2013

APPLICATION OF SWARM AND REINFORCEMENT LEARNING TECHNIQUES TO REQUIREMENTS TRACING

Hakim Sultanov
University of Kentucky, hisult2@g.uky.edu

APPLICATION OF SWARM AND REINFORCEMENT LEARNING TECHNIQUES TO REQUIREMENTS TRACING

DISSERTATION

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the College on Engineering at the University of Kentucky

By
Hakim Sultanov

Lexington, Kentucky
Director: Dr. Jane Huffman Hayes, Professor of Computer Science
Lexington, Kentucky
2013

ABSTRACT OF DISSERTATION

APPLICATION OF SWARM AND REINFORCEMENT LEARNING TECHNIQUES TO REQUIREMENTS TRACING

Today, software has become deeply woven into the fabric of our lives. The quality of the software we depend on needs to be ensured at every phase of the Software Development Life Cycle (SDLC). An analyst uses the requirements engineering process to gather and analyze system requirements in the early stages of the SDLC. An undetected problem at the beginning of the project can carry all the way through to the deployed product.

The Requirements Traceability Matrix (RTM) serves as a tool to demonstrate how requirements are addressed by the design and implementation elements throughout the entire software development lifecycle. Creating an RTM matrix by hand is an arduous task. Manual generation of an RTM can be an error prone process as well.

As the size of the requirements and design document collection grows, it becomes more challenging to ensure proper coverage of the requirements by the design elements, i.e., assure that every requirement is addressed by at least one design element. The techniques used by the existing requirements tracing tools take into account only the content of the documents to establish possible links. We expect that if we take into account the relative order of the text around the common terms within the inspected documents, we may discover candidate links with a higher accuracy.

The aim of this research is to demonstrate how we can apply machine learning algorithms to software requirements engineering problems. This work addresses the problem of requirements tracing by viewing it in light of the Ant Colony Optimization (ACO) algorithm and a reinforcement learning algorithm. By treating the documents as the starting (nest) and ending points (sugar piles) of a path and the terms used in the documents as connecting nodes, a possible link can be established and strengthened by attracting more agents (ants) onto a path between the two documents by using pheromone deposits. The results of the work show that ACO and RL can successfully establish links between two sets of documents.

KEYWORDS: Software Engineering, Requirements Engineering, Traceability, Swarms, Reinforcement Learning

Hakim Sultanov

Student’s Signature

March 27, 2013

Date

APPLICATION OF SWARM AND REINFORCEMENT LEARNING TECHNIQUES TO REQUIREMENTS TRACING

By

Hakim Sultanov

Dr. Jane Huffman Hayes

Director of Dissertation

Dr. Raphael Finkel

Director of Graduate Studies

March 27, 2013

I dedicate my dissertation to my family and my parents. I express a deep appreciation of the possibilities and brave world outlook given to me by my loving parents, Ismat Sultanov and Sanobar Azimova.

I dedicate this work to my wife, Marianna Zagurovskaya, for being a gentle and loving support, so needed, for completing this work. A special thanks to my children Arthur and Marc, who shared my interest in learning about ants and provided cheerful and reflective drawings of the process.

ACKNOWLEDGEMENTS

I would like to thank Dr. Jane Hayes for her guidance and for being a patient and outstanding advisor. I would also like to thank Dr. Judy Goldsmith for her valuable input. Thanks also to my committee members Dr. Brent Seales and Dr. Kevin Donohue for their support.

I would like to offer thanks to fellow graduate students Wei-Keat Kong, Wenbin Li, and Jody Larsen for their support.

Table of Content

List of Figures

List of Tables

1. Introduction

Today, software has become deeply woven into the fabric of our lives. Software controls a pump meter at a gas station, manages concurrent display of maps and conversation on a cell phone, and controls the ascent of rockets into space. Software malfunctions can cause disasters both small and large. For example, a malfunction in a rocket's control software may cause the rocket to disintegrate in pieces like the Ariane 5 [1]. For these reasons, the quality of the software we depend on needs to be ensured at every phase of the Software Development Life Cycle (SDLC).

The SDLC consists of four main phases: Planning, Analysis, Design, and Implementation and Testing [2]. The planning phase aims to address justifications for the software system, feasibility studies, risk management, etc. During the analysis phase, the requirements for the future software system are elicited, gathered, negotiated, and validated. When the SDLC enters the design phase, these requirements are transformed into design elements describing how the required functionality is achieved. The implementation phase encompasses the development and test of the designed elements.

1.1 Requirements Tracing

An analyst uses the requirements engineering process to gather and analyze system requirements. During this process, the analyst clarifies customer needs, conducts feasibility studies, presents and specifies a solution, and cross validates the specifications. In a large-scale project, it is quite possible to miss or misinterpret some of the identified requirements. In his book, *Patterns of Software System Failure and Success,* Jones says that more than 80% of the failures in large-scale mission-critical projects are attributed to undetected problems in the early phases of the SDLC [3]. An undetected problem at the beginning of the project can carry all the way through to the deployed product; this is called a latent defect or latent error.

Such undetected problems can have the additional effect of lengthening a project's timeline and expanding the development budget. Boehm and Basili point out that as the software life cycle progresses, the cost of fixing or changing software increases. They claim that finding and fixing a software problem after delivery is often one hundred times more expensive than finding and fixing it during the requirements and design phase [4]. Boehm's curve, shown below, illustrates a simple idea: create a proper set of requirements accompanied by good and detailed design or face the strong possibility of paying a higher price later [5].

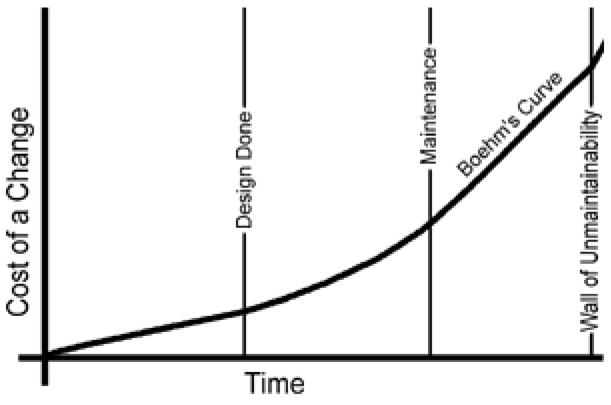


Figure 1.1 Boehm curve

To address and mitigate the possibility of costly latent errors, an analyst should collect, note, and track the software requirements during the early phases of the SDLC.

Two sets of documents are typically created in the early phases of any software project: the Software Requirements Specification (SRS) and the Software Design Description (SDD). These two sets of documents capture the information needed to properly identify the required functionality (SRS) and then define how the software should be structured to provide the functionality required (SDD).

According to the Software Engineering Body of Knowledge (SWEBOK), a software requirement is "a property which must be exhibited by software developed or adapted to solve a particular problem [6]." These requirements are captured in the Software Requirements Specification (SRS). This document is defined by IEEE Standard 1012-1998 as "documentation of the essential requirements (i.e., functions, performance, design constraints, and attributes) of the software and its external interfaces. The software requirements are derived from the system specification [6]."

The Software Design Description (SDD) is a "representation of software created to facilitate analysis, planning, implementation, and decision making. The software design description is used as a medium for communicating software design information, and may be thought of as a blueprint or model of the system [7]."

1.2 Requirements Traceability Matrix

The process of Validation and Verification (V&V) uses artifacts created during early phases of the Software Development Life Cycle (SDLC). Among other things, V&V ensures that the every requirement specification element is adequately reflected by at least one design description element.

The Requirements Traceability Matrix (RTM) serves as a tool to demonstrate how requirements are addressed by the design and implementation elements throughout the entire software development lifecycle.

