Improving Requirements Tracing via Information Retrieval

Jane Huffman Hayes
Computer Science Department
Laboratory for Advanced Networking
University of Kentucky
hayes@cs.uky.edu
(corresponding author)

Alex Dekhtyar
Computer Science Department
University of Kentucky
dekhtyar@cs.uky.edu

James Osborne
Computer Science Department
University of Kentucky
jas@netlab.uky.edu

Abstract

This paper presents an approach for improving requirements tracing based on framing it as an information retrieval (IR) problem. Specifically, we focus on improving recall and precision in order to reduce the number of missed traceability links as well as to reduce the number of irrelevant potential links that an analyst has to examine when performing requirements tracing. Several IR algorithms were adapted and implemented to address this problem. We evaluated our algorithms by comparing their results and performance to those of a senior analyst who traced manually as well as with an existing requirements tracing tool. Initial results suggest that we can retrieve a significantly higher percentage of the links than analysts, even when using existing tools, and do so in much less time while achieving comparable signal-to-noise levels.

Research

1. Introduction

There are two primary motivators for performing requirements tracing: ensuring that a new system does indeed satisfy all its specified requirements, and performing impact analysis on proposed changes. Both of these can be facilitated if a developer builds a detailed requirements trace as development proceeds. The adoption of CASE tools such as DOORS [25], RDD-100 [13], and Rational RequisitePro [21], or initiating process improvement initiatives, such as ISO–9000 [14], Capability Maturity Model [6], or Personal Software Process (PSP)/Team Software Process (TSP) [24], can spark such discipline in organizations that were formerly remiss. Though our auditing and verification and validation experience has shown the old adage “you can lead a horse to water but you can’t make it drink” to be true in many cases.

Requirement tracing is at best a mundane, mind numbing activity, as anyone who has spent any time performing this activity will tell you. Even with automation support, it is still a time consuming, error prone, person-power intensive task. It has been our experience that the tools that do exist to support this activity have numerous shortcomings: they require the user to perform interactive searches for potential linking requirements or design elements, they require the user to assign keywords to all the elements in both document levels prior to tracing, they return many potential or candidate links that are not correct, they fail to return correct links, and they do not provide support for easily retracing new versions of documents.

Since there are still many times when a requirements traceability matrix (RTM) does not exist and there is a need to ensure requirement completion and to understand change impact, a method for easy “after-the-fact” requirements tracing is needed. Requirements traces can be evaluated by calculating two metrics: the percentage of actual matches that are found (*recall*) and the percentage of correct matches as a ratio to the total number of candidate links returned (*precision*). As mentioned above, current methods are prone to error and require intense effort on the part of the analyst. Recall, precision, and performance values for these methods are not widely known or generalized. This paper presents the results of NASA-funded research to improve the state of the art of after the fact requirements tracing. The problem was cast in as an Information Retrieval problem, three algorithms were developed, an analysis tool was developed, and an evaluation study was performed. Our retrieval with thesaurus algorithm provided recall of 85.3% and precision of 40.6% in a much shorter period of time than analysts performing the same task.

Section 2 discusses related work in requirements tracing. IR Background on Information Retrieval (IR) is presented in Section 3. We discuss how requirements tracing can be represented as an IR problem and the algorithms we used in our study in Section 4. Section 5 discusses the results obtained from evaluation. Finally, Section 6 presents conclusions and areas for future work.

2. Related work

In the context of our work, there are two areas of interest: requirements tracing and IR as it has been applied to the problem of requirements analysis. Each will be addressed below.

2.1 Requirements tracing

We have been tackling the requirements tracing problem for many decades. In 1978, Pierce [18] designed a requirements tracing tool, basically a way to build and maintain a requirements database, to facilitate requirements analysis and system verification and validation for a large Navy undersea acoustic sensor system.

Hayes et al [12] built a front end for a requirements tracing tool called the Software Automated Verification and Validation and Analysis System (SAVVAS) Front End processor (SFEP). This was written in Pascal and interfaced with the SAVVAS requirements tracing tool that was based on an Ingres relational database. SFEP allows the extraction of requirement text as well as the assignment of requirement keywords through the use of specified linkwords such as *shall*, *must*, *will*, etc. These tools are largely based on keyword matching and threshold setting for that matching. Several years later the tools were ported to hypercard technology on Macs, and then to Microsoft Access and Visual Basic running on PCs. This work is described by Mundie and Hallsworth in [17]. These tools have since been further enhanced and are still in use as part of the Independent Verification and Validation (IV&V) efforts for the Mission Planning system of the Tomahawk Cruise Missile as well as for several NASA Code S science projects.

