ASME Journal of Computing and Information Science in Engineering
Online journal at:
https://asmedigitalcollection.asme.org/computingengineering

ReqNet and ReqSim: A Network and Semantic Similarity Dataset of Requirements From the Tree Structure of System Requirement Specifications

Chandan Kumar Sahu[1]
Department of Automotive Engineering,
Clemson University,
Greenville, SC 29607
e-mail: csahu@clemson.edu

Rahul Rai[1]
Dean's Distinguished Professor
Department of Automotive Engineering,
Clemson University,
Greenville, SC 29607
e-mail: rrai@clemson.edu

Margaret Wiecek
School of Mathematical and Statistical Sciences,
Clemson University,
Clemson, SC 29631
e-mail: wmalgor@clemson.edu

David Gorsich
US Army DEVCOM Ground Vehicle Systems Center,
Warren, MI 48397
e-mail: david.j.gorsich.civ@army.mil

Systems are developed as a solution to the problem space defined by their requirements. The requirements are acquired during the elicitation process. The creative nature of the elicitation process, proprietary nature of requirements, the need of extensive preprocessing, and the diverse techniques for analysis restricts the development of a requirement dataset. There exists no standard method to create a requirement dataset. Thus, we devise a semi-formal method to create a multi-purpose open-source requirement dataset that harnesses human knowledge in the system requirement specification documents (SyRSDs). We devise a method to extract a tree from each SyRSD. Our dataset has three forms. (1) ReqList*, a list of requirements from* 86 *distinct systems with their document structure in pure text form. The* 12,701 *requirements are ready to leverage natural language processing techniques and unsupervised machine learning techniques; (2)* ReqNet*, a large network of requirements consisting of* 17,375 *nodes to deploy graph-theoretic algorithms for requirement engineering.* ReqNet *portrays small-world network characteristics with an average distance of* ≈ 9.5619 *links; (3)* ReqSim*, a dataset consisting of* 10,933 *pairs of requirements annotated with their similarity scores.* ReqSim *enables sentence-level supervised learning tasks to exploit the semantics of requirements. The similarity scores are coherent with human knowledge. The dataset is grounded by the tree structure of SyRSDs. The tree structure resonates with the hierarchical nature of the requirement allocation process. The ReqList, ReqNet and ReqSim dataset will facilitate the deployment of modern computing algorithms for requirements engineering.* [DOI: 10.1115/1.4065786]

Keywords: requirements engineering, requirement dataset, natuaral language processing, network analysis, machine learning, graph theory, model-based systems engineering, knowledge engineering

1 Introduction

Engineering systems are developed to satisfy the needs of their stakeholders [1,2]. The needs are transformed into requirements to serve as a formal agreement between the stakeholders over the system's development cycle. These requirements define the problem space, whereas the system under development defines the corresponding solution space. The requirements are iteratively decomposed into low-level requirements during requirement allocation to delineate the problem space effectively [3,4]. Furthermore, these requirements, which formalize the idea of the system, are continually revised during the system development process based on the evolving constraints [5]. Requirement engineering (RE) is the process that manages the requirements to comprehend the problem space better to develop a better solution space by the system. The cost to fix defects in systems grows exponentially if they are fixed in the design or code or development or operations phases, which succeed the requirements phase [6]. It is crucial to delineate the requirements early in the system design process to avoid system failure, time, cost, and resource overruns [7]. RE spans from the elicitation of requirements to the representation/modeling, analysis (such as traceability, change propagation, prioritization), documentation, verification, and validation [8].

All the RE tasks are contingent upon acquiring a set of requirements during the elicitation process. The lack of an RE dataset and a method to create any RE dataset is evident by the plenty of research papers focusing on identifying requirements from source documents by classifying if a particular sentence is a requirement or not [9,10]. Unlike the abundance of NL corpora, requirement datasets are scarce because of the following reasons. First, as requirements are a product of system ideation, having access to detailed requirements can disclose the system under development.

[1]Corresponding authors.

Manuscript received February 3, 2024; final manuscript received June 13, 2024; published online March 20, 2025. Assoc. Editor: Yan Wang.

The organization risks losing its competitive edge. Second, it is a challenging task to acquire a set of well-formed requirements with all the characteristics of ISO/IEC/IEEE 29148:2018 [11]. Third, the creative nature of the requirement elicitation process and the ensuing human involvement during elicitation limits the number of elicited requirements. Fourth, having a dataset that is usable for all the tasks of RE is challenging. We need to create multiple variants of the dataset to enable multi-faceted analysis. Finally, as the requirements vary depending on its domain and application, annotating the requirements for any specific analysis (such as traces, dependence) becomes a resource-intensive task. Thus, most of the existing RE work happened in silos with small requirement datasets. Reference [12] emphasized the necessity of a good requirement dataset to foster progress in RE.

The overarching goal of our research is to offer a multi-purpose requirement dataset in order to accelerate requirement analysis. As the requirements are elicited in natural language, they call for *natural language processing* (NLP) techniques for analysis [13]. The requirements are recorded in a system requirement specification document (SyRSD). Even though matrix-based representations are widely used to represent requirements [14], a *hierarchical tree structure* was endorsed by the NASA System Engineering Handbook [15] and the INCOSE Needs, Requirements, Verification, Validation Lifecycle Manual [16]. The tree structure opens the portal for deploying *network analysis* techniques. Furthermore, the limited capability of rule-based NLP techniques has plateaued their utility for requirement analysis, especially in RE applications involving the semantics of requirements. *Artificial intelligence* (AI) techniques emerged as a promising solution to capture the *semantics in RE* [17]. Nonetheless, the lack of a pertinent requirement dataset hinders the full-fledged adoption of AI in RE.

We create the requirement dataset from the SyRSDs. Our dataset is underpinned by the tree structure extractable from the SyRSDs. Our work attempts to address the fourth research direction from the review of semantic networks for engineering design by Han et al. [18], a comprehensive knowledge graph of requirements with a framework to extend it further. We seek the answers to the following research questions (RQ). RQ1: How to represent the requirements in SyRSDs? Can a tree structure be extracted from the requirements in SyRSDs? RQ2: How to compute the semantic similarity between pairs of requirements? To answer these RQs, we adopt a semi-automatic method [19] to create an open-source[2] [20] requirement dataset by reusing requirements from existing open-source SyRSDs. The process is illustrated in Fig. 1. While doing so, we make the following contributions:

(1) We devise a method to represent the requirements from a SyRSD as a tree. We refer to these requirement trees as *ReqTree*s.
(2) We formulate a method to compute the semantic similarity between pairs of requirements from the SyRSDs.
(3) We release three different forms of data to accelerate requirement analysis as follows:
   (a) *ReqList*s: lists comprising of 12,701 requirements in pure text form acquired from 86 SyRSDs. They only contain the requirements and corresponding clause numbers to retain their structure. Their document structure supplements them as the metadata. *ReqList*s are ready to use NLP and unsupervised machine learning (ML) techniques without further data preprocessing.
   (b) *ReqNet*: a large network of requirements consisting of 17,375 nodes to facilitate graph-theoretic investigations for RE. It assembles the *ReqTree*s of 86 distinct systems.
   (c) *ReqSim*: a dataset consisting of 10, 933 pairs of requirements annotated with their similarity scores. The similarity scores align with human knowledge [21]. *ReqSim*, the first dataset that estimates the pairwise similarity among requirements, enables sentence-level supervised learning tasks.

The remainder of the paper is organized as follows. Section 2 offers a critical appraisal of related research work and positions different components of our work. Section 3 proposes a formal method to (1) extract a tree of requirements from SyRSD and (2) compute the semantic similarity of pairs of requirements. The results of the formal method proposed in Sec. 3 deployed on SyRSDs are deliberated in Sec. 4. It includes the sources of the data and preprocessing for reproducibility of the results. Section 5 discusses the utility of *ReqList*, *ReqNet,* and *ReqSim*. Lastly, Sec. 6 concludes our work.

