
# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**AUTOMATING REQUIREMENTS TRACEABILITY USING NATURAL LANGUAGE PROCESSING: A COMPARISON OF INFORMATION RETRIEVAL TECHNIQUES**

by

Christopher D. Laliberte

September 2021

Thesis Advisor: Ronald E. Giachetti
Co-Advisor: Mathias N. Kolsch

THIS PAGE INTENTIONALLY LEFT BLANK

| REPORT DOCUMENTATION PAGE | | | Form Approved OMB No. 0704-0188 |
|---|---|---|---|



| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | September 2021 | Master's thesis |

**4. TITLE AND SUBTITLE**
AUTOMATING REQUIREMENTS TRACEABILITY USING NATURAL LANGUAGE PROCESSING: A COMPARISON OF INFORMATION RETRIEVAL TECHNIQUES

**5. FUNDING NUMBERS**

**6. AUTHOR(S)** Christopher D. Laliberte

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Naval Postgraduate School
Monterey, CA 93943-5000

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
N/A

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
Approved for public release. Distribution is unlimited.

**12b. DISTRIBUTION CODE**
A

**13. ABSTRACT (maximum 200 words)**

This thesis compares histogram distance and cosine similarity measures used as information retrieval (IR) techniques in automated requirements tracing. We first build a software application that computes a Term Frequency–Inverse Document Frequency (TD-IDF) matrix of a National Aeronautics and Space Administration (NASA) public requirements dataset; classify requirement pairs using each similarity measure across a variety of similarity thresholds; derive performance achieved by each IR-based similarity measure in terms of precision, recall and F-score; and compare them for real-world effectiveness when used for requirements tracing. Given the analyzed dataset, cosine similarity outperformed histogram distance with respect to overall precision and recall. Overall, further research is needed to yield higher levels of precision and recall for automated tracing methods, simplify automated tracing use, and to ultimately instill enough confidence in systems engineers to supplant time-consuming and error prone conventional requirements tracing methods.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| systems engineering, requirements management, requirements traceability, automated requirements tracing, information retrieval, natural language processing | 99 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UU |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release. Distribution is unlimited.**

# AUTOMATING REQUIREMENTS TRACEABILITY USING NATURAL LANGUAGE PROCESSING: A COMPARISON OF INFORMATION RETRIEVAL TECHNIQUES

Christopher D. Laliberte
Contractor, Lockheed Martin
BS, University of Massachusetts Dartmouth, 2009
MS, Worcester Polytechnic Institute, 2014

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN SYSTEMS ENGINEERING MANAGEMENT**

from the

**NAVAL POSTGRADUATE SCHOOL**
**September 2021**

Approved by: Ronald E. Giachetti
Advisor

Mathias N. Kolsch
Co-Advisor

Oleg A. Yakimenko
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

This thesis compares histogram distance and cosine similarity measures used as information retrieval (IR) techniques in automated requirements tracing. We first build a software application that computes a Term Frequency–Inverse Document Frequency (TD-IDF) matrix of a National Aeronautics and Space Administration (NASA) public requirements dataset; classify requirement pairs using each similarity measure across a variety of similarity thresholds; derive performance achieved by each IR-based similarity measure in terms of precision, recall and F-score; and compare them for real-world effectiveness when used for requirements tracing. Given the analyzed dataset, cosine similarity outperformed histogram distance with respect to overall precision and recall. Overall, further research is needed to yield higher levels of precision and recall for automated tracing methods, simplify automated tracing use, and to ultimately instill enough confidence in systems engineers to supplant time-consuming and error prone conventional requirements tracing methods.

THIS PAGE INTENTIONALLY LEFT BLANK

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| DAU | Defense Acquisition University |
| DOD | Department of Defense |
| DOORS | Dynamic Object-Oriented Requirements System |
| DSIG | Domain Special Interest Group |
| GUI | Graphical User Interface |
| IBM | International Business Machines |
| IR | Information Retrieval |
| LSI | Latent Semantic Indexing |
| MBSE | Model-Based Systems Engineering |
| ML | Machine Learning |
| NASA | National Aeronautics and Space Administration |
| NLP | Natural Language Processing |
| OMG | Object Management Group |
| TF-IDF | Term Frequency – Inverse Document Frequency |
| VSM | Vector Space Model |

THIS PAGE INTENTIONALLY LEFT BLANK

# EXECUTIVE SUMMARY

Requirements management is a fundamental systems engineering activity that involves managing requirements traceability throughout a system's life cycle. Given issues with conventional requirements tracing methods, automated tracing methods using Natural Language Processing (NLP) based Information Retrieval (IR) techniques can help systems engineers ensure links between parent and child requirements are correct while preventing common requirements traceability issues, such as missing traces or orphan requirements. In this thesis, histogram distance was compared against the commonly used similarity measure, cosine similarity, with the objective of understanding the performance and utility of histogram distance when used in automated requirements tracing.

A software tool was developed to compare the similarity measures and address the research objective of whether histogram distance can outperform the cosine similarity measure when used for requirements tracing. Using publicly available requirements documentation from National Aeronautics and Space Administration (NASA), the software tool analyzed 215 requirements, generated a Term Frequency–Inverse Document Frequency (TF-IDF) matrix of the document collection, and classified parent-child requirement pairs using the histogram distance and cosine similarity measures under eighteen different similarity measure thresholds for each similarity measure. Precision, recall, and F-scores were calculated, yielding maximum F-scores for each similarity measure under a variety of thresholds. The resulting precision, recall, and F-scores provided insight into the performance of each IR–based similarity measure. The methodology presented can be used to develop a software tool and objectively compare performance of other similarity measures.

This analysis revealed that cosine similarity achieved a higher overall F-score than histogram distance and is better suited for automated requirements tracing. While high recall may be tolerable to systems engineers at the expense of low precision when a human is part of the solution to validate the false positives, the number of resulting false positives rendered by histogram distance arguably involved more work to manually resolve than to manually establish without the use of automated tracing.

While research toward more robust similarity methods is important, using IR as the sole basis of automated tracing has inherent limitations due to  degrees of precision and recall too low to instill sufficient confidence for most systems engineers to use such tools without human intervention. A multifaceted approach using other techniques and Machine Learning (ML)-based methods is likely needed to achieve acceptable levels of precision and recall for use by systems engineers.

Overall, improvements in automated tracing can help systems engineers achieve fewer errors in requirements management, help systems engineers conduct requirements management in less time, and ultimately help programs achieve a better product through increased requirements management rigor. To achieve this, further research is needed to seek out other, potentially more robust similarity measures that yield higher recall and precision than the cosine method. Second, automated tracing solely based on IR will likely remain limited unless a multifaceted approach is taken to augment IR-based methods, improve human-in-the-loop integration, and integrate other automated techniques with IR—such as ML-based approaches—to improve recall and precision to admirable levels. Third, it is recommended that the Department of Defense (DOD) places more emphasis on researching automated tracing given the time savings and potential error reduction afforded by automated tracing tools as compared to existing requirements management tools. Fourth, it is recommended that automated tracing be used as a tool to inspect requirement links manually established by systems engineers and to not obviate human involvement completely. Further research is needed to yield higher levels of precision and recall, simplify automated tracing use, and to ultimately instill enough confidence in systems engineers to supplant time-consuming and error prone conventional requirements tracing methods.

# ACKNOWLEDGMENTS

I would like to express my appreciation to everyone who contributed to my academic accomplishments and personal growth at the Naval Postgraduate School. My interactions with professors and staff during my time in the PD21 program, and shortly during the 581 program, have amazed me by how much the school is devoted to its people, our nation's armed forces, intellectual growth, and the betterment of systems engineering. On a personal level, I would like to thank the individuals supporting the PD21 systems engineering program, specifically Walter Owen and Kristin Giammarco, for guiding me through a journey I never anticipated at the start of the program.

As for this thesis, the completion of such an undertaking would not have been possible without the commitment and support of my advisors, Professor Ronald Giachetti and Professor Mathias Kolsch. I am deeply grateful for their time, patience, wisdom, and unwavering support. This research evolved from its inception, and I thank you for helping guide me along the way.

Lastly, I want to thank my wife, Rebecca, and our two children, Harris and Maeve, for allowing me to take precious moments from them to spend on my academic pursuits. I could not have done it without your love, sacrifices, and daily encouragement, and I hope you someday read this and feel proud of the work I accomplished.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. INTRODUCTION

## A. BACKGROUND

Requirements management, as defined by Acqnotes (2021), is a fundamental systems engineering activity that involves "documenting, analyzing, tracing, prioritizing, and controlling changes to requirements" throughout a system's life cycle. The purpose of requirements management is to "assure that the system's requirements meet the needs and expectations of its stakeholders" and ultimately trace back to user-defined capabilities (Acqnotes 2021).

Requirements traceability is one activity of requirements management. Over the course of the system development process, requirements are decomposed to lower levels of the design. As lower levels of the design are decomposed, it is important that bi-directional traceability is maintained between the source document or parent-level requirements and the lower level requirements, as shown in Figure 1.

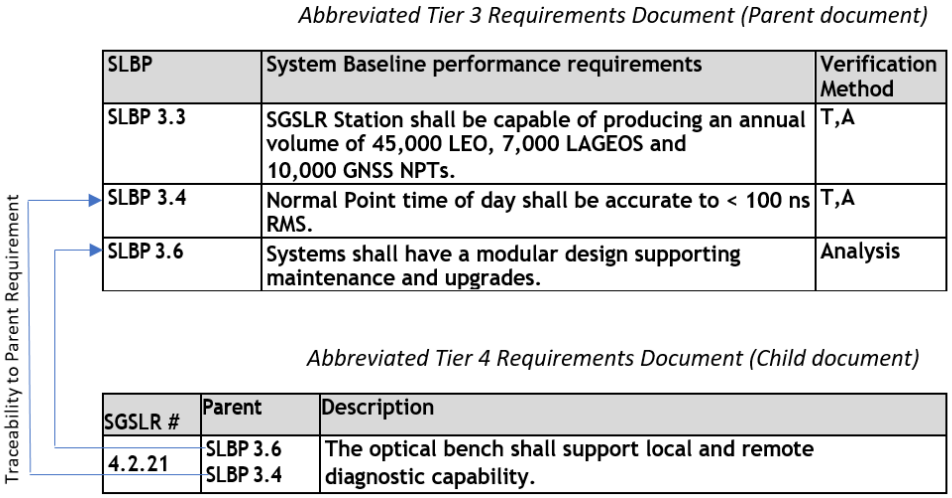



Figure 1. Example of Requirements Traceability

Maintaining bi-directional traceability allows systems engineers to provide a clear lineage between parent and child requirements, ensuring that the originating stakeholder requirements are completely and accurately reflected in the final system. Requirements

traceability also has other benefits such as helping systems engineers identify impacts to lower-level requirements associated with changes to higher-level requirements as changes occur during the system's life cycle.

Given the complexity of managing the relationship between requirements particularly as the system's development advances, requirements traceability should ideally be maintained through a relational database, numbering scheme, or other method that maintains the relationship between the lowest level requirements, their corresponding parent requirements, and the customer's source documentation from which it was ultimately derived (Defense Acquisition University [DAU] 2010). Errors that arise during this activity, if not caught early, can lead to significant program impacts. In some cases, errors in requirements flow-down can result in a system that does not meet its intended capability needs required by the customer.

Although requirements traceability has been practiced in numerous programs, particularly in the Department of Defense (DOD), errors made during this activity have and will likely continue to exist given that traceability is a human-based process. The problems associated with requirements traceability are not new to systems engineering, and according to Huang et al. (2012) include "technical issues related to physically creating, maintaining, and using [potentially] thousands of interrelated and relatively brittle traceability links. As a result, many organizations struggle to implement and maintain traceability links," resulting in improper requirements flow-down and traceability gaps, "even though it is broadly recognized as a critical element of the [system's] development life cycle" (Huang et al. 2012, 7). Practitioners have attempted to curb such issues through increased process rigor, automation of the traceability process, and employment of industry developed relational database tools, such as International Business Machines (IBM) Rational Dynamic Object-Oriented Requirements System (DOORS). Although requirements management tools help reduce errors in terms of requirements traceability and baseline control, they are manually intensive and susceptible to common requirements management issues that ultimately stem from errors in data entry or difficulty of use (Arkley et al. 2005). Despite systems engineers' best intentions, errors in this activity are likely to persist especially as DOD systems become increasingly complex. Traceability is

an activity that is prone to errors, particularly when systems engineers lack proper training, requirements management tools, or when project scope changes.

## B. RESEARCH OBJECTIVE

While academic, technological, and other attempts to curb known issues with requirements traceability may help, systems engineers may also benefit from automated traceability methods using Natural Language Processing (NLP) techniques, which can decrease the "effort needed to construct and maintain a set of traceability links across [requirements] documents" (Cleland-Huang et al. 2007).

Automated methods can potentially help ensure links between parent and child requirements are correct while preventing common requirements issues, such as omissions in requirements flow-down or requirements lacking a parent. Automated methods can provide systems engineers additional confidence in the requirements development phases of a program, as shown in Figure 2, by helping practitioners identify errors sooner regardless of whether formal requirements management tools have been used.

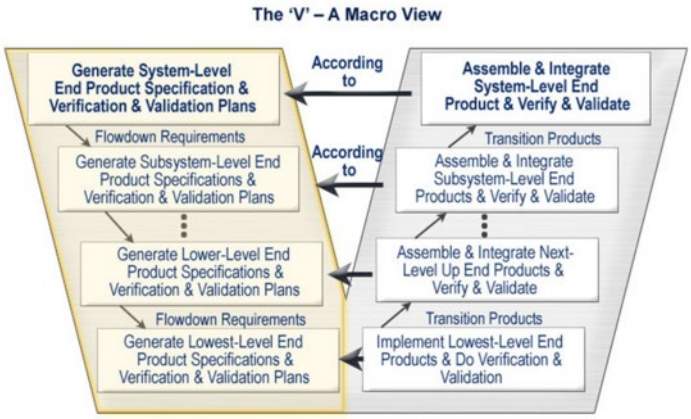



Figure 2. The Systems Engineering V-model. Source: Fairley et al. (2021).

While research in the domain of automated traceability has explored various techniques including Machine Learning (ML), research has primarily focused on using an NLP technique known as Information Retrieval (IR). In essence, IR-based techniques use a similarity measure to determine the text similarity of requirement pairs. Cosine similarity

is one of numerous methods used to derive similarity and is commonly used in IR and automated tracing research.

The objective of this thesis is to determine if other metrics, namely histogram distance, can provide higher quality automated tracing results than the standard cosine similarity method. Quantifying precision of these automated traceability methods will be done by evaluating two metrics commonly used in related research: recall and precision. According to Cleland-Huang et. al (2007), "recall measures the extent to which all of the desired [requirement] links are retrieved," while precision "measures the percentage of retrieved links that are relevant" (Cleland-Huang et al. 2007). To support this research, a software program is developed to help understand the effectiveness of the similarity measures using publicly available requirements documentation.

Overall, the desired goal for this thesis is to determine whether histogram distance can improve upon the cosine similarity method which is commonly used in automated requirements tracing. This study helps systems engineers understand the effectiveness of histogram distance as a similarity measure and whether it can better assist systems engineers with requirements management. This thesis aims to improve upon existing IR-based techniques since "IR-based methods have the most potential to be adopted by industry, as they have been validated from multiple viewpoints" (Wang et al. 2018a). Additionally, this thesis aims to garner more attention by systems engineers in this research area, which has been predominantly researched in the field of software engineering. Continued improvements in automated traceability could help achieve superior requirements management techniques and tools that reduce classic requirements issues encountered across DOD programs. This thesis also aims to identify shortfalls in the use of histogram distance as an IR technique in automated traceability.

## C. THESIS ORGANIZATION AND METHODOLOGY OVERVIEW

Given that this thesis validates the research question using a software tool, this thesis includes the literature review, the methodology behind the development of the software tool and its similarity measure scoring criteria, the corresponding experiments and

results, and the resulting conclusions and recommendations for using alternative similarity measures to support automated traceability.

Chapter II presents a survey of methods traditionally used for requirements traceability and methods used to automate traceability. Chapter III describes the software design and the methodology of how the software program supports the research question. Chapter IV describes the experiments performed using the software tool and the corresponding results. Chapter V provides conclusions regarding the effectiveness based on the results obtained in Chapter IV. Chapter V also highlights areas for improvement and further research.

Overall, the research method involved the development of a software program that independently traces requirements based on the similarity of unique word terms. For the purposes of exploring this research, it was assumed that related requirements are similarly worded and that the uniqueness of particular words in the top-level requirements document will act as features allowing the IR methods to assign a similarity value used for classifying a parent-child requirement pair. The source data used to test the software program was publicly available requirements documentation provided from NASA and structured in a hierarchical fashion, decomposed from higher levels to lower levels. Data pre-processing was performed to facilitate processing of the software tool, as provided in Appendix A. Effectiveness of the IR-based similarity measures is evaluated by measuring several criteria described in Chapter IV. Details regarding the software program and the methodology used for this research is provided in Chapter III and the source code developed for this thesis is provided in Appendix B.