The activity of building an RTM is a part of the requirements tracing process. The process involves seven steps [8]:

1. Identify each requirement and design element.
2. Assign a unique identifier to each requirement and design element.
3. For each requirement, locate all matching design elements.
4. For each design element, locate a parent element in the collection of requirements.
5. Determine if each requirement has been completely satisfied.
6. Prepare a report that presents the traceability matrix.
7. Prepare a summary report that expresses the level of traceability of the document pair.

Creating an RTM matrix by hand is an arduous task. For each combination of requirements and design documents, an analyst must open two documents (the requirement document and the design document) using a word processor application and then search for key terms and phrases that may be relevant or important for establishing a possible logical link between two documents.

For example, an analyst opens a requirement document from the requirements collection, analyzes the content, and notes main points. Then the analyst opens a document from the design elements collection and searches for key ideas, terms, or phrases in the opened design document. Here, we make an assumption that all documents are text based. This process of opening, analyzing, and searching within each document is repeated for every pair of requirement and design documents.

An RTM provides a view of the requirements to design elements mapping in a matrix form. Each row corresponds to a requirement. Each column corresponds to a design element. Requirement elements addressed by the design elements are marked in an appropriate row and column. Figure 1.2 displays a sample RTM for Requirements to Design Traceability.

The following are sample requirements (R.1 through R.4) and design elements (D.1 through D.4) corresponding to Figure 1.2.

R.1 The image viewer will allow the viewing of images.
R.2 The system shall mark images checked for printing.
R.3 The system shall allow printing image sections.
R.4 The system shall provide information about displayed images.

D.1 Annotation overlay to indicate marked images.
D.2 A list to present the indexes and information about the images.
D.3 A user interface to display images and respective information and controls to print images.
D.4 A cropper tool to select sections of an image.

	D.1	D.2	D.3	D.4
R.1			X	
R.2	X		X	
R.3				X
R.4		X		

Figure 1.2. RTM requirements vs. design elements.

We can see that requirements R1 and R2 are addressed by design element D3. The requirements state that the system needs to be able to view and print images. These requirements are addressed by the design description of a user interface to view and print the images.

As the example shows, in addition to being labor intensive, a manual generation of an RTM can be an error prone process as well. The manual process requires a human analyst to cross check every pair of documents. Luckily, there are automatic tools designed to alleviate the process of matching requirements artifacts with design elements [9][10].

1.3 Problem Statement

As the size of the requirements and design document collection grows, it becomes more challenging to ensure proper coverage of the requirements by the design elements, i.e. assure that every requirement is addressed by at least one design element. The techniques used by the existing requirements tracing tools take into account only the content of the documents to establish possible links. We expect that if we take into account the relative order of the text around the common terms within the inspected documents, we may discover candidate links with a higher accuracy.

The aim of this research is to demonstrate how we can apply machine learning algorithms to software requirements engineering problems. This work addresses the problem of requirements tracing by viewing it in light of the Ant Colony Optimization (ACO) algorithm [11] and a reinforcement learning algorithm [12]. By treating the documents as the starting (nest) and ending points (sugar piles) of a path and the terms used in the documents as connecting nodes, a possible link can be established and strengthened by attracting more agents (ants) onto a path between the two documents by using pheromone deposits. The results of the work show that ACO and RL can successfully establish links between two sets of documents [13].

1.4 Research Thesis

The research demonstrates two approaches, one based on the Ant Colony Optimization algorithm and the other is based on Reinforcement Learning, to identify candidate links between two collections of documents: the requirements and the design documents.

The requirements tracing tool is based on the existing tool Retro.NET [9]. Our tool establishes the candidate links by applying the Ant Colony Optimization algorithm and Reinforcement Learning.

1.5 Scope of Research

The research is aimed at English textual software requirements and design documents. An assumption is made that requirements and design documents are presented as two separate collections.

1.6 Research Contributions

This research makes the following contributions: establish candidate links based on the common textual segments between documents; and emphasize and demonstrate the benefit of treating documents as collection of phrases, rather than "bag of words." As consequence of this approach, the suggested method establishes links of a higher quality between textual documents. The quality of the links can be evaluated through the ratio of the number of correctly suggested links vs. the total number of suggested links. The higher the ratio, the better is the quality of the links. A correct list of links between the document pair ensures higher efficiency for the human analyst performing the tracing process.

The remainder of the dissertation is organized as follows: Chapter 2 provides necessary background information. Chapter 2 consists of sections on Requirements Traceability, Information Retrieval, Swarm Intelligence, and Reinforcement Learning. Chapter 3 surveys the related work in the field. Chapter 4 discusses pheromone swarm technique and the results obtained through this technique. Chapter 5 presents Reinforcement Learning algorithm applied to the requirements tracing problem. Chapter 6 contains the dissertation conclusions and directions for possible future work.

2 Background

To understand the proposed ideas using the swarm technique and the reinforcement learning for requirements tracing, it is necessary to understand the basic concepts in the following areas: information retrieval (IR), requirements tracing, swarm intelligence, and reinforcement learning (RL).

2.1 Information Retrieval

Information retrieval (IR) is the process of finding documents relevant to an information request within a collection of documents, usually a search query. In a typical scenario, the documents returned in response to a query are sorted by weight relevance. The relevance weight is a computer calculated numeric value indicating how closely the returned document matches the requesting query; the higher the weight, the more relevant the document is to the query. From a user perspective, a document is *relevant* if the user considers the document relevant to the original query. The user may not agree with the high weight relevance of every returned document.

The effectiveness and accuracy of the IR method can be evaluated through *recall* and *precision* measurements. *Recall* is evaluated as the total number of relevant retrieved documents divided by the total number of relevant documents in the whole collection.

$$\text{Recall} = \frac{\#\text{of relevant retrieved}}{\#\text{of relevant in collection}} \qquad \textbf{2.1.1}$$

Precision is evaluated as the total number of relevant retrieved documents divided by the total number of retrieved documents:

$$\text{Precision} = \frac{\#\text{of relevant retrieved}}{\#\text{of retrieved}} \qquad \textbf{2.1.2}$$

Precision and recall can be combined into a weighted harmonic mean:

$$F = \frac{(\beta^2+1)P*R}{\beta^2 P + R} \text{ ,where } \beta^2 \in [0,\infty)\,. \qquad \textbf{2.1.3}$$

When β^2 = 1, precision and recall are balanced in the measure, this is called F_1 measure. When $\beta^2 = 2$, recall has more weight than precision, this is called F_2 measure.

Higher recall and precision measurements indicate higher completeness and accuracy of the retrieved data.

A secondary measurement such as Mean Average Precision (MAP) measures "the quality across the recall levels" [14]. The higher the MAP, the closer the true links are to the top of the candidate link list. For h_j in a set of textual artifacts $H=\{h_{-1},\ldots, h_{-n}\}$, a subset of relevant documents $\{d_{-1},\ldots, d_{-m_j}\}$, *and* $L_{jT} \in L=\{(d,h)|sim(d,h)\}$ *a subset of true links ranked by relevance,* MAP is evaluated as follows:

$$\text{MAP(H)} = \frac{1}{|H|}\sum_{j=1}^{|H|}\frac{1}{m_j}\sum_{k=1}^{m_j}\Pr ecision(L_{jT}). \qquad \textbf{2.1.4}$$

A high value of MAP implies that true links are ranked higher in the list of the returned results.

2.2 IR Methods

There are several IR methods. The following two methods are the most common:

- Boolean
- Vector space

In the rest of this section, we introduce these techniques.