2 Literature Review

2.1 Requirement Datasets. The requirement datasets are scarce because of the reasons discussed in Sec. 1. The most used requirement dataset is the PROMISE non-functional requirements dataset [22]. It contains 625 requirements in the form of a list. Most of the RE work leveraged this dataset. Notably, most of the requirement datasets contain less than a hundred or just a few hundred (often <1000) requirements pertaining to a single system (Fig. 4). The best available open-source requirements dataset is the PUblic REquirements (PURE) dataset [23]. It is an aggregation of publicly available SyRSDs from the web in their native forms. While the corpus spans diverse domains, the data are quite restricted in terms of its utility owing to the extensive preprocessing it entails. We offer a requirement dataset in which the data are available in the native form, as well as in a clean list form and a network form, with their semantic similarity scores. Our dataset facilitates multi-faceted analysis as they are variants with the same core set of requirements. Additionally, we offer a formal method to expand the corpus.

2.2 Representation of Requirements. Requirements are represented in mathematical forms to enable analysis. The representations can be broadly split into four categories. In the simplest form, the elicited requirements are represented as *list*s [24]. The lists need minimal preprocessing of requirements. Thereby, the information content of lists is minimal. *SysML* models are widely used [5] by RE professionals. SysML does not have the entity type, "need," nor the associated need diagrams [2]. SysML cannot represent the behaviors of sets of requirements. *Matrices* offer a generic and flexible form of representation [14]. The versatility of matrices can be attributed to their effectiveness in capturing the correspondence between pairs of objects, such as requirements-to-requirements for allocation and traceability, requirements-to-systems/components, and requirements-to-tests for verification and validation. Design structure matrices [25], house of quality modeling schemes [26] are a few instances of matrix-based approaches. *Networks* are another form of representation [27]. Networks and matrices are interconvertible. The requirement allocation produces a tree structure as the requirements are decomposed into low-level ones during allocation [15,16]. Our *tree* form of representation, which is more specific than networks, extracts the requirement allocation in a SyRSD.

2.3 Semantics of Requirements. The semantics of requirements allows us to understand unknown concepts and requirements by comparing and contrasting the subjectivity and imprecision of requirements expressed in NL. Most of the RE work that focus on the semantics of requirements restrict themselves to word-level semantics instead of sentence-level semantics. They leverage word-level semantics [28] to classify if a sentence is a requirement (to extract requirements [9,10]); or categorize requirements into functional and non-functional requirements, and other categories [29,30]. Techniques focusing on sentence-level semantics supplement word-level semantics with other information like syntactic and word-order information [31]. Greedy pairing, optimal matching

[2]The*ReqList*, *ReqNet* and *ReqSim* dataset is available at https://github.com/ChandanKSahu/ReqList_ReqNet_ReqSim with DOI:10.5281/zenodo.10976096

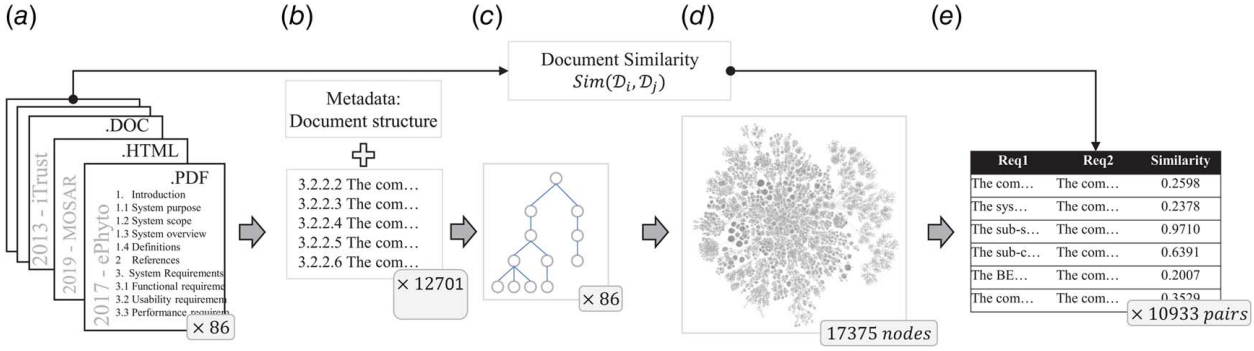

Fig. 1 Computational framework to create *ReqList*, *ReqTree*, *ReqNet,* and *ReqSim* from SyRSDs: (*a*) SyRSD: system requirement specification documents, (*b*) ReqList: list of requirements, (*c*) ReqTree: tree of requirements of a system, (*d*) ReqNet: network of requirements, and (*e*) ReqSim: Pairs of requirements with their similarity scores

are a few techniques of acquiring sentence level semantics from word-level semantics [32].

However, utilizing word-level semantics for comparing requirement sentences questions the credibility of those methods as they (1) ignore the order of the words in the requirement sentences and (2) fail to capture the contextual information offered by the composition of keywords [33]. Thus, the focus is minimal on other RE tasks such as generating requirements, identifying redundancy and contradiction, semantic traceability, and predicting change [13]. The key impediments include the need for (1) a germane dataset; and (2) sentence-level semantics of requirements. Furthermore, vector semantics based on sentence-embeddings are not yet fully adopted for usage in RE. While the sentence-embeddings can be used for inference in RE, their utility is restricted due to the unavailability of a dataset or a method to compute the semantic similarity among requirements to fine-tune the sentence-embedding models.

Our method of computing the semantic similarity between requirements is inspired from lexical semantics, such as WordNet [34], which enable computing the semantic similarity between words. The underlying idea is to construct a graph of the words and leverage the graph-theoretic distance to compute the similarity [35,36]. Such approaches assume that two objects are similar if their distance is short. The distances have been scaled [37,38] to overcome the fact that the semantic distance between nodes at different levels of the taxonomy are not same. More fine-grained concepts are semantically more similar than high-level concepts. We adopt a hierarchical weighing scheme to incorporate these generalizations to compute the semantic similarity between requirements. Furthermore, we incorporate the document-level similarity to consider the highest level of semantic similarity, which is between the SyRSDs. Our method of computing semantic similarity leverages the tree structure of requirements created by RE professionals, thereby harnessing the implicit information stored in the SyRSDs.

3 Methodology

Our representation of requirements is based on the clause numbers of the requirements in a SyRSD. These clause numbers are unique identifiers of the requirements in their SyRSD. The requirements are referred by these clause numbers during discussions and changes. The clause numbers in SyRSDs follow two types of numbering: sequential numbering (e.g., 2021—ReqView.pdf) and/or hierarchical numbering (e.g., 2022—MobileSurveillance.pdf) [39]. Often, they are used together where the requirements follow a sequential order within the hierarchy of the SyRSD. Sequential numbering is used with (e.g., 2004—SGVTF Integration.pdf) or without (e.g., 2005—clarus low.pdf) a categorical prefix. The prefix, part of the clause number preceding the terminal separator, indicates the order of the requirement in the document's structure expressed by the sections and subsections. We parse the clause numbers of the requirements to create a path from the requirement node to the document node to construct a tree. The process extracts the ancestor nodes of the requirement node. The path passes through the ancestor nodes to reach the document node which serve as the root node.

3.1 ReqTree: Tree of Requirements From a SyRSD. We create a requirement tree for each SyRSD. Our network creation process is inspired by creating a tree from the *table of contents* of a document. We illustrate the tree creation process by considering a set of requirements as shown in Fig. 2. Consider a requirement R_1 with clause number $c_1.c_2.c_3.c_4$. Here R_1 belongs to the chapter c_1 and lies in the sub-sub-section $c_1.c_2.c_3$. We prefix the clause number with the document node making the clause number as $D.c_1.c_2.c_3.c_4$. This clause creates the nodes D, $D.c_1$, $D.c_1.c_2$, $D.c_1.c_2.c_3$, and $D.c_1.c_2.c_3.c_4$ in the tree where the node $D.c_1.c_2.c_3.c_4$ corresponds to R_1. The node $D.c_1.c_2.c_3.c_4$ is a requirement node and others are non-requirement nodes. It creates the edges $(D, D.c_1)$, $(D.c_1, D.c_1.c_2)$, $(D.c_1.c_2, D.c_1.c_2.c_3)$, $(D.c_1.c_2.c_3, D.c_1.c_2.c_3.c_4)$ by connecting each pair of nodes according to their order in the clause $D.c_1.c_2.c_3.c_4$. Essentially, each separator becomes an edge. This process creates a path from the requirement node $D.c_1.c_2.c_3.c_4$ to the document node D where the path passes through the ancestor nodes of the requirement. The path is a tree rooted at the document node D.

