represented using a network structure to enlighten the traceability process.

In this work we investigate whether network analysis tech-

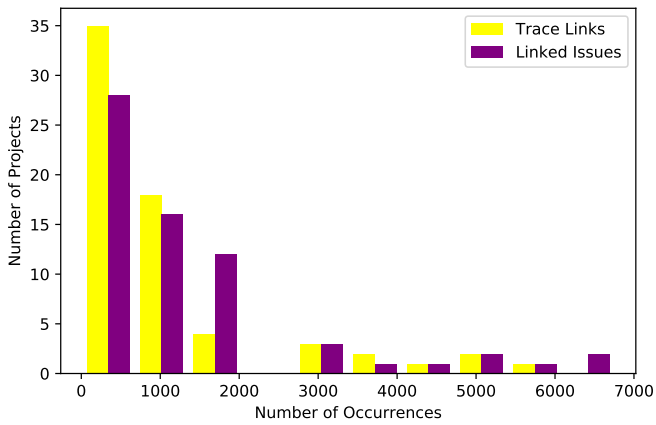


Fig. 1: Number of Linked Issues and Trace Links across projects.

TABLE I: The distribution of link types across projects. JIRA built-in link types are in bold.

Link	Occurrences	Percentage of Total	In Number of Projects.
relates to	**32386**	**45.56%**	**65 (98.48%)**
duplicates	**12696**	**17.86%**	**66 (100.00%)**
blocks	**6669**	**9.38%**	**63 (95.45%)**
depends upon	6269	8.82%	62 (93.94%)
incorporates	3683	5.18%	62 (93.94%)
breaks	2456	3.45%	53 (80.30%)
contains	1771	2.49%	49 (74.24%)
requires	1731	2.44%	53 (80.30%)
is a clone of	**1593**	**2.24%**	**63 (95.45%)**
supercedes	1359	1.91%	57 (86.36%)
is a parent of	169	0.24%	20 (30.30%)
Blocked	167	0.23%	41 (62.12%)
Dependent	59	0.08%	18 (27.27%)
causes	54	0.08%	18 (27.27%)
Parent Feature	14	0.02%	10 (15.15%)
Dependency	11	0.02%	4 (6.06%)

niques [28] help in analyzing the nature of trace links and their types in open source projects. Specifically, we answer the following research questions:

- **RQ1**: *Do issue traceability networks demonstrate salient network properties that could assist in analyzing issues and their relations?*
- **RQ2**: *To what extent can we leverage those properties for traceability tasks such as link type prediction if such properties can be observed?*

The main contributions of this paper are as follows. We perform a case study exploring the issue tracker data of sixty-six open source projects from JIRA, the characteristics of which are described in Section II-A. We have made the dataset available to support reproducibility of this work as well as future research in this domain.[1] With this data, we first define and construct issue traceability networks from manually tagged trace links, detailed in Section II-B. We then analyze two important network properties. First, in Section III we explore the quantitative nature of trace link connections and show that issues with a high number of connections are central to both the workflow and arrangement of a project's requirements. We also show that software projects show similar connection structures to networks in other domains and this similarity makes identifying highly connected issues straightforward. Second, in Section IV, we study the transitivity property of links. We show that triadic closure can be used to predict trace link types with 72% accuracy. We discuss our findings for our research questions in Section V.

II. Traceability Network Construction

A. Dataset

Using the Jira Rest API[2], we collected the details of all the historical issues of sixty-six Apache Open Source projects, in February 2018. Each issue is characterized by its textual content including a summary and description of the issues, as well as several metadata items including the issue type (i.e. *Bug, New Feature, Sub-task, Improvement*, etc.), creator, assignee, creation date, resolution (i.e. *Unresolved, Fixed, Duplicate, Won't Fix*, etc.) and resolution date. For the trace links between issues, we extracted their semantic types that were tagged by the project contributors. To encourage reproducibility and further research, we have made this data publicly available.[1]

Across the sixty-six projects, there is a total of $71,087$ trace links between $90,936$ linked issues. Quantities range from between 3 and 5811 trace links and between 5 and 6760 linked issues per project. Figure 1 shows the distribution of linked issue and trace link frequencies across projects.

While JIRA provides four default trace link types, project administrators are able to create additional types to suit organizational or team needs. Within our dataset, there are sixteen unique trace-link types with a wide range of frequencies between projects. Table I shows the distribution of each link type across projects, with the JIRA defaults in bold. It is evident that the distribution of link-types is highly skewed with the top three types accounting for 72.8% of all links. Recent work has also observed this long-tailed link type distribution [35].