THIS PAGE INTENTIONALLY LEFT BLANK

# II. LITERATURE REVIEW

This chapter provides a survey of current literature available on traditional requirements tracing methods and automated tracing methods, including background on IR as it relates to automated traceability. Although the requirements tracing methods described in this section are not comprehensive, they are relevant to the research performed for this thesis.

## A. TRADITIONAL REQUIREMENTS TRACING METHODS

It is important to understand traditional methods that currently exist and their shortfalls. Numerous requirements traceability approaches have been developed and used by practitioners including basic techniques, basic automated tools, and model-based methods.

Gotel and Finkelstein (1994) provides several basic traceability techniques that are paper-based and include cross-referencing schemes, key phrase dependencies, and requirements traceability matrices as shown in the example depicted in Table 1. In general, these methods "differ in the quantity and diversity of information they can trace between, in the number of interconnections they can control, and in the extent to which they can maintain [traceability] when faced with ongoing changes to requirements" (Gotel and Finkelstein 1994, 2).

Table 1. Requirements Traceability Matrix. Source: Wheeler (2016).

| METHOD OF VERIFICATION<br>N/A - NOT APPLICABLE<br>1 - ANALYSIS<br>2 - DEMONSTRATION<br>3 - EXAMINATION<br>4 - TEST | | | | | | CLASSES OF VERIFICATION<br>A - DESIGN VERIFICATION<br>B - FIRST ARTICLE<br>C - ACCEPTANCE | | | |
|---|---|---|---|---|---|---|---|---|---|
| SECTION 3 | VERIFICATION METHOD | | | | | VERIFICATION CLASS | | | SECTION 4 |
| REQUIREMENT | N/A | 1 | 2 | 3 | 4 | A | B | C | VERIFICATION |
| 3.1 | | X | X | X | X | X | | | 4.2 |
| 3.2 | | X | X | X | X | | X | | 4.3 |
| 3.3 | X | | | | | | | | |
| 3.3.1 | | X | | | | X | | | 4.6.1.1 |

Gotel and Finkelstein (1994) also describes several basic automated tools that support traceability by utilizing automated forms of the basic techniques. These include word processors, spreadsheets, and database systems. These methods are largely manually performed like basic techniques and can be hand-configured to allow previously paper-based traceability techniques to be performed using a computer. These methods generally involve manually created hyper-text links for cross-referencing or manually created and managed cross-reference matrices. These methods, while an improvement over basic paper-based methods, do not significantly reduce the manual intensity of conducting traceability and are prone to human error particularly when several practitioners are involved or when practitioners manage the traceability linkages between hundreds or even thousands of requirements. As a result, computer-based automated tools have improved to simplify requirements management and allow practitioners to manage traceability linkages more effectively. Modern software tools such as IBM DOORS and JAMA Software simplify the creation and management of traceability links between requirements by simplifying change management through user-friendly Graphical User Interfaces (GUI) that are coupled with databases, allowing practitioners to easily create, modify, or remove linkages between requirements, as depicted in Figure 3.

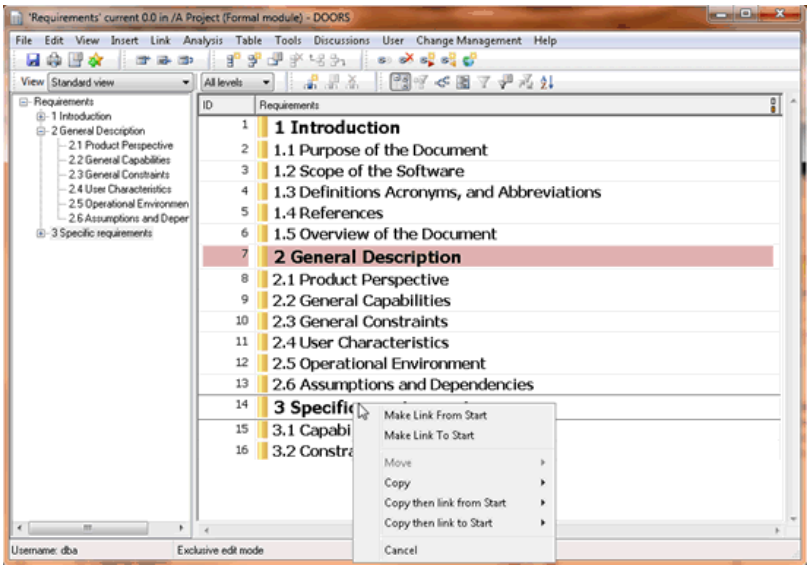



Figure 3. IBM Rational DOORS. Source: International Business Machines (2020).

In lieu of basic and automated methods discussed, Model-Based Systems Engineering (MBSE) tools have become increasingly popular due to their support from International Council on Systems Engineering (INCOSE), Object Management Group (OMG) Systems Engineering Domain Special Interest Group (DSIG), DOD, industry, and academic institutions. According to OMG (2021), "applying MBSE is expected to provide significant benefits over the document centric approach by enhancing productivity and quality, reducing risk, and providing improved communications among the system development team" (OMG 2021). These benefits are primarily associated with ease of use, visualization of the technical baseline, and modeled behavior that corresponds with functionality stipulated by requirements (Hart 2015). MBSE tools that facilitate requirements management include No Magic's Magic Draw, Innoslate, and others. An example of requirements captured in an MBSE model is shown in Figure 4.

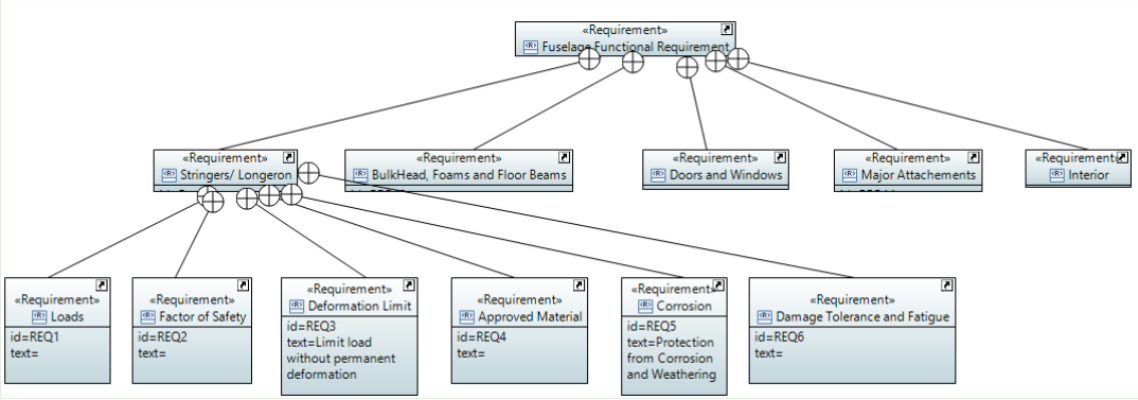



Figure 4. MBSE Requirements Traceability. Source: Veluri and Agarwal (2019).

Despite the benefits provided by MBSE and automated tools such as DOORS, these methods are not impervious to errors that are encountered when using more basic methods. The quality of the resulting requirements traceability using basic, automated, and model-based methods ultimately depends on rigid adherence to the process of decomposing and linking requirements, which is manually done by the practitioner regardless of the method used. According to Gotel and Finklestein (1994), automated tools such as DOORS require training and effort to "initially configure, involve mundane and repetitive activities for use,

and often provide little more than an electronic version of paper-based requirements traceability" (Gotel and Finkelstein 1994, 7). Although model-based tools may help reduce errors even more than traditional tools such as DOORS given their ease of use and behavior modeling, these tools also rely on process rigor and manual interaction, which are susceptible to human error. Common errors in requirements traceability include but are not limited to parent requirements without children requirements and children requirements lacking a link to one or more parent requirements, (i.e., an orphan requirement). These are usually symptoms of failures in the traceability process, including failing to decompose or allocate all parent requirements to lower-level specifications, or failing to ensure proper bi-directional traceability between parent and child requirements. The latter typically results from errors in the requirement development process during decomposition or during later phases of the systems engineering process when requirement changes occur.

## B. ADVANCED AUTOMATED REQUIREMENTS TRACEABILITY METHODS

The basic automated methods previously described primarily address computer-based requirements management tools commonly used to help practitioners manually perform tracing during requirements development and management. With advances in computer technology, practitioners realized the potential of automating the requirements tracing process by inferring requirements linkages and supporting after the fact requirements tracing (Hayes et al. 2003); these advancements can be referred to as advanced automated requirements traceability methods. Research in this area has mainly focused on NLP based IR techniques such as the Vector-Space Model (VSM) (Wang et al. 2018b). Further, research in this area has been done primarily in the domain of Computer Science to improve requirement tracing to source code and documentation. However, research in this area is also highly relevant to the field of systems engineering, which also performs requirements tracing throughout the project life cycle as requirements are decomposed, test procedures are developed, and requirement verification is performed.

Given the similarity of requirements tracing to IR, IR methods have been introduced for automating and helping overcome limitations associated with manual tracing (Wang 2018b). IR is the "science of finding material of an unstructured nature,

such as text, that satisfies an information need from within large collections" (Manning et al. 2008, 1). IR seeks to determine which documents (i.e., requirements) from the collection (i.e., requirements documents) are relevant to a particular requirement (Hayes et al. 2006). Research has shown that IR methods can be effectively applied in an automated fashion for text artifacts and that IR methods reduce the effort required to perform a manual trace (Cuddeback et al. 2010). Effective use of IR methods in automating requirements tracing is based on the premise of requirement similarity, which IR algorithms try to approximate (Hayes et al. 2006). According to Wang et al. (2018b) Well known text-based IR methods include Term Frequency – Inverse Document Frequency (TF-IDF) retrieval, TF-IDF with Thesaurus, Latent Semantic Indexing (LSI), and others (Wang et al. 2018b). While these techniques are mostly automated, some research has focused on including humans in the automated traceability loop; however, this has achieved mixed results as some studies have shown that humans tend to decrease the accuracy of auto-generated trace matrices (Antoniol et al. 2017, 25–27). Wang et al. (2018a) identifies strategies to improve IR-based methods, such as documentation structure and structural analysis (Wang et al. 2018a). Additional detail regarding TF-IDF is provided in the following paragraphs given that the software developed for this thesis is based on TF-IDF. History on the development and performance of the other IR methods can be found in Hayes (2003) and in Chen (2019).

According to Campbell et al. (2016), TF-IDF "is a statistical measure that reflects how important a word is to a document in a collection or corpus" (Campbell et al. 2016, 337). According to Missaoui and Idrissa (2014), "the value of TF-IDF increases proportionally with the number of times term (t) appears in the document (d) and is offset by the frequency of documents of corpus (D) that contain the word" (Missaoui and Idrissa 2014, 77). In general, words with high TF-IDF weighting are used frequently in a document but rarely in other documents in the overall corpus (Missaoui and Idrissa 2014). In the case of requirements artifacts, terms with a high value of TF-IDF are terms used frequently in one requirement, but seldomly in other requirements contained in the relevant requirements artifacts.

According to Hayes et al. (2006)

to calculate TF-IDF, let V = $\{k_1, \ldots k_N\}$ be the vocabulary (list of keywords) of a given document collection. Then, a vector model of a document $d$ is a vector $(w_1, \ldots, w_N)$ of keyword weights, where $w_i$ is computed as $w_i = tf_i(d) * idf_i$. Here, $tf_i(d)$ is the term frequency in document $d$. The normalized frequency of keyword $k_i$ in the document $d$ as $idf_i$, called inverse document frequency, is computed as $idf_i = \log_2(n/df_i)$, where $n$ is the number of documents in the collection and $df_i$ is the number of documents in which keyword $k_i$ occurs. (9)

With a document vector $d = (w_1, \ldots w_N)$ and a query vector $q = (q_1, \ldots, q_N)$, the challenge becomes how to best determine similarity between $d$ and $q$. To help the reader visualize TF-IDF, an example TF-IDF matrix is given in Figure 5. Please note that documents (e.g., requirements) with zero or no weight indicates that the document or requirement does not contain the term.

| | a | an | apple | ate | banana | eat | i | today | will | yesterday |
|---|---|---|---|---|---|---|---|---|---|---|
| Doc 1 | | 0.0811 | 0.0811 | 0.0811 | | | 0 | | | 0.0811 |
| Doc 2 | | 0.0676 | 0.0676 | | | 0.1831 | 0 | 0.1831 | 0.1831 | |
| Doc 3 | 0.2197 | | | 0.0811 | 0.2197 | | 0 | | | 0.0811 |

Figure 5. Example TF-IDF Matrix. Source: Nishida (2016).

In summary, advanced automated traceability, hereafter referred to simply as automated traceability, relies on IR techniques such as VSM to derive vector similarity. Vectors are characterized in terms of frequency and distribution of word terms in each document (e.g., requirement) within the document collection (e.g., family of relevant requirements artifacts) using TF-IDF. The following section describes methods by which to derive similarity of vectors.

## C. VECTOR-SPACE MODEL SIMILARITY MEASURES AND APPLICATION TO INFORMATION RETRIEVAL

In general, VSM characterizes the closeness between a pair of vectors in terms of the pair-wise similarity or distance. Similarity measures measure the similarity or distance between two vectors and map them into a single numeric value, which depends on two

factors—the features of the two vectors and the similarity measure itself (Huang et al. 2008). VSM and the concept of vector similarity is depicted in Figure 6.

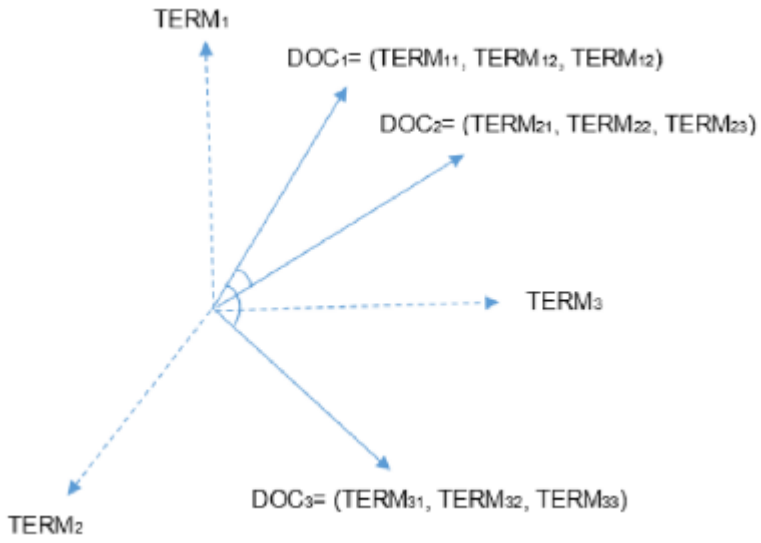



Figure 6. VSM and Vector Similarity. Source: Campbell (2016).

A variety of mathematical models such as Euclidean distance, Manhattan distance, Minkowski distance, cosine similarity, the Jaccard correlation coefficient, Pearson Correlation Coefficient, Averaged Kullback-Leibler Divergence, and others can be used as a similarity measure to measure how closely two vectors are related (Huang 2008). Similarity can be in the form of the angle between vectors as in the case of cosine similarity, in terms of relative distance as with the Euclidean method, or in terms of a correlation coefficient as in the case of Pearson or Jaccard. Details regarding their calculations can be found in Huang (2008) and Cha (2008).

Among the existing similarity measures, cosine similarity is commonly used in automated traceability research (Hayes et al. 2003). According to Hayes et al. (2003), cosine similarity is calculated as:

$$S_{cos}(d,q) = \frac{\sum_{i=1}^{N} w_i\, q_i}{\sqrt{\sum_{i=1}^{N} w_i^2 \sum_{i=1}^{N} q_i^2}}$$

Although automated tracing methods based on TF-IDF commonly use this measure to trace requirement pairs that yield the highest similarity, other similarity measures could yield requirement pairs that potentially improve upon the cosine similarity measure. One candidate measure, known as histogram distance, is the focus of this thesis. Since a histogram can be considered as a vector, numerous geometrical distances can be applied to compare histograms (Cha 2008). According to Bellet et al. (2005), the chi-square or $\chi$ 2 distance is "common in text processing and computer vision, where documents are represented as a frequency vector of (visual) words" and "is a histogram pseudo-distance derived from the chi-square statistical test" (Bellet et al. 2005, 11). According to Bellet et al. (2005), the histogram distance is defined as:

$$\chi^2(d,q) = \frac{1}{2}\sum_{i=1}^{N}\frac{(w_i - q_i)^2}{w_i + q_i}$$

Despite the variety of similarity measures one could use, their effectiveness in characterizing text similarity is still not well understood (Huang et al. 2008). Therefore, research has relied on recall and precision to quantify effectiveness between similarity measures, which this thesis will also assess.

## D. MACHINE LEARNING BASED TRACEABILITY METHODS

While IR-based tracing is well studied, IR methods have known limitations including an inability to consider the relationships between requirements beyond term similarity (e.g., semantics or context) and an inability to use past-memory to predict future traces (Gervasi and Didar 2014). Unlike IR-based methods, research has shown that Machine Learning (ML)-based methods can address these shortfalls by using an initial set of requirements traces that have been previously established as a training set (Wang et al. 2018a). In essence, "ML approaches seek to extract information from pre-existing traces, use the information to build a model of the previous linking patterns, and leverage the model in suggesting candidates for future traces" (Gervasi and Didar 2014, 143).