The same method is followed for all the clauses in the extracted list of requirements. The succeeding requirements will use the nodes/edges already appended to the tree, and non-existing nodes/edges will be added to the tree to add new requirements. The requirement R_2 with clause number $c_1.c_2.c_3.c_4.c_5$ will create only one new node $D.c_1.c_2.c_3.c_4.c_5$ and add the requirement R_2 to it. It will create a new edge $(D.c_1.c_2.c_3.c_4, D.c_1.c_2.c_3.c_4.c_5)$ to connect to the tree. The requirements R_3 with clause number $c_1.c_2.c_3$ will not add any new node or edge as the corresponding node is

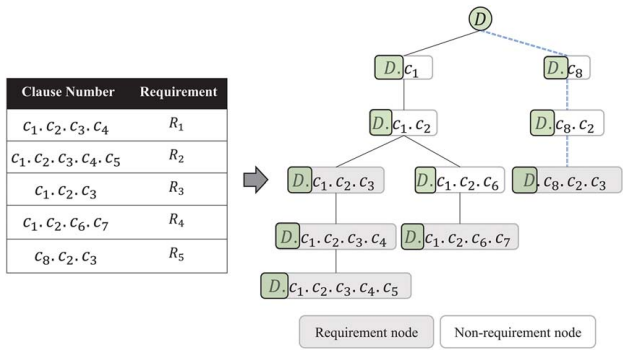

Fig. 2 Illustration of the tree creation process using a set of sample requirements

already in the tree. R_3 will be added to the node $D.c_1.c_2.c_3$. If no requirement exists for the clause $c_1.c_2.c_3$ in the extracted list of requirements, the node $D.c_1.c_2.c_3$ will remain as a non-requirement node. The requirement R_4 with clause number $c_1.c_2.c_6.c_7$ will add the nodes $D.c_1.c_2.c_6$ and $D.c_1.c_2.c_6.c_7$ and the edges $(D.c_1.c_2, D.c_1.c_2.c_6)$, $(D.c_1.c_2.c_6, D.c_1.c_2.c_6.c_7)$. Now, the requirement R_5 with clause number $c_8.c_2.c_3$ belongs to chapter c_8 in the document D. Adding R_5 will add the nodes $D.c_8$, $D.c_8.c_2$, and $D.c_8.c_2.c_3$ and the respective edges as shown in Fig. 2. This tree, rooted at the document node D, has two subtrees. The first sub-tree is rooted at the node $D.c_1$ corresponding to Chapter c_1 in the SyRSD in which the requirements R_1, R_2, R_3, and R_4 exists. The other sub-tree is rooted at the chapter node $D.c_8$ to which the requirement R_5 belongs. Exclusion of the document node D would create a forest with each tree rooted at the chapter nodes.

The *ReqTree* is essentially a collection of the paths from the requirements nodes to the document node D. We refer to this *tree* of *req*uirements as **ReqTree**, and denote as T. The general mechanism for creating a *ReqTree* from a *ReqList* is illustrated in Algorithm 1. The *ReqTree* is essentially a system-requirement tree. A closer look at the SyRSDs reveals that the requirements are listed under sections and subsections of the SyRSD. Thus, the sections and subsections of the SyRSD become the branch nodes in the tree. The requirements in the SyRSD become the leaf nodes. The document becomes the root node. As we parse through the clause numbers of requirements, the ancestors of the requirement nodes (e.g., sections and subsections) also get added to the *ReqTree*. The nodes in the *ReqTree* correspond to the sub-system or components or functions or operational states of the system because the sections and subsections of the SyRSD specify them. Each edge reflects the decomposition of the parent node's role into child nodes. As the high-level sub-systems/functions/requirements/operational-states decompose into low-level ones, the *ReqTree* reflects the requirement allocation process. Allocation of functions as in data flow diagrams, states as in state diagrams, goals, or capability from the stakeholder scenarios are a few examples of hierarchical allocation in SyRSDs [40].

Algorithm 1 Construction of *ReqTree*

Input : *ReqList*
Output : *ReqTree*
1 **for** each *clause number* in *ReqList* **do**
2 Prefix the *clause number* with a document node;
3 Extract the nodes and append to a list;
4 Extract the edges and append to a list;
5 **end**
6 Extract all the unique nodes;
7 Extract all the unique edges;
8 Create a network with the set of unique nodes and edges;

3.2 ReqNet: Network of Requirements From Multiple SyRSDs. The distance between a pair of requirements in the *ReqTree*, T, allows us to compare any pair of requirements from the same SyRSD. To compare a pair of requirements spanning across two different systems, we need to connect those systems. We achieve this by joining the trees. We define a global node (G) as the root node to connect the *ReqTree*s of any two systems. The underlying idea of connecting the trees is that all the systems are components of a global system. This idea is inspired by computing the distance between disconnected synsets in WordNet [41], where WordNet uses a simulated root node to compute the distance which is otherwise infinity [42]. Moreover, we connect *ReqTree*s from multiple SyRSDs to the global root node to form a huge requirement network to compare any pair of requirements. We refer to this requirement network as **ReqNet**, denoted as $\mathcal{N}$. *ReqNet* is an unweighted undirected tree. Mathematically, the *ReqNet* can be expressed as

$$\mathcal{N} = \bigcup_{i=0}^{\mathbf{N}} \{G\} \cup T_i \tag{1}$$

where **N** represents the number of SyRSDs.

3.3 Weighted Tree. The similarity between a pair of requirements can be computed from the distance between the corresponding nodes in *ReqNet*. The distance in *ReqNet* reflects the number of edges to be hopped to reach one requirement from another. The number of edges to be traversed is a coarse measure of distance. It has a significant shortcoming. The unweighted distance ignores the difference between comparing a pair of requirements within a *ReqTree* and across a pair of *ReqTree*s (see Fig. 3). It lays equal importance to both cases. Conceptually, requirements within the same *ReqTree* are more similar than those across different *ReqTree*s as the latter pertain to different systems whereas the former is from the same system. The latter should produce a larger distance. Thus, we convert the unweighted *ReqTree*s to weighted *ReqTree*s.

The unweighted *ReqTree* can be converted to a weighted *ReqTree* by leveraging the requirement allocation process [4,16,43]. We adopt two methods to acquire the weighted trees. Our first consideration is that "all the non-reducible requirements are allocated with equal importance." The leaf nodes are the non-reducible nodes in the *ReqTree*. Thus, we allocate a unit weight to each of the edges connecting the leaf nodes of the *ReqTree* and propagate the weights until the root node. This consideration adopts a bottom-up mode of weight update. The bottom-up allocation mimics the system design behavior where the system is designed based on existing or reusable components. Reverse engineering [44], component analysis and interface analysis methods give rise to a bottom-up requirement allocation. The second consideration is that "all the requirements allocated from the same parent requirement are allocated with equal importance." We start by allocating a unit weight to the edges connected to the root node of the *ReqTree*. Then, we distribute the weight equally among its child nodes until we reach the leaf nodes. The second consideration translates to a top-down mode of weight update. The top-down approach mimics the system design behavior where the system is designed using functional analysis or structural analysis. The higher-level requirements/systems/functionalities are decomposed into a set of low-level requirements/systems/functionalities that satisfy the parent node upon composition. The weight update starts by

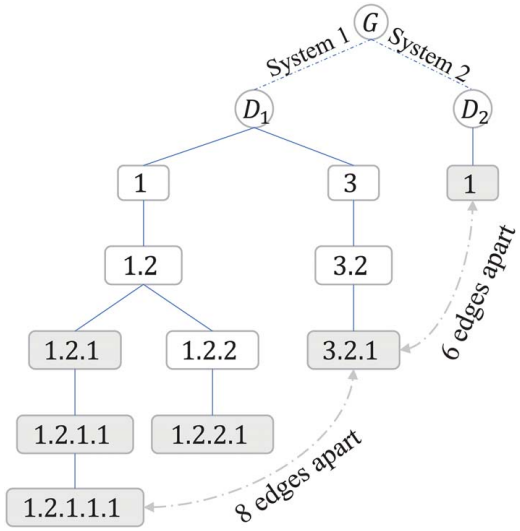

Fig. 3 Comparison of unweighted distance from 3.2.1 to 1.2.1.1.1 (within the a *ReqTree*) with 3.2.1 to 1 (across *ReqTree*s) in a notional *ReqTree*

considering the unweighted undirected *ReqTree* (UUT) as an unweighted directed *ReqTree* (UDT) and ends as a weighted undirected *ReqTree* (WUT). Mathematically, the weight update process starts with nodes with in-degree zero and terminates with nodes with out-degree equal to zero during execution in either mode.