Abrahams and Barkley, Ramesh, and Watkins and Neal [1, 19, 27] discuss the importance of requirements tracing from a developer's perspective and explain basic concepts such as forward tracing, backward tracing, vertical tracing, and horizontal tracing. Casotto [7] examined run-time tracing of the design actvity. Her approach uses requirement cards organized into linear hierarchical stacks and supports retracing. Tsumaki and Morisawa [26] discuss requirements tracing using UML. Specifically they look at tracing artifacts such as use-cases, class diagrams, and sequence diagrams from the business model to the analysis model and to the design model (and back) [26].

There have also been significant advances in the area of requirements elicitation, analysis, and tracing. Work has largely been based on lexical analysis, such as extraction and analysis of phoneme occurrences to categorize and analyze requirements and other artifacts [22]. Bohner's work on software change impact analysis using a graphing technique may be useful in performing tracing of changed requirements [4]. Anezin and Brouse present advances in backward tracing and multimedia requirements tracing in [2, 5].

Cleland-Huang et al [8] propose an event-based traceability technique for supporting impact analysis of performance requirements. Data is propagated speculatively into performance models that are then re-executed to determine impacts from the proposed change. Ramesh et al examine reference models for traceability. They establish two specific models, a low-end model of traceability and a high-end model of traceability for more sophisticated users [20]. They found that a typical low end user created traceability links to model requirement dependencies, to examine how requirements had been allocated to system components, to verify that requirements had been satisfied, and to assist with change control. A typical high-end user, on the other hand, uses traceability for full coverage of the life cycle, includes the user and the customer in this process, captures discussion issues, decision, and rationale, and captures traces across product and process dimensions [20].

2.2 Information retrieval in requirements analysis

Dag et al [9] perform automated similarity analysis of textual requirements using IR techniques. They found this to be a promising method that helped identify relationships between requirements. Our work differs from theirs in that we examine requirements tracing "after the fact" while they focus on assisting developers who must deal with a rapid arrival of new requirements from numerous diverse sources. They propose to continuously analyze the flow of incoming requirements to increase the efficiency of the requirements engineering process [9].

Antionol et al [3] applied a probabilistic and a vector space IR model in two case studies to trace C++ source code onto manual pages and to trace Java code to functional requirements. They examined the effect of requiring 100% recall and found that the probabilistic model achieves the highest recall values, less than 100 percent, with a smaller number of documents retrieved and then performs better when 100% recall is required.

3. Information retrieval (IR)

IR is a field that studies the problem of finding relevant documents in document collections given user queries.

While IR research first appeared in the 1960s, it became a separate discipline of Computer Science in the late 1970s. The advent of the World Wide Web and the growth of data storage capacity of computers, which lead to the growth in the number and size of document repositories, generated a new wave of IR research in the 1990s. A recent introduction to IR is found in [10]; [11] provides an excellent roadmap for developing IR systems from scratch. We refer the reader interested in the history and development of IR to Sparck Jones, and Willet [23], a large collection of seminal and influential papers from the field.

IR attempts to model individual documents within document collections and to model user information needs. IR methods determine how *relevant* the document representation is to the query that represented user information need.

Among a large variety of methods of IR, keyword-based retrieval is arguably the most-studied and often-used method. In keyword-based IR, each document in the repository is analyzed to determine the (key)words or terms that are important for this document and can be used to query it. User queries are also analyzed for keywords and these keywords are compared with the ones associated with each document in the collection in order to determine matches. In most keyword-based methods, the *relevance* of the document to the query is expressed using a *similarity measure* that computes how closely the representations of a document and a query match. The answer to the query is given in the form of a list of documents in descending order of their expected relevance to the query.

The quality of IR methods is measured by how well the documents returned match the user's expectations. This is typically formalized with two metrics: *precision* and *recall*. *Precision* is computed as the fraction of the relevant documents in the list of all documents returned by the IR method given a query. *Recall* is the fraction of the retrieved relevant documents in the entire set of documents, retrieved and omitted, that are relevant to the query. We notice here that precision is a quantity that usually is relatively easy to measure given the query and the list of answers. Measuring recall is a much harder task, as it requires knowledge of the entire document collection. IR methods are usually designed to work on large collections of data: a typical test collection for an IR system is around 5-7 GB, whereas industrial-strength applications, such as WWW search engines are expected to handle data collections that are at least 2-3 orders of magnitude greater, **and** provide answers within seconds. Because of this, performance of the IR methods also plays a major role in their evaluation, as sometimes methods that give higher precision and recall become impractical due to the time it takes for them to deliver the answer.