To overcome limitations associated with IR techniques, ML techniques can be applied "instead of or in addition to classical IR techniques" (Gervasi and Didar 2014,

143). For example, this thesis initially sought to use unsupervised ML-based clustering to generate requirements traces; however, as research progressed, it was determined that cluster-based approaches were ultimately dependent on the underlying IR methodology. As a result, the focus of this thesis shifted to the underlying component of IR and ultimately the similarity measures that characterize it. ML-based methods can also be used independent of IR-based methods. Research related to ML-based requirements tracing has focused on using supervised methods to improve the accuracy of requirements to code traceability (Antoniol et al. 2017, 22–24) and automating traceability maintenance for new requirements added to requirement sets with established traces (Mills et al. 2018). Although supervised ML techniques can result in high precision and recall, supervised methods require a labeled dataset, which is time-consuming to produce (Chen et al. 2019). Limited research with respect to unsupervised learning approaches has been performed on software artifacts, including the use of sequential semantic patterns to enhance precision of requirement links with no labeled training data (Chen et al. 2019).

THIS PAGE INTENTIONALLY LEFT BLANK

# III. METHODOLOGY

This chapter describes the process for validating whether the histogram distance similarity measure improves upon the cosine similarity measure when applied to automated requirements tracing. The overall process is depicted in Figure 7 and consists of preprocessing source data, parsing requirements from source data, generating requirement links, and evaluating requirement links. The structure of this chapter is based on this process flow and includes sections for steps A through D. A software tool was developed to perform several functions necessary to automate steps two through four.

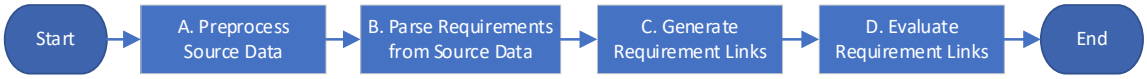



Figure 7. Thesis Methodology

## A. PREPROCESS SOURCE DATA

To standardize and streamline processing of the source data's requirements for automated requirements tracing, a series of steps were taken to preprocess the source data. These steps are delineated in Figure 8 and are described in Sections 1 and 2 below.

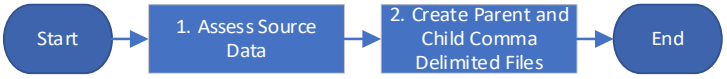



Figure 8. Preprocess Source Data

### 1. Assess Source Data

Although requirements documentation is generally similar in structure, not all requirements documents follow a standard organization or file format. For example, some requirements documents are created using a text processing application, spreadsheet application, or in a more formal tool such as DOORS while not following any standardized structure. To standardize processing of the source data's requirements for subsequent

automated requirements tracing the source data was assessed for the minimum required attributes. Minimum attributes include the following elements:

- Upper-level requirements and lower-level requirements; hereafter, referred to as parent and child requirements, respectively.
- Parent and child requirements include text and unique identifiers to describe and identify each requirement.
- Child requirements include explicit traces to their respective parent requirement to convey traceability, which are defined by the requirements document's author(s).

### 2. Create Parent and Child Comma Delimited Files

Preprocessing the source data enabled consistent and streamlined extraction of requirements by the software. Two separate comma delimited files were created; one file contained parent requirements and another file contained child requirements. The steps taken for preprocessing the source data's parent requirements into a parent requirement comma delimited file are shown in Figure 9.


Figure 9. Preprocessing Parent Requirements from Source Data

The headers in the preprocessed parent requirements file were populated with a unique numeric identifier (Parent_ID) and requirement text (Requirement). To simplify array processing, the comma delimited file also modified the parent identifiers such that they are purely integers instead of alphanumeric objects. To complete the file containing parent requirements, the text associated with each parent requirement was populated in descending order relative to its parent identifier.

With the parent source data preprocessed, the next step required preprocessing the child source data. The steps taken for preprocessing the source data's child requirements into a child requirement comma delimited file is shown in Figure 10.

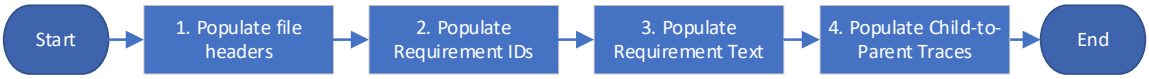



Figure 10. Preprocessing Child Requirements from Source Data

The file headers in the preprocessed file attributes included a unique numeric identifier (*Child_ID*), requirement text (*Description*), and trace to parent requirements (*Parent_ID*). Unlike the source document, the comma delimited file used unique attributes for each parent requirement.

Traces between the child requirement and parent were represented in a binary fashion; a one represented a trace while a zero represented no trace. To simplify array processing, the comma delimited file also modified the child identifiers such that they were purely integers instead of alphanumeric objects. To complete the file containing child requirements, the text associated with each child requirement was populated in descending order relative to its parent identifier.

## B. PARSE REQUIREMENTS FROM SOURCE DATA

With the source data manually structured and ready to be processed, the first step in automated requirements tracing involved parsing the requirements from the preprocessed source data. This process is summarized in Figure 11 and is described in Sections 1 through 3.

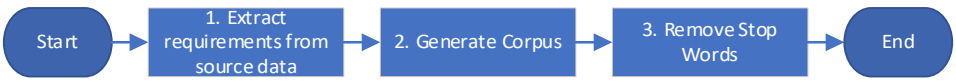



Figure 11. Parsing Requirements from Source Data

### 1. Extract Requirements from Source Data

Using the preprocessed source data generated in III.A, the software tool extracted all parent requirements from the parent and child comma delimited files using a library known as Pandas and stored the associated attributes—including the actual parent-child traces—in separate arrays as shown in Table 2 and Table 3 for the full set of parent requirements and child requirements contained within each file. During this step, the total number of parent and child requirements were stored as variables. The actual parent-child traces were performed by the *build_actual_traces* function and the results were contained in the array *actual_traces*, as shown in Appendix B. During software development, verification was performed to ensure proper extraction of requirements and *actual_traces* from a subset of parent and child requirements and then the full dataset. Similarly, verification was performed on the parent, child, and *actual_traces* arrays during software development to ensure the values corresponding with array indices aligned with the values delimited in the preprocessed source data.

Table 2. Parent Array

| **Parent_ID** | **Requirement** |
|---|---|
| *Element 0 contains parent requirement ID* | *Element 1 contains the requirement text* |

Table 3. Child Array

| Child_ID | Parent_1 | Parent_n | Requirement |
|---|---|---|---|
| *Element 0 contains child requirement ID* | *Element 1 contains binary integer; 1 = Trace, 0 = No trace* | *Elements n through 12 contain binary integer* | *Element 13 contains the requirement text* |

### 2. Generate Corpus

Once the parent array and child arrays were created, the next step merged the collection of parent and child requirements text into a collection of documents, (i.e., a

corpus). This was completed by combining *parent_corpus* and *child_corpus* into a single variable named *corpus* as shown in Appendix B. Verification was performed to ensure that the merged corpus contained the full set of requirements text associated with all parent and child requirements in the source data.

### 3. Remove Stop Words

To help improve effectiveness of subsequent TF-IDF calculations, common words deemed too frequent and insignificant were excluded from the corpus. The list of excluded words is known as stop words. According to Manning et al. (2008), stop words are "extremely common words which would appear to be of little value in helping select documents (i.e., requirements) matching a user need," which are "excluded from the vocabulary entirely" (Manning et al. 2008, 27). The software developed included the standard English stop word list provided by the Sci-kit library with the inclusion of the word "shall" which is a term used in every requirement and therefore not beneficial for distinguishing requirements from each other. Verification was performed to ensure that the merging of parent and child requirements into a single corpus produced the expected result. Further, verification was performed to ensure omission of stop words by using a parent and child requirements list containing known stop words such as "shall."

## C. GENERATE REQUIREMENT LINKS

With the corpus established, the next step in automating requirements tracing involved generating the TF-IDF matrix, generating similarity measure scores, and determining parent-child traces based on the scores and predefined thresholds. A summary of this process is depicted in Figure 12; steps 1 through 3 are described below.


Figure 12. Process for Generating Requirement Links

### 1. Generate TF-IDF Matrix

The first step in deriving requirement pairs from the corpus was generating a TF-IDF matrix, which was based on all unique terms in the corpus. Calculations associated with generating the TF-IDF matrix are described in II.C. Built-in TF-IDF functionality provided with the scikit-learn Python libraries was used to streamline TF-IDF vectorization of each requirement in the corpus, as shown in Appendix B. The resulting output of the TF-IDF vectorization function was a matrix with TF-IDF calculated weights for each term in the requirement's text. An example of a TF-IDF matrix is shown in Figure 5 in Chapter II.

### 2. Generate Similarity Measure Scores

With a TF-IDF matrix established the software then compared parent-child requirements and predicted whether a requirement pair was related. The requirement pair was classified based on the similarity measure and predefined threshold used. Similarity measures examined included histogram distance and cosine similarity, which correspond to the *histogram_similarity* and *cos_similarity* functions in Appendix B. Calculations for these similarity measures are described in II.C. The output of both similarity measure functions was a two-dimensional array named *scores*. The first dimension of this array represented the parent requirement number and the second dimension represented the child requirement number. The value of the corresponding array indices represented the similarity measure score resulting from the *histogram_similarity* or *cos_similarity* functions. During software development, verification was performed to ensure that the *histogram_similarity* and *cos_similarity* functions produced the expected result by comparing the software's output against manual calculations.

### 3. Generate Parent-Child Traces

With the scores collected for each similarity measure against all potential parent-child requirement pairs, the next step established which pairs should be considered parent-child traces. As mentioned, the similarity measure and corresponding threshold value ultimately determine whether two requirement vectors are related. For histogram distance, the threshold cut-off distance was less than or equal to the predefined threshold.

Requirement pairs that were less than or equal to the distance threshold were classified as a requirement pair for the given threshold since distances greater than the threshold are less stringent. For cosine similarity, the threshold cut-off was greater than or equal to the predefined threshold since similarity or orientation values higher than the threshold values are more stringent. In other words, the threshold value is the most relaxed distance or similarity value allowed to classify a requirement pair.

Based on the formulations provided in II.C and as captured in the *perform_histogram_tracing* and *perform_cosine_tracing* functions provided in Appendix B, the scores array and user-defined threshold were passed to each function, which effectively searched the *scores* array for values within the threshold cut-off range and assigned a binary value to *predicted_histogram_traces* and *predicted_cosine_traces* arrays, which are two dimensional. The first dimension of these arrays represented the parent requirement number, and the second dimension represented the child requirement number. The corresponding indices contained either a one to represent a trace or a zero to represent no trace between the parent-child pair. This activity built the *predicted_histogram_traces* and *predicted_cosine_traces* arrays with parent-child requirement links for a given threshold for both similarity measures and concluded the process of automated requirements tracing.

The final step in this thesis' methodology evaluated the generated requirements links and compared the histogram and cosine similarity measure predicted results against the actual requirements traces.

## D. EVALUATE REQUIREMENT LINKS

With automated tracing performed for both similarity measures, the final step evaluated the predicted requirement links to understand the overall performance of each similarity measure and how they compared. The process for this evaluation is summarized in Figure 13.


Figure 13. Process for Evaluating Requiring Links

### 1. How to Calculate Recall and Precision Scores

As described in Chapter II, similarity measures characterize the proximity of a pair of vectors in terms of the pair-wise similarity or distance between the vectors. Understanding the result quality of different similarity measures is important when choosing the best one for automated requirements traceability (Huang et al. 2012).

According to Hayes et al. (2003), "the quality of IR [similarity measures] is measured by how well the documents [requirement pairs] match the user's expectations" (Hayes et al. 2003, 3). The user's expectation in this application is the accurate tracing of requirement pairs, which is measured by comparing the measure against the requirements document author's derivation. The quality of similarity measures is typically formalized with the two metrics described in Chapter II: precision and recall (Hayes et al. 2003). According to Hayes et al. (2003), precision is defined as "the fraction of the relevant documents in the list of all documents returned by the similarity measure," while "recall is the fraction of the retrieved relevant documents in the entire set of documents, retrieved and omitted, that result from the similarity measure" (Hayes et al. 2003, 3). According to Scikit-learn (2021), precision and recall are calculated as follows:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

where:

- *True positives are predicted requirements traces that are true*
- *False positives are predicted requirements traces that are false*
- *False negatives are omitted traces that are actually true traces*

Using the *predicted_histogram_traces, predicted_cosine_traces, and actual_traces* arrays, the software determined true positives, false positives, and false negatives for each

similarity measure and threshold used. As described below and as shown in the *recall* and *precision* functions in Appendix B, the true positives, false positives, and false negatives were determined in the following manner:

- If *predicted_traces* [parent ID][child ID] equals 1 and *actual_traces* [parent ID][child ID] equals 1, then the predicted pair is a true positive.
- If *predicted_traces* [parent ID][child ID] equals 1 and *actual_traces* [parent ID][child ID] equals 0, then the predicted pair is a false positive.
- If predicted_traces [parent ID][child ID] equals 0 and actual_traces [parent ID][child ID] equals 1, then the predicted pair is a false negative.

The true positive, false positive, and false negative scores were aggregated for each similarity measure for final comparison. For verification purposes, the computed true positive, false positive, and false negative results were compared with manually scored results using preprocessed source data with a subset of parent and child requirements.

### 2. How to Calculate F-Score

To simplify evaluating a similarity measure's performance, recall and precision were combined into a single measure known as F-score (Koehrsen 2018). According to Koehrsen (2018), "the [F-score] is the harmonic mean of precision and recall taking both metrics into account in the following equation" and is defined by:

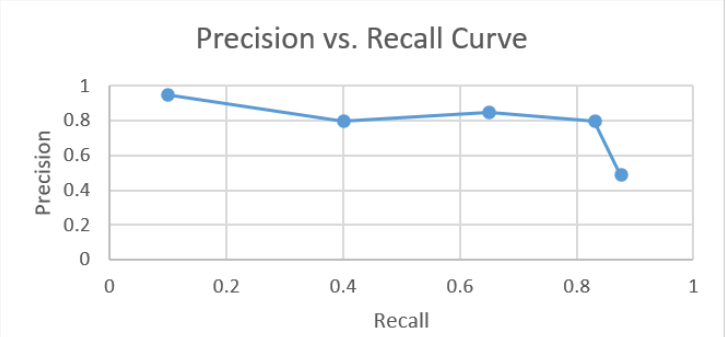
$$F - score = 2\frac{(Precision)(Recall)}{Precision + Recall}$$

According to Koehrsen, the harmonic mean is used instead of an average because it penalizes extreme recall and precision values. While there are other methods for characterizing precision and recall as a single metric, the F-score is commonly used in the application of information retrieval (Koehrsen 2018). For this thesis, F-score was calculated based on the respective recall and precision scores that resulted from the histogram and cosine similarity measures.

### 3. Compare Similarity Measure Results

The final step of the methodology used in this thesis compared the quality of histogram and cosine similarity measures using the approach and precision, recall, and F-score calculations described above. As shown in Appendix B, the software developed contains functions named *recall* and *precision* which compared the arrays *predicted_cosine_traces and predicted_histogram_traces to actual_traces*. This comparison evaluated the true positives, false positives, and false negatives needed to derive recall, precision, and ultimately the F-score for both similarity measures. These scores were calculated over a variety of thresholds, which were calculated per the formulations provided above.

Recall, precision, and F-score graphs were automatically generated to evaluate the performance of each similarity measure over a variety of threshold values. Similarity measure threshold values are the distance or correlation cut-off value resulting from a similarity measure's calculation of the parent-child requirement pair. When graphing precision and recall, the resulting curve depicts the tradeoff between precision and recall for different similarity measure thresholds as shown in the example provided in Figure 14.




| Threshold | 0.8 | 0.7 | 0.6 | 0.5 | 0.4 |
|---|---|---|---|---|---|
| Recall (X) | 0.1 | 0.4 | 0.65 | 0.83 | 0.875 |
| Precision (Y) | 0.95 | 0.8 | 0.85 | 0.8 | 0.49 |

Figure 14. Example of Precision vs. Recall Curve

Based on the computed recall and precision scores, the similarity measure with the highest F-score generally results in the best performance for automated tracing. According to Hayes, a general baseline for comparison of acceptable, good, and excellent recall and precision is captured in Table 4 (Hayes et al. 2006).

Table 4. General Baseline of Performance Quality. Source: Hayes et al. (2006)

| **Measure** | **Acceptable** | **Good** | **Excellent** |
|---|---|---|---|
| **Recall** | 60% - 69% | 70% - 79% | 80% - 100% |
| **Precision** | 20% - 29% | 30% - 49% | 50% - 100% |

THIS PAGE INTENTIONALLY LEFT BLANK

# IV. EXPERIMENTS AND RESULTS

This chapter describes the experiments performed to validate whether histogram distance improves upon cosine similarity for the purposes of automated traceability. An analysis of the results is provided to compare the two similarity measures, their performance, and their utility to systems engineers.

## A. EXECUTION OF EXPERIMENTS

Using the methodology described in Chapter III, this section summarizes the results of experimentation in each of the four steps: Preprocess Source Data, Parse Requirements from Source Data, Genera Requirement Links, and Evaluate Requirement Links.