B. Network Construction

The **Issue Traceability Network** is a representation of trace links existing between issues. We define this network as a set of nodes V and edges E. Each node v, where $v \in V$, corresponds to an issue from our dataset. An edge e, defined by $e = \langle v_1 \in V, v_2 \in V \rangle$, where $e \in E$, corresponds to a trace link between two issues. We use the link type that was tagged by the project contributors, such as *relates to*, *blocks*, *depends upon*, etc. to label the edges in the traceability network. A number of trace link types that are directional, e.g. $blocks(A, B)$ and that necessarily imply an inverse relationship such as $blocked\_by(B, A)$. For consistency, we collapse these forward and backward link

[1] https://doi.org/10.5281/zenodo.3942332
[2] https://docs.atlassian.com/software/jira/docs/api/REST/7.8.0/

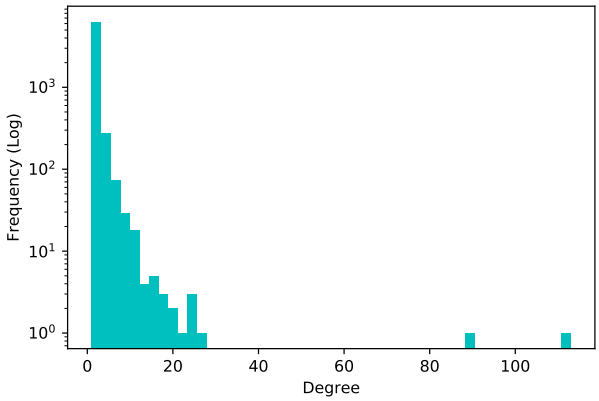


Fig. 2: Degree distribution of the HIVE project.

types into singular, undirected edge labels. The network graphs in this work were implemented using python and the open source NetworkX library[3].

III. Degree Distributions of Traceability Networks

Projects vary depending on factors such as the nature of the project, team conventions and the current development stage. However, do the issue traceability networks of different projects exhibit certain similarity in their structures that we can use for project management? In this section, we quantitatively examine the similarities between trace link networks created from each project in our dataset.

A. Power Law Degree Distributions

The degree of a node refers to the number of edges it has with other nodes in the network. The degree distribution $P(k)$ then is concerned with how often each degree value occurs. More formally, $P(k)$ is calculated as $N(k)/|N|$, or the ratio of the number of nodes with a degree of k to the total number of nodes in the network. For example, in a fully connected network, i.e. one in which each node is connected to all other nodes, each node has a degree of $|N|-1$ and the degree distribution is centered on this degree value. This ratio can also be seen as the empirical probability that a random node in the network will have k edges. This probabilistic perspective becomes useful when analyzing the highly connected issues within the network.

We find that the degree distribution of a number of projects follows a power law. That is, $P(k) \sim k^{-\gamma}$ for some value of γ known as the scale coefficient or scale exponent. As an example, we show the degree distribution of the *HIVE*[4] project ($\gamma = 2.16$) in Figure 2. We note that as the degree value increases, the proportion of nodes with that degree decreases exponentially. Figure 3 shows the distribution of

[3]https://networkx.github.io/

[4]https://jira.apache.org/jira/projects/HIVE/

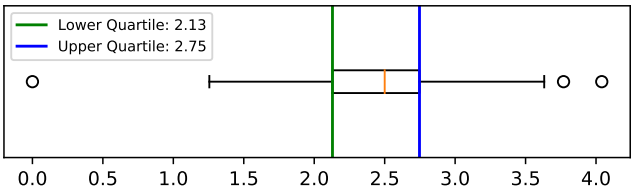


Fig. 3: The distribution of γ-values across projects.

γ-values across all sixty-six open source projects. One proposed explanation for power law degree distributions is the concept of a scale-free network [2]. A scale-free network can be interpreted as a network in which the shape of the degree distribution is invariant to the scale (number of nodes and connections) of the network. Scale free networks have been observed in many different domains, such as procedure call networks [26], actor co-starring networks and scientific literature citation networks [2].

One aberration in our dataset is the IBATIS[5] project which has $\gamma = 0$. Upon further inspection, the dataset only contains 17 links between 34 issues despite the fact that there are 777 issues in total. Such an anomaly might be an indication that issue links are not actively created or maintained in this project.

B. Identifying Issue Hubs

One implication of a power law degree distribution is that a small number of nodes have a much larger number of connections than the majority of nodes. These highly-connected nodes can be considered as *hubs* of trace link activities. The objective of the following experiment is to identify such issue hubs and to discover to what extent their characteristics are distinct.