There is a wide array of keyword-based retrieval models for document collections. Boolean model is the simplest: a representation of a document is a Boolean vector identifying the keywords present in the document. *Vector model* extends Boolean model by associating with each term in the document representation a *weight* that signifies its assumed importance to the document collection. Consider a standard vector retrieval model. Given a document ***d*** in the collection, let us denote its representation in the vector model as a vector $d=(w_1, w_2,\ldots,w_N)$, where N is the number of terms in the document collection's vocabulary and w_i is the abovementioned weight of the ith term. This weight is computed as

$$w_i = tf_i(d) \cdot idf_i,$$

where $tf_i(d)$ is the *term frequency* of the ith keyword in document d and idf_i is the *inverse document frequency of the ith term in the document collection.* Term frequency is usually the number of occurrences of the term in the document and is usually normalized. Inverse document frequency is computed as $idf_i = \log_2\left(\frac{n}{df_i}\right)$, where df_i is the total number of documents containing the ith term in the document collection and n is the size of the document collection. Basically, the importance of the term is judged by how often this term is found in the document and by how discriminating the term is. That is., the less frequent the term is in the collection, the more its presence is important for the document. A user query is also converted into a similar vector $q=(q_1,\ldots,q_N)$ of term weights. In this model, given a document vector d and a query vector q, the similarity between them is computed as the *cosine of the angle between vectors d and q in the N-dimensional space:*

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

Different extensions of the standard vector retrieval model exist, based on modifications to the computation of term weights in the document and similarity between the document and query vectors. There are also extensions of the vector model based on the use of additional information:

Retrieval based on user feedback. After the original list of the answers to the user query is compiled, the user is asked to specify which of the returned documents were relevant and which were not. Using this information it is possible to re-weigh the query vector and adjust the similarity computation in a way that documents similar to the ones the user declared relevant will get a higher relevance rating, while the documents similar to the ones declared irrelevant will drop significantly in their relevance rating.

Thesaurus—based retrieval. Classical vector model compares only the occurrences of individual keywords and key phrases. However, in many situations, one needs to take into account the presence in the document of the keywords synonymous or otherwise related to the query keywords. For example, the query "car retailer" will not match the document describing "Toyota dealership" in classical vector retrieval, but it may be very relevant to the query. *Thesauri* are collections of information about the relationship between different terms. Use of thesauri in IR allows one to extend classical vector retrieval to account for the presence of synonyms, words representing subcategories of the query terms, etc. Thesauri can come in a variety of different flavors: from very detailed descriptions of term hierarchies to ad-hoc lists of synonym pairs. The exact way of incorporating the thesaurus into the IR method depends of its type.

3. Automating requirements tracing

Among the tasks that must be performed during the requirements tracing, the most time-consuming and crucial activity is the *generation of candidate links*. Even with the aid of currently available support tools, this is still largely an analyst-driven process. Whether performing forward-, lateral- or back-tracing, the majority of the time analysts spend is devoted to *the generation of sensible lists of candidate matches.* . It is this portion of the requirements tracing process that we are automating.

In addressing the problem of automating the requirements tracing process, our main objectives were to improve the quality of the candidate lists as well as decrease the time needed for their generation. We notice that IR metrics of recall and precision are appropriate characteristics of the quality of candidate lists: recall measures the fraction of true matches that had been included, while precision measures the signal-to-noise ratio. Between these two metrics, we chose recall as our most important objective. This ordering ensures that an analyst feels confident that all related links have, indeed, been retrieved. Precision comes next, so that the analyst would not have to sift through numerous erroneous potential links. We notice however, that without good precision, total recall is a meaningless accomplishment. For example, in forward tracing, it can be achieved by simply including every single lower-level requirement in the candidate lists for every higher-level requirement.

As is apparent, automatic generation of candidate lists is bound to be orders of magnitude faster than their manual generation by the analyst, even assisted by currently available interactive tools. Automated generation drastically reduces the burden on the analyst of two time-consuming and frustrating activities: keyword assignment and interactive search for candidate links.

There were six major activities as we performed this work: (i) framing the problem in terms of an IR problem, (ii) selecting IR algorithms to implement, (iii) preparing the input requirement text, (iv) analyzing the output from the algorithms, (v) selecting strategies for trimming algorithm output, and (vi) comparing algorithm performance to human analyst performance. Each activity will be discussed in subsections below except for the algorithm evaluation activities that will be deferred until Section 5.