### 1. Preprocess Source Data

Preprocess Source Data contains two sub-processes: Assess Source Data and Create Parent and Child Comma Delimited Files. The results from these sub-processes are described in Sections a and b below.

#### *a. Assess Source Data*

The source data used by the software developed as part of this thesis for comparing similarity measures is based on publicly available requirements documentation from the National Aeronautics and Space Administration (NASA) (McGarry 2016). To standardize processing of the source data's requirements for automated tracing, the source data was assessed for the minimum attributes described in chapter III.A.1. The structure of the source data consisted of parent and child requirements, as required. The source data also identified child-to-parent tracing, which was determined by the requirements document's author, Jan McGarry.

The source data's parent requirement structure is depicted in Figure 15, which included a subset of the source data's parent level requirements. In general, the parent requirement attributes consist of a unique requirement number and requirement text.

| | SGSLR Level 3 Requirements |
|---|---|
| | |
| | |
| SGSLR # | Description |
| SLF | Functional requirements |
| SLF 3.1 | With a standard clear atmosphere or better, SGSLR stations shall be capable of 24 x 7 tracking of satellites whose arrays satisfy the ILRS retro-reflector guidelines, and whose altitudes are 300 km to 22,000 km. |
| SLF 3.1.1 | With a standard clear atmosphere or better, SGSLR stations shall be capable of tracking geosynchronous satellites whose arrays satisfy the ILRS retro-reflector guidelines. |

Figure 15. Example of Source Parent Requirement Structure.
Source: McGarry (2016).

The source data structure of child requirements is depicted in Figure 16, which includes a subset of the NASA source document's child level requirements. Similar to the parent-level requirements, each child requirement has a corresponding number for identification purposes in addition to requirement text that is derived from the associated parent. Child requirements have an additional attribute that contains the numeric identification of associated parent requirements, and all child requirements have at least one parent.

| | Controlled by: Jan McGarry | SGSLR Level 4 Requirements |
|---|---|---|
| | | 7-Aug-16 |
| | | Version 2.0 |
| | | |
| SGSLR # | Parent | Description |
| 4.1 | | Telescope and Gimbal (GTA) |
| 4.1.1 | | Telescope |
| 4.1.1.1 | SLF 3.1<br>SLF 3.1.1 | The telescope subsystem shall be designed to transmit from the optical bench subsystem and return receive light from the satellite to the optical bench subsystem |
| 4.1.1.2 | SLBP 3.4 | The telescope shall be capable of operation within -40oC to +50oC and wind speeds up to 18 m/s. |
| 4.1.1.3 | SLBP 3.7<br>SLBP 3.4 | The telescope shall be capable of survival at temperatures ranging from -50oC to +55oC. |

Figure 16. Example of Source Child Requirement Structure.
Source: McGarry (2016).

In total, the source NASA documentation contained 12 parent requirements and two hundred and four child requirements.

### *b. Create Parent and Child Comma Delimited Files*

While the source data was relatively structured, the lack of structure consistency throughout the document required manual preprocessing of the data in comma delimited files to ensure absolute consistency and correct mapping of identifiers to requirement text. Preprocessing the source data enabled consistent and streamlined extraction of requirements by the software. A subset of the resulting comma delimited file for parent requirements is depicted in Figure 17.

| Parent_ID | Requirement |
|---|---|
| 1 | With a standard clear atmosphere or better, SGSLR stations shall be capable of 24 x 7 tracking of satellites whose arrays satisfy the ILRS retro-reflector guidelines, and whose altitudes are 300 km to 22,000 km. |
| 2 | With a standard clear atmosphere or better, SGSLR stations shall be capable of tracking geosynchronous satellites whose arrays satisfy the ILRS retro-reflector guidelines. |
| 3 | Data precision for LAGEOS NPT shall be < 1.5 mm when averaged over a one month period. |
| 4 | The LAGEOS Normal Point range bias shall be stable to 1.5 mm over 1 hour. |
| 5 | Over one year the RMS of station's LAGEOS NPT range biases shall be < 2mm. |

Figure 17. Subset of Preprocessed Parent Requirements. Adapted from McGarry (2016).

With the parent source data preprocessed, the next step preprocessed the child source data. The structure of the comma delimited file for child requirements is depicted in Figure 18, which contains a subset of the full set of child requirements that were preprocessed. The full parent and child comma delimited files used for this thesis are included in Appendix A for reference.

| Child_ID | Parent_1 | … | Parent_12 | Description |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | The telescope subsystem shall be designed to transmit from the optical bench subsystem and return receive light from the satellite to the optical bench subsystem |
| 2 | 0 | 0 | 0 | The telescope shall be capable of operation within -40oC to +50oC and wind speeds up to 18 m/s. |
| 3 | 0 | 0 | 0 | The telescope shall be capable of survival at temperatures ranging from -50oC to +55oC. |
| 4 | 0 | 0 | 0 | The telescope optical paths shall be sealed against dust and contamination. |
| 5 | 0 | 0 | 0 | The telescope shall support the mounting of system support equipment. |

Figure 18. Subset of Preprocessed Child Requirements. Adapted from McGarry (2016).

### 2. Parse Requirements from Source Data

With the source data manually preprocessed, the software automated parsing of the requirements from the source data by following the sub-processes described in chapter III.B, which included extracting requirements from source data, generating the corpus, and removing stop words. At the completion of executing these steps, a corpus containing the requirements text of all parent and child requirements was generated based on the source data with stop words removed. A subset of the corpus is shown in Figure 19.

```
corpus =  ['With a standard clear atmosphere or better, SGSLR stations shall be capable of
```

Figure 19. Subset of the Corpus

### 3. Generate Requirements Links

With the corpus established, the next step in automating requirements tracing involved generating the TF-IDF matrix, generating similarity measure scores, and determining parent-child traces based on the scores and predefined thresholds.