2.2.1 Boolean Retrieval

In the Boolean Retrieval model, a query is constructed in the form of a Boolean expression of terms. In this model, each document is treated as a collection of terms/words. One way to determine the presence of a word in a document is to scan the documents linearly. To facilitate the search, the *incidence matrix* is constructed. The *incidence matrix* indicates the presence of a term in a document; one (1) indicates the document contains the term, zero (0) indicates the document does not contain the term.

	A1.txt	A2.txt	C2.txt	C3.txt	D3.txt	F1.txt
Personal	1	0	0	0	0	0
Distribution	1	1	0	0	0	0
List	1	1	1	0	1	0
Email	0	1	1	1	0	1
System	0	0	0	1	1	0
..						
Store	1	0	0	0	1	1

Figure 2.1 A term document incidence matrix.

Examining Figure 2.1, a query of *Personal* AND *Distribution* AND *List* AND *Store*, we will take the vectors for these terms and do a bitwise AND:

100000 AND 110000 AND 111010 AND 100011 AND = 100000

The result for this query is document A1.txt. A1.txt is the only document containing the term *Personal.*

The limitation of Boolean Retrieval can easily be discovered by querying a collection of 1 million documents with 100,000 distinct terms. It would be hard to fit a matrix of $10^6 * 10^5 = 10^{11}$ bits in the operating memory of a computer.

To overcome the limitations of the incidence matrix for a huge collection of documents, the *inverted index* has become a major concept in the field of information retrieval [15]. All of the distinct terms across the documents in the collection comprise a *dictionary*. For each term in the dictionary, the inverted matrix maintains a list of documents indicating where the term is encountered as shown in Figure 2.2. The list of document occurrences is called a *posting*.

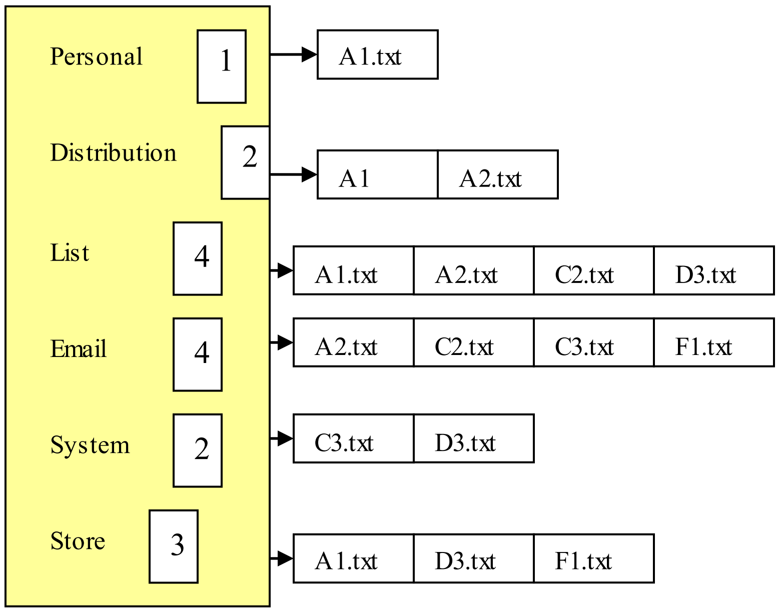


Figure 2.2 Inverted index

During the construction of the inverted index, the *document frequency* of the term is stored along with the document postings. The *document frequency* indicates how many documents in the collection contain the term.

Even though the Boolean Retrieval model does not utilize the document frequency count, there are other IR methods that use the document frequency to calculate the relevance weight for query results. One such method is the Vector Space model with Term Frequency Inverted Document Frequency (TF-IDF) weighting.

2.2.2 Vector Space TF –IDF

Unlike the Boolean Retrieval, free text queries do not use any connecting search operators such as AND, OR, or NOT. The Vector Space Model (VSM) supports document searches for these types of queries by representing the queries and documents as multi-dimensional vectors. The multi-dimensional space is constructed using all terms in the dictionary, using each term as an orthogonal measurement in the multidimensional space.

To measure the similarity between two vectors in the multi-dimensional space, the VSM uses the Euclidean cosine similarity between the vectors. The size of the vector space is equal to the size of the dictionary (each term represents a dimension). If *d* is a document, then we can denote a vector derived from the document as $\vec{V}(d)$. The vector coordinates can be represented as:

$\vec{V}(d) = \{v_1, v_2, \ldots, v_n\}$,

where $v_i = 0$ if the term ***i*** is not present in the document.

Cosine similarity in the Euclidean multi-dimensional space is estimated by the following formula:

$$\text{Sim}(\vec{V}_1, \vec{V}_2) = \frac{\vec{V}_1 \bullet \vec{V}_2}{|\vec{V}_1||\vec{V}_2|}, \qquad \textbf{2.2.1}$$

where $\vec{V}_1 \bullet \vec{V}_2$ is a dot product of two vectors. The dot product between two vectors $\vec{x}$ and $\vec{y}$ is estimated as:

$$\vec{x} \bullet \vec{y} = \sum_{i=1}^{N} x_i y_i \qquad \textbf{2.2.2}$$

The Euclidian length $|\vec{x}|$ is estimated as:

$$|\vec{x}| = \sqrt{\sum_{i=1}^{N} x^2_i} \qquad \textbf{2.2.3}$$

The effect of $\frac{\vec{V}_1}{|\vec{V}_1|}$ is to normalize $\vec{V}_1$ to a unit vector. The unit vector is obtained from a vector in N-dimensional space that has the same orientation, but its length is equal to 1.

When we consider a document as a vector in the multi-dimensional space represented by dictionary terms, we can treat the term frequency as a coordinate corresponding to the term. For example, if the documents A1.txt, C1.txt, and F1.txt consist of the following text, respectively:

> *"A1. The system shall have an address book available to store contacts. The address book shall store contacts in groups as well."*

> *"C1. The system shall support a text-based interface to compose mail, use mail addresses from an address book, and attach mail stored in folders."*

"F1. The system shall support the ability for users to create a folder to store mail. The system shall support uploading mail that is stored in folders."

The dictionary shown in Figure 2.3 presents the terms and their respective counts for documents A1, C1, and F1. The column TF stands for *term frequency.* TF is the total count of the term in the collection of documents.

	A1	C1	F1	TF
the	2	1	2	5
system	1	1	2	4
shall	2	1	2	5
have	1	0	0	1
an	1	0	0	1
address	2	2	0	1
book	2	1	0	2
available	1	0	0	1
to	1	0	0	1
store	2	0	0	2
contact	1	0	0	1
group	1	0	0	1
well	1	0	0	1
text-based	0	1	0	1
interface	0	1	0	1
compose	0	1	0	1
use	0	1	0	1
mail	0	3	1	3
attach	0	1	1	1
support	0	0	2	2
ability	0	0	1	1
user	0	0	1	1
folder	0	1	2	3
store	0	1	2	3
upload	0	0	1	1
compose	0	0	1	1

Figure 2.3 Dictionary terms with TF count for documents A1, C1, and F1.

In our example, if we treat the term frequencies as coordinates in the multi-dimensional space, the vector corresponding to the documents A1, C1, and F1 will look like this:

$\vec{V}$ (A1) = (2, 1, 2, 1, 1, 2, 2, 1, 1, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
$\vec{V}$ (C1) = (1, 1, 1, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 3, 1, 0, 0, 0, 1, 1, 0, 0)
$\vec{V}$ (F1) = (2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 2, 2, 1, 1)

A cursory look at the three vectors shows common terms among the vectors. The problem with this approach is that all terms are treated equally. The term's importance is not considered when assessing a query. For example, the terms "the," "system," and "shall" are encountered in all documents. Thus, the documents may have these terms in common, but may not be related. Frequently used words such as the articles "a," "an," and "the" and prepositions "to," "for," and "from" are removed from consideration. The removed terms belong to a *stopword* list. The *stopword* list contains all terms that should be extracted and ignored before analyzing the documents.