The challenge in identifying hubs is that the minimal number of connections a hub has may vary with the size of the network. To overcome this, we consider the following approach to identify hubs: we first rank the issues within a project using the links they have in descending order; we then consider the highest ranked issues that in total cover 90% of the link activity as the issue hubs in that project.

C. Results and Analysis

We identify hubs in sixty-five projects (with the exception of the IBATIS project). The mean number of hub issues per project is 18, with a mean of 10 trace links per hub (compared to one link per issue in general). On average, hubs make up 1.79% of the issues within a project, ranging from $0.28-20\%$. This reinforces the idea of hubs as rare but central points of activity within the network resulting from a power law degree distribution.

Figure 4a shows the distribution of the types of links both hub and non-hub issues tend to have. While both hubs and non-hubs maintain *relates to* as the most frequently occurring link

[5]https://jira.apache.org/jira/projects/IBATIS/

TABLE II: Additional statistics about Hubs and Non-hubs.

	Hubs	Non-hubs
Median number of issues a creator opens	**34**	3
Total number resolved	959 (80.86%)	**77399 (85.11%)**
Total number unresolved	**227 (19.14%)**	13537 (14.89%)
Average number of tokens (text length)	**102**	95
Median resolution time (days)	**159.50**	55.25

type, the distributions of the remaining link types vary. Non-hubs contain more duplication while hubs exceed in a number of link types including *blocks*, *depends upon*, *incorporates*, *requires* and *contains*.

The top five most common issue types is the same for both hub and non-hub issues, although they present at different frequencies (see Figure 4b). *Improvements* were the most frequent types for hubs, closely followed by *Bugs* and *New Features*. On the contrary, *Bugs* are the dominant type for non-hubs with a frequency close to 50%.

We also compare the resolution status for hub and non-hub issues in Figure 4c. The distributions are overall similar. The most common resolution status are both *Fixed* followed by *Unresolved*. Among other possible statuses, hubs are much less likely to be resolved as *Duplicate* compare to non-hubs.

Furthermore, we compare several additional statistics between the hubs and non-hubs in Table II. Our results reveal that hubs take on average almost three times longer to resolve if they are eventually resolved. Hub creators are also markedly more active the non-hub creators. Hubs also have on average slightly more textual content than regular issues.

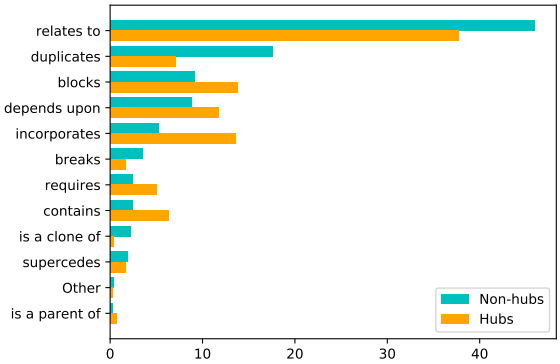
IV. Triadic Closure of Traceability Networks

The previous section has primarily focused on the distributions of issues as network nodes, as well as several important characteristics of those nodes. However, analysis of a network with respect to the edges is also essential for understanding the properties of the network. Further, edge-centred data analysis approaches have shown potential for prediction tasks in social networks in the past [21]. In this section we focus on analyzing the distribution of semantically typed links as network edges based on the *Triadic Closure* property, and discuss the potential implications of our observations.

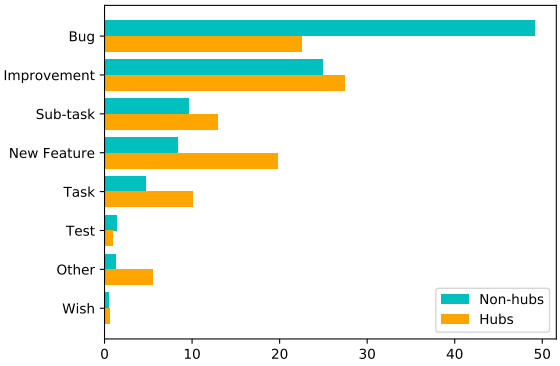
A. Triads

A triad refers to a group of three nodes: A, B, C when there are direct or indirect links between any two of the nodes. A closed triad indicates that the three nodes are fully connected, i.e. all possible pairs of nodes are connected to each other via a link. Hence, a closed triad is a triangular structure. In contrast, open triads are groups of three nodes in which only two links exist, i.e. $link(A, B)$ and $link(B, C)$, but a link between A and C does not exist. Figure 5 illustrates the types of triads.