The resulting TF-IDF matrix contained 682 unique terms and numbers based on the source data with stop words removed. A subset of the TF-IDF matrix generated by the software tool is shown in Figure 20. The header contains the unique terms while each row below the header contains the TF-IDF term weights for each of the 215 requirements.

```
The TF-IDF matrix is:
['000', '001', '05', '10', '100', '1000', '1064'
0 :  0.21840904634655348 0.0 0.0 0.0 0.0 0.0 0.0
1 :  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2 :  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3 :  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4 :  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
5 :  0.6712034044855253 0.0 0.0 0.18218296912616
6 :  0.0 0.0 0.0 0.0 0.3923270268192407 0.0 0.0
7 :  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
8 :  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Figure 20. Subset of the TF-IDF matrix

With the TF-IDF matrix generated, requirement pairs were then classified based on the similarity measure and predefined threshold used, which determined the cut-off distance as described in Chapter III.C. A variety of thresholds were arbitrarily applied to each similarity measure to derive the similarity measure's performance curve for classifying parent-child requirement pairs. The thresholds applied to the histogram and cosine similarity measures are shown in Table 5.

Table 5. Similarity Thresholds Selected for Each Similarity Measure

| **Threshold Sample** | **Histogram Distance Thresholds** | **Cosine Similarity Thresholds** |
|---|---|---|
| 1 | 2 | 0.0025 |
| 2 | 2.1 | 0.003 |
| 3 | 2.2 | 0.005 |
| 4 | 2.3 | 0.006 |
| 5 | 2.4 | 0.007 |
| 6 | 2.5 | 0.008 |
| 7 | 2.6 | 0.009 |
| 8 | 2.7 | 0.01 |
| 9 | 2.8 | 0.02 |
| 10 | 2.9 | 0.03 |
| 11 | 3 | 0.04 |

| Threshold Sample | Histogram Distance Thresholds | Cosine Similarity Thresholds |
|---|---|---|
| 12 | 3.1 | 0.05 |
| 13 | 3.2 | 0.06 |
| 14 | 3.3 | 0.07 |
| 15 | 3.4 | 0.08 |
| 16 | 3.5 | 0.09 |
| 17 | 4 | 0.1 |
| 18 | 5 | 0.11 |

#### 4. Evaluate Requirement Links

With automated tracing performed for both similarity measures for each threshold in Table 5 , the final step evaluated the predicted requirement links to understand the overall performance of each similarity measure and how they compared. For each threshold and similarity measure, precision, recall, and F-score were calculated in the software tool as described in Chapter III.D. The results are presented in the following section.

## B. SUMMARY AND ANALYSIS OF EXPERIMENTAL RESULTS

This thesis performed automated requirements tracing on a public dataset to address the fundamental research question: how effective is histogram distance as a similarity measure in predicting trace links compared with the cosine similarity measure? Based on the thresholds used in Table 5, the resulting precision, recall, and F-scores were generated using a software tool to answer this question. The following sections summarize the performance of the histogram distance similarity measure and cosine similarity measure, concluding with a comparison of both similarity measures for the application of automated requirements tracing.

### 1. Performance of Histogram Distance as an Automated Tracing Similarity Measure

Given the thresholds outlined in Table 6, histogram distance's resulting precision and recall were inversely related as shown in Figure 21. In other words, threshold values that resulted in improved recall would negatively affect precision, and vice-versa.

Table 6. Summary of Histogram Distance Results

| Histogram Results | | | | |
|---|---|---|---|---|
| Threshold Sample | Threshold | Precision (Y) | Recall (X) | F-Score |
| 1 | 2 | 0.6667 | 0.0131 | 0.0258 |
| 2 | 2.1 | 0.5333 | 0.0175 | 0.0339 |
| 3 | 2.2 | 0.4545 | 0.0328 | 0.0612 |
| 4 | 2.3 | 0.4333 | 0.0569 | 0.1006 |
| 5 | 2.4 | 0.3393 | 0.0832 | 0.1336 |
| 6 | 2.5 | 0.3179 | 0.1357 | 0.1902 |
| 7 | 2.6 | 0.2571 | 0.1969 | 0.2230 |
| 8 | 2.7 | 0.2202 | 0.2910 | 0.2507 |
| 9 | 2.8 | 0.2153 | 0.4114 | 0.2827 |
| 10 | 2.9 | 0.1949 | 0.5142 | 0.2826 |
| 11 | 3 | 0.1850 | 0.5930 | 0.2820 |
| 12 | 3.1 | 0.1726 | 0.6477 | 0.2726 |
| 13 | 3.2 | 0.1765 | 0.7243 | 0.2839 |
| 14 | 3.3 | 0.1773 | 0.7681 | 0.2881 |
| 15 | 3.4 | 0.1794 | 0.8249 | 0.2948 |
| 16 | 3.5 | 0.1801 | 0.8665 | 0.2982 |
| 17 | 4 | 0.1857 | 0.9847 | 0.3125 |
| 18 | 5 | 0.1871 | 1.0000 | 0.3153 |

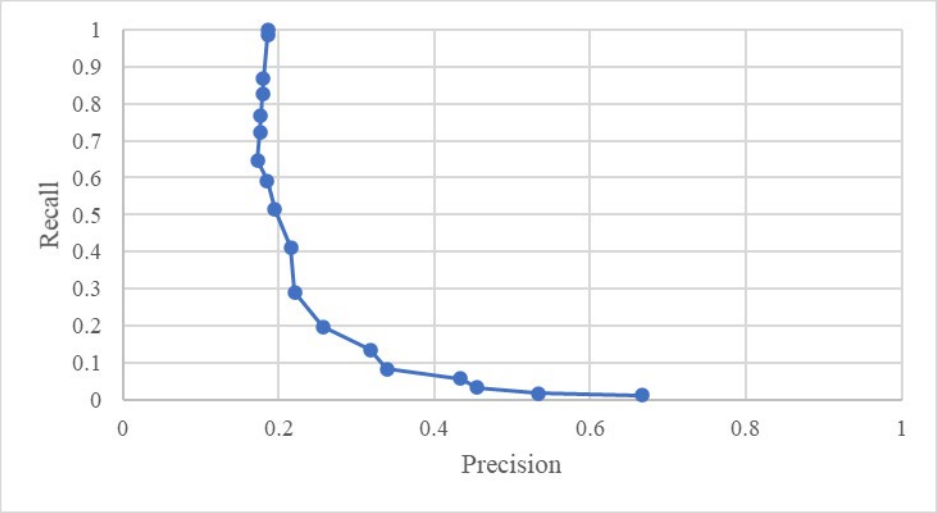



Figure 21. Histogram Recall vs. Precision

To simplify evaluating histogram distance's performance, F-score was computed using the precision and recall values. Histogram distance F-scores generally improved with higher threshold values as shown in Figure 22; in the case of this experiment, the highest F-score achieved was 0.315 when the histogram distance threshold was set to five.

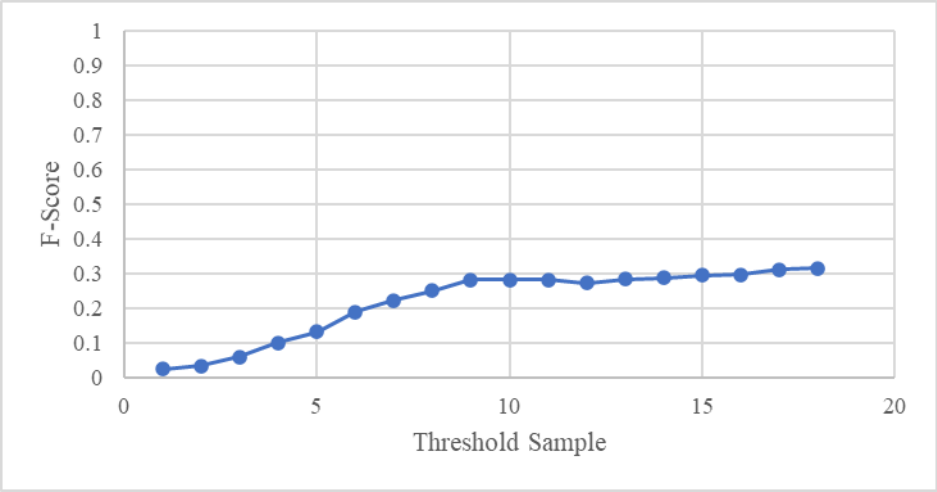



Figure 22. Histogram F-Score

### 2. Performance of Cosine Similarity as an Automated Tracing Similarity Measure

Using the thresholds outlined in Table 7, cosine similarity's resulting precision and recall reveal no clear pattern as shown in Figure 23. There was no apparent pattern associated with threshold values and improved recall or precision, which is likely due to the fundamental difference between the similarity measures. While the histogram method returns a distance between vectors, the cosine method returns the cosine of the angle between two vectors. In other words, "the cosine similarity captures the orientation (the angle) of the documents and not the magnitude" between vectors in a multi-dimensional space, where the number of dimensions is the number of words in the corpus (Prabhakaran 2018).

Table 7. Summary of Cosine Similarity Results

| Cosine Results | | | | |
|---|---|---|---|---|
| Threshold Sample | Threshold | Precision (Y) | Recall (X) | F-Score |
| 1 | 0.0025 | 0.3726 | 0.2976 | 0.3309 |
| 2 | 0.003 | 0.3431 | 0.4880 | 0.4029 |
| 3 | 0.005 | 0.3441 | 0.4880 | 0.4036 |
| 4 | 0.006 | 0.3447 | 0.4880 | 0.4040 |
| 5 | 0.007 | 0.3442 | 0.4836 | 0.4022 |
| 6 | 0.008 | 0.3417 | 0.4748 | 0.3974 |
| 7 | 0.009 | 0.3403 | 0.4661 | 0.3934 |
| 8 | 0.01 | 0.3458 | 0.4661 | 0.3970 |
| 9 | 0.02 | 0.3793 | 0.3370 | 0.3569 |
| 10 | 0.03 | 0.3456 | 0.2473 | 0.2883 |
| 11 | 0.04 | 0.3379 | 0.2144 | 0.2624 |
| 12 | 0.05 | 0.3398 | 0.1904 | 0.2440 |
| 13 | 0.06 | 0.3363 | 0.1641 | 0.2206 |
| 14 | 0.07 | 0.3855 | 0.1510 | 0.2170 |
| 15 | 0.08 | 0.4514 | 0.1422 | 0.2163 |
| 16 | 0.09 | 0.5043 | 0.1291 | 0.2056 |
| 17 | 0.1 | 0.4842 | 0.1007 | 0.1667 |
| 18 | 0.11 | 0.5238 | 0.0963 | 0.1627 |

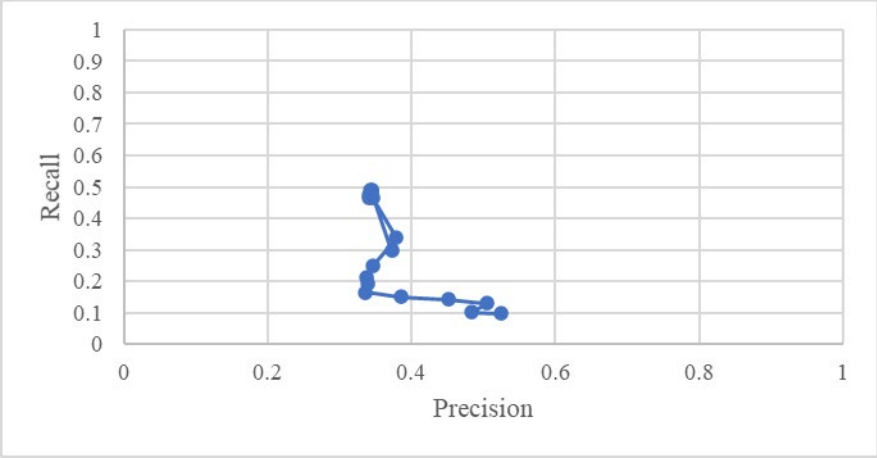



Figure 23. Cosine Recall vs. Precision

Cosine similarity's F-score generally yielded improved scores with threshold values closer to zero as shown in Figure 24; in the case of this experiment, the highest F-score achieved was 0.402 when the cosine angle threshold was set to 0.003.

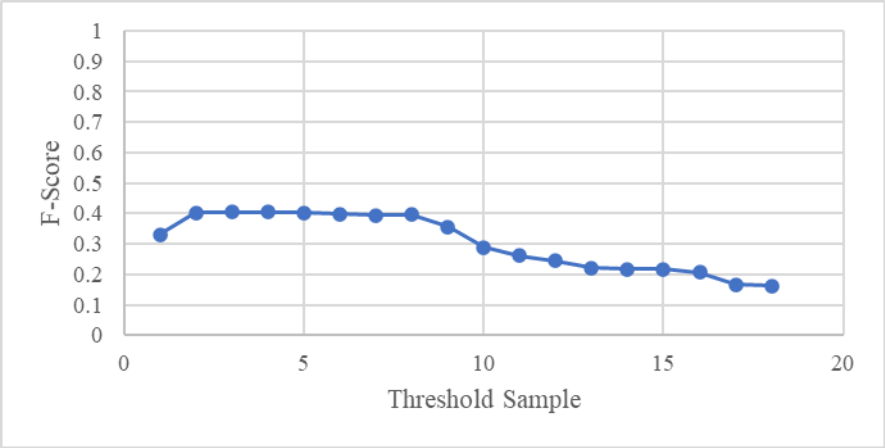



Figure 24. Cosine F-Score

Cosine similarity's best F-score outperformed histogram distance's best F-score as shown in Table 8. However, depending on the systems engineer's goals when performing requirements tracing, one similarity measure may be deemed more desirable than the other. Systems engineers must consider the tradeoffs of recall and precision when selecting a similarity measure for automated requirements tracing.

Table 8. Summary of Highest F-Scores for the Given Dataset

| **Histogram** | | | **Cosine** | | |
|---|---|---|---|---|---|
| *Precision* | *Recall* | *F-score* | *Precision* | *Recall* | *F-score* |
| 0.18 | 1.00 | 0.32 | 0.34 | 0.48 | 0.40 |

Histogram distance, as shown in Figure 25, has the potential to generate a higher level of precision than the cosine method at the expense of recall for the given dataset. This is important in a situation where the systems engineer may value precision at the expense of returning fewer true traces. In practice, this may occur when manual tracing has been performed, but the systems engineer is seeking a second vote on a handful of requirements

in question. This outcome can be achieved by using histogram distance with lower threshold values, though this may depend on the dataset.

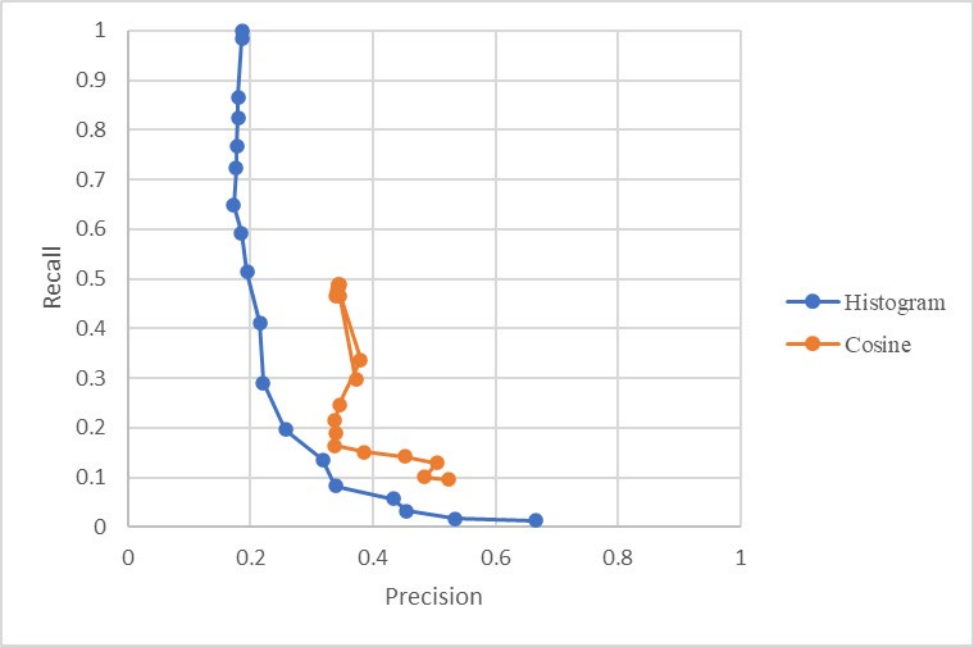



Figure 25. Histogram and Cosine Recall vs. Precision

Histogram distance also has the potential to generate a higher level of recall than the cosine method for the given dataset at the expense of lower precision when high threshold values are used. In some cases, this may be desirable by systems engineers who desire high recall yet can tolerate low precision. This situation may be desirable particularly when there is a human in the loop to validate the false positives; however, depending on the dataset, histogram distance may be counterproductive as determined against this study's dataset.

In the case of the studied dataset, the highest F-score for histogram distance was achieved when the recall reached 1.0, which means that every actual trace was identified at the expense of elevated false positives. In this instance, histogram distance predicted all 457 actual traces; however, the false positives amounted to 1,985. The systems engineer(s) would need to review 2,442 of a total 2,448 total possible traces to identify the 457 actual traces. Given the low degree of precision, false positives of this magnitude could be

counterproductive and arguably involve more work to manually resolve than to manually establish without the use of automated tracing. Given that many actual requirement pairs in the dataset are based on the dataset author's domain knowledge and not based on matching terms, the low degree of precision demonstrates the well-known IR limitation of classifying pairs on a literal term basis. For example, child requirement 1 refers to the "optical bench subsystem" while the associated parent requirements refer to the terms "satellites" and "retro-reflector." While the derivation is valid, term similarity is low. Had the dataset's author used terms from the parent requirements more consistently in the children requirements, precision would have likely improved.

Although cosine similarity did not demonstrate an ability to reach high degrees of precision or recall, the similarity measure performed better when considering both parameters simultaneously. In general, the cosine similarity function is the best choice as it will generally achieve higher recall and precision for a particular threshold, and therefore achieve a higher overall F-score than histogram distance as shown in Figure 26.

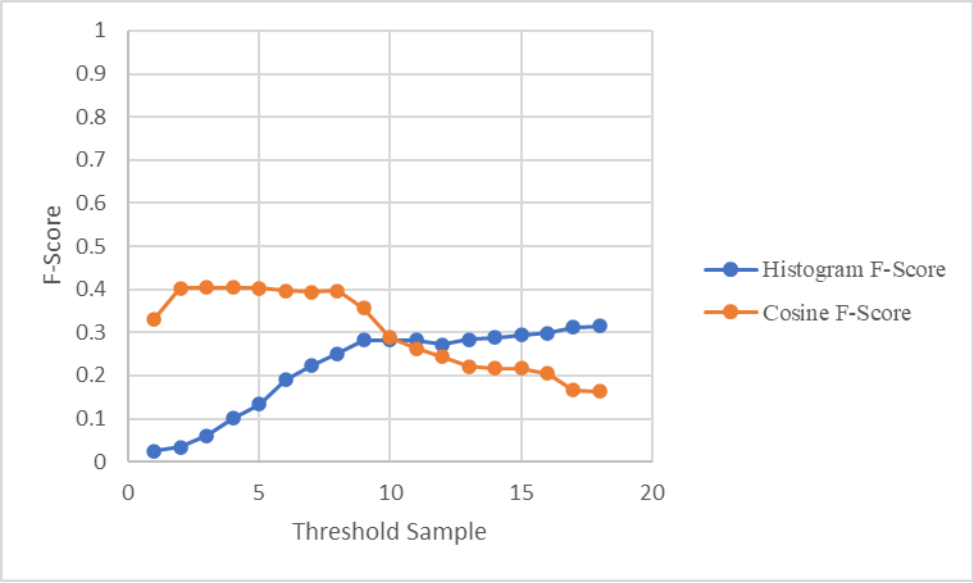



Figure 26. Histogram and Cosine F-Scores for Varying Thresholds

In the case of the studied dataset, the highest F-score for histogram distance was achieved when the recall reached 0.48, which means that approximately half of all actual

traces were identified at the expense of increased precision. When a threshold of 0.003 was used, cosine similarity predicted 223 actual traces with 424 false positives—fewer than one quarter of histogram distance's false positives.

To put the resulting precision and recall scores into the context of real-world performance quality, histogram and cosine similarity scores were compared with the general baseline of performance quality provided in Table 4. While histogram distance's recall and threshold can vary considerably depending on the threshold used, the highest F-score achieved given the dataset had excellent recall, yet less than acceptable precision. For histogram thresholds that yielded acceptable recall, precision was considered less than acceptable. For the highest cosine similarity F-score achieved given the dataset, recall can also be considered less than acceptable, while its precision can be considered good. Table 9 summarizes the performance of the two similarity measures in this thesis' dataset against the general baseline of acceptable, good, and excellent scores. Recall and precision scores associated with the highest F-scores achieved in this study were used against the baseline of performance.

Table 9. Histogram and Cosine Performance vs General Baseline of Performance

| **Similarity Measure** | | **Less than Acceptable** | **Acceptable** | **Good** | **Excellent** |
|---|---|---|---|---|---|
| Histogram Distance | Recall | - | - | - | ✓ |
| | Precision | ✓ | - | - | - |
| Cosine Similarity | Recall | ✓ | - | - | - |
| | Precision | - | - | ✓ | - |

THIS PAGE INTENTIONALLY LEFT BLANK

# V. CONCLUSIONS

## A. SUMMARY

Requirements management is a fundamental systems engineering activity that involves managing traceability throughout a system's life cycle. Given issues with conventional requirements tracing methods, automated tracing methods can help systems engineers ensure links between parent and child requirements are correct while preventing common requirements traceability issues, such as missing traces. In this analysis a new similarity measure was studied and compared against the commonly used measure, cosine similarity, with the objective of understanding the performance and utility of new similarity measures in automated requirements tracing.

A software tool was developed to compare similarity measures and ultimately address the research objective of whether histogram distance, when used as a similarity measure, can outperform the commonly cosine similarity method. Using publicly available requirements documentation from NASA, which was manually preprocessed, the software tool analyzed 215 requirements and classified parent-child requirement pairs using the histogram distance and cosine similarity measures and 18 different similarity measure thresholds for each similarity measure. Precision, recall, and F-scores were produced for each permutation, yielding maximum F-scores for each similarity measure under a variety of thresholds enabling insight into the performance of each IR-based similarity measure.

## B. INSIGHTS

This study provided several insights into the use and selection of similarity measures to support systems engineers with automated requirements tracing. The act of experimenting with a functional software tool provided a deeper understanding of the behavior associated with different similarity measures beyond solely considering F-score, which is typically the focus of automated requirements tracing research.

This study revealed that histogram distance is generally less suitable as a similarity measure when compared to cosine similarity. While high recall may be tolerable in some cases at the expense of lower precision—particularly if a human in the loop is part of the

solution to validate the false positives associated with low precision—the amount of time needed to resolve the false positives may be greater than the time spent conducting the process without automated tools if precision is too low. In the case of this study, the number of false positives requiring human review rendered histogram distance counterproductive, arguably involving more work to manually resolve than to manually establish without the use of automated tracing. This study also revealed that cosine similarity, while outperforming histogram distance, resulted in less than acceptable recall for the given dataset. Therefore, augmenting the IR methodology used in this thesis with other IR or ML-based approaches—such as those mentioned in II.C and II.D—is needed to increase the overall effectiveness of the similarity measures to acceptable levels.

Despite research towards more robust similarity methods, using IR as the sole basis of automated tracing has inherent limitations that may not result in degrees of precision and recall high enough to supplant human involvement or even foster use as a requirements management tool amongst systems engineers. IR, if used exclusively for automated tracing, fundamentally measures similarity between requirement pairs based on requirement term weights assigned by frequency. This approach ignores context of the requirement within the overall requirements artifacts, and it does not consider the requirement author's underlying subjective intentions or rationale used when conducting the original derivation. This leads to two additional insights with respect to improving performance of IR-based techniques. First, improvements in preprocessing source requirements to take advantage of TF-IDF principles could potentially yield improved precision and recall. For example, this study included the standard scikit-learn stop words list with the inclusion of the term "shall." A term frequency analysis of the source requirements could reveal additional high-frequency low-value terms that may result in a more optimized TF-IDF weighting. Second, given knowledge of TF-IDF weighting principles, systems engineers could take advantage of automated tracing by writing requirements such that parent requirements and their derived child requirements contain matching unique terms seldomly used elsewhere in the requirements artifacts, as feasible. While some requirements development efforts may facilitate this approach, this practice may not always be straightforward or feasible depending on number of systems engineers managing requirements or the terminology

characterizing the system at higher and lower levels of system design. This likely holds true regardless of domain (e.g., medical, aeronautical) since requirements derivation is not a domain-specific process and requirements terminology is typically technical in nature, where terms describing the system or functionality are carried across higher to lower levels of derivation. Although improving preprocessing of the source data and writing requirements to take advantage of TF-IDF weighting may enhance IR-based results, fully automated IR-based methods may likely not yield enough precision to instill confidence for most systems engineers to use such tools without human intervention or without other techniques, such as ML, used in tandem.

Although this study did not objectively analyze time associated with automated tracing or compare it with conventional methods, a final insight obtained through this research was the benefit of time-savings made possible by automating requirements tracing. Throughout a system's life cycle, requirements at different levels will change and it is the systems engineer's responsibility to ensure that requirements are correctly mapped along the way. This activity requires repeated analysis of requirements traces as requirements are added and removed, which can be a time-consuming and error prone endeavor especially for larger programs involving hundreds or thousands of requirements. Time spent maintaining requirement linkages over the course of a program as requirements evolve can add significant costs. The author's experience with requirements tracing using conventional methods and the automated methods described herein allowed realization of the stark difference in time required by each method. Manually tracing requirements using a tool such as DOORS can take a systems engineer minutes or significantly longer per requirement depending on the complexity and number of system requirements analyzed. Automated tracing can potentially perform this task in seconds using a relatively modern desktop computer. While this insight does not consider time spent by a human validating the automated tool's results, this anecdote highlights the potential time-savings that can be realized by automating this otherwise manually intensive task, warranting further research with respect to automating requirements traceability.

## C. RECOMMENDATIONS AND FUTURE WORK

This work represents a contribution to research on automated traceability based on IR techniques. This work compared the performance of histogram distance with the commonly used cosine method and concluded that cosine similarity achieved superior F-scores and better overall performance in terms of recall and precision. Based on this work, the following recommendations and future work are proposed to help advance research on automated requirements tracing.

First, it is recommended that researchers continue to seek out other, potentially more robust similarity measures that yield higher recall and precision than the cosine method. While cosine similarity is commonly used in research and other strategies are being researched to augment its performance, the methodology presented in this study can be used to develop a software tool and objectively compare its performance with other similarity measures.

Second, it is recommended that a multifaceted approach is taken to improve IR-based automated tracing. According to Wang et al. (2018a), "IR-based methods have the most potential to be adopted by industry, as they have been validated from multiple viewpoints." However, IR-based method, if used alone, will likely remain limited, particularly for evolving requirement sets requiring new traces over time. Improvements includes augmenting IR-based methods (such as preprocessing and other strategies), improving human-in-the-loop integration (i.e., human validation of tool results or reinforcement of ML-based algorithms), and integrating other automated techniques with IR, such as ML-based approaches which are growing in popularity amongst researchers. Additionally, requirements writing that takes advantage of the automated technique's fundamentals—such as using common terms in parent and child requirements when employing TF-IDF—could help further improve automated tracing results. With a multifaceted approach, these steps may help improve recall and precision to acceptable levels and foster widespread interest amongst systems engineers to use such tools.

Third, it is recommended that the DOD places more emphasis on researching automated tracing given the time savings and potential error reduction afforded by

automated tracing tools as compared to existing requirements management tools. For program managers, use of automated tools could result in higher quality systems and significant time and cost savings due to reduction in errors that commonly occur in DOD programs particularly as requirements evolve. Larger programs with greater number of requirements stand to benefit even more from automated tracing.

Fourth, it is recommended that automated tracing be used as a tool to inspect requirement links manually established by systems engineers and to not obviate human involvement completely. Prior research identified in the literature review found that automated tracing was often used in software engineering to trace parent and child requirements artifacts with no prior mapping. Given the complexity of natural language processing and the underlying intentions of the original requirement document's author during derivation, systems engineers should also use automated tracing to help maintain requirements changes throughout the system's life cycle and not just at its inception. IR-based approaches can help systems engineers establish traceability of initial requirement sets, followed by ML-based approaches as requirements evolve. Not only do automated tools help identify potential errors made by the systems engineer, but they can also help reduce the amount of time taken to perform the task.

Future work should consider improvements in recall and precision afforded by additional preprocessing of requirement source data, consider the effects of using similarity measures in sequence, consider ways systems engineers can write requirements to leverage IR-based techniques, consider seeking out novel multifaceted methods including ML, and consider improved ways to incorporate user feedback into the requirements tracing process similar to work done by Hayes et al. Additionally, future work should aim to simplify useability of automated tracing to help garner interest in such tools by systems engineers beyond the framework provided in this thesis.

In summary, automated tracing is a fruitful research area that can have enormous benefits to DOD programs from a cost, schedule, and system quality standpoint. Improvements in automated tracing can help systems engineers conduct requirements management in less time and with fewer errors, and ultimately help programs achieve a better product through increased requirements management rigor. Further research is

needed to simplify automated tracing use, yield higher levels of precision and recall, and to ultimately instill enough confidence in systems engineers to supplant time-consuming and error prone conventional requirements tracing methods.

# APPENDIX A. SOURCE DATA

Please note that Table 10 and Table 11 in Appendix A contain the requirements text attributed to McGarry (2016), but in a reformatted manner to facilitate software processing done as part of this thesis.

Table 10. Preprocessed Source Data Containing Parent Requirements

| Parent_ID | Requirement |
|---|---|
| 1 | With a standard clear atmosphere or better, SGSLR stations shall be capable of 24 x 7 tracking of satellites whose arrays satisfy the ILRS retro-reflector guidelines, and whose altitudes are 300 km to 22,000 km. |
| 2 | With a standard clear atmosphere or better, SGSLR stations shall be capable of tracking geosynchronous satellites whose arrays satisfy the ILRS retro-reflector guidelines. |
| 3 | Data precision for LAGEOS NPT shall be < 1.5 mm when averaged over a one month period. |
| 4 | The LAGEOS Normal Point range bias shall be stable to 1.5 mm over 1 hour. |
| 5 | Over one year the RMS of station's LAGEOS NPT range biases shall be < 2mm. |
| 6 | SGSLR Station shall be capable of producing an annual volume of 45,000 LEO, 7,000 LAGEOS and 10,000 GNSS NPTs. |
| 7 | Normal Point time of day shall be accurate to < 100 ns RMS. |
| 8 | Systems shall have a modular design supporting maintenance and upgrades. |
| 9 | Systems shall be capable of local and remote operation by an operator with a path to full automation. |
| 10 | Systems and operations shall satisfy local and NASA safety requirements. |
| 11 | Systems shall be capable of following ILRS procedures and formats and handle ILRS-defined restricted tracking. |
| 12 | SGSLR Stations shall not introduce any unquantified biases into the legacy SLR network. |

Table 11. Preprocessed Source Data Containing Child Requirements

| Child_ID | Parent_1 | Parent_2 | Parent_3 | Parent_4 | Parent_5 | Parent_6 | Parent_7 | Parent_8 | Parent_9 | Parent_10 | Parent_11 | Parent_12 | Description |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | The telescope subsystem shall be designed to transmit from the optical bench subsystem and return receive light from the satellite to the optical bench subsystem |
| 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The telescope shall be capable of operation within -40oC to +50oC and wind speeds up to 18 m/s. |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The telescope shall be capable of survival at temperatures ranging from -50oC to +55oC. |
| 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The telescope optical paths shall be sealed against dust and contamination. |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The telescope shall support the mounting of system support equipment. |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The telescope optical elements shall be designed to meet MIL-SPEC-C-675 sections 3.8.2, 3.8.3, and 3.8.4.1 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The telescope shall be capable of maintaining alignment. |
| 8 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | The rotation caused by the optical Coude path with respect to gimbal angular space shall be defined by a fixed model. |
| 9 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The telescope optical elements shall optimize system performance and lose no more than 2% per surface at wavelengths of 532, 1064 and 1550 nm and not more than 4% (TBR) loss per surface in the broadband from 400 nm to 800 nm |
| 10 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The telescope optical elements shall be designed to allow alignment of the system in the field |
| 11 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The telescope shall be designed to have a 10^-6 reduction in stray light through the optical path even while pointing to within 10 degrees of the sun |
| 12 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The telescope shall be capable of having a FOV of 40 arc sec unvignetted, 60 arc sec with some vignetting for acquisition, and a total FOV of 2 arc minutes for star calibrations. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 13 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The telescope shall be designed to achieve the fundamental performance of the baseline prescription |
| 14 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The Gimbal shall be capable of a maximum slew velocity of at least 20 deg/ s. |
| 15 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The Gimbal shall be designed to follow the angular commands to within 1 arc second RMS while tracking satellites from 300 km altitude to geosync, as well as stars for star calibrations, from 7 to 90 degrees elevation for all azimuth angles. |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The Gimbal shall provide digital status information on critical parameters, like temperature and voltages, for monitoring. |
| 17 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The Gimbal shall be capable of operation over the temperature range of -40oC to +50oC. |
| 18 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The Gimbal shall be capable of survival at the expected extremes of temperature form -50oC to +55oC(TBR). |
| 19 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The Telescope and Gimbal subsystem shall be designed to have a system downtime no greater than 0.75% (TBR), which consists of scheduled maintenance, MTBF and MTTR. |
| 20 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | The tracking subsystem shall be able to support a payload of the telescope and an additional 120 lbs.(TBR) of equipment necessary for operation and diagnostics. |
| 21 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | The location of the intersection of the axes for the gimbal shall be known to within 1mm in 3D space(TBR) and referenced to an external survey point on the gimbal. |
| 22 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The Gimbal interface shall be capable of transmitting position information from the GTA to the software at least a rate of 1 kHz to confirm accurate tracking of the target (satellite) trajectory based on the operational laser fire rate. |
| 23 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The Gimbal shall include an internal error model table that will map repeatable errors in pointing and tracking. |
| 24 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The Gimbal shall be designed to meet absolute open loop pointing requirements of <= 3 arcsec RMS through the use of a GTA mount model based on tracking stars (star calibration). |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 25 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | The Gimbal shall be designed to allow for calibration of the system by pointing at ground targets. |
| 26 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | The Gimbal shall be capable of mechanically mating to the pier through the use of a riser. |
| 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | The Gimbal shall be capable of being reshipped. |
| 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The Gimbal shall be designed to meet all of the NASA, GSFC and local safety standards. |
| 29 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The Tracking subsystem shall be capable of a tracking azimuth velocity of 0°– 10°/sec with an azimuth acceleration of 0°-2°/sec2; a tracking elevation velocity of 0°-2° deg/sec with an elevation acceleration of 0°- 0.5°/sec^2 |
| 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The Tracking Subsystem shall be capable of full control by the SGSLR system software, and provide all relevant data to the system software. |