The *stopword* preprocessing helps to reduce the amount of noise coming from the frequent terms that do not carry much information. The importance of a term in the collection can be evaluated though the term's relative frequency. The *document frequency,* $df_{,t}$, is the total number of the terms in a document. The inverse document frequency (*idf*) of a term *t,* is estimated as follows:

$$idf_t = \log \frac{N}{df_t}, \qquad \textbf{2.2.4}$$

where *N* is the total number of documents in the collection.

The *idf* for a frequent term is low and is high for the rare term. The *tf-idf* promotes the importance of a term in a document using the composite weight of the term frequency and inverse document frequency:

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \text{ x } \text{idf}_t \qquad \textbf{2.2.5}$$

Thus, the importance of a term in a document is high for a rare term (relative to the whole collection). The importance weight is amplified by term frequency in the document.

2.3 Requirements Tracing

In the introduction, we covered the importance of requirements tracing, the RTM provides the results of the tracing activity. Requirements tracing plays an important role in the project life cycle because it enables analysts "to describe and follow the life of a requirement, in both a forward and a backward direction, through the whole system's life cycle [16]."

As the software project evolves, the project documentation is augmented by use cases. The use cases typically yield the software requirements. Sometimes, the use cases are used as design artifacts. In this case the requirements are interpreted though the use cases. When the requirements serve as a basis for layout of design elements, the testing ensures the correctness of the produced code from the source requirements. To trace the requirements forward, we trace the use cases to the requirements specification or

requirements specifications to the design elements. To trace backwards, we might trace from the use cases back to the requirements or from design elements to the requirements. Figure 2.4 and
Figure 2.5 show the forward and backward tracing, respectively.

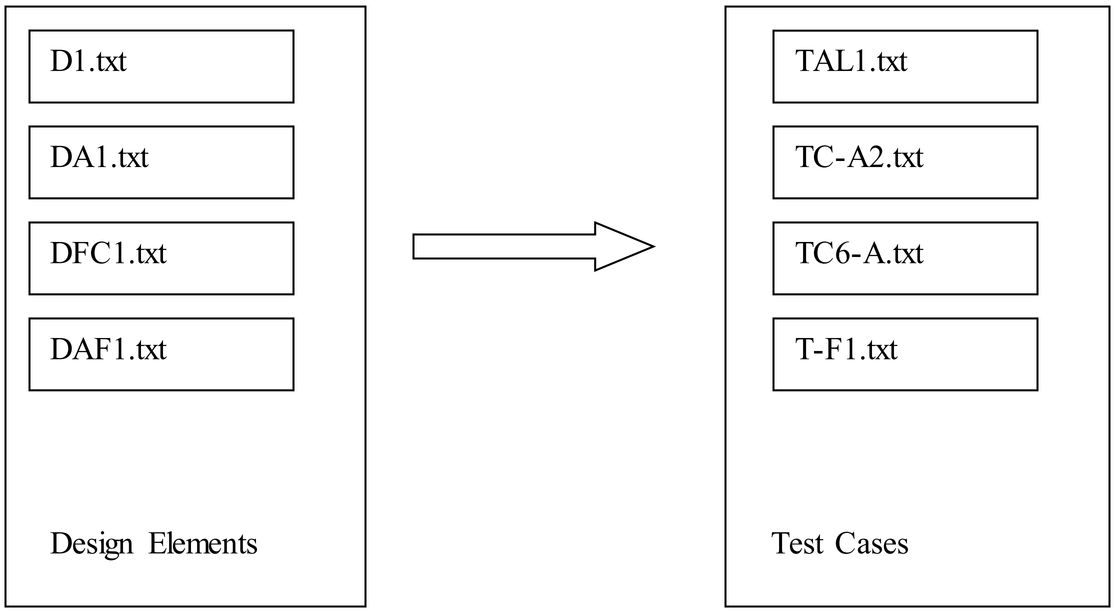


Figure 2.4 Forward tracing from design elements to test cases.

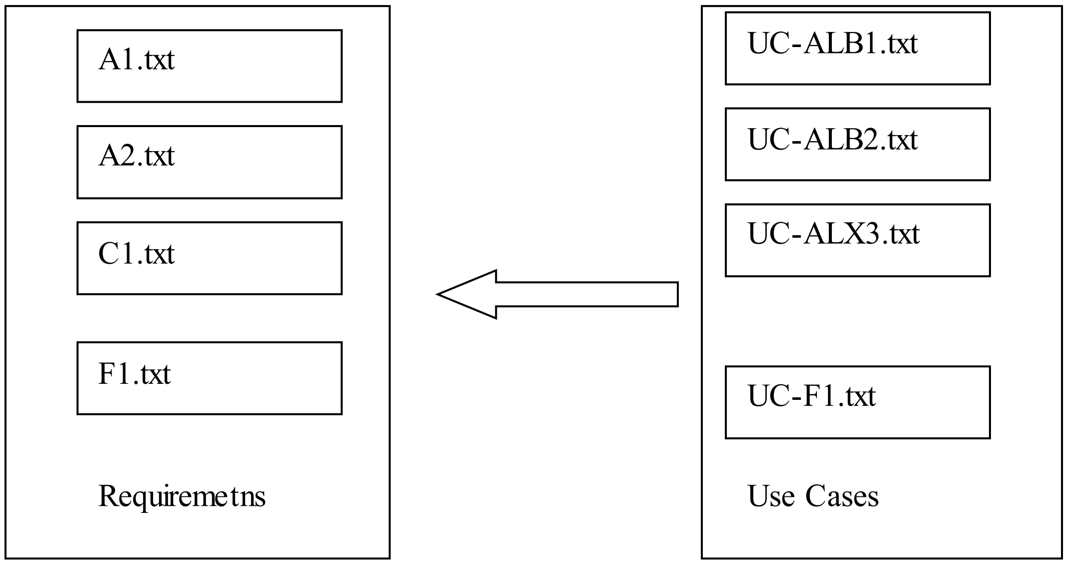


Figure 2.5 Backward tracing from use cases to requirements.

As the result of tracing, we establish candidate links between two collections of documents. A candidate link is a logical connection between two documents; if a document TAL1.txt (Figure 2.6) addresses ideas mentioned in document DA1.txt, we say there is a link between DA1.txt and TAL1.txt. For example, if in the forward tracing from design elements to test cases (Figure 2.4) there is no link coming from element DAF1.txt, we can immediately assess that our test cases do not fully address all of the design elements.