3.3.1 Bottom-Up Mode. We temporarily consider the UUT as a UDT where the edges are directed away from the leaf nodes *toward* the root node. The weight update process starts at the leaf nodes in the bottom-up UDT. The in-degree of the leaf nodes is 0, and the out-degree of the leaf nodes is 1. A unit weight is allocated to each edge outgoing from the leaf nodes as all the leaf nodes are equally important. These are the same edges that are incoming to the parents of the leaf nodes. The in-degree of these branch nodes is ≥ 1, whereas their out-degree equals one. Then, we update the weight of the only outgoing edge from the parent nodes with the sum of the weights of the incoming edges. The process continues, and terminates upon reaching the root node, whose out-degree is zero. The weight update process can be mathematically written as

$$w_{bu}(v^{out}) = \sum_{v \in T} w_{bu}(v^{in}) \tag{2}$$

where w represents the weight; v^{out} and v^{in} represent the edge outgoing and incoming to the vertex v in the *ReqTree*, T. The subscript *bu* denotes the bottom-up mode of weight update. The bottom-up mode of weight update translates to an *additive* mode of weight update.

3.3.2 Top-Down Mode. Alternately, in the top-down mode, we consider a UDT where the edges are directed *away from* the root node toward the leaf nodes. The weight update starts at the root node, whose in-degree is zero. We allocate a unit weight to each edge outgoing from the root node. The root node's child nodes (or branch nodes) have an in-degree of 1 but an out-degree of ≥ 1. As all the child nodes of a node are equally important, the weight of the incoming edge of the node is equally distributed among its outgoing edges. The process terminates upon reaching the leaf nodes, whose out-degree is zero. Mathematically, the weight update process can be written as

$$w_{td}(v^{out}) = \frac{w_{td}(v^{in})}{|\Gamma(v^{out})|},\ v \in T \tag{3}$$

where,w represents the weight; v^{out} and v^{in} represent the edge outgoing and incoming to the vertex v in the *ReqTree* T. $|\Gamma(v^{out})|$ denotes the number of edges outgoing from the node v. The subscript *td* denotes the top-down approach of weight update. The top-down mode of weight update translates to a *divisive* mode of weight update.

3.3.3 Weight Normalization. The weighted trees suffer from exploding (in the additive mode) or vanishing (in the divisive mode) weights. The weights become system-dependent and become too large or too small in the case of large trees. An adverse consequence of these exploding/vanishing weights is that any comparison between the requirements of the two systems will be affected by the number of requirements listed for each of them. So, we normalize the weights of the *ReqTree* to make all the systems equally important. The weights of the edges are scaled by double the weight of the tree, as shown follows:

$$w_{u,v} = \frac{w(u, v)}{2 \sum_{u,v \in T} w(u, v)} \tag{4}$$

where $w(u, v)$ denotes the weight of an edge connecting a pair of nodes u and v in the *ReqTree* T. Normalization with double the tree's total weight slashed the maximum distance of a requirement from the global root to 0.5. Thus, the distance between a pair of nodes is ≤ 1. The weighted trees are denoted as T^w. Algorithm 2 summarizes the bottom-up mode of weight update with weight normalization.

Algorithm 2 Conversion of an unweighted *ReqTree* into a weighted *ReqTree* using the bottom-up approach

Input *ReqTree*
Weighted *ReqTree*
Extract adjacency matrix of *ReqTree*;
Identify the leaf nodes;
Consider the leaf nodes as the initial set of *child nodes*;
while *child nodes* ≠ *root node* **do**
  Find *parent nodes* of the *child nodes*;
  for each *parent node* **do**
    In the adjacency matrix, $w^{out}_{parent} \leftarrow \sum w^{in}_{parent}$
  end
  child nodes ← *parent nodes*;
end
Normalize the weights of the adjacency matrix;
Construct a weighted tree from the updated adjacency matrix;

3.3.4 Weighted ReqNet. The weighted *ReqTrees* of all the RSDs are connected to the global root node to construct the weighted *ReqNet*. Mathematically

$$\mathcal{N}^w = \bigcup_{i=0}^{\mathbf{N}} \{G\} \cup T_i^w \tag{5}$$

where $\mathcal{N}^w$ is the weighted *ReqNet*. The weighted *ReqNet* offers a fine-grained measure of the distance between pairs of requirements following the practices of requirement allocation. It aligns better with human judgment. The weighted distances overcome the shortcomings (See Sec. 3.3) of unweighted distances as the weights accumulate toward the root node. Since the path between two requirements across two systems passes through the root node, the weighted distances become higher. The distance between a pair of requirement nodes within a *ReqTree* is low as their connecting path does not pass through the root node. This is in coherence with human knowledge that a pair of requirements from the same system are more similar (from the same *ReqTree*) than a pair from different systems (from different *ReqTrees*).

3.4 ReqSim: Similarity of Requirements. Our method of computing the semantic similarity rely on the assumption that the requirements are semantically organized in the SyRSDs. The foundation of this assumption is that the creators of the SyRSDs organize the requirements based on their similarity. In other words, the creators aggregate similar requirements and segregate dissimilar requirements. The *ReqTree* captures this organization of requirements. Second, the distance between a pair of requirement nodes in the tree reflects their semantic similarity [35,36].

We leverage *ReqNet*, $\mathcal{N}^w$, to compute the similarity between pairs of requirements. As ReqNet has three variants, we will have three different distances between a pair of requirements. And, correspondingly, three different similarity scores: a coarse-grained similarity, an additive fine-grained similarity, and a divisive fine-grained similarity. We can construct the *ReqNet* to compute the distance between any pair of nodes and the similarity. We use the path similarity [42,45], the reciprocal of the distance incremented by one, to compute the similarity between a pair of requirements. The path similarity did not consider the similarity across the source SyRSDs where the pair of requirements exists. Thus, we incorporate the document-level similarity, defined as the similarity between a pair of SyRSDs, into the path similarity. The similarity between a pair of requirements R_i and R_j is defined as

$$\underset{\mathcal{N}}{Sim}(R_i, R_j) = \frac{Sim(\mathcal{D}_i, \mathcal{D}_j)}{1 + \underset{\mathcal{N}}{Dist}(R_i, R_j)} \tag{6}$$

where $R_i \in \mathcal{D}_i$ and $R_j \in \mathcal{D}_j$.