| 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The Tracking Subsystem shall be designed to support local, remote, and fully automated operations |
| 32 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | The optical bench subsystem shall serve as the optical interface between the laser transmitter, receiver, GTA, and star camera. |
| 33 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The optical bench subsystem shall be designed to have a system downtime no greater than 1.65% (TBR), which consists of scheduled maintenance, MTBF and MTTR. |
| 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The optical bench subsystem shall be capable of full control by the SGSLR system software, and provide all relevant data to the system software. |
| 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The optical bench subsystem shall be capable of being automatically placed into all operational configurations by the software. |
| 36 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The components on the optical bench shall be compatible with all laser output characteristics. |
| 37 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The optical bench subsystem design shall optimize system ranging performance at 532 nm, 1064 nm, and 1550 nm, and star imaging performance from 400 nm to 800 nm. |
| 38 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The optical bench subsystem shall be designed to minimize backscatter across all operational wavelengths. |
| 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | The optical bench subsystem shall be capable of manual configuration for testing, troubleshooting, and special calibrations of the system. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The optical bench subsystem shall be capable of directing the transmit laser beam along a path angularly different from the telescope optical axis for point ahead capability. |
| 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The optical bench subsystem design shall provide space for the required optical attenuators and beam blocks to address laser safety. |
| 42 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The optical bench subsystem shall be isolated from vibrations from the shelter that are greater than the equivalent to Vibration Criterion Curve A (VC-A) 50 micrometers/sec RMS (TBR). |
| 43 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The optical bench subsystem design shall support alignment to within 1 arc seconds (TBR) of the transmit, receive, and star camera optical paths to the telescope optical axis. |
| 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The optical bench subsystem shall be capable of being placed into a “safe mode” in the event of an unexpected power failure. |
| 45 | 1 | 1 | 1 | 1 | 1 | | | 1 | | | | | The optical bench subsystem design shall include sufficient space for the laser head of the laser subsystem. |
| 46 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | The optical bench subsystem design shall include sufficient space for the receiver subsystem. |
| 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The optical bench subsystem shall be designed to support local, remote, and fully automated operations. |
| 48 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | The optical bench subsystem shall be designed and optimized to support the required range measurement precision and stability. |
| 49 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The optical bench subsystem shall capture usable nighttime images of stars down to magnitude 6 (TBR). |
| 50 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The optical bench subsystem shall be designed to support the 2 arcmin FOV for star imaging and 60 arcsec FOV for satellite acquisition. |
| 51 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The optical bench subsystem shall provide stray light protection for the detector and all cameras on the bench. |
| 52 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The optical bench shall support local and remote diagnostic capability. |
| 53 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | The range receiver subsystem shall make all timing measurements relative to the system base frequency with < 5 ps (TBR) precision and < 10 ps (TBR) stability over an hour. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 54 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | The components of the range receiver subsystem shall each have a known error which collectively do not exceed the subsystem's ranging error budget. |
| 55 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | The range receiver subsystem shall be designed to detect photons from the laser pulse in the transmit optical path. |
| 56 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | The range receiver subsystem shall be designed to detect single photons from the receive optical path. |
| 57 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | The range receiver subsystem shall be designed to operate in three modes: internal calibration, external calibration, and satellite ranging. |
| 58 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The range receiver subsystem shall have a system dead time that is less than 10 ns (TBR). |
| 59 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The range receiver subsystem shall be capable of avoiding > 90% (TBR) of collisions between the transmit and receive events. |
| 60 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The range receiver subsystem shall be capable of blanking the detector during laser energy transmission. |
| 61 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The range receiver shall provide timing and spatial information from transmit and receive events needed for closed loop tracking. |
| 62 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The range receiver subsystem shall be able to correctly distinguish and accurately process satellite range returns with (1) background noise rates up to 13 MHz (TBR) with a return signal rates of between 0.05 and 0.2 pes/ fire (TBR), and (2) background noise rates up to 5 MHz (TBR) for return signal rates between 0.001 and 0.05 pes/fire |
| 63 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The range receiver system shall be able to survive and recover from 30 MHz (TBR) background rates. |
| 64 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The range receiver subsystem shall provide gating to the detector for internal and external calibrations, and satellite ranging. |
| 65 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The range receiver shall be capable of full control by the SGSLR system software, and provide all relevant data to the system software. |
| 66 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The range receiver subsystem shall be designed to have a system downtime no greater than 1.8% (TBR), which consists of scheduled maintenance, MTBF and MTTR. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 67 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The range receiver subsystem shall be capable of being placed into a “safe mode” in the event of an unexpected power failure. |
| 68 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The receiver subsystem shall be designed to support local, remote and fully automated operations. |
| 69 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The range receiver subsystem shall provide spatial information to =< 2 arc second accuracy (TBR). |
| 70 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The range receiver subsystem shall provide signal processing information in the form of a spatial histogram across the tracking FOV to the system software at 20 Hz (TBR) rate. |
| 71 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The laser subsystem shall generate optical pulses for ranging with an adjustable power output and repetition rate for a set wavelength, pulse width and enough energy to successfully range to the highest required satellites. |
| 72 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The laser subsystem shall be designed to have a system downtime no greater than 2.85% , which consists of scheduled maintenance, MTBF and MTTR. |
| 73 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The laser subsystem shall be capable of full control by the SGSLR system software, and provide all relevant data to the system software. |
| 74 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The laser output shall be stable over defined periods of time. |
| 75 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The laser subsystem shall be capable of firing continuously for one month between scheduled maintenance periods. |
| 76 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The laser subsystem shall be able to resume firing at nominal output parameters after defined periods of interruption. |
| 77 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The laser subsystem shall be capable of being placed into a “safe mode” in the event of an unexpected power failure. |
| 78 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The laser subsystem shall support local, remote, and fully automated operations. |
| 79 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | The laser head of the laser subsystem shall be mountable on the optical bench subsystem. |
| 80 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The laser subsystem shall support external firing using input from the RCE. |
| 81 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The laser safety subsystem shall ensure that no one is exposed to non eye safe laser light during normal operations. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 82 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The laser safety subsystem shall ensure that no one is exposed to non eye safe laser light outside of the laser operations area and dome during non operational periods. |
| 83 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The laser safety subsystem shall be designed to have a system downtime no greater than 1.5% (TBR), which consists of scheduled maintenance, MTBF and MTTR. |
| 84 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | Outdoor components of the laser safety subsystem shall be capable of operation within -40 deg C and +50 deg C. |
| 85 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | Outdoor components of the laser safety subsystem shall be capable of survival at -50 deg C and +55 deg C. |
| 86 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The laser safety subsystem shall be capable of automated and manual reset. |
| 87 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The laser safety subsystem shall be capable of detecting aircraft and ensuring that aircraft are not exposed to laser light. |
| 88 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The laser safety subsystem shall not allow transmission of non eye safe laser radiation below the minimum tracking elevation angle 10 deg (TBR) |
| 89 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The laser safety subsystem shall allow for full power operation, 5 Watts (TBR), of the laser for alignment purposes without exposing persons outside of the laser operations area (nominal hazard zone). |
| 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The laser safety subsystem shall provide status and configuration information to the system software. |
| 91 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The laser safety subsystem shall be capable of commanding by the system software without allowing the software to override safety settings. |
| 92 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The laser safety subsystem shall be designed to support local, remote and fully automated operations. |
| 93 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | Applicable laser safety subsystem components shall be designed to occupy a minimal footprint on the optical bench. |
| 94 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The laser safety subsystem shall be capable of being placed into a “safe mode” in the event of an unexpected power failure. |
| 95 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The laser safety subsystem shall default to a fail safe mode in case of a subsystem failure. |
| 96 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | Laser safety system shall be fully compliant with NASA safety standards. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 97 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | The time and frequency subsystem shall provide stable and accurate date/ time and frequency signals relative to GPS. |
| 98 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The time and frequency subsystem shall be designed to have a system downtime no greater than 0.3% , which consists of scheduled maintenance, MTBF and MTTR. |
| 99 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The outdoor components of the time and frequency subsystem shall be capable of operation from -40 deg C to +50 deg C. |
| 100 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The outdoor components of the time and frequency subsystem shall be capable of survival from -50 deg C to +55 deg C. |
| 101 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | The time and frequency subsystem shall provide the timing signals (analog and digital) required by the SGSLR subsystems. |
| 102 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The timing and frequency subsystem shall be capable of full control by the computer and software subsystem, and shall provide all relevant data to the computer and software system. |
| 103 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | The time and frequency subsystem shall be able to accept external time and frequency sources. |
| 104 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | The time and frequency subsystem shall be capable of self- monitoring its frequency and timing pulses by comparison to an included independent GPS source. |
| 105 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The time and frequency subsystem shall be designed to support local, remote and fully automated operations. |
| 106 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The time and frequency subsystem shall be capable of being placed into a “safe mode” in the event of an unexpected power failure. |
| 107 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | The time and frequency subsystem shall meet its required performance within a 24 hour period of time from power on. |
| 108 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | The meteorological subsystem shall measure wind speed and direction (average and gust), atmospheric pressure, temperature, relative humidity, visibility, precipitation, and cloud cover. |
| 109 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The Meteorological subsystem shall be designed to have a system downtime no greater than 0.6%, which consists of scheduled maintenance, MTBF and MTTR. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 110 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | The meteorological subsystem shall be capable of producing consistently accurate data within the operational range of -40 deg C to +50 deg C. |
| 111 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The meteorological subsystem shall be capable of survival between -50 deg C and +55 deg C. |
| 112 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The meteorological subsystem shall provide environmental data when requested by the system software within a 60 s response for all except precipitation and wind which require a 10 s response time. |
| 113 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The meteorological subsystem shall be capable of full control by the SGSLR system software, and provide all relevant data to the system software. |
| 114 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The meteorological subsystem shall be designed to support local, remote, and fully automated operations. |
| 115 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The meteorological subsystem shall be capable of being placed into a “safe mode” in the event of an unexpected power failure. |
| 116 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | The dome and shutter shall protect the GTA and associated components from the elements and support system performance. |
| 117 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The dome subsystem shall be designed to have a system downtime no greater than 2.1% (TBR), which consists of scheduled maintenance, MTBF and MTTR. |
| 118 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The dome subsystem shall be able to support operations in temperature between -40 deg C and +50 deg C, and wind speed of up to 18 m/s. |
| 119 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The dome subsystem shall be capable of survival at -50 deg C to +55 deg C and wind speeds up to 60 m/s. |
| 120 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The dome subsystem shall be designed to mitigate the effects of condensation. |
| 121 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | The dome subsystem shall keep precipitation from entering the interior of the dome with the shutter fully closed. |
| 122 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | Environmental conditions inside the dome shall be available for monitoring by software. |
| 123 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The dome and shutter shall be capable of closed loop position control by the computer subsystem to an accuracy of 5 degrees (TBR). |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 124 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The dome subsystem shall be capable of full control by the SGSLR system software, and provide all relevant data to the system software. |
| 125 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The dome shall be capable of being slaved to the Telescope and Gimbal subsystem to an accuracy of 5 degrees (TBR). |
| 126 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The dome and shutter shall allow the telescope and auxiliary equipment an unobstructed view of satellites and ground targets. |
| 127 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | The dome subsystem shall allow full telescope Az and El rotation with the shutter open or closed. |
| 128 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | The dome subsystem shall allow for manual control of the dome and shutter motors without the use of the system operational software. |
| 129 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | The dome shall be able to be rotated and the shutter opened/closed without the use of motors. |
| 130 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The dome subsystem shall be capable of being placed into a “safe mode” in the event of an unexpected power failure. |
| 131 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The dome shall be capable of a minimum 20 deg/s (TBR) angular velocity. |
| 132 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The dome subsystem shall be designed to support local, remote, and fully automated operations. |
| 133 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | The shelter shall be of suitable size and construction to support SLR operations, house system components, provide workspace for repairs, maintenance, and logistics, and provide a buffer from the environment. |
| 134 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The shelter subsystem shall be designed to have a system downtime no greater than 2.1% (TBR), which consists of scheduled maintenance, MTBF and MTTR. |
| 135 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The shelter subsystem shall be capable of operation between -40 deg C and +50 deg C and wind speeds up to 18 m/s. |
| 136 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | The shelter subsystem shall be capable of survival and protection of the system equipment from -50 deg C to +55 deg C and in wind conditions of up to 60 m/s. |
| 137 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | Environmental conditions inside the shelter shall be capable of being monitored and controlled by software. |
| 138 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The lights, cameras, communications, and power of the shelter shall be capable of being monitored and controlled by software. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 139 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The shelter roof shall be capable of supporting the dome dead load of 3000 pounds; a uniform live load of 50 pounds per square feet (psf) (TBR) or a concentrated live load of 1000 pounds over 12 inch x 12 inch area (TBR) |
| 140 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The shelter shall be partitioned to allow for separate environmental conditions and safety considerations. |
| 141 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | The shelter shall be sealed to prevent contamination to the shelter interior. |
| 142 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | The shelter floor and roof shall provide openings for the telescope and telescope support structure. |
| 143 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | The shelter shall be physically isolated from the telescope and telescope support structure to provide an isolation efficiency of 95% (TBR). |
| 144 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The shelter subsystem shall have the capability to monitor security and emergency conditions and report to the software subsystem. |
| 145 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The shelter power shall be sufficient to supply the 30 kilowatts total for 3 phases needed to operate the system with margin. |
| 146 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The shelter subsystem shall provide a suitable ground for safety and system performance. |
| 147 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The shelter subsystem shall provide conditioned power to operational equipment. |
| 148 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The shelter subsystem shall mitigate lightning damage. |
| 149 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The power provided to the shelter subsystem shall be capable of monitoring by the SGSLR system software. |
| 150 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The power provided by the shelter subsystem shall be capable of monitoring and control by the SGSLR system software. |
| 151 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The shelter subsystem shall provide fiber optic cable interface for all equipment and data connections external to the shelter. |
| 152 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | The pier and riser shall be constructed to rigidly support the GTA while maintaining the optimum performance of the GTA. Fundamental frequency is greater than or equal to 80 Hz (TBR). |
| 153 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | The pier and riser shall be constructed to support the Coude path to the optical bench. |
| 154 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | The pier shall have > 95% (TBR) vibration isolation efficiency from the shelter and its pad. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 155 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | The riser shall be designed to include a leveling mechanism for the GTA with a +/- 2 arc seconds accuracy. |
| 156 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The computer and software subsystem shall be designed to support (a) local operations, (b) remote operations, and (c) fully automatic operations with no human present on site but with remote monitoring of the system. |
| 157 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | Remote Access Terminal (RAT) shall include a user interface to allow both local and remote access to the data generated by the system. |
| 158 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | RAT shall maintain a display of critical subsystem and operational parameters accessible to both local and remote users. |
| 159 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | Remote Access Terminal (RAT) client and server software shall be able to manage at least two (TBR) internet connections. |
| 160 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | Remote Access Terminal (RAT) software shall allow remote and local control of operations and of operating parameters such that there is no effect on the health and safety of the system or surrounding environment (e.g., aircraft avoidance). |
| 161 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | Any computers requiring deterministic timing shall have a real-time operating system with a known latency that is < 100 microseconds. |
| 162 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The computers selected shall have backplanes that can support the types of interfaces (serial, USB, Ethernet, parallel) and number of cards required for the SGSLR subsystems. |
| 163 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The computer subsystem shall have > 4 GB (TBR) memory, > 2 GHz (TBR) CPU, and > 500 GB (TBR) drive space capacity. |
| 164 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | The computer subsystem shall be designed to keep down time to less than 2.55% (TBR) on average over a year. |
| 165 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of running successfully in an independent operational state for > 7 days (TBR) . |
| 166 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | The computers shall have the capability of sharing data. |
| 167 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of processing satellite, ground calibration, and star calibration data. |
| 168 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | The software shall be capable of handling the ILRS predictions for tracking satellites. |
| 169 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | The software shall follow the ILRS procedure for handling leap seconds. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 170 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | The software shall be capable of generating science data in the ILRS formats. |
| 171 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | The software shall comply with all ILRS restricted tracking requirements. |
| 172 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | The software shall follow ILRS guidelines for normal point formation. |
| 173 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | The software shall be capable of transmitting the SLR data products in the timeframe specified by the ILRS. |
| 174 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of handling error conditions, logging them locally and making them accessible to remote user(s). |
| 175 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The computer & software subsystem shall interact with all of the hardware subsystems. |
| 176 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The time critical software tasks shall run in a real-time environment, and finish in specified time intervals associated with each task. |
| 177 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The computer hardware shall be capable of generating timing interrupts and the software shall be capable of handling the interrupts. |
| 178 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | The software shall be capable of handling the time tagging of the data to provide the Normal Point range measurement time with < 0.1 microsecond (TBR) time tags. |
| 179 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | The software shall be capable of combining the data acquired from the hardware to construct the range measurement in a manner which provides the Normal Point range precision and stability required for LAGEOS ranges. |
| 180 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall record data, activities, events and report status and error information to the IGSOC on defined intervals. |
| 181 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of sending alerts and warnings immediately to the IGSOC and selected personnel. |
| 182 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | The computers and software shall adhere to NASA's IT Security policy (NPR 2810.1a Security of Information Technology). |
| 183 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of receiving star and sky image data from the camera systems and transferring them to the IGSOC. |
| 184 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of processing star images to generate star centroids for star calibrations, and sky images to generate cloud cover for sky clarity determination. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 185 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The computers shall be capable of receiving and monitoring status information from the subsystems. |
| 186 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of determining each subsystem's status and determining the overall system status. |
| 187 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | The software shall have an automatic backup procedure for system and data recovery. |
| 188 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | The software shall be capable of command and control of the system 24/7, to perform satellite tracking and ranging, for a majority of the ILRS list of active satellites, when conditions permit. |
| 189 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | The software shall be capable of performing ground calibrations 24/7 when conditions permit. |
| 190 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of performing star calibration 24/7 when conditions permit. |
| 191 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of performing star assessments 24/7 when conditions permit. |
| 192 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be able to control operations by making decisions based on weather, sky clarity, pointing bias, and system and subsystem monitoring. |
| 193 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | The software shall be capable of automatically following the daily SGSLR schedule, including satellite passes, scheduled ground calibration, scheduled star calibration, and routine maintenance. |
| 194 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of making real-time changes to the tracking schedule based on system information, real-time information from the VLBI antenna and external input from the IGSOC. |
| 195 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | The computer and software subsystem shall allow for diagnostic control and simulation both locally and remotely. |
| 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | The software shall not be able to override the laser safety subsystem. |
| 197 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The software shall check that all conditions are nominal before requesting the laser to fire. |
| 198 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The software shall operate in a manner to protect any nearby VLBI antenna from the SLR radar. |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 199 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | The software shall be capable of making decisions and taking action to protect the system. |
| 200 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | The software shall be capable of restricting lasing in certain exclusion areas at, or near, the ground, through the use of a mask. |
| 201 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | The software shall be capable of protecting the system from erroneous user input. |
| 202 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of accepting a schedule generated by the IGSOC or generating a satellite prioritized schedule onsite. |
| 203 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of site specific and target specific configurations. |
| 204 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | The software shall be capable of setting system configurations for optimum return rates, including blanking, PRF, ND wheels, time & pointing bias, beam divergence and TBD. |

# APPENDIX B. SOURCE CODE