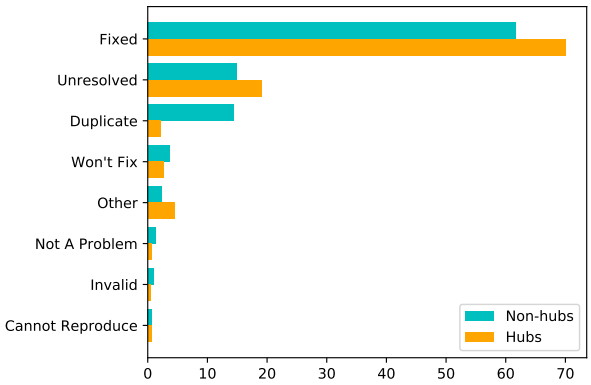
Triadic closure is the transitive property of links in a network. That is, it is the property such that among nodes A, B, and C, if $link(A, B)$ and $link(B, C)$ exist, then there exists $link(A, C)$, the link between nodes A and C. This property



(a) Link Semantics of Hubs and Non-hubs.



(b) Issue Types of Hubs and Non-hubs.



(c) Resolution Status of Hubs and Non-hubs.

Fig. 4: Comparison of Issues that are identified as Hubs versus Non-hubs

has been discovered and utilized in various studies on social network analysis [14, 19] in both directed and undirected network settings [22]. If *triadic closure* can be observed in the traceability networks, then it can be used to infer new link between two issues when they are indirectly linked through a third issue.

Figure 6 displays the distribution of the number of open triads with respect to the number of edges in each of the projects studied in this work. We find that the number of open triads is in general, large, and increases exponentially

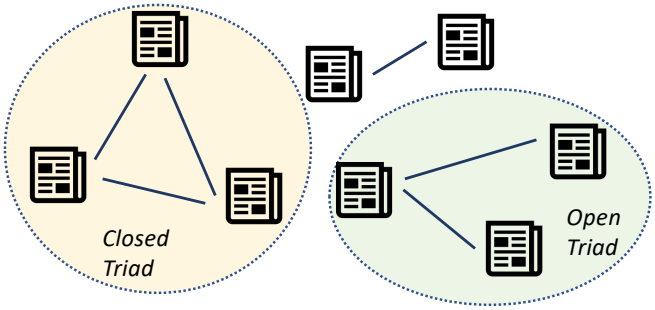


Fig. 5: Illustration of closed triad, i.e. three nodes with direct links and open triad, i.e. one direct link between two nodes is missing

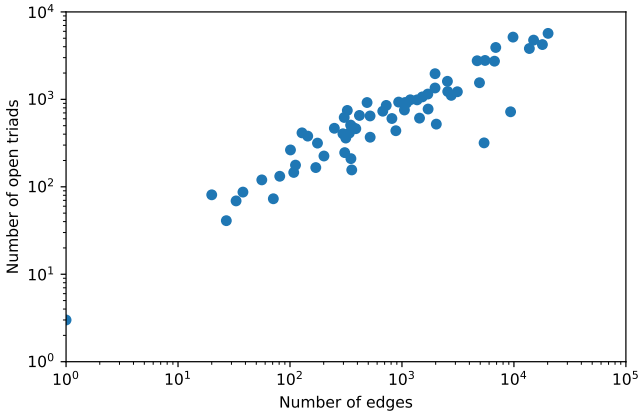


Fig. 6: The distribution of the number of open triads with respect to the number of edges for each of the projects.

as the size of the graph increases. This is likely because link creation remains a manual process requiring a large amount of time and effort of comparing a new issue with all the existing issues to determine relevant links. This finding shows potential for *triadic closure* to predict new links in our dataset.

Triads in networks with semantically typed edges are of particular interest. Given that links between issues are semantically typed, if there are recurring patterns of link type compositions of triads, then such patterns can be used to predict not only the existence of links but also the specific types of the links. Furthermore, certain patterns can be indicators of potential concerns for the project. For example a triad consisting of three *blocks* links might denote a deadlock between the three constituent issues.

B. Link Type Prediction

We investigate to what extent triadic closure can be observed in the traceability network and whether certain patterns of link type compositions emerge from the observed triads. We also assess the accuracy when using these patterns to predict the link semantic types.

To leverage triadic closure for the prediction of link types, we first identify all the closed triads in the traceability networks constructed in Section II-B. We then extract link type triples $\{link_1, link_2, link_3\}$ from the closed triads and rank them according to the frequency of their appearance, denoted by $Count(\{link_1, link_2, link_3\})$. We use $Count(\{link_1, link_2\})$ to indicate the total number of appearance of $\{link_1, link_2\}$ regardless of $link_3$. The closure property of the triad is measured as $Count(\{link_1, link_2, link_3\})/Count(\{link_1, link_2\})$.