3.1 Requirements tracing as an IR problem

We illustrate how requirements tracing can be represented as an IR problem using forward tracing as an example. Note that the same technique can be applied to tracing design descriptions backward to requirement specifications, to tracing requirement specifications laterally to test specifications, to tracing design elements forward to code elements, etc. The collection of high-level requirements can be extracted from the high-level requirements document. Similarly, the lower-level document can be broken into the collection of individual lower-level requirements (also called design elements here). Each requirement and each design element can be treated as a separate document in an IR document repository. Generally it should contain all text and supplemental information (e.g., tables, graphs if such exist) necessary for the requirement/design element to be readable and understandable on its own.

Now, given the list $R = (r_1,\ldots,r_N)$ of requirements and the list $S=(s_1,\ldots,s_M)$ of lower-level design elements, a requirements trace is a mapping $tr:R \rightarrow 2^S$, where every design element $s \in tr(r)$ satisfies part or all of requirement r and no other design elements satisfy any parts of r.

The approach we adopt for this research is to consider requirements tracing as an IR problem. Because most manual or semi-automatic technologies for requirements tracing are keyword-based, keyword-based IR methods appear to be a natural extension of this process. In particular, we formalize the requirements tracing problem as follows. The universe of documents $D = R \cup S$ is the union of all individual requirements and design elements. Let $V_D = \{k_1,k_2,\ldots,k_L\}$ be the vocabulary of D, i.e., the list of all terms that appear in both higher-level and lower-level requirements. Each document $d_i \in D$ is represented as the vector of term weights $d_i=(w_{i,1},\ldots,w_{i,L})$. Assume the existence of a similarity measure, *sim*, that, given two vectors d_i and d_j, quantifies the similarity between them.

The process of building candidate link lists for the requirements tracing problem is then reduced to the procedure of computing the matrix of similarities between vectors $r_1,\ldots,r_N$,representations of high-level requirements, and vectors $s_i,\ldots,s_M$,representations of lower-level requirements. For each high-level requirement r_i the list $s'_1,\ldots,s'_{Mi}$ of design elements, such that $sim(r_i,s'_j)\geq 0$,sorted in the order of descending similarity value, serves as the first approximation of the candidate list. This list can further be pruned in a variety of different ways: for example, by considering only the top five vectors on the list, or by setting up some similarity threshold α and pruning out all specifications that exhibit smaller similarity.

3.2 Features of the requirements tracing process

While IR methods seem to provide a good match for the problem of automating the generation of the lists of candidate matches, the requirements tracing problem has a number of specific features that typical IR settings do not have. We briefly address these features here and discuss their implications on our attempts to apply IR techniques to requirements tracing.

1. *Size of the domain.* IR methods are designed for working with large numbers of large documents in the presence of large vocabularies. In requirements tracing, our domain is a fairly small collection of documents: there are on the order of thousands of requirements in a large-scale software development project, whereas, typical document collections number hundreds of thousands to millions of documents. An individual requirement is also quite short: it often contains just 2-3 sentences. Finally, the limited document collection that the requirements form has a relatively limited vocabulary.

   Implications of domain size. (i) Traditional IR methods become robust on large collections of data. Their performance on smaller collections can be not as good because the influence of individual components of the model on the final result grows, and, sometimes, coincidental matches outscore the true similarities. Therefore, we must be careful in evaluation of our IR methods. *(ii)* On the other hand, because of small domain sizes, we can apply some of the more complex IR techniques that are typically deemed to be too slow for large data collections. This is part of our future work.

2. *Query interdependence.* It is customary in IR systems to consider all queries as being independent. It is a reasonable assumption for Internet search engines which process thousands of queries each second coming from thousands of different users. In the requirements tracing problem, though, the queries are the higher-level requirements, which are, very often, related to each other.

   Implications of query interdependence. The result of our automated process is the matrix of similarities between higher-level and lower-level requirements. Knowing that the rows of this matrix may be not independent, we can perform secondary analysis on this matrix comparing the candidate lists for different requirements and the similarity measures.

3.3 Selection and modification of IR algorithms

In our initial study we have explored three different IR methods: classical vector IR model and two extensions of it with simple thesauri constructs. All algorithms followed the same path from data preparation to generation of answers. First, individual requirements were extracted from higher- and lower-level requirements documents using automatic extraction scripts, similar to those found in SuperTracePlus™ and commercial tools. Each requirement/design element was stored as a separate file. The repository thus generated was used as the input for the model-building tool.

On the model-building stage the following is done: (i) each requirement is parsed and tokenized; (ii) stopwords (i.e., words that are not useful for the purposes of retrieval, like “shall”, “the” or “for”) are detected and removed from the token stream; (iii) the remaining tokens are stemmed to ensure that different forms of the same word are treated as one term (e.g., “information” and “informational”); (iv) the vector representation of the document is created and stored. As a byproduct, the master vocabulary of the repository is constructed.