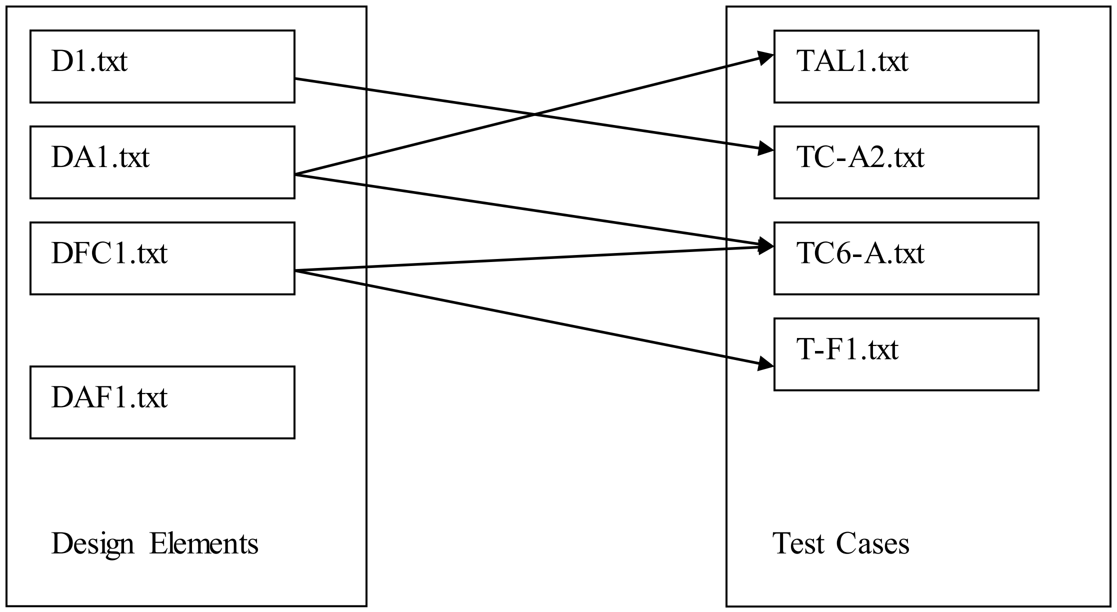


Figure 2.6 Candidate links. One design element is missing a link to test cases.

The TF-IDF method creates a list of candidate links between the two document collections with the "weight" assigned to the links for each suggested pair of documents. The weight represents a "similarity" between the documents. The higher the weight the "closer" the documents are to each other. The closeness is evaluated by having similar terms. Also, the value of the "weight" is used as a filter. Links below a certain threshold are cut off from consideration. A low value of the weight implies that the documents share only a few terms; a higher value of the weight indicates that documents share many terms. By lowering the threshold, we create a large list of candidate links. The documents in such links may share just a few terms, but have very little meaning in common. The low threshold value pulls many document pairs for consideration; hence we may obtain a higher recall, but the precision of such candidate links will suffer: only a small fraction of the document pairs can be identified as true links. With a higher threshold value, we obtain more precise results, but not all possible true links are identified. Thus the results of the TF-IDF method may range from a very low recall and high precision to a high recall and low precision.

Another shortcoming of the TF-IDF method is that it treats a textual document as a bag of words. The relative order of the terms is not important for the TF-IDF method. We propose a method that identifies common segments between the documents; thus shifting the focus onto treating documents as collections of phrases. One of the objectives of the proposed research is to discover the candidate links between two sets of software requirement documents automatically by using swarm intelligence. Another objective is to provide candidate links that do not have either high recall and low precision or low recall and high precision. We want to have our recall and precision values come a step closer to the ideal location in the precision recall graph – top right corner, i.e., high recall and high precision.

2.4 Swarm Intelligence

Insects such as bees and ants, small and simple individually, can accomplish tremendous tasks in a collective effort. Swarm intelligence describes computational algorithms that inspire computer scientists by the fact that the insects' achievements and actions are all accomplished through local peer-to-peer interactions. A number of scientists have studied the behavior of ants in foraging for food. Jean-Louis Deneubourg described the self-organizing behavior of ant colonies, where ants used pheromone communication [17]. The idea of using pheromone trails as a method of communicating through the environment is at the heart of the ant colony optimization (ACO) algorithm [11]. This algorithm has been used in a number of computer science applications, such as the traveling salesperson problem, and has applicability to requirements engineering problems.

Consider a graph G = (V, E), where V is a set of vertices and E is a matrix representing connections between the vertices. For each edge, (***i, j***), between the nodes ***i*** and ***j*** in the graph, we assign a pheromone value $\boldsymbol{\tau_{ij}}$. In the initial step, the ACO will assign each edge in the graph a zero pheromone value, $\boldsymbol{\tau_{ij}}(0)$. Also, a group of ants k = 1,...,n is positioned at the source node.

For every iteration, each ant builds a path to the destination node. Also, at every node, each ant decides the next link to take. If ant ***k*** is at node ***i***, the probability $p_{ij}^{k}(\boldsymbol{t})$ of selecting the next node $j \in N_i^k$, which belongs to a set of nodes adjacent to ***i*** [11], is:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)}{\sum_{j \in N_i^k} \tau_{ij}^{a}(t)} & if \quad j \in \mathrm{N}_{\mathrm{i}}^{\mathrm{k}} \\ 0 & if \quad j \notin \mathrm{N}_{\mathrm{i}}^{\mathrm{k}} \end{cases} \qquad \textbf{2.4.1}$$

where N_i^k is the set of nodes accessible for agent ***k*** from the node ***i***. If node ***j*** is not accessible for ant ***k*** from the node ***i***, the probability $p_{ij}^k(t)=0$. In the formula above, α is a parameter which amplifies the attractiveness of the pheromone trail. Large values of α attribute importance to pheromone.

2.5 Reinforcement Learning

The reinforcement learning (RL) model is a machine learning technique dealing with the actions an agent needs to take in order to maximize collected rewards as a result of these actions. The agents in RL learn the actions to maximize the long term, discounted, expected reward by interacting with the environment.

The RL model can be presented as ($\boldsymbol{S}$, $\boldsymbol{A}$, $\{P_{sa}\}$, γ, R), where

- $\boldsymbol{S}$ is a set of environment states
- $\boldsymbol{A}$ are actions available to agents
- P_{sa} is a state transition distribution, i.e., the probability of transitioning into next state s' by taking an action ***a*** while being at state *s* and $\sum_{s'} P_{sa}(s') = 1$
- γ is a discount factor
- R is a reward function, R: $\boldsymbol{S} \times \boldsymbol{A} \times \boldsymbol{S} \rightarrow \mathbb{R}$, $\mathbb{R}$ is domain of real numbers. Reward is a scalar value associated with transitioning into states.

In reinforcement learning (RL), agents probe the environment though a discrete sequence of steps and actions over time *t*, where t = 0, 1, 2, 3 etc. At each step *t*, the agent evaluates the state $s_t \in \boldsymbol{S}$, where $\boldsymbol{S}$ is a set of all possible states. Based on the state s_t, the agent selects an action $a_t \in \boldsymbol{A}(s_t)$, where $\boldsymbol{A}$ is a set of possible actions available to the agent in state s_t. As the result of the action taken at the moment ***t***, i.e. t-th time step, the agent gains reward r_{t+1}, and moves to the state s_{t+1} [12]. Figure 2.7 displays the interaction between the agent and environment [12].

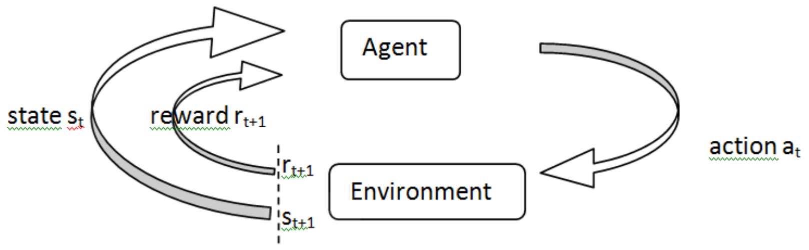


Figure 2.7 The interaction of the agent and the environment in reinforcement learning.

As shown in, Figure 2.7 the agent receives the state $\boldsymbol{s_t}$ as an input and produces action a_t as an output. The mapping of the states into actions is determined by a policy π_t. Since each state s_t can present a set of possible actions $\boldsymbol{A}(s_t)$, the policy π_t denotes the probabilities of selecting one of the possible actions determined by the state s_t. The mapping of states to actions is represented as $\pi_t(s,a)$, the probability of selecting action $a=a_t$, when state $s=s_t$. The agent's goal is to maximize the total rewards acquired in the long run by choosing actions according to the distribution specified by π.