```
'''
Created on January 15, 2021
@author: Christopher D. Laliberte
'''

import math

from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import pandas as pd
import csv

number_of_parent_requirements = 0
number_of_child_requirements = 0
number_of_total_requirements = 0

#This function performs parent-to-child traceability (i.e., parent-
child pairs that meet similarity measure's threshold)
def print_tdidf_matrix(arr):
    # creates a range corresponding to the indexes in our list 0 to
len(arr) -1
    array = arr.toarray()
    print()
    print("The TF-IDF matrix is:")
    print(vectorizer.get_feature_names())
    for i in range(len(array)):
        for j in range(len(array [i])):
            if j==0:
                print(i, ': ',array [i][j], end=' ')
            else:
                print(array [i][j], end=' ')
        print()
    return

#Cosine similarity. The higher the value (up to a max of 1) the better.
def cos_similarity(arr):
    array = arr.toarray()
    print()
    #print(len(array))

    # number_of_parent_requirements = 2 #12
    # number_of_total_requirements = 3 #14 #len(array)-
number_of_parent_requirements

    scores = np.zeros((number_of_parent_requirements,
                       number_of_total_requirements))  # Make a score
array that is # of parent reqs by # of child reqs

    summation_numerator = 0
    summation_denominator1 = 0
```

```
    summation_denominator2 = 0

    #Numerator summation for wi * qi, i.e., summed_numerator
    for i in range(number_of_parent_requirements):  # 0 through
number_of_parent_requirements - 1 per https://www.w3schools.com/python/
python_for_loops.asp
        for j in range(number_of_parent_requirements, len(array)):  #
from start of child requirements through the end of the TF-IDF matrix
            #print(i)
            #print(j)
            for k in range(len(array [i])):  # for each element in
either vector...
                # print((array [i][k])) #for debugging purposes --
parent array element
                # print((array [j][k])) #for debugging purposes --
child array element (query or q)
                summation_numerator = summation_numerator + (array
[i][k] * array [j][k])
                summation_denominator1 = summation_denominator1 +
(array [i][k] * array [i][k])
                summation_denominator2 = summation_denominator2 +
(array [j][k] * array [j][k])
                # print(k," ",summation) #for debugging purposes --
shows the summation for each element j in the vector pair
            if math.sqrt(summation_denominator1*summation_denominator2)
!= 0:  # checks if denominator is zero; if not, it performs the
calculation. If yes, set score for pair to zero.
                scores [i, j] = summation_numerator /
math.sqrt(summation_denominator1 * summation_denominator2)  # pair
score for pair i,j
            else:
                scores [i, j] = 0

            summation_numerator = 0
            summation_denominator1 = 0
            summation_denominator2 = 0

            #print("Cosine similarity score for parent ," i+1, " and
child ," j-number_of_parent_requirements+1, " is: ," scores [i, j])

    return scores

def histogram_similarity(arr): #The lower the value (i.e., distance)
the better
    array = arr.toarray()
    scores = np.zeros((number_of_parent_requirements,
number_of_total_requirements))  # Make a score array that is # of
parent reqs by # of child reqs
    summation = 0

    for i in range(number_of_parent_requirements):  #0 through
number_of_parent_requirements - 1 per https://www.w3schools.com/python/
python_for_loops.asp
        for j in range(number_of_parent_requirements,len(array)): #from
start of child requirements through the end of the TF-IDF matrix
```

```
            #print(i)
            #print(j)
            for k in range(len(array [i])): # for each element in
either vector...
                #print((array [i][k])) #for debugging purposes --
parent array element
                #print((array [j][k])) #for debugging purposes -- child
array element
                if (array [i][k] + array [j][k]) != 0: #checks if
denominator is zero; if not, it performs the calculation. If yes, it
moves to the next increment.
                    summation = summation + ((array [i][k] - array
[j][k])*(array [i][k] - array [j][k]))/(array [i][k] + array [j][k])
                else:
                    summation = summation + 0
                #print(k," ",summation) #for debugging purposes --
shows the summation for each element j in the vector pair
            scores [i, j] = 0.5 * summation #pair score for pair i,j

            #print("Histogram distance score for parent ," i+1, " and
child ," j-number_of_parent_requirements+1, " is: ",scores [i,j])
            #print()
            summation = 0

    #print(scores)

    return scores

def perform_histogram_tracing(scoresArray, threshold): #This function
is the same as perform_cosine_tracing, except the threshold is <=
    predicted_traces_binary = np.zeros((number_of_parent_requirements,
number_of_total_requirements))
    predicted_parent_to_child_traces_binary =
np.zeros((number_of_parent_requirements, number_of_child_requirements))
# This is similar to the above array except this one resets the size of
the array to size parent,child instead of parent,total
    predicted_child_to_parent_traces_binary =
np.zeros((number_of_child_requirements, number_of_parent_requirements))

    count = 0
    parentCount = 0
    childCount = 0
    print()
    # print("Length of the Scores array is: ",len(scoresArray))

    print("Histogram distance threshold is: ," threshold)

    for i in range(
            number_of_parent_requirements):  # 0 through
number_of_parent_requirements - 1 per https://www.w3schools.com/python/
python_for_loops.asp
        for j in range(len(scoresArray [i])):  # from start of child
requirements through the end of the TF-IDF matrix
            # print(scoresArray [i][j])
            if (scoresArray [i][j] <= threshold) and (scoresArray
```

```
[i][j] != 0):
                #predicted_traces_binary [i][j] = 1
                predicted_parent_to_child_traces_binary [parentCount][j
- number_of_parent_requirements] = 1
        parentCount = parentCount + 1
        childCount = 0

    # This prints out the predicted_parent_to_child_traces_binary array
    '''
    for i in range(number_of_parent_requirements):
        print("Parent index ," i , " (or true parent #",i+1,") traces
to children represented by elements with 1: ")
        for j in range(number_of_child_requirements):
            print(int(predicted_parent_to_child_traces_binary [i][j]))
    '''

    # This converts the predicted_parent_to_child_traces_binary Array
to predicted_child_to_parent_traces Array
    # For each parent, search child k
    childCount = 0
    for i in range(number_of_parent_requirements):
        for j in range(number_of_child_requirements):
            if (predicted_parent_to_child_traces_binary [i][j] == 1):
# if there is a parent to child trace...
                predicted_child_to_parent_traces_binary [j][i] = 1

    # This prints out the predicted_child_to_parent_traces_binary array
    '''
    for i in range(number_of_child_requirements):
        print("Child index ," i , " (Child_ID: ",i+1,") traces to
parent represented by elements with 1: ")
        for j in range(number_of_parent_requirements):
            print(int(predicted_child_to_parent_traces_binary [i][j]))
    '''

    for i in range(
            number_of_parent_requirements):  # 0 through
number_of_parent_requirements - 1 per https://www.w3schools.com/python/
python_for_loops.asp
        print()
        print("Parent ," i+1, "traces to child: ",
              end=" ")  # +1 to "show" the first parent as 1 instead of
0. Note that it starts at 0 in array.
        for j in range(len(scoresArray [i])):  # from start of child
requirements through the end of the TF-IDF matrix
            if predicted_traces_binary [i][j] == 1:
                # print("Parent requirement ",i,"traces to child
requirement ",j-number_of_parent_requirements, " with a score of
",scoresArray [i][j])
                print(j - number_of_parent_requirements+1, end=" ")  #
prints number of child
                # if there is a trace, update the predicted_traces
array
                count = count + 1
                #predicted_traces_binary [i][count] = j -
```

```
number_of_parent_requirements  # this is an array that
                # print()
                # print("Parent ," i, ," Child ," j)

    print()
    print("There are ," count, " traces")

    return predicted_child_to_parent_traces_binary
#predicted_parent_to_child_traces_binary

def perform_cosine_tracing(scoresArray, threshold): #This function is
the same as perform_histogram_tracing, except the threshold is >=
    predicted_traces_binary = np.zeros((number_of_parent_requirements,
number_of_total_requirements))
    predicted_parent_to_child_traces_binary =
np.zeros((number_of_parent_requirements, number_of_child_requirements))
#This is similar to the above array except this one resets the size of
the array to size parent,child instead of parent,total
    predicted_child_to_parent_traces_binary =
np.zeros((number_of_child_requirements,
number_of_parent_requirements,))

    count = 0
    parentCount = 0
    childCount = 0
    print()
    #print("Length of the Scores array is: ",len(scoresArray))

    print("Cosine threshold is: ",threshold)

    for i in range(number_of_parent_requirements):  # 0 through
number_of_parent_requirements - 1 per https://www.w3schools.com/python/
python_for_loops.asp
        for j in range(len(scoresArray [i])):  # from start of child
requirements through
            #print(scoresArray [i][j])
            if (scoresArray [i][j] >= threshold) and (scoresArray
[i][j] != 0):
                #predicted_traces_binary [i][j] = 1
                predicted_parent_to_child_traces_binary [parentCount][j
- number_of_parent_requirements] = 1
        parentCount = parentCount + 1
        childCount = 0