Once the vector models of requirements are built, the actual retrieval process proceeds as follows. The list of queries, higher-level requirements for the case of forward-tracing, is processed one-by-one and converted into query vectors using the same parse -> remove stopwords -> stem sequence. After that, similarity computation is performed for each query-design element pair. A list of design elements with non-zero similarity is created for each query, sorted in the descending order of the similarity value. These lists are returned to the analyst.

The first method implemented, *vanilla vector retrieval,* has been described in Section 2. As the basis, we have used a generic IR system developed by a graduate student during an IR course taught by one of the co-authors. The provided software had been modified to work with repositories of requirements and design elements, but the main computational procedures were kept intact.

For the second method, *retrieval with key-phrases*, we have augmented the traditional vector model, by associating a list of technical terms or key-phrases with the document repository. When the model-building software detected a technical term, it was added to the vocabulary and treated as any other term from then on. This allowed us to raise the relevance of matches related to technical terminology and exclude some coincidental matches. For example, our requirements and design elements contain the phrases "ecs production environment" and "ecs archive metadata." In the standard vector model, the match on keyword "ecs" generated a false positive, but by qualifying the phrases above as vocabulary terms we were able to decrease the relevance of this match. On the other hand, because "ecs archive metadata" became a much more discriminating term than any of its individual components, the key-phrase match between two documents started to carry much more weight. We note here, that the generation of the list of technical terms is a reasonably simple and straightforward task: it is readily found in the definitions or acronyms sections of most requirement documents.

The third method, *thesaurus retrieval*, took the idea of incorporating technical lingo into the retrieval process one step further. To aid vector retrieval we used a simple *thesaurus*. Each entry of our thesaurus is a triple (k_i,k_j,α_{ij}), where k_i and k_j are vocabulary terms (either words or key-phrases) and $\alpha_{ij}\in[0,1]$ is a *perceived similarity coefficient* of k_i and k_j. The analyst assigns this coefficient to each thesaurus entry , hence the qualifier "perceived". During the model-building stage, thesaurus entries are recognized and added to the vocabulary as new terms, similar to the addition of key-phrases in the previous method. The main change in the behavior of this method with respect to the other two, however, comes during the query processing stage. When computing the similarity between a query requirement $r=(r_1,\ldots,r_N)$ and a design element $d=(w_1,\ldots,w_N)$, the standard cosine computation receives an add-on that is generated by matches found via the thesaurus. More formally, letting T denote the thesaurus, the new similarity measure, sim_T, used in this method is computed as:

$$sim_T(d,q)=\frac{\sum_{i=1}^{N} w_i\cdot q_i+\sum_{(k_i,k_j,\alpha_{ij})\in T}\alpha_{ij}\cdot(w_i\cdot r_j+w_j\cdot r_i)}{\sqrt{\sum_{i=1}^{N} w_i^2\cdot\sum_{i=1}^{N} r_i^2}}.$$

Construction of a thesaurus for our thesaurus-based retrieval method is a reasonably straightforward procedure. Most of the information linking technical terms is present in data dictionaries and acronym lists that are typically found in the appendices of requirements documents. The analyst has to assign similarity weights to each constructed pair, but the final computation of similarity is fairly robust with respect to small fluctuations in these weights so the analyst needs only provide the "ballpark" estimate. If the analyst chose not to assign such a value, a default value is assigned. Using a thesaurus entry ("corrupted data", "missing packets", 1.0), we can establish similarity between a requirement and a design element which otherwise contain no matching terms.

All our IR algorithms were implemented in C++ running under Linux.

4. Evaluation

We undertook a multi-faceted evaluation effort to ensure that our research objectives had been met. We built two datasets from open source NASA Moderate Resolution Imaging Spectroradiometer (MODIS) documents [16] for this purpose. One dataset contains ten high level requirements and ten lower level requirements and the other contains 19 high level and 50 low-level requirements. The 10x10 dataset was a subset of the 19x50 dataset. We then verified the 10x10 trace and the 19x50 trace. To accomplish this, we had a senior analyst with 20 years of experience examine the traceability matrix provided in Table 7-1 of the MODIS Science Data Processing Software Requirements Specification Version 2 document [16]. These high level requirements were traced down to the Level 1A (L1A) and Geolocation Processing Software Requirements Specification [15]. Several changes were made based on this review. For example, some links were deleted from the RTM and some links were added. Approximately 90% of the RTM remained unchanged though.