We then evaluate the effectiveness of identifying and using triad triples as heuristics to predict link types. We randomly split the traceability network into a training set that contains 80% of the links and testing set that contains 20% of the links. The training set is used to discover the heuristic in the form of type triples $H_{eu} = \{link_1, link_2, link_3\}$ as described earlier. We then apply the heuristics to all the nodes in the testing set: if among three nodes A, B, C there exist $link_1(A, B)$ and $link_2(B, C)$, we make a prediction using the highest ranked heuristic H_{eu} from the training set. The accuracy of H_{eu} is then calculated as $Count(H_{eu})$, divided by the $Count(\{link_1, link_2\})$ in the testing set. This experiment is performed considering two scenarios: discovering heuristics across all projects and discovering heuristics within a single project. In both scenarios, the accuracy is averaged across all heuristics.

C. Results and Analysis

The five most frequent triples are displayed in Table III along with the number of appearances and their closure property. The top triple has a dominant number of 1434 appearances. The closure property reveals the fact that among closed triads, when Issue A is *related to* Issue B, and B is *related to* Issue C, then 88.74% of the time Issue A is also *related to* Issue C in the existing traceability network. While the triple of $\{depends\_upon, incorporates, incorporates\}$ appears less frequently, i.e., 298 times, its closure measure is as high as 97.70%. Given that all the top triples demonstrate strong closure measure (above 60%), those triples could potentially serve as useful indicators of semantic link type prediction.

Indeed, when we use 80% of the randomly selected trace links across all project to discover triad heuristics, those heuristics achieve on average 71.88% accuracy to predict trace link types in the remaining 20% dataset. In comparison, a random classifier would only predict a link type with 6.25% average accuracy. In the within project prediction setting, the average accuracy per project is 60.56% with a standard deviation of 0.3984. To further assess the gene of this method, we compare the learned heuristics of each project, i.e. project context, with those learned from across all project, i.e. the global context. We use $H_{project} \cap H_{global}$ to denote the overlap between heuristics learned from project context and global context. Each project is mapped to a point in Figure 7 with is value in the x-axis calculated as $|H_{project} \cap H_{global}|/|H_{global}|$ and value in y-axis as $|H_{project} \cap H_{global}|/|H_{project}|$. We observe that most projects have at least 50% of their heuristics appearing in the global context with a few exceptions. A few

TABLE III: Table showing top five heuristics by frequency across all projects and their closure measure.

$Link_1$	$Link_2$	$Link_3$	Freq.	Closure Measure
relates to	relates to	relates to	1434	88.74%
depends upon	depends upon	depends upon	570	93.44%
depends upon	incorporates	incorporates	298	97.70%
blocks	blocks	blocks	103	62.42%
duplicates	relates to	relates to	92	67.15%

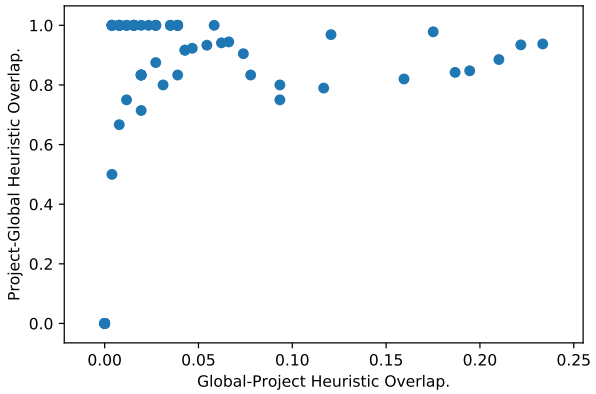


Fig. 7: Fraction of project learned heuristics found in the global context vs. Fraction of globally learned heuristics found in the project context.

projects also contain as many as 20% of the global heuristics. Those results suggests that closed triads discovered in the project context are not only helpful for predicting trace link types within that project, but also can be used to enrich heuristics in global context, and vice-versa.

V. Discussion

Modelling trace links and other issue tracking data using a network structure gives us a higher level view of the relational patterns found within software projects. We discuss our findings for each research question below.

RQ1: *Do issue traceability networks demonstrate salient network properties that could assist in analyzing issues and their relations?*

We find that issue traceability networks show the *scale-free* network property. The implication of a network being scale-free is that such a network contains a few highly-connected nodes, known as hubs, while most nodes have low connectivity. This is a favorable property for many real world networks as opposed to highly interconnected ones since this allows the network to be robust to failure. For an issue traceability network, it indicates that a problem in resolving most non-hub issues will not impact most other issues.