The reward the agent collects depends upon the actions it takes and their probabilistic effects. To estimate the desirability of a state, some RL algorithms use the notion of value function. Formally, the value function is represented as:

$$V^{\pi}(\boldsymbol{s}) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s\}, \qquad (2.5.1)$$

where R_t is a function of the reward sequence [12]. The value $E_{\pi}\{\}$ is the expected reward value given to the agent that follows the policy π. The discount coefficient $\gamma \in [0, 1]$ signifies preference for the immediate or future rewards. If γ approaches 0, the immediate rewards are assigned the most value. When γ approaches 1, the future rewards and immediate rewards are valued more nearly equally.
Bellman's equation [12] provides another way to express the value of a state s:

$$V^*(s) = \sum_a \pi(s,a) \sum_{s`} P_{ss`}^a [R_{ss`}^a + \gamma V^{\pi}(s`)] \qquad (2.5.2)$$

where, $P_{ss`}^a$, is the probability of reaching state $s`$ from s if action a is taken; $R_{ss`}^a$ is the reward associated with reaching state $s`$ from s by taking action a.

A policy that maximizes expected return for all states is called an optimal policy and is denoted π^*. Formally, $\pi^* \geq \pi^`$, if and only if, $V^{\pi^*}(\boldsymbol{s}) \geq V^{\pi`}(\boldsymbol{s})$ for all $\boldsymbol{s} \in \boldsymbol{S}$. Alternatively, we can define V^* as:

$$V^*(s) = max_{\pi} V^{\pi}(s) \qquad (2.5.3)$$

There exist at least one policy and its expected return is better than or equal to that of π^* for all the states, Bellman's theorem [12]. If there are several policies, i.e., more than one policy, that allow agents to reach maximal expected return, we still denote these policies as π^*.

One way to determine an optimal policy is to use the value iteration algorithm [12]. The value iteration algorithm is an iterative backup operation. The algorithm combines an immediate policy improvement for the current state and the values of states reachable from the current state in the following form:

$$V_{k+1}(s) = max_a \textstyle\sum_{s`} P_{ss`}^a [R_{ss`}^a + \gamma V_k(s`)] \qquad (2.5.4)$$

where $P_{ss`}^a$ and $R_{ss`}^a$ bear the same meaning as defined in equation (2.5.2). The value of state s is maximized across all actions a available at s. The pseudo code for the value iteration algorithm [12] is listed in Fig. 2.8.

```
Initialize V(s) =0, for all s ∈ S
Repeat
    $\Delta \leftarrow 0$
        For each s ∈ S
        V ← $V(s)$
        V(s) ← $max_a \sum_{s`} P_{ss`} [R_{ss`} + \gamma V(s`)]$
                $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
        Until $\Delta < \varepsilon$ ($\varepsilon$ a small positive number)
Output a deterministic policy, $\pi$, such that
        $\pi(s) = argmax_a \sum_{s`} P^a_{ss`} [R^a_{ss`} + \gamma V(s`)]$
```

Figure 2.8 Value Iteration reinforcement learning pseudo code.

To apply the reinforcement learning approach to the traceability problem, we constructed a search space, i.e., an environment. After the states, actions, and rewards are established, the value iteration algorithm is executed. The value iteration algorithm outputs actions for each state. The actions established for the states determine the navigation heuristics for the agents.

The idea of building a path from the source node to the destination node resonates well with the activity of establishing candidate links in the requirements traceability process.

3 Related Work

3.1 Requirements Traceability

In this section, we address traceability link generation, swarm techniques, and reinforcement learning. As mentioned earlier, candidate link generation is concerned with retrieving the relevant elements from a given textual artifact pair. The candidate link list is reviewed by an analyst to determine if each link is a true relevant link or not.

In 1994, Gotel and Finkelstein identified a lack of automatic tools to conduct requirements traceability activities [18]. Since then, much work has been done to remedy this problem by applying information retrieval techniques to the candidate link generation problem. Antoniol et al. [19] used the vector space model (VSM) and a probabilistic model to recover traceability from source code modules to man pages and functional requirements. In a probabilistic model the documents are ranked based on the probability of being relevant to a query. The authors used a Bayesian classifier "to score the sequence of mnemonics extracted from each source code components against the models. [19] " With VSM, they achieved the highest recall (100%) for the Albergate dataset by setting the threshold to 10% of the highest similarity measure. However, they only achieved a precision of 11.98%.

In the VSM and probabilistic models, links are established between documents using common weighted terms. Specifically, terms are assigned weights based on term frequency and term count in the document collection. The swarm technique differs in that links between documents are established by discovering and promoting the importance of common phrases in the inspected documents. The reinforcement learning method discovers candidate links by optimizing the search heuristics (Chapter 5).

Another perspective on requirements traceability is goal-centric traceability, as demonstrated by Huang et al. [20]. Huang proposed a model to establish links among subsets of artifacts that an analyst considered as covering a certain objective. For example, the artifacts may describe the security features of a system. The authors demonstrated how goal-centric traceability keeps track of the traces between goals and documents. The model provided change impact analysis through automated traceability. In our approach, we use the swarm technique instead of the goal-centric traceability model. This was done since the swarm technique does not require an initial classification of the documents as related to a particular goal or objective. This was also done because Huang's approach potentially increases the possibility of creating too many traces between documents in the subset. To manage this possibility, we looked at a scoped approach to traceability management as described by Lago et al. [21]. The method described by Lago et al. takes on the traceability task by focusing on selected activities, rather than by using an automatic "trace all" approach. Similar to the goal-centric traceability in Huang's work, Lago's method requires an initial selection of artifacts related to the target activities. Unlike our swarm technique, the scoped approach traces only selected items.

Panis [22] states that 26 engineers at Teradyne expressed their preference to see the traced content of a requirement rather than see a simple identifier. He found that engineers place the most value on traceability information when they are creating documents.

Further, according to Egyed et al. [23], an analyst takes one to two minutes, on average, to manually establish traces from code to requirements. They also found that recovery of method traces takes 3 - 6 times longer than recovering class traces (also manually). The swarm technique provides a method to generate trace links in an automatic fashion. While this time reduction is significant, there are still additional issues to deal with in order to improve the quality of the candidate links generated. Specifically, we had to select a context to establish the trace links. To do this, we first looked at work done by DeLucia et al. [24].

DeLucia et al. used a traceability recovery tool based on Latent Semantic Indexing (LSI). By introducing categorization, the DeLucia et al. reached a precision of 25% with 90% recall. Without categorization and at the same 90% level of recall, the precision reached only 17%. Marcus and Maletic [25] applied the LSI technique to the same Albergate dataset used by Antoniol et al. The LSI technique identifies the patterns and concepts contained in a collection of text by establishing associations among terms occurring in similar contexts [24]. Marcus and Maletic achieved a precision of 16.38% at 100% recall using this technique.

In effect, the LSI technique uses a document as the context. The swarm technique differs by establishing candidate links between two collections of documents based on similar terms occurring in the neighborhood of common terms; the neighborhood of a linking term acts as a "context."

Swarm techniques and the RL method further expand neighborhood terms by using a thesaurus. This approach discovers links through synonymous terms. The value of using a thesaurus was validated by Hayes, Dekhtyar, and Osborne [26] when they applied VSM with a thesaurus to a dataset and compared this method to manual tracing and to a proprietary tool. They achieved a higher precision using manual tracing compared to the proprietary tool: 46% vs. 38.8%. Also the manual tracing scored better in terms of precision than the VSM + thesaurus method: 46% vs. 40.7%. At the same time, VSM + thesaurus method outperformed the other two approaches in terms of recall, (85.4% compared to 43.9% for manual and 63.4% for the proprietary tool). Thus, the use of the thesaurus expanded the term base. As a result, additional links were discovered between textual chunks expressing similar ideas and phrases using different terms.