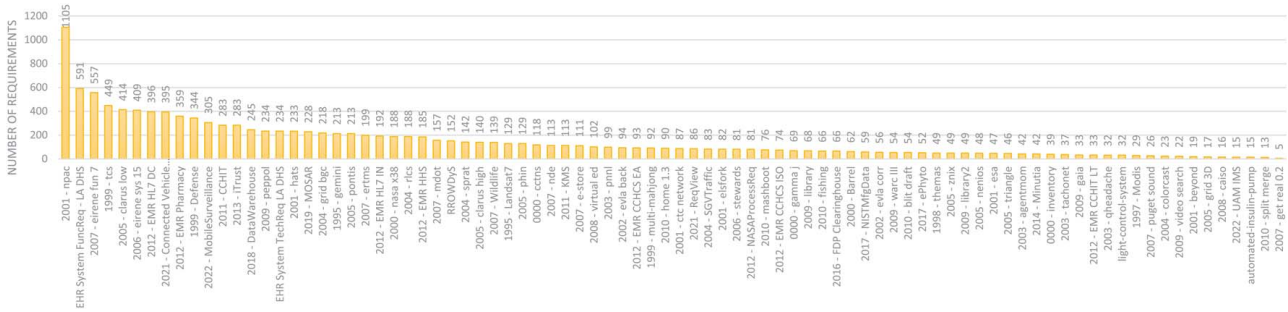

Fig. 4 Distribution of requirements in *ReqLists*

4 Data Curation and Results

4.1 Data Sources—SyRSDs. Our principle of creating this dataset is based on requirement reuse. In accordance with the principles of systematic literature review [46], we have queried for the keywords (ϕ OR *System* OR *Software*) AND (*Requirement Specification*) OR (*Document*) OR (ϕ OR *PDF* OR *WORD* OR HTML). Additional queries include *Requirement Dataset* and *SRS*. The search string in each query consists of multiple keywords. One keyword is selected from the keywords within each parentheses based on the "OR" condition. The selected keyword is included/excluded from the search string based on the "AND/OR" condition preceding the enclosing parentheses. An example query is "System Requirement Specification Document." It includes a keyword from the first, second, and third parentheses. It excludes the keyword from the fourth parenthesis using the "OR" condition preceding it. ϕ indicates that the keyword from the parentheses is optional, as in the query "Requirement Specification Document" which makes the first keyword optional. The queries have been made in Google Scholar[3], Google Search,[4] Google Dataset Search,[5] GitHub,[6] ACM Digital Library,[7] IEEE Xplore,[8] ScienceDirect,[9] and SpringerLink.[10] We excluded the results from the search based on the following criteria. First, the search results that lacked a structure in the SyRSDs are discarded, as the structured clause numbers are crucial to our tree structure. Second, the SyRSDs that focus exclusively on use-case scenarios where the requirement statements omit the subject or object of the sentences are excluded. Third, we excluded the SyRSDs created as part of capstone projects. We included only six SyRSDs created from capstone projects that had a consistent structure for diversity. Based on the number of SyRSDs retained, Google Search produced the best results, followed by Google Dataset Search and Github. While GitHub produced the most results, almost all were capstone projects, which led to their exclusion. Our search results have discovered the PURE dataset [23]. The exclusion criteria rejected 25 out of the 79 documents from the PURE dataset. We added 32 more SyRSDs and created a dataset comprising 86 SyRSDs.

4.2 Data Preprocessing. The requirements are organized logically in the source SyRSDs to ease their interpretation to the reader [24]. Predominantly, the system/software RSDs follow IEEE Std 1233-1998 [47] and IEEE Std 830-1998 [48] guidelines irrespective of the form (such as PDF, HTML, DOC, XLS) of the SyRSDs. SyRSDs graciously use tables, sub-clauses, figures, flowcharts, and cross-references to ease interpretation and texts in different forms (bold, italics, underlines) to emphasize. They vary significantly from each other in terms of their structure, e.g., order and inclusion of header/footer. Over two-thirds of the SyRSDs contained unstructured content [23]. Additionally, the requirement statements contain modal verbs such as "has," "have," "must," and "should" in addition to "shall." These facets hinder the deployment of any standard NLP technique to extract the requirements. Thus, we adopted a semi-automatic method to extract requirements from these documents [19]. We only used automated text processing techniques to convert the PDF/DOC/ HTML documents into pure text form. We extracted the requirements manually.

The automated conversion to pure text form has a few unique characteristics requiring manual cleaning. It discards raster images. It extracts the text from vector images (e.g., flowcharts) and tables by ignoring their structure. We discarded such information completely. We initiated the manual processing by comparing the extracted texts with their source SyRSDs. We removed the titles, table of contents, headers, and footers, and other texts which are not requirements. Annotations, such as "M" for mandatory and "O" for optional requirements (e.g., 2007—ertms.pdf, 2014—Minutia.pdf) or traces (e.g., 2021—Connected Vehicle Pilot NYC.pdf, 1997—Modis.pdf), are ignored. Often, the documents contain requirements with sub-clauses and bullet points. If they cannot be split into different requirements due to their common context, we packed the bullet points in a single clause. The conversion from PDFs to texts accrued noise, such as fragmented words, additional line breaks, and conjoined words. The line breaks were removed. Fragmented words were merged, and conjoined words were split using a word processor.

During the manual cleaning of the extracted texts, we retained the structure of the RSDs by keeping the hierarchical order within the documents. As the structure is reflected by the hierarchical ordering (sections, subsections, clause numbers, etc.), we retained the section/clause numbers. In the absence of a unique clause number or if there are multiple requirements in a single clause, we allocate a unique clause number to each requirement based on its position in the SyRSD. For instance, if clause 2.1 has two requirements, we used 2.1.1 and 2.1.2 to have a unique clause number corresponding to each of those requirements. If clause 2.1 contains two requirements and 2.1.1 also includes a requirement, we used 2.1.0.1 and 2.1.0.2 to allocate unique clause numbers to them. This structure is crucial because it is created by a domain expert who has organized the document logically [49,50] to ease comprehension.

After preprocessing, we created a clean *list of requirements* from each SyRSD with their corresponding unique clause numbers, preserving the structure of the SyRSD. It is referred to as **ReqList**. Each of the *ReqList* is supplemented with its metadata to capture the contextual information. The metadata file includes the hierarchical structure (e.g., system → sub-system → component) of the SyRSD. The number of requirements extracted from each source document is shown in Fig. 4. The 54 SyRSDs from the PURE [23] dataset produced 7310 requirements even though they include 34,268 sentences. Our 32 SyRSDs added 5391 requirements. *ReqList* has 12,701 requirements from 86 systems.

[3] https://scholar.google.com/
[4] https://www.google.com/advanced_search
[5] https://datasetsearch.research.google.com/
[6] https://github.com/
[7] https://dl.acm.org/search/advanced
[8] https://ieeexplore.ieee.org/search/advanced
[9] https://www.sciencedirect.com/search/entry
[10] https://link.springer.com/advanced-search

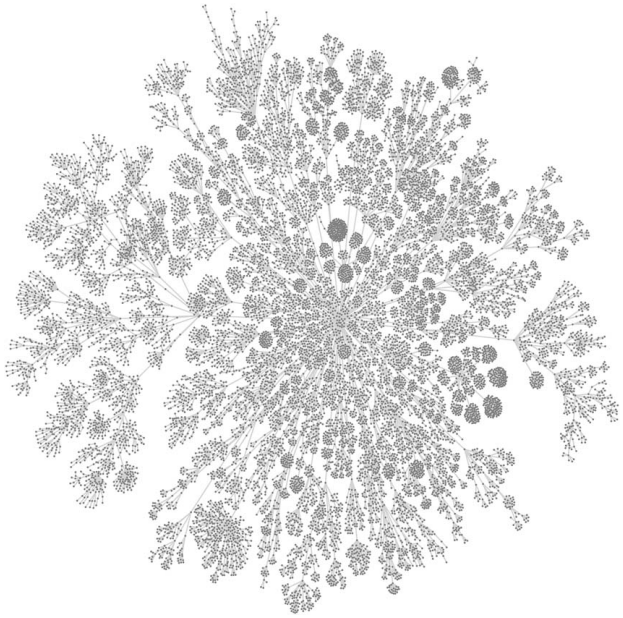

Fig. 5 ***ReqNet*—A network of requirements**

4.3 ReqNet: Network of Requirements. The 86 *ReqTree*s created from the 86 SyRSDs are assembled as per Sec. 3.2 to create the *ReqNet*. Figure 5 shows the *ReqNet* with 17,375 nodes. It has 12,701 requirement nodes and 4674 non-requirement nodes.

4.4 ReqSim: Similarity of Requirements. The *ReqNet*, a union of all the *ReqTree*s, is a huge network with heavy resource demand. We adopt specific properties of trees to ease the computation of $Dist(R_i, R_j)$. The path that connects a pair of nodes in a tree is unique. In the case of *ReqNet*, a tree, the unique path connecting a pair of requirements connects the corresponding *ReqTree*s. The path passes through the global node, G, if the pair of requirements pertain to different *ReqTree*s. The path is a subset of the union of the paths from the selected requirements to the global node, G. So, we avoid using the complete *ReqNet* while computing $Dist(R_i, R_j)$ by leveraging the path of the requirements from the global node. The distance between a pair of requirements can be computed if their paths from the root node and the weights of the connecting edges are known. We extract the weights of the edges along the path that connects each requirement node to the global node. Then, the requirements are randomly paired, and the graph-theoretic distances between the pairs are calculated. We compute the additive distance as well as the divisive distance. Then, we factor in the document similarity to compute the semantic similarity between the pair of requirements. This is illustrated in Algorithm 3.