The knowledge of which issues are hubs can play a major role in prioritizing effort during issue triaging and resolving. Our analysis reveals that the hubs are less likely to be duplicated to other issues and are more likely to have critical connections with other issues (e.g. *blocks*, *depends upon*, *incorporates*). Resolution on those issues, therefore, will make a great impact on project development and maintenance.

RQ2: *To what extent can we leverage those properties for traceability tasks such as link type prediction if such properties can be observed?*

We find that the transitive triadic closure property of networks can be used in the form of closure heuristics to predict the type semantics of links. By applying the heuristics collected from a large set of projects, we achieved a 72% accuracy on link type prediction. The project-specific heuristics can also predict the link types with 61% accuracy. The considerable amount of overlapping between global and project-specific heuristics for triadic closure indicates that despite occurring in different projects, some link types are used consistently and their relations can collectively inform expected links in other projects.

Our findings prompt the exploration of related characteristics to the network properties presented in this work. For example, hubs in a graph could have been created over time by the property of *preferential attachment* in which a node is more likely to connect to another node that already has a large number of links. Preferential attachment informs new links and hence has been used in prior work for link prediction in networks [21]. For software traceability, representation of traceability graphs as networks opens doors to new techniques for leveraging network properties for traceability tasks.

That the use of heuristics for link type prediction across project achieves higher accuracy of 71.88% as opposed to within-project with 60.56% accuracy shows promise for generalizability of issue link traceability research. Same link types in projects are largely semantically similar to those in other projects, assuring a consistency in link type use in different projects. This information is integral, since future work can hope to standardize link types across all open source projects with the intention of creating a generalized mechanism for link type prediction.

VI. Threats to Validity

There are two main threats to the validity of this work. Firstly, our work only analyzed open source projects using the Apache JIRA Issue Tracker. There are potentially consistencies within Apache open source projects that do not appear in other open source project ecosystems. Given the size of the Apache organization however, analyzing a large number of projects may have served to mitigate this limitation. This analysis could be made more robust by including the same or similar data from other issue trackers. We note that the type of data available within different issue trackers may also vary. In this work, we leverage only the links and the link types which are retrievable from other common issue trackers. However, utilising less common metadata items in our dataset may render some analysis not transferable to other issue tracker datasets.

The second threat is that of project variability. Although we have shown both link types and issue types in aggregate, the data does not reveal the project team's interpretation of these custom data points. Furthermore, while in general we observe interesting trends from applying network structure across projects, we find it difficult to make recommendations for projects that do not exhibit favourable network properties. Projects having a very small number of connected issues, or a network with mostly isolated groups of issues prove ambiguous. We note however, that this might be an artifact of mismanaged or poor existing trace data for the project. However, since such projects were found to be extremely rare, we remain optimistic that network structure is a favourable perspective for issue traceability and remains an avenue for much future work.

VII. Related Work

A. Software Traceability

In the field of software engineering, traceability has become an integral part of the development and maintenance process [32]. Cleland-Huang et al. performed an in-depth analysis of recent research in the field of software traceability to discover that work has focused on tooling and visualization, applying traceability in specific domains, and using information retrieval techniques to at least semi-automate trace link creation [7].

Tomova et al. assessed the connection between issue trace link types *clones* and *relates* and the textual similarity between issues [35]. Their application of information retrieval techniques reveal that the *cloned* issues do not necessarily have the highest textual similarity and *related* issues have varying levels of similarity across projects. This provides an interesting implication that textual similarity is not necessarily a good indicator of link semantics between issues, opening avenues for research on other contributing factors for link semantics prediction.

B. The Use of Network Structure in Software Engineering

Previous work has shown network structure to be useful in revealing trends in various software engineering communities and artifacts. For example, network science techniques have been applied to describe properties of the reverse engineering research community [16], as well as open source communities [24]. These techniques have also been used to identify experts across projects [27] and anticipate whether contributors will form project teams [15]. There is also a history of work using networks to illustrate and reason about program properties and state such as connected modules, memory [41], and control flow [12, 18]. More recent work also uses graphs for tasks such as debugging [34, 37] and defect prediction [40].