    #This prints out the predicted_parent_to_child_traces_binary array
    '''
    for i in range(number_of_parent_requirements):
        print("Parent index ," i , " (Child_ID: ",i+1,") traces to
children represented by elements with 1: ")
        for j in range(number_of_child_requirements):
            print(int(predicted_parent_to_child_traces_binary [i][j]))
    '''

    #This converts the predicted_parent_to_child_traces_binary Array to
predicted_child_to_parent_traces Array
```

```
    #For each parent, search child k
    childCount=0
    for i in range(number_of_parent_requirements):
        for j in range(number_of_child_requirements):
            if(predicted_parent_to_child_traces_binary [i][j]==1): #if
there is a parent to child trace...
                predicted_child_to_parent_traces_binary [j][i] = 1

    # This prints out the predicted_child_to_parent_traces_binary array
    '''
    for i in range(number_of_child_requirements):
        print("Child index ," i , " (or true child #",i+1,") traces to
parent represented by elements with 1: ")
        for j in range(number_of_parent_requirements):
            print(int(predicted_child_to_parent_traces_binary [i][j]))
    '''

    for i in range(number_of_parent_requirements):  # 0 through
number_of_parent_requirements - 1 per https://www.w3schools.com/python/
python_for_loops.asp
        print()
        print("Parent ," i+1, "traces to child: ",end =" ") #+1 to
"show" the first parent as 1 instead of 0. Note that it starts at 0 in
array.
        for j in range(len(scoresArray [i])):  # from start of child
requirements through the end of the TF-IDF matrix
            if predicted_traces_binary [i][j] == 1:
                #print("Parent requirement ",i,"traces to child
requirement ",j-number_of_parent_requirements, " with a score of
",scoresArray [i][j])
                print(j-number_of_parent_requirements+1, end =" ")
#prints number of child
                #if there is a trace, update the predicted_traces array
                count = count + 1
                #predicted_traces_binary [i][count] = j-
number_of_parent_requirements #Removed this on 5.15.2021 as I didn't
see the need for it any more with
predicted_child_to_parent_traces_binary
                #print()
                #print("Parent ," i, ," Child ," j)

    print()
    print("There are ," count, " traces")

    return predicted_child_to_parent_traces_binary
#predicted_parent_to_child_traces_binary

def parse_parent_CSV():
    df = pd.read_csv(r'C:\Users\Chris\Desktop\Parent_CSV.csv')
    global number_of_parent_requirements
    number_of_parent_requirements= len(df.values)  # this will get the
total number of parent requirements
    parent_corpus = []

    for i in range(len(df.values)):
```

```
        parent_corpus.append(df.values [i][1])
        #parent_corpus [i] = df.values [i][1]

    #print(parent_corpus)

    return parent_corpus

def parse_child_CSV():
    df = pd.read_csv(r'C:\Users\Chris\Desktop\Child_CSV.csv')
    global number_of_child_requirements
    number_of_child_requirements = len(df.values)  # this will get the
total number of parent requirements
    child_corpus = []

    for i in range(len(df.values)):
        child_corpus.append(df.values
[i][number_of_parent_requirements+1])
        #child_corpus [i] = df.values [i][1]

    #print(child_corpus)

    return child_corpus

def build_actual_traces():
    #Scan CSV, parse out actual traces, and put into actual_traces
[i,j], where i is the parent and j is the sequence of children in
integer form

    df = pd.read_csv(r'C:\Users\Chris\Desktop\Child_CSV.csv')
    number_of_child_requirements = len(df.values)  # this will get the
total number of parent requirements

    #actual_child_to_parent_traces_binary = []
    actual_child_to_parent_traces_binary =
np.zeros((number_of_child_requirements, number_of_parent_requirements))

    #With CSV parsed, now build actual_child_to_parent_traces_binary
array
    for i in range(len(df.values)):
        for j in range(1, number_of_parent_requirements+1): #goes from
1 to #ofparentrequirements-1 :
            #print(" i,j=",i,",",j, ":",df.values [i][j]," ," end="")
            #actual_child_to_parent_traces_binary.append(df.values
[i][j])
            actual_child_to_parent_traces_binary [i][j-1] = df.values
[i][j]


    #print(actual_child_to_parent_traces_binary [2]) #LEFT OFF HERE!
    #print(actual_child_to_parent_traces_binary [3])

    return actual_child_to_parent_traces_binary


def recall(predicted_traces, actual_traces):
```

```
    recall_score = 0
    true_positives = 0 # TruePositives --> For each i (parent
requirement) in predicted_traces [i], count the number of actual ID
matches against actual_traces [i]
    false_positives = 0  # FalsePositives --> For each i (parent
requirement) in predicted_traces [i], count the number of predicted IDs
that are NOT in the actual_traces [i]
    false_negatives = 0 # FalseNegatives --> For each i (parent
requirement) in predicted_traces [i], count the number of missing IDs
that are in actual_traces [i]

    #Recall = TruePositives / (TruePositives + FalseNegatives)

    # NOTE: in predicted_traces or actual_traces --> i is the child, j
is the parent element. 1 indicates a trace.

    #print("***predicted_traces length is: ," len(predicted_traces))
    #print("predicted_traces...: ",predicted_traces)
    #print("***actual_traces length is: ," len(actual_traces))
    #print("actual_traces...: ",actual_traces)

    for i in range(number_of_child_requirements):
        for j in range(number_of_parent_requirements):
            if predicted_traces [i][j] == 1 and actual_traces [i][j] ==
1: #count the number of actual ID matches against actual_traces [i]
                true_positives = true_positives + 1
            elif predicted_traces [i][j] == 0 and actual_traces [i][j]
== 1:  #count the number of missing predictions that ARE in
actual_traces [i]
                false_negatives = false_negatives + 1
    print()
    print("false_negatives: ," false_negatives)

    if true_positives + false_negatives == 0:
        recall_score = 0
    else:
        recall_score = true_positives / (true_positives +
false_negatives)

    return recall_score

def precision(predicted_traces, actual_traces):
    precision_score = 0
    true_positives = 0  # TruePositives --> For each i (parent
requirement) in predicted_traces [i], count the number of actual ID
matches against actual_traces [i]
    false_positives = 0  # FalsePositives --> For each i (parent
requirement) in predicted_traces [i], count the number of predicted IDs
that are NOT in the actual_traces [i]
    #Precision = TruePositives / (TruePositives + FalsePositives)

    # NOTE: in predicted_traces or actual_traces --> i is the child, j
is the parent element. 1 indicates a trace.

    for i in range(number_of_child_requirements):
```

```
        for j in range(number_of_parent_requirements):
            if predicted_traces [i][j] == 1 and actual_traces [i][j] ==
1:  # count the number of actual ID matches against actual_traces [i]
                true_positives = true_positives + 1
            elif predicted_traces [i][j] == 1 and actual_traces [i][j]
== 0: # count the number of predicted IDs that are NOT in the
actual_traces [i]
                false_positives = false_positives + 1
    print()
    print("true_positives: ," true_positives)
    print("false_positives: ," false_positives)
    print()

    if true_positives + false_positives == 0:
        precision_score = 0
    else:
        precision_score = true_positives / (true_positives +
false_positives)

    return precision_score

#MAIN

parent_corpus = parse_parent_CSV()
child_corpus = parse_child_CSV()
number_of_total_requirements = number_of_parent_requirements +
number_of_child_requirements
corpus = parent_corpus + child_corpus

print()
print()
print("corpus = ," corpus)
print()
print()

print("number_of_parent_requirements is: ,"
number_of_parent_requirements)
print("number_of_child_requirements is: ,"
number_of_child_requirements)
print("number_of_total_requirements is: ,"
number_of_total_requirements)

my_stop_words = text.ENGLISH_STOP_WORDS.union(["shall"])
vectorizer = TfidfVectorizer(ngram_range=(1,1),
stop_words=my_stop_words)
X = vectorizer.fit_transform(corpus)
#idf_values = dict(zip(vectorizer.get_feature_names(),
vectorizer.idf_))

# printing the tfidf vectors
#print(X)

# printing the vocabulary of TF-IDF matrix
#print(vectorizer.vocabulary_)
```

```
print_tdidf_matrix(X)

histogram_scores = histogram_similarity(X)
predicted_histogram_traces =
perform_histogram_tracing(histogram_scores, 2)

cosine_scores = cos_similarity(X) #perform_cosine_tracing() returns
predicted_parent_to_child_traces_binary
predicted_cosine_traces = perform_cosine_tracing(cosine_scores, 0.006)
#perform_histogram_tracing() returns
predicted_parent_to_child_traces_binary

#Now that we have traces predicted, let's get true traces (based on
author)
actual_traces = build_actual_traces()

#Given predicted and actual, let's perform scoring
print()
print("Histogram precision and recall:")
histogram_recall = recall(predicted_histogram_traces, actual_traces)
histogram_precision = precision(predicted_histogram_traces,
actual_traces)
histogram_fscore =
2*((histogram_precision*histogram_recall)/(histogram_precision+histogra
m_recall))

print("Cosine precision and recall:")
cosine_recall = recall(predicted_cosine_traces, actual_traces)
cosine_precision = precision(predicted_cosine_traces, actual_traces)
cosine_fscore =
2*((cosine_precision*cosine_recall)/(cosine_precision+cosine_recall))

print()
print("Histogram recall is: ",histogram_recall," precision is: ,"
histogram_precision, " and F-score is: ," histogram_fscore)
print()
print("Cosine recall is: ",cosine_recall," and precision is: ,"
cosine_precision, " and F-score is: ," cosine_fscore)
```

# LIST OF REFERENCES


Acqnotes. 2018. “Requirements Management.” June 1, 2018. https://acqnotes.com/acqnote/careerfields/requirements-management.

Antoniol, Giuliano, Jane Cleland-Huang, Jane Huffman Hayes, and Michael Vierhauser. 2017. “Grand Challenges of Traceability: The Next Ten Years.” Cornell University. https://arxiv.org/abs/1710.03129.

Arkley, P., and S. Riddle. 2005. “Overcoming the Traceability Benefit Problem.” In 13th *IEEE International Conference on Requirements Engineering* (RE’05), 385–89. IEEE. https://doi.org/10.1109/RE.2005.49.

Bellet, Aurélien, Amaury Habrard, and Marc Sebban. 2015. *Metric Learning*. San Rafael, California: Morgan and Claypool Publishers. https://doi.org/10.2200/S00626ED1V01Y201501AIM030.

Campbell, Andrew, Elena Navarro, Emmanuel Adam, Fernando de la Prieta, María Jiménez-López, María Escalona, María Moreno, Philippe Mathieu, Rafael Corchuelo, Silvia Rossi, Zita Vale. 2016. *Trends in Practical Applications of Scalable Multi-Agent Systems, the PAAMS Collection*. Germany: Springer International Publishing, 2016.

Cha, Sung-Hyuk. 2008. “Taxonomy of Nominal Type Histogram Distance Measures.” In *American Conference On Applied Mathematics.* 325–326. Cambridge, MA: Harvard University Press.

Chen, Lei, Dandan Wang, Junjie Wang, and Qing Wang. 2019. “Enhancing Unsupervised Requirements Traceability with Sequential Semantics.” In 2019 26th Asia-Pacific Software Engineering Conference (APSEC), 23–30. IEEE. https://doi.org/10.1109/APSEC48747.2019.00013.

Cleland-Huang, J, R Settimi, E Romanova, B Berenbach, and S Clark. 2007. “Best Practices for Automated Traceability.” *Computer* 40 (6): 27–35. https://doi.org/10.1109/MC.2007.195.

Cuddeback, D, A Dekhtyar, and J Hayes. 2010. “Automated Requirements Traceability: The Study of Human Analysts.” In 2010 18th IEEE International Requirements Engineering Conference, 231–40. IEEE. https://doi.org/10.1109/RE.2010.35.

Defense Acquisition University. “Defense Acquisition Guidebook.” February 19, 2010. https://www.dau.edu/tools/dag.

Fairley, Dick, Kevin Forsberg, and Ray Madachy. Vee. n.d. “System Life Cycle Process Models: Vee.” Accessed January 31, 2021. https://www.sebokwiki.org/wiki/System_Life_Cycle_Process_Models:_Vee.

Gervasi, Vincenzo, and Didar Zowghi. 2014. “Supporting Traceability through Affinity Mining.” In 2014 IEEE 22nd International Requirements Engineering Conference (RE), 143–52. IEEE. https://doi.org/10.1109/RE.2014.6912256.

Gotel, O.C.Z, and C.W. Finkelstein. 1994. “An Analysis of the Requirements Traceability Problem.” In *Proceedings of IEEE International Conference on Requirements Engineering*, 94–101. IEEE Comput. Soc. Press. https://doi.org/10.1109/ICRE.1994.292398.

Hart, Laura. 2015. “Introduction To Model-Based System Engineering (MBSE) and SysML.” Presentation at Delaware Valley INCOSE Chapter Meeting. Lockheed Martin Corporation. July 30, 2015. https://www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf.

Hayes, J.H, A Dekhtyar, and J Osborne. 2003. “Improving Requirements Tracing via Information Retrieval.” In *Proceedings. 11th IEEE International Requirements Engineering Conference,* 2003, 138–47. IEEE. https://doi.org/10.1109/ICRE.2003.1232745.

Hayes, J.H, A Dekhtyar, and S.K Sundaram. 2006. “Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods.” *IEEE Transactions on Software Engineering* 32 (1): 4–19. https://doi.org/10.1109/TSE.2006.3.

Huang, Anna. 2008. “Similarity Measures for Text Document Clustering.” In *NZCSRSC 2008, Christchurch, New Zealand*. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.4480&rep=rep1&type=pdf.

Huang, Jane, Orlena Gotel, and Andrea Zisman. 2012. *Software and Systems Traceability*. London: Springer London. https://doi.org/10.1007/978-1-4471-2239-5.

International Business Machines. 2020. “DOORS option to move objects is disabled.” 1 May 2020. https://www.ibm.com/support/pages/doors-option-move-objects-disabled.

Koehrsen, Will. 2018. Beyond Accuracy: Precision and Recall. TowardsDataScience.com. March 3, 2018. https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c.

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. 2008. Introduction to Information Retrieval. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511809071.

McGarry, Jan. 2016. *Space Geodesy Satellite Laser Ranging System Requirements Document Public Version*. Greenbelt, MD: NASA. https://ntrs.nasa.gov/api/citations/20160011972/downloads/20160011972.pdf.

Mills, Chris, Javier Escobar-Avila and Sonia Haiduc. 2018. "Automatic Traceability Maintenance via Machine Learning Classification." 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME) (2018): 369–380. https://ieeexplore.ieee.org/document/8530044.

Missaoui, Rokia., and Idrissa. Sarr. 2014. *Social Network Analysis - Community Detection and Evolution*. 1st ed. Cham: Springer International Publishing, 2014. https://doi.org/10.1007/978-3-319-12188-8.

Nishida, Kan. 2016. "Demystifying Text Analytics Part 4– Dimensionality Reduction and Clustering." *Learn Data Science*. June 26, 2016. https://blog.exploratory.io/demystifying-text-analytics-part-4-dimensionality-reduction-and-clustering-in-r-cbc8c90e0b14.

Object Management Group (OMG). 2021. "MBSE Wiki." January 28, 2021. https://www.omgwiki.org/MBSE/doku.php

Prabhakaran, Selva. 2018. "Cosine Similarity – Understanding the math and how it works." Machinelearningplus. Last modified October 22, 2018. https://www.machinelearningplus.com/nlp/cosine-similarity/.

Scikit-learn. 2021. Precision-Recall. Last accessed May 31, 2021. https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html.

Veluri, Sastry, and Agarwal, Vishal. 2019. Model-Based Systems Engineering: Digitized Requirements Management for Aerospace and Defense Products. Infosys. infosys.com/engineering-services/white-papers/Documents/model-based-systems-engineering.pdf.

Wang, Bangchao, Rong Peng, Yuanbang Li, Han Lai, and Zhuo Wang. 2018a. "Requirements Traceability Technologies and Technology Transfer Decision Support: A Systematic Review." The Journal of Systems and Software 146: 59–79. https://doi.org/10.1016/j.jss.2018.09.001.

Wang, Wentao, Nan Niu, Hui Liu, and Zhendong Niu. 2018b. "Enhancing Automated Requirements Traceability by Resolving Polysemy." In *2018 IEEE 26th International Requirements Engineering Conference* (RE), 40–51. IEEE. https://doi.org/10.1109/RE.2018.00-53.

Wheeler, Richard. 2016. "verification-cross-reference-matrix." *Rich Wheeler's Thoughts on SE, BA, RM, BPM, PM* (blog), June 2011. http://richwheeler.blogspot.com/2011/06/verification-cross-reference-matrix.html.

THIS PAGE INTENTIONALLY LEFT BLANK

# INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California