Second, we ran the vanilla vector retrieval algorithm on the 10x10 and 19x50 datasets. We developed an analysis tool to compare the result matrix generated by the query tool to the manually verified RTM (see Section 5). The tool, written in C++, computes precision and recall for each document and for the whole of the dataset. We used the data analysis tool to examine the results. We also asked two junior analysts with less than 5 years of experience to manually trace the 10x10 dataset. The results of these tracing activities are shown in Table 1.

As can be seen, the analysts tied or outperformed the vanilla vector algorithm in overall recall (by 0 – 20%). One analyst outperformed precision by 12.4% while the other analyst was outperformed by 2.6%. The vanilla vector algorithm had slightly better recall on the larger dataset, but lower precision. The analysts recorded the amount of time it took them to build candidate lists and perform a relative/similarity assessment of these (the same tasks performed by our algorithm). It took the analysts 65 minutes and 150 minutes, respectively, to perform the trace. Not surprisingly, our algorithms ran in less than a minute.

Table 1. Results of baseline algorithm compared to analysts.

	Analyst 1	Analyst 2	Vanilla vector (10 x 10)	Vanilla vector (20 x 50)
Recall	23.0%	42.9%	23.0%	25.4%
Precision	15.0%	30.0%	17.6%	11.4%
Performance (min.)	65	150	seconds	Seconds

These results are not surprising. The vanilla vector algorithm works well only when the vocabularies of the high and low level requirements are close enough to generate multiple keyword matches. In this case, the two levels of requirements had been written using very different terminology. This contributed to low recall. Low precision can be explained by the fact that coincidental matches of common English words often obscured technical terminology.

Next, we implemented two additional IR algorithms that extend the vanilla vector algorithm as discussed in Section 4. The key-phrase retrieval algorithm slightly improved recall on the 10x10 dataset (see Table 3). However, the precision went down. This is also not surprising: while the introduction of technical terminology allowed us to capture some previously undetected matches, more noise was also introduced.

The retrieval with thesaurus algorithm was tested on the 19x50 dataset. In addition, we had a senior analyst at Science Applications International Corporation (SAIC) trace the same dataset using the latest version of SuperTracePlus™, the requirements tracing tool discussed in [12] and [17] in Section 2.1. The results are shown in Table 2. We show two sets of recall and precision measures. The average recall and precision metrics represent the respective means of precision and recall values for the 19 individual requirements. The overall precision and recall are computed as the fractions of the total number of correctly found matches to the total numbers of supplied answers and correct links, respectively.

Table 2. Results of enhanced algorithms compared to SuperTracePlus.

	SuperTracePlus Tool	Analyst	Retrieval with thesaurus algorithm
Correct links	41	41	41
Correct links found	26	18	**35**
Total number of candidates	67	39	86
Missed requirements	**3**	6	4
Average recall	69.37%	53.30%	**71.69%**
Average precision	**56.48%**	53.55%	32.76%
Overall recall	63.41%	43.9%	**85.36%**
Overall precision	38.80%	**46.15%**	40.69%
Performance (hours)	N/A – included in analyst performance	9	Seconds for algorithm, 0.33 for thesaurus building

The 19x50 dataset included a number of high-level requirements with no matching low-level requirements. For the purposes of evaluation, we considered precision and recall for these requirements to be 100% if no candidates were produced and 0% otherwise. This is accounted for in the average recall and precision measures but not in the overall recall and precision because the latter measures look only at the total number of correct links retrieved.

Note that the retrieval with thesaurus algorithm achieved recall of over 85% with 40.6% precision on a dataset that is only 19x50. The recall outperforms SuperTracePlus™ by 22% and the analyst by a whopping 42%. The algorithm's average recall also outperformed the analyst by 18% and the SuperTracePlus™ tool by 2%. The analyst outperformed the retrieval with thesaurus algorithm in precision per requirement by 21% and in overall precision by 5%. The SuperTracePlus™ tool outperformed the algorithm only in precision per requirement, by 24%. It should be noted that it took the analyst 13 hours to perform the trace. Nine of the thirteen hours were spent on building a link library (keyword assignment, two hours) and tracing and data review (examining links and interactively searching for others, seven hours). Our thesaurus was built in less than 20 minutes by cutting and pasting from the data dictionary and other appendices of the document.

Taking a closer look, we see that our algorithm only missed 6 links while SuperTracePlus™ missed 15 and the analyst missed 17. We found that there were three high level requirements that did have low level links for which SuperTracePlus™ found no match, four such requirements that were missed by our retrieval with thesaurus algorithm, while the analyst's count was six (see "missed requirements" row of Table 3). Two of the requirements that SuperTracePlus™ missed were caught by our algorithm. One of the requirements that our algorithm missed was found by SuperTracePlus™. There were three requirements that both SuperTracePlus™ and the thesaurus retrieval algorithm missed. As part of our future work, we will examine these three requirements carefully to understand how our algorithms might be improved.