C. The Use of Network Structure in Traceability

There has also been analysis on the practicality and appropriateness of network structure to traceability from a visualization perspective [17, 25] as well as the utility of trace networks on various tasks [20]. Li et. al [20] recruited twenty-four participants and used a survey to assess their preference of visualization types to perform management, design, implementation and testing tasks. J Cleland-Huang et. al [8] proposed a framework and tools for using an augmented traceability network to support specific organizational goals. From an implementation perspective, Elamin and Osman considered the benefits of using graph databases to represent traceability networks and perform link prediction with respect to data storage and information retrieval optimization [11].

The closest traceability network to that of issues is the bug report network (BRN) which represents the connections between bug reports and their assignees [31]. Prior work has focused on using the properties of networks to automate triaging in order to reduce the time and effort taken to assign bugs to developers manually. Zhang and Lee developed a method to recommend the top relevant developers for fixing a particular bug based on their experience and cost of fixing [38]. Other work has studied developer contributions and interactions to assist triaging [36, 39].

In contrast to previous work, we attempt to view the desirable characteristics of traceability networks that may prove to be useful for the issue traceability task. Additionally, we take the perspective of focusing on trace link types and exploiting the semantics found therein.

VIII. Conclusion and Future Work

In this work, we examine the effectiveness of analyzing issue trace link data using a network structure. We show that there are considerable quantitative differences in the connection tendencies of issues that are identified as hubs versus non-hubs in the network. These differences correlate with visible high-level metrics such as resolution time, type of issue, and the nature of connections. We have also observed that among small groups of issues, or triads, there are patterns of connection that can be used to fairly effectively predict link semantics within and across different projects.

In future work, we intend to further investigate the potential of considering traceability problems as a link prediction task using either network-based metrics or network/graph embedding based techniques, in comparison to existing data-driven prediction techniques. We also envision that the interaction between different networks, e.g. an issue traceability network, an issue-creator network and a developer collaboration network, would bring insight to tasks such as issue triaging and trace link construction.

References

[1] Eytan Adar. Guess: A language and interface for graph exploration. *In Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 791–800, 2006.

[2] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. In *Science*, volume 286, pages 509–512. American Association for the Advancement of Science, 1999.

[3] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *In Proceedings of International Conference on Web and Social Media (ICWSM)*, pages 361–362, 2009.

[4] Vladimir Batagelj and Andrej Mrvar. Pajek-program for large network analysis. *Connections*, 21(2):47–57, 1998.

[5] T. F. Bissyand, D. Lo, L. Jiang, L. Rveillre, J. Klein, and Y. L. Traon. Got issues? who cares about it? a large scale investigation of issue trackers from github. In *In Proceedings of International Symposium on Software Reliability Engineering (ISSRE)*, pages 188–197, 2013.

[6] Fergal Mc Caffery, Valentine Casey, M. S. Sivakumar, Gerry Coleman, Peter Donnelly, and John Burton. *Medical Device Software Traceability*, pages 321–339. Springer London, London, 2012.

[7] Jane Cleland-Huang, Orlena C. Z. Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software traceability: Trends and future directions. *In Proceedings of the Future of Software Engineering*, pages 55–69, 2014.

[8] Jane Cleland-Huang, Jane Huffman Hayes, and JM Domel. Model-based traceability. *In Proceedings of the ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, pages 6–10, 2009.

[9] Csar Ducruet and Igor Lugo. Structure and dynamics of transportation networks: Models, methods, and applications. 2011.

[10] Mark E.J. Newman. The structure of scientific collaboration networks. *In Proceedings of the National Academy of Sciences of the United States of America*, 98:404–409, 2001.

[11] Randa Elamin and Rasha Osman. Implementing traceability repositories as graph databases for software quality improvement. *In Proceedings of IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 269–276, 2018.

[12] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. In *ACM Transactions on Programming Languages and Systems*, volume 9. ACM, 1987.

[13] Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giuliano Antoniol, Jonathan Maletic, and Patrick Mäder. Traceability fundamentals. *In Proceedings of Software and Systems Traceability*, pages 3–22, 2012.

[14] Mark Granovetter. The strength of weak ties: A network theory revisited. In *Sociological theory*, pages 201–233. JSTOR, 1983.

[15] Jungpil Hahn, Jae Yun Moon, and Chen Zhang. Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. In *Information Systems Research*, volume 19, pages 369–391. INFORMS, 2008.

[16] Ahmed E. Hassan and Richard C. Holt. The small world of software reverse engineering. *In Proceedings of Working Conference on Reverse Engineering*, pages 278–283, 2004.

[17] Philipp Heim, Steffen Lohmann, Kim Lauenroth, and Jürgen Ziegler. Graph-based visualization of requirements relationships. In *In Proceedings of Requirements Engineering Visualization (REV'08).*, pages 51–55. IEEE, 2008.