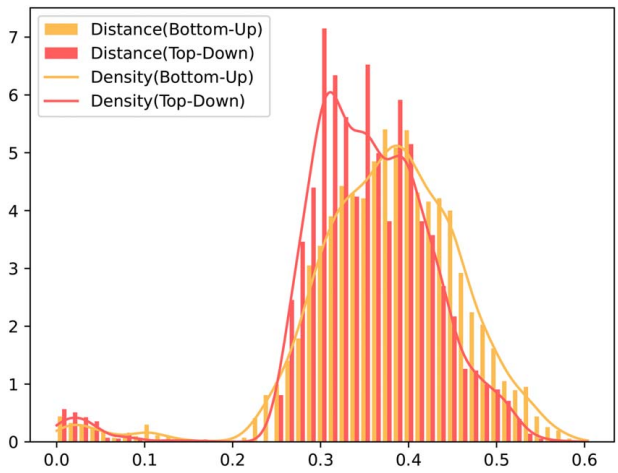

Fig. 6 Histogram plot of fine-grained distances in the bottom-up and top-down approaches

To estimate the similarities across different SyRSDs, we computed the embeddings of the SyRSDs. The 512 token limits of most transformer-based models [51] deem them unsuitable for embedding SyRSDs as the number of tokens in a SyRSD far exceeds 512 tokens. Thus, we embed the *ReqList*s and their corresponding metadata using the Longformer-Encoder-Decoder model [52], which has a token limit 16,384. The document similarity, $Sim(\mathcal{D}_i, \mathcal{D}_j) \in [0.3623, 1]$. The document similarity is the highest when the two documents are identical.

Algorithm 3 Construction of *ReqSim*

```
   Input : ReqLists of all systems
   Output ReqSim
1  for ReqList of each SyRSD do
2      Construct the ReqTree (See Algorithm 1);
3      Convert the unweighted ReqTree to a weighted one (See Algorithm 2);
4      for each clause number in a ReqList do
5          Find and store the weights of the edges connecting to the global
           root node;
6      end
7      Pair the requirements within the system;
8  end
9  Assemble the ReqLists of all the systems;
10 Shuffle and pair the requirements of the assembled list;
11 Merge the two kinds of requirement pairs;
12 for each requirement pair do
13     Compute the graph-theoretic distance between each pair;
14     Estimate the similarity between the requirement pairs while factoring
       the document similarity (Eq. (6));
15 end
```

Random pairing of the requirements produces a distribution of the weighted distances where the data are biased toward dissimilar requirements against similar requirements (see Fig. 6). The reason being that the highly similar requirements are at the tail end of the distribution. Moreover, if a reference SyRSD has 10 requirements and the dataset has 100 requirements, there is a 90% probability that a requirement will be paired with a dissimilar requirement in case of random pairing. So, we additionally paired the requirements within a document with each other (Step 7 in Algorithm 3) to include more similar pairs. 12, 701 requirements from *ReqList*s produced 6350 pairs. We created an additional 4583 pairs by pairing the requirements within SyRSDs. We set a maximum limit of 100 pairs from any document to prevent any system from dominating the highly similar pairs. Overall, we have 10,933 requirement pairs with their semantic similarity scores.

5 Discussion

5.1 ReqList. *ReqList*, being a list of requirements of a system, is cleaner and closer to the source SyRSDs. Most RE applications that can be practiced on SyRSDs can be practiced on *ReqList*s. The primary one is NLP for RE [13,53]. We quote a few ideas for potential applications. First, the *ReqList*s can serve as the ground truth benchmark for research works that focus on identifying and extracting requirements from source SyRSDs [54,55]. Second, the requirements can be related and traced among each other using clustering techniques and word frequency-based techniques. Moreover, system-to-requirement level traces can be established by deploying NLP techniques on the *ReqList*s and their corresponding document structure. Third, the requirements can be evaluated for complying with linguistic rules for compliance to ISO/IEC/IEEE 29148:2018(E) [11]. Fourth, if NLP techniques deployed on the *ReqList* discover its *ReqTree*, it will eliminate the need for manual

structuring of the requirements for documentation. Lastly, *ReqList*s are well-suited for deploying unsupervised or self-supervised learning techniques. For instance, the requirements in *ReqList* can help in prompting generative AI tools to generate requirements.

5.2 *ReqTree*. An effect of the weighted *ReqTree*s is that the parent nodes (one degree of separation) accumulate more weight as the number of children increases. Alternately, the similarity between a requirement and its child requirements reduces as the number of child requirements increases. The effect of weight accumulation gets further amplified as we compare a requirement with its grandparents and other higher ancestors (≥ 2 degrees of separation). This aligns with requirement allocation, where the correlation between parent and child requirements reduces as the parent requirement is decomposed into more child requirements. Due to weight accumulation, a requirement shares more similarities with its siblings than its grandparent requirement, even though both are two edges apart.

We want to emphasize that weighted trees are essential for computing the semantic similarity. Our weighing methods consider either the leaf requirements to be of equal importance or the child requirements of the same parent node to be of equal importance. In reality, all the requirements may not be equally important. Thus, a weighted requirement allocation process will eliminate the need of our weighing process while being more close to reality. Furthermore, a different hierarchy in the SyRSD will affect the weighted trees.

5.3 *ReqNet*. *ReqNet*, being a graph, opens up the portal to deploy graph-theoretic and network analysis techniques. Moreover, tree-specific algorithms can also be exploited. Notably, traceability, detecting change in requirements, and analyzing the propagation of its impact are a few applications of network analysis techniques in the domain of RE.

ReqNet revealed a few interesting characteristics about the structure of the SyRSDs. First, the depth of *ReqNet* is low. The depth of a tree is defined as the average unweighted distance of all the nodes from the root node. The average unweighted distance of a requirement node from the document node is 3.8384. This results from the document hierarchy, usually three layers deep: section, sub-section, and sub-sub-section. The requirements typically lie within the sub-sub-section. They account for the ≈ 4 degrees of average separation of a requirement node from the document node. Beyond those three layers, the authors of the SyRSDs use bullet points, sub-clauses, and enumerated lists. Their presence adds a few more layers in *ReqNet*. Nonetheless, these deeper layers (>6 edges from the global node, *G*) are a bit of noise than the norm. A consequence of the low-depth nature of *ReqNet* is that *ReqNet* portrays small-world network behavior. The average unweighted distance between any pair of nodes is 9.5619. The links connect the global node to the document node to section heading to sub-section heading to sub-sub-section heading to requirement. Five edges for each requirement in the pair approximate to 9.5619.