The SAIC analyst made a number of observations during the tracing activity:

- It was difficult to do some of the tracing because the two documents were incomplete. Section titles would have helped. Also, some requirement text was incomplete and ambiguous. For example, some sentences were incomplete sentences, did not have a subject, or in one case said "Deleted."
- Not knowing acronyms hindered the linking process.
- The linking of parent and child requirements does not take into account the analyst-assigned status. For example, I might have selected a link or several links, but selected a status of "Partial."

The first observation is accurate. The requirements were extracted from the RTM of the documents and the section headings were not repeated with the requirement text. We also found the second observation to be true as we verified the trace and also as we interviewed the two analysts who manually traced the 10x10 dataset. The final comment is also valuable. Basically the analyst is saying that (s)he may not have been convinced that a high level requirement was satisfied. But the analyst could not find any more links for it, so had to leave it. The analyst therefore would have noted that requirement as only partially satisfied. Our study did not allow for collection of that information. We will consider this for improving our future studies.

Analysis of our retrieval algorithms showed the presence of many false positives. We also noticed that many of these were returned with very low relevance. In order to analyze the true effectiveness of our algorithms, we chose to implement various thresholds to trim the lists of candidate links. Decreasing the size of the lists this way allows us to improve precision at a potential cost to recall. We have decided upon four thresholds: Top 4 candidates, any candidates with similarity above 0.25, and any candidates with similarity within 0.33 and 0.50 of the similarity of the top candidate.

Table 3 shows how overall recall ([R]) and precision ([P]) for all three of our algorithms change for different trimming thresholds. Note that Above 25% trimming yields the highest results for the first two algorithms, but not for the retrieval with thesaurus algorithm.

Table 3. Algorithm results after trimming.

	Vanilla algorithm (10x10) [R,P]	Retrieval w/key phrases algorithm (10x10) [R,P]	Retrieval w/thesaurus algorithm (19x50) [R,P]
No trimming	[23%,17.6%]	[27.2%,5.2%]	**[85.4%,40.6%]**
Top 4	[23%,17.6%]	[27.2%,8.3%]	[36.5%,30.6%]
Above 25	[23%,**75%]**	[27.2%,**25%]**	[9.7%,40%]
Within 33	[23%,23%]	[27.2%,15.7%]	[48.7%,**44.4%]**
Within 50	[23%,20%]	[27.2%,15.7%]	*[58.5%,42.1%]*

5. Conclusions and future work

In this paper, we have studied a method for improving candidate link generation by applying IR techniques. We started with a classical vector space model algorithm, the vanilla vector algorithm. We found that this algorithm does not outperform analysts or existing tools in terms of recall or precision, but that it does perform faster and with no keyword assignment required of analysts. Next, we developed two extensions to this algorithm. The first uses a simple key-phrase list, one that can be easily pulled from the definitions or acronym section of a requirement document. The retrieval with key-phrases algorithm resulted in improved recall but with decreased precision. Next, we added a simple thesaurus. We found this information to be readily available in the definition list and the data dictionary in the appendix of the traced documents. Testing the thesaurus-based retrieval algorithm on the 19x50 dataset, we found that recall improved to 85% and that precision moved up to 40%.

Evaluation of the algorithms against a comparable keyword-based tool and analysts showed that the retrieval with thesaurus algorithm outperforms all in terms of recall and sometimes - in terms of precision.

Note that the techniques have been evaluated using a forward trace process. However, these techniques can just as easily be applied to back tracing and lateral tracing.

We found that a number of things pose problems for analysts: incomplete or ambiguous requirement documents; undefined acronyms; lack of intimate domain area or project knowledge; and different lingo in which the high- and low-level documents are written.

We note that the methods studied in this paper address the problem of automating the candidate link generation. It is imperative to have the analyst examine the final candidate list to effectively complete requirements tracing. By improving the candidate link lists, we reduce the burden on the analyst. In addition to that, using feedback-based retrieval techniques, we can make this last stage of the process more efficient. We are currently in the process of developing such a feedback agent for our candidate link generator software. Another interesting area for future research is using IR techniques to predict the coverage or satisfaction of traced requirements by their lower level requirements.

The initial results are promising and indicate that additional work is warranted. The results, however, are limited and the effectiveness of our IR models on a broader scale remains to be seen. A much larger scale study is required before any broad conclusions can be reached. We have secured agreement from the International Space Station to allow use to use their documents for tracing studies. This will allow us to trace several thousand high-level requirements to tens of thousands of low-level elements. We are confident that our algorithms will only perform better as the document collection size increases.