By using phrasing as a way to improve the precision of automated IR traces, Zou et al. [27] obtained improvements of almost 20% for one dataset when examining the top 5% of the returned candidate links. Their work focuses on establishing "similar" areas between documents. The similar areas are established through shared common terms in the neighborhood of the linking terms. In this regard, "similar neighborhoods" in our work resemble the phrasing technique used by Zou et al. [27].

Phrasing is similar to the idea of 'lexical affinities' as expressed by Maarek et al. [28] and by Niu and Easterbrook [29]. Their research considered two word units within a single sentence. The 'lexical affinities' limit the neighborhood window to a maximum of five terms apart. In other words, terms occurring relatively close to each other in two documents form related phrases. The related phrases in two documents can be viewed as common segments, creating a logical link between the documents.

This idea of small common segments between two documents appears to be a valid starting point for investigating the swarm behavior on the traceability problem. Unlike the 'lexical affinities' method, the swarm technique considers terms that may cross the boundaries of a sentence. Furthermore, the swarm technique does not require any knowledge about the part of speech for a given term, whereas the 'lexical affinities' method deals with two-word phrases: noun verb pairs.

Zisman and Spanoudakis [30] examined ways to generate traceability links by applying rules to artifacts that had been tagged with the parts of speech. In their work, the authors established four types of traceability relationships based on the grammatical tagging of the textual artifacts. The proposed swarm technique does not perform such fine-grained classification of traceability links. The swarm agents simply identify the links based on the common vocabulary base with the purpose of simplifying the algorithm and the search heuristics.

The effect of the vocabulary base on traceability accuracy (using both artifacts versus just the low-level artifact to build the vocabulary) was studied by Sundaramet et al. [31]; in the study, they found support for using only the low-level artifact.

In general, the above techniques have been able to achieve excellent recall [26] [31] [32], but often at the expense of precision that is only borderline acceptable at best. The work described in this dissertation differs in that it uses a "greedy algorithm" approach to generate the candidate link lists with the goal of increasing precision.

A greedy algorithm will potentially increase precision because it selects the optimal link to follow, which is optimal from the agent's point of view. This algorithm also does not require tagging parts of speech or phrasing, simplifying the process of building links and reducing the amount of time required to conduct searches.

To evaluate the performance of this method, we use traditional IR measurements: recall, precision, F harmonics, mean average precision (MAP), as well as several other secondary measurements.

Zou et al. [27] use average precision (AP) to measure the internal quality of candidate link lists. AP looks at a number of recall levels such as 10% recall, 20% recall, etc., and averages the precision changes of each, thus returning only one value. For similar reasons, we prefer mean average precision (MAP) to AP. It has the advantage of returning a single value, but it does not require one to set recall levels, and it does not require interpolation. The swarm method also uses secondary measurement, such as MAP, to evaluate the performance of the algorithm.

3.2 PSO and ACO Techniques

There are other researchers who have applied the particle swarm optimization (PSO) algorithm to analyze textual documents. PSO is a direct method that searches for an optimal solution in a search space. The main characteristic of the PSO algorithm is that each member of the swarm adjusts its behavior based on the information obtained from its neighbors in the search space. The swarm agents are modeled to have a specific position and velocity in a search space. The agents iteratively evaluate a fitness function where the agents' position and velocity are used as input parameters. The agents operate on the premise of their own "best" position and the swarm's and the neighbors' "best" position, where "best" implies a point in the search space where the fitness function has reached some optimal value [33].

To test this approach, Merwe and Engelbrecht applied data clustering using PSO on six different classification problems [34]. Four hundred vectors were randomly created in a two-dimensional space from the Wisconsin breast cancer database, with the objective of classifying the data as representing benign or malignant tumors. Another PSO clustering work was carried out by Cui, Potok, and Palathingal on textual documents [35].

Also, PSO was used to rank the results of IR methods. Diaz-Aviles and Nejdl proposed a swarm ranking method for IR using the particle swarm optimization on the benchmark database LETOR. The swarm first undertook a learning phase to rank IR results and attempted to reduce over-fitting [36].

In the above work, the researchers modeled the search space as a hyperspace of words or terms. The fitness function was, in some form or fashion, a Euclidian distance in the vector space of terms between the multidimensional points. The vector space model treats each term as a dimension of the multidimensional space. For example, for data clustering, Merwe and Engelbrecht [34] used a variation of a distance vector to randomly seed centroid vectors, e.g., to seed some starting points in the search space. When compared to the PSO method described above, a drawback of a VSM approach discussed earlier becomes apparent. Namely, it treats terms as separate dimensions of the search space. Each new term increases the vector space's dimension size and hence increases the complexity and number of necessary computations.

To overcome this weakness in the VSM approach, Diaz-Aviles and Nejdl [36] used training (learning to rank IR results) for a collection of queries and the resulting retrieved documents. They used a training set, as well as a validation set, to attempt to reduce over-fitting. They proposed the method of *SwarmRanking* to optimize the combination of the content and links. This method used mean average precision (MAP) as the fitness function to evaluate the results. They found that the approach significantly outperformed standard approaches.

Our method is similar in that we use a swarm algorithm to rank retrieved low-level requirement elements that may be relevant to a given high-level requirement. Our approach differs in that we do not take a semi- or supervised learning approach, and thus do not require a training set.

Aghdam, Ghasem-Aghaee, and Basiri used ACO to select text features [37]. Azzag and Guinot [38] used ant colony optimization (ACO) to cluster data in trees. In their work, ([37],[38]), the authors mentioned that due to "the probabilistic behavior of artificial ants they can produce quality results without any prior knowledge of data structures." In our work, randomness is considered a positive factor as well, since it allows ants to explore the search space of the document collection.

Further, in the ACO algorithm, the agents do not have any prior knowledge of the text features. The proposed swarm method also does not involve supervised learning, and the agents do not have a predetermined knowledge of the space they traverse.

In typical ACO, the pheromone deposited by the ants evaporates over time. The evaporation enables a dynamic behavior to take place in the search process. A path with more pheromone deposits becomes more attractive to the ants. The more ants that traverse the path, the more attractive the path becomes.

The proposed swarm method uses pheromone deposits on the links and terms to influence the path selection behavior of a swarm agent. The pheromone deposits on the links and terms influence the path selection behavior of a swarm agent. Note, that there is no predetermined knowledge of the traversed space. The search and discover phase of the algorithm is like "random roulette" and it is greedy. The term and document frequencies of the text collection are used as guiding heuristics for the agent's behavior. Technically, the algorithm still resembles an ant colony, but it is not as intelligent and cooperative as ACO. In our approach, the swarm agents are given freedom to operate on their own, determining the search path based on the environment, i.e., term frequency, weight, etc.

The next logical step from the pheromone swarm technique is to "learn" the search space environment. The RL method maps out the search space and "learns" the environment. As a result of this learning, our RL method equips the agents with the search space traversal heuristics to discover candidate links (Chapter 5).

Abraham and Ramos [39] explored ACO clustering with linear genetic programming. Their model of clustering web documents was based on the behavior of ants forming cemetery clusters (deposits of dead ant bodies) within the colony's territory. From the computational point of view, the main factors that influenced the behavior of artificial ants are the number of objects in the neighborhood and their similarities. The proposed swarm technique also builds the behavior of an artificial ant based on the similarities between neighborhoods in the documents.

Li and Lam used ant-like agents to generate test threads from unified model language (UML) diagrams [40]. The authors used three-dimensional UML diagrams as directed graphs to provide a search space for artificial ants. The swarm technique also creates a three-layered graph as a search space for the ants.