5.4 *ReqSim*. We notice that the similarity, $Sim_{bu} \in [0.21456, 0.99955]$ and $Sim_{td} \in [0.20400, 0.99999]$ where the subscripts *bu* and *td* correspond to bottom-up and top-down modes respectively. This is a result of the normalized distances which are $Dist_{bu} \in [0.00045, 0.83705]$ and $Dist_{td} \in [4.33 \times 10^{-}6, 0.83838]$. The pairs of requirements with the maximum (Max) and minimum (Min) semantic similarity in the bottom-up and the top-down approaches are shown in Table 1. It is evident from the table that the highest similarity is exhibited by requirements from the same SyRSD. They are separated by one or two edges, reflecting that such pairs share a parent-child, sibling, or grandparent-child relationship in the *ReqNet*. This behavior aligns with human judgment as the child requirement nodes inherit several properties from their parent requirement nodes. Another attribute of requirement pairs with

Table 1 Pairs of requirements with their similarity score

Requirement (R_i)	Document ($\mathcal{D}_i$)	Requirement (R_j)	Document ($\mathcal{D}_j$)	$Sim_{(\mathcal{D}_i, \mathcal{D}_j)}$	$Dist(R_i, R_j)$	$Dist_{bu}(R_i, R_j)$	$Sim_{bu}(R_i, R_j)$	$Dist_{td}(R_i, R_j)$	$Sim_{td}(R_i, R_j)$
The user shall be prompted to enter a text string describing a string pattern.	2001—hats	The HATS-GUI shall provide the user the capability to search the text display for text sub-strings. The search criteria are described in Appendix F.	2001—hats	1.0 (Max)	1 (Min)	4.5×10^{-4} (Min)	0.9995 (Max)	8.69×10^{-6}	0.9999
The Clarus program shall determine how it will provide data and information to contributors.	2005—clarus low	The Clarus program shall determine the need for bilateral Clarus Data Sharing Agreements with countries, agencies, states, and regions.	2005—clarus low	1.0 (Max)	2	7.27×10^{-4}	0.9993	4.33×10^{-6} (Min)	0.9999 (Max)
Network Services shall support IEEE 802.11	1999—Defense	The dose of insulin to be delivered shall be computed by measuring the current level of blood sugar, comparing this to a previous measured level and computing the required dose as described in Sec 3.3.	automated-insulin-pump	0.3633 (Min)	9	0.6931	0.2145 (Min)	0.7808	0.2040 (Min)
All displayed nodes hidden before the most recent unhide instruction shall become hidden.	2001—hats	NPAC SMS shall, upon placing a Subscription Version into conflict, record the current date and time as the conflict date and time stamp.	2001—npac	0.6757	18 (Max)	0.4936	0.4524	0.3387	0.5047

high similarity scores is that such requirements are leaf nodes or lie far from the root node where the weights do not accumulate. It is important to emphasize that the similarity score is derived from the tree structure created from the clause numbers. The tree structure disregards the textual information contained in the requirements corresponding to the clause numbers. A pair of textually identical requirements from two different SyRSDs (e.g., the requirement for the service life of the motor from the SyRSD of a DC motor and that from the SyRSD of an AC motor) may not produce the expected high similarity score. Their similarity score will depend on the similarity of the SyRSDs and the locations of the requirements in the SyRSDs.

Additionally, we observe that the least similarity is exhibited by a pair of requirements from distinct SyRSDs. Their lowest similarity score arise from their low document similarity score, even though their weighted distance is moderate. Moreover, the most distant nodes (separated by 18 edges) exhibit moderate similarity due to weight normalization and relatively high document similarity scores. Thus, the similarity scores of *ReqSim* align with human interpretation. We can conclude that *ReqNet* and *ReqSim* capture human knowledge. So, *ReqSim* can push the boundaries of existing NLP and ML approaches from word-level to sentence-level semantics.

6 Conclusion

We represent the requirements from a SyRSD as a tree by leveraging the structure of the SyRSDs. The tree structure reflects requirement allocation and facilitates mathematical analysis. We supplemented the tree structure with the document-level similarity among the SyRSDs to estimate the similarity between pairs of requirements. We deploy our method on 86 SyRSDs to create a requirement dataset that can facilitate the deployment of a diverse class of computing algorithms. Our dataset has three forms with the same core set of requirements. First, we release *ReqList*, a list of 12,701 requirements from 86 distinct systems to facilitate natural language processing and unsupervised machine learning algorithms (such as prompting for generative artificial intelligence). They can help in requirement identification and extraction tasks, creating requirement-to-requirement and requirement-to-system traces, and characterizing well-formed requirements using rule-based NLP, among many others. Each of these *ReqList* is supported by their metadata consisting of the structure of the SyRSDs. Second, we create *ReqNet*, a large network of requirements consisting of 17,375 nodes, to support graph-theoretic explorations. *ReqNet* can help in traceability analysis, identification of changes in requirements, and the study of cascading changes in requirements. *ReqNet* portrays small-world characteristics with an average distance of ≈ 10 links. The small-world characteristics arises from the path consisting of five edges from the global node to each requirement node that passes through the document node and the three layers of the section hierarchies of SyRSDs. Third, we create *ReqSim*, a dataset containing 10, 933 pairs of requirements and their similarity scores, to gear RE toward sentence-level semantics from word-level semantics. The semantic similarity scores align with human judgment as they are created from the structure of the SyRSDs, which are created by humans. Thus, they will serve as the gold standard for supervised learning tasks. The datasets can be expanded further by incorporating SyRSDs of new systems while following the proposed formal method. Our method considers all the systems equally important and the requirements with the same parent node equally important. We lay higher importance on requirements that are higher in the hierarchy of the document structure. Our dataset will accelerate the deployment of natural language processing, graph theory, and unsupervised and supervised learning techniques in requirement engineering.

Acknowledgment

This work was supported by the Automotive Research Center (ARC), a US Army Center of Excellence for modeling and simulation of ground vehicles, under Cooperative Agreement W56HZV-19-2-0001 with the US Army DEVCOM Ground Vehicle Systems Center (GVSC).

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The dataset is available online[11] with DOI: 10.5281/zenodo.10976096.

Distribution Statement A

Approved for public release; distribution is unlimited (OPSEC 7820).

References

[1] Sommerville, I., 2005, "Integrated Requirements Engineering: A Tutorial," IEEE Softw., **22**(1), pp. 16–23.
[2] Ryan, M., Wheatcraft, R., and INCOSE Requirements Working Group, 2023, "Incose Guide to Writing Requirements, July–2023," International Council on Systems Engineering, pp. 1–144.
[3] MacLellan, C. J., Langley, P., Shah, J., and Dinar, M., 2013, "A Computational Aid for Problem Formulation in Early Conceptual Design," ASME J. Comput. Inf. Sci. Eng., **13**(3), p. 031005.
[4] Jackson, M., and Wilkerson, M., 2016, "MBSE-Driven Visualization of Requirements Allocation and Traceability," 2016 IEEE Aerospace Conference, Big Sky, MT, IEEE, pp. 1–17.
[5] Hoepfner, G., Nachmann, I., Zerwas, T., Berroth, J. K., Kohl, J., Guist, C., Rumpe, B., and Jacobs, G., 2023, "Towards a Holistic and Functional Model-Based Design Method for Mechatronic Cyber-Physical Systems," ASME J. Comput. Inf. Sci. Eng., **23**(5), p. 051001.
[6] Boehm, B., Valerdi, R., and Honour, E., 2008, "The ROI of Systems Engineering: Some Quantitative Results for Software-Intensive Systems," Syst. Eng., **11**(3), pp. 221–234.
[7] Hofmann, H. F., and Lehner, F., 2001, "Requirements Engineering as a Success Factor in Software Projects," IEEE Softw., **18**(4), p. 58.
[8] Masoudi, N., Rai, R., Ortiz, J., Sutton, M., Khade, V., Acena, D., Freeman, G., Summers, J., Gorsich, D., Rizzo, D., et al., 2022, "Elicitation, Computational Representation, and Analysis of Mission and System Requirements," SAE Int. J. Adv. Curr. Pract. Mobility, **5**, pp. 315–325.
[9] Guzman, E., Ibrahim, M., and Glinz, M., 2017, "A Little Bird Told Me: Mining Tweets for Requirements and Software Evolution," 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal, IEEE, pp. 11–20.
[10] Williams, G., and Mahmoud, A., 2017, "Mining Twitter Feeds for Software User Requirements," 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal, IEEE, pp. 1–10.
[11] ISO 29148:2018(E), "ISO/IEC/IEEE International Standard – Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," ISO/IEC/IEEE 29148:2018(E), pp. 1–104.
[12] Ferrari, A., Dell'Orletta, F., Esuli, A., Gervasi, V., and Gnesi, S., 2017, "Natural Language Requirements Processing: A 4D Vision," IEEE Softw., **34**(6), pp. 28–35.
[13] Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., and Batista-Navarro, R. T., 2021, "Natural Language Processing for Requirements Engineering: A Systematic Mapping Study," ACM Comput. Surv. (CSUR), **54**(3), pp. 1–41.
[14] Wang, N., 2006, "ERMM: An Engineering Requirements Management Method," ASME J. Comput. Inf. Sci. Eng., **6**(2), pp. 196–199.
[15] Hirshorn, S. R., Voss, L. D., and Bromley, L. K., 2017, "NASA Systems Engineering Handbook," Technical Report, Washington, DC.
[16] Wheatcraft, L., Katz, T., Ryan, M., Wolfgang, R., and INCOSE Requirements Working Group 2022, ""Needs and Requirements Manual: Needs, Requirements, Verification, Validation Across the Lifecycle, May–2022," International Council on Systems Engineering, pp. 1–457.
[17] Williams, G., Meisel, N. A., Simpson, T. W., and McComb, C., 2022, "Design for Artificial Intelligence: Proposing a Conceptual Framework Grounded in Data Wrangling," ASME J. Comput. Inf. Sci. Eng., **22**(6), p. 060903.
[18] Han, J., Sarica, S., Shi, F., and Luo, J., 2022, "Semantic Networks for Engineering Design: State of the Art and Future Directions," ASME J. Mech. Des., **144**(2), p. 020802.
[19] Knauss, E., and Ott, D., 2014, "(Semi-) Automatic Categorization of Natural Language Requirements," International Working Conference on Requirements Engineering: Foundation for Software Quality, Essen, Germany, Springer, pp. 39–54.
[20] Sahu, C. K., 2024, Chandanksahu/reqlist reqnet reqsim: Zenodo release, Apr.