6. Acknowledgments

Our work is funded by NASA under grant NAG5-11732. Our thanks to Ken McGill, Tim Menzies, Stephanie Ferguson, Pete Cerna, Mike Norris, Bill Gerstenmaier, Bill Panter, the International Space Station project, and the MODIS project for maintaining their website that provides such useful data. We thank Hua Shao for assistance with the vanilla retrieval algorithm and K.S. Senthil for his assistance with evaluation.

7. References

[1] Abrahams, M. and Barkley, J., "RTL Verification Strategies," IEEE WESCON/98, 15 - 17 September 1998, pp. 130-134.

[2] Anezin, D., "Process and Methods for Requirements Tracing (Software Development Life Cycle)," Dissertation, George Mason University, 1994.

[3] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E. Recovering Traceability Links between Code and Documentation. IEEE Transactions on Software Engineering, Volume 28, No. 10, October 2002, 970-983.

[4] Bohner, S., "A Graph Traceability Approach for Software Change Impact Analysis," Dissertation, George Mason University, 1995.

[5] Brouse, P., "A Process for Use of Multimedia Information in Requirements Identification and Traceability," Dissertation, George Mason University, 1992.

[6] Capability Maturity Model, http://www.sei.cmu.edu/cmm/cmms/cmms.html

[7] Casotto, A.. Run-time requirement tracing, Proceedings of the IEEE/ACM International Conference on Computer-aided Design, Santa Clara, CA, 1993.

[8] Cleland-Huang, J., Chang, C.K., Sethi, G., Javvaji, K.; Hu, H., Xia, J. (2002) Automating speculative queries through event-based requirements traceability. *Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02),* Essex, Germany, 9-13 September, 2002, pages: 289- 296.

[9] Dag, J., Regnell, B., Carlshamre, P., Andersson, M., Karlsson, J. A feasibility study of automated natural language requirements analysis in market-driven development, Requirements Engineering, Vol. 7, Issue 1, p.20, June 2002.

[10] Daeza-Yates, R. and Ribeiro-Neto, B. *Modern Information Retrieval,* Addison-Wesley, 1999.

[11] W. Frakes, R. Baeza-Yates (Eds.), *Information Retrieval: Data Structures and Algorithms,* Prentice Hall, 1992.

[12] Hayes, J. Huffman. Risk reduction through requirements tracing. In The Conference Proceedings of Software Quality Week 1990, San Francisco, California, May 1990.

[13] Holagent Corporation product RDD-100, http://www.holagent.com/new/products/modules.html

[14] ISO 9000-3:1997 Quality management and quality assurance standards -- Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software, http://www.iso.ch/

[15] Level 1A (L1A) and Geolocation Processing Software Requirements Specification, SDST-059A, GSFC SBRS, September 11, 1997.

[16] MODIS Science Data Processing Software Requirements Specification Version 2, SDST-089, GSFC SBRS, November 10, 1997.

[17] Mundie, T. and Hallsworth, F. Requirements analysis using SuperTrace PC. In Proceedings of theAmerican Society of Mechanical Engineers (ASME) for the Computers in Engineering Symposium at the Energy & Environmental Expo 1995, Houston, Texas.

[18] Pierce, R. A requirements tracing tool, Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues, 1978.

[19] Ramesh, B., "Factors Influencing Requirements Traceability Practice," Communications of the ACM, December 1998, Volume 41, No. 12 pp. 37-44.

[20] Ramesh, B.; Jarke, M. Toward reference models for requirements traceability***;*** IEEE Transactions on Software Engineering, Volume 27, Issue 1, January 2001, page(s): 58 –93.

[21] Rational RequisitePro, http://www.rational.com/products/reqpro/index.jsp

[22] Savvidis, I. "A Multistrategy Framework for Analyzing System Requirements (Software Development)," Dissertation, George Mason University, 1995.

[23] Sparck Jones, K. and Willet, P. *Readings in Information* Retrieva,l Morgan Kaufmann Series in Multimedia Information and Systems, Morgan Kaufmann, 1997.

[24] Team Software Process and Personal Software Process, http://www.sei.cmu.edu/tsp/

[25] Telelogic product DOORS, http://www.telelogic.com/products/doorsers/doors/index.cfm

[26] Tsumaki, T. and Morisawa, Y. "A Framework of Requirements Tracing using UML," Proceedings of the Seventh Asia-Pacific Software Engineering Conference 2000, 5 - 8 December 2000, pp. 206 - 213.

[27] Watkins, R. and Neal, M. "Why and How of Requirements Tracing," *IEEE Software*, Volume 11, Issue 4, July 1994, pp. 104-106.

.