Another interesting aspect of Li and Lam's work is a limited "energy" supply for the ants. This way the ants can avoid looping indefinitely while traversing the graph. In our work, we limit the length of a path that the swarm agent traverses; the length of a path is equal to two – from a high level document to a low level though a common term. In other words, the ant can only move from a high level document to a low level document through a common term before it finishes its journey. The three layer topology of the search space implies the agent cannot travel more than 3 hops. One extra hop is permitted to jump to a synonym if it is chosen.

3.3 Machine Learning Techniques

In the past several years, the interest in machine learning techniques applied to requirements engineering has been growing. Machine learning techniques can help establish some knowledge or rules from requirements engineering artifacts [41],[42].

Background knowledge from a set of examples of the system description and system's properties is derived by a method proposed by d'Avila-Garces et al. [42]. The method uses a machine learning technique, *inductive learning (IL)*[1]. From the set of positive and negative examples, the inductive learning technique finds hypotheses, i.e., definitions of domain concepts. The authors use the technique to analyze and revise specifications if any system property violations are discovered. Our work is different because we use reinforcement learning. Our method does not use positive or negative examples to train the system; the discovery of candidate links is executed autonomously.

Another example of inductive learning can be found in work by Spanoudakis, d'Avila-Garces, and Zisman. They use a machine learning technique to generate requirements traceability relations [41]. The traceability rules are established between two sets of documents: textual requirement statements and object models. Based on user feedback on the undetected traceability relations, the existing traceability rules are transformed to match the indicated traceability relations. To implement the method, the authors utilize *abduction (AL)*[2] *and induction learning (IL)* techniques and the *part of speech tagging* method. In our work, we also establish logical links between two sets of documents, but our method does not use the *part of speech tagging* and we used RL not AL or IL.

In addition to extracting knowledge from the documents, the machine learning techniques can be used in recommender systems. Seo and Zhang describe a reinforcement learning (RL) technique for the Web based personalized filtering system [45]. The work by Seo and Zhang presents an interest for our work because the personalized filtering system gives a boost to selecting relevant documents. The personalized information filtering method learns from the profiles of individual users and their responses to presented documents. Our method is similar to the work by Seo and Zhang using greedy term

[1] Induction Learning evaluates and generates conclusion based on some examples, i.e. premises. In the inductive logical argument, the premises support the conclusion to some degree of certainty [43].
[2] In abduction learning an explanatory hypothesis is adopted to account for all the facts or some of them [44]

selection through the RL technique to locate relevant documents. However, our method differs because it does not use any form of feedback.

Cleland-Huang, Czauderna, Gibiec, and Emenecker present two machine learning approaches to improve traces between regulatory codes and product requirements [46]. The terms in requirements are assigned probabilistic scores with respect to a regulatory code. To classify the requirements, the manually created traces were used for cross-training and testing. The second approach, web based, was used to retrieve indicator terms from the Internet for a specific regulatory code. Only in this second case, the machine learning classification took place based on the web-mined documents.

Asuncion H, Asunsion A, and Taylor [47] use the *latent Dirichlet allocation (LDA)* machine learning technique to assign topics to traceability artifacts: requirements and design documents. For this technique, the initial input for the LDA method consists of the documents and number of topics to assign. The authors suggest that topic modeling provides semantic information about traceability artifacts.

Establishing links between the documents can also be based on related textual segments. Hatziavasilloglu, Klavans, and Eskin present the composite similarity metric to measure the semantic distance between a pair of small textual segments [48]. The authors use a machine learning approach to select the potential optimal features between documents. The potential matches are established through word co-occurrence. This approach resonates well with our technique. We also use common linking terms and the terms located close to a lining term in the text. The composite similarity performs the matching through the noun phrases, synonyms, the semantic class of verb (verb implying similar actions), and common proper nouns.

In our work, we also use synonyms to conduct matching. The composite similarity uses the relative order of terms in evaluating the matching. The authors use and train a classifier on manually marked pairs of units. This aspect of the matching used by the authors echoes with our work. In chapter 5, we describe how the textual segments are probed for similarities. The relative order of the terms is also considered for the similarity evaluation.

The main focus of our work is to establish the logical links between the textual documents by using common textual segments. The work presented by Menczer and Belew lists many of features similar to our work [49]:

1. The authors describe how autonomous agents make decisions to automate the web document search and discovery process. The agents in the work of Mencer and Belew have a heuristic behavior by which the agents select links to follow. In our work, the autonomous agents also discover a heuristic to traverse the search space, i.e., select a link to follow.
2. An agent in Menczer and Belew's work senses the "current neighborhood" by analyzing the text where the agent is situated. This matching feature is similar to the concept of term neighborhood that we use (Chapters 4 and 5).

3. The agents in Menczer and Belew's work use reinforcement learning (RL) to modify the behavior to follow the "best link" possible. In our work, we use the RL technique to enable agent to traverse the search space and establish the candidate links between the documents.

Even with so many similarities between the agents in Menczer and Belew's work and ours, there exist three notable differences:

1. The links between documents in the work of Menczer and Belew are web links. In our work, the links between documents are established via common terms (Chapter 4 and 5).
2. The agents of Menczer and Belew receive user feedback on the suggested links; in our work the agents do not receive feedback.
3. The agents in Menczer and Belew's work are created with "initial reservoir of 'energy' [49]." The agents in our research do not utilize any energy measurements for the search space traversal.

To sum up the features of the related work, we can state the following:

- It has been proven useful to link documents by treating them as a collection of phrases, not a bag of words [48].
- Small textual segments and the similarity between them can be evaluated based on semantic distance [49].
- The textual segments of linking terms, i.e. neighborhoods of the linking terms, provide useful location data of the compared textual segments [48] [49].
- The machine learning approach in general, and reinforcement learning in particular, proved to be useful computational agents to modify and select an optimal search space behavior [45] [49].

In Chapter 5, we describe further how we probe textual segments for similarities and order terms considered for similarity evaluation.

4 Research Approach

To trace high level textual elements (from a requirements document for example) to low level textual elements (from a design document), we use swarm agents that traverse the collection of all documents and the vocabulary shared by all documents. The main idea of the proposed method is based on constructing a search space traversable by software ants. The search space is composed of documents on both levels, high and low, and common terms.

To use an analogy of the overall structure, documents with links to common terms can be visualized as a "tree trunk" of common terms at the core of the search space. Documents can be viewed as leaves on the tree's branches (Figure 4.1).

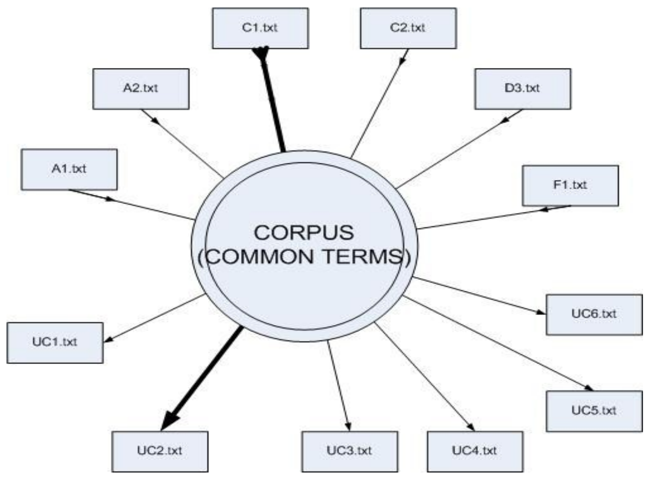


Figure 4.1 Vocabulary with documents compose the search space.

The vocabulary, i.e., the collection of terms from all documents, connects all documents in the search space. The swarm agents can travel from high level documents to the vocabulary using a positional index in the vocabulary.