[11] https://github.com/ChandanKSahu/ReqList_ReqNet_ReqSim

[21] Li, Y., McLean, D., Bandar, Z. A., O'shea, J. D., and Crockett, K., 2006, "Sentence Similarity Based on Semantic Nets and Corpus Statistics," IEEE Trans. Knowl. Data Eng., **18**(8), pp. 1138–1150.
[22] Cleland-Huang, J., Mazrouee, S., Liguo, H., and Port, D., 2007, x Nfr.
[23] Ferrari, A., Spagnolo, G. O., and Gnesi, S., 2017, "Pure: A Dataset of Public Requirements Documents," 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal, IEEE, pp. 502–505.
[24] Doors, T., 2008, "*Manual "Get It Right the First Time: Writing Better Requirements"*, " IBM® Corporation.
[25] Chen, Y., Cheng, P., and Yin, J., 2010, "Change Propagation Analysis of Trustworthy Requirements Based on Dependency Relations," 2010 2nd IEEE International Conference on Information Management and Engineering, Chengdu, China, IEEE, pp. 246–251.
[26] Beitz, W., Pahl, G., and Grote, K., 1996, "Engineering Design: A Systematic Approach," MRS Bull., Springer London, **71**.
[27] Giffin, M., de Weck, O., Bounova, G., Keller, R., Eckert, C., and Clarkson, P. J., 2009, "Change Propagation Analysis in Complex Technical Systems," ASME J. Mech. Des., **131**(8), p. 081001.
[28] Femmer, H., Müller, A., and Eder, S., 2020, "Semantic Similarities in Natural Language Requirements," Software Quality: Quality Intelligence in Software and Systems Engineering: 12th International Conference, SWQD 2020, Proceedings 12, Vienna, Austria, Jan. 14–17, 2020, Springer, pp. 87–105.
[29] Cleland-Huang, J., Settimi, R., Zou, X., and Solc, P., 2007, "Automated Classification of Non-Functional Requirements," Requir. Eng., **12**(2), pp. 103–120.
[30] Abad, Z. S. H., Karras, O., Ghazi, P., Glinz, M., Ruhe, G., and Schneider, K., 2017, "What Works Better? A Study of Classifying Requirements," 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal, IEEE, pp. 496–501.
[31] Farouk, M., 2018, "Sentence Semantic Similarity Based on Word Embedding and Wordnet," 2018 13th International Conference on Computer Engineering and Systems (ICCES), Cairo, Egypt, IEEE, pp. 33–37.
[32] Shahzad, K., Pervaz, I., and Nawab, A., 2018, "Wordnet Based Semantic Similarity Measures for Process Model Matching," BIR Workshops, Stockholm, Sweden,Sept. 24–26, pp. 33–44.
[33] Ferreira, R., Lins, R. D., Simske, S. J., Freitas, F., and Riss, M., 2016, "Assessing Sentence Similarity Through Lexical, Syntactic and Semantic Analysis," Comput. Speech Lang., **39**, pp. 1–28.
[34] Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. J., 1990, "Introduction to Wordnet: An On-Line Lexical Database," Int. J. Lexicography, **3**(4), pp. 235–244.
[35] Rohde, D. L., Gonnerman, L. M., and Plaut, D. C., 2006, "An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence," Commun. ACM, **8**(627–633), p. 116.
[36] Alvarez, M. A., and Lim, S., 2007, "A Graph Modeling of Semantic Similarity Between Words," International Conference on Semantic Computing (ICSC 2007), Irvine, CA, IEEE, pp. 355–362.
[37] Fellbaum, C. and Miller, G., 1998, "*Combining Local Context and Wordnet Similarity for Word Sense Identification*," *WordNet: A Lexical Reference System and Its Application*, MIT Press, Cambridge, MA, pp. 265–283.
[38] Wu, Z., and Palmer, M., 1994, "Verb Semantics and Lexical Selection," Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics, Las Cruces, NM, pp. 133–138.
[39] Sparx Systems and S. Maguire, 2016, Enterprise Architect: Requirements Engineering, Version 1.0.
[40] Hull, E., Jackson, K., and Dick, J., 2005, *Requirements Engineering in the Solution Domain*, Springer.
[41] Miller, G. A., 1995, "Wordnet: A Lexical Database for English," Commun. ACM, **38**(11), pp. 39–41.
[42] Goodman, M. W., 2020, Wordnet: Taxonomy-Based Metrics https://github.com/goodmami/wn.
[43] Flanigan, D., and Brouse, P., 2012, "System of Systems Requirements Capacity Allocation," Procedia Comput. Sci., **8**, pp. 112–117.
[44] Feng, S. C., Moges, T., Park, H., Yakout, M., Jones, A. T., Ko, H., and Witherell, P., 2023, "Functional Requirements of Software Tools for Laser-Based Powder Bed Fusion Additive Manufacturing for Metals," ASME J. Comput. Inf. Sci. Eng., **23**(3), p. 031005.
[45] Wan, S., and Angryk, R. A., 2007, "Measuring Semantic Similarity Using Wordnet-Based Context Vectors, Man and Cybernetics," 2007 IEEE International Conference on Systems, Montreal, QC, Canada, IEEE, pp. 908–913.
[46] Petersen, K., Vakkalanka, S., and Kuzniarz, L., 2015, "Guidelines for Conducting Systematic Mapping Studies in Software Engineering: An Update," Inf. Software Technol., **64**, pp. 1–18.
[47] IEEE Std 1233-1998, 1998, "IEEE Guide for Developing System Requirements Specifications," 1998 Ed., IEEE Std 1233, pp. 1–36.
[48] IEEE Std 830-1998, 1998, "IEEE Recommended Practice for Software Requirements Specifications," IEEE Std 830-1998, pp. 1–40.
[49] Mao, S., Rosenfeld, A., and Kanungo, T., 2003, "Document Structure Analysis Algorithms: A Literature Survey," Document Recognition and Retrieval X, Vol. 5010, pp. 197–207.
[50] Klink, S., Dengel, A., and Kieninger, T., 2000, "Document Structure Analysis Based on Layout and Textual Features," Proceedings of International Workshop on Document Analysis Systems, DAS2000, Rio de Janeiro, Brazil, pp. 99–111.
[51] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I., 2017, " "Attention is All You Need," *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S.Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, eds., Vol. 30, Curran Associates, Inc.
[52] Beltagy, I., Peters, M. E., and Cohan, A., 2020 "Longformer: The Long-Document Transformer". arXiv preprint arXiv:2004.05150.
[53] Lash, A., Murray, K., and Mocko, G., 2012, "Natural Language Processing Applications in Requirements Engineering". pp. 541–549.
[54] Senger, T., 2022, "A Unified Open-and Closed-Source Software Requirements Dataset," B.S. thesis.
[55] Slankas, J., and Williams, L., 2013, "Automated Extraction of Non-Functional Requirements in Available Documentation," 2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE), San Francisco, CA, IEEE, pp. 9–16.