[18] Susan Horwitz and Thomas Reps. The use of program dependence graphs in software engineering. In *Proceedings of the 14th International Conference on Software Engineering*, pages 392–411. ACM, 1992.

[19] Gueorgi Kossinets and Duncan J Watts. Empirical analysis of an evolving social network. In *Science*, volume 311, pages 88–90. American Association for the Advancement of Science, 2006.

[20] Yang Li and Walid Maalej. Which traceability visualization is suitable in this context? a comparative study. *In Proceedings of International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 194–210, 2012.

[21] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. In *Journal of the American society for information science and technology*, volume 58, pages 1019–1031. Wiley Online Library, 2007.

[22] Tiancheng Lou, Jie Tang, John Hopcroft, Zhanpeng Fang, and Xiaowen Ding. Learning to predict reciprocity and triadic closure in social networks. In *ACM Transactions on Knowledge Discovery from Data*, volume 7. ACM, 2013.

[23] Patrick Mader, Paul L Jones, Yi Zhang, and Jane Cleland-Huang. Strategic traceability for safety-critical projects. In *IEEE software*, volume 30, pages 58–66. IEEE, 2013.

[24] Gregory Madey, Vincent Freeh, and Renee Tynan. The open source software development phenomenon: An analysis based on social network theory. *In Proceedings of Americas Conference on Information Systems*, page 247, 2002.

[25] Thorsten Merten, Daniela Jüppner, and Alexander Delater. Improved representation of traceability links in requirements engineering knowledge using sunburst and netmap visualizations. In *In Proceedings of 4th International Workshop on Managing Requirements Knowledge*, pages 17–21. IEEE, 2011.

[26] Christopher R Myers. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. In *Physical Review E*, volume 68, page 046116. APS, 2003.

[27] Masao Ohira, Naoki Ohsugi, Tetsuya Ohoka, and Kenichi Matsumoto. Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. In *SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.

[28] Márcia Oliveira and João Gama. An overview of social network analysis. In *Wiley Interdisciplinary Reviews:*

Data Mining and Knowledge Discovery, volume 2, pages 99–115. Wiley Online Library, 2012.

[29] Michael Rath, Jacob Rendall, Jin LC Guo, Jane Cleland-Huang, and Patrick Mäder. Traceability in the wild: automatically augmenting incomplete trace links. In *Proceedings of the 40th International Conference on Software Engineering*, pages 834–845. ACM, 2018.

[30] John Rushby. New challenges in certification for aircraft software. *In Proceedings of the International Conference on Embedded Software (EMSOFT)*, pages 211–218, 2011.

[31] Robert Sandusky, Les Gasser, and Gabriel Ripoche. Bug report networks: Varieties, strategies, and impacts in a f/oss development community. 01 2004.

[32] George Spanoudakis and Andrea Zisman. Software traceability: a roadmap. In *Handbook Of Software Engineering And Knowledge Engineering: Vol 3: Recent Advances*, pages 395–428. World Scientific, 2005.

[33] Olaf Sporns. Structure and function of complex brain networks. *Dialogues in clinical neuroscience*, 15:247–62, 09 2013.

[34] Wang Tao. Post-mortem dynamic analysis for software debugging. 2007.

[35] Mihaela Todorova Tomova, Michael Rath, and Patrick Mäder. Use of trace link types in issue tracking systems. *In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 181–182, 2018.

[36] S. Wang, W. Zhang, Y. Yang, and Q. Wang. Devnet: Exploring developer collaboration in heterogeneous networks of bug repositories. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 193–202, 2013.

[37] Cheng Zhang, Juyuan Yang, Dacong Yan, Shengqian Yang, and Yuting Chen. Automated breakpoint generation for debugging. In *JSW*, volume 8, pages 603–616, 2013.

[38] Tao Zhang and Byungjeong Lee. An automated bug triage approach: A concept profile and social network based developer recommendation. In *Intelligent Computing Technology*, pages 505–512, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[39] W. Zhang, S. Wang, Y. Yang, and Q. Wang. Heterogeneous network analysis of developer contribution in bug repositories. *In Proceedings of International Conference on Cloud and Service Computing*, pages 98–105, 2013.

[40] Thomas Zimmermann and Nachiappan Nagappan. Predicting defects using network analysis on dependency graphs. *In Proceedings of the 30th International Conference on Software Engineering (ICSE)*, pages 531–540, 2008.

[41] Thomas Zimmermann and Andreas Zeller. Visualizing memory graphs. In *Software Visualization*, 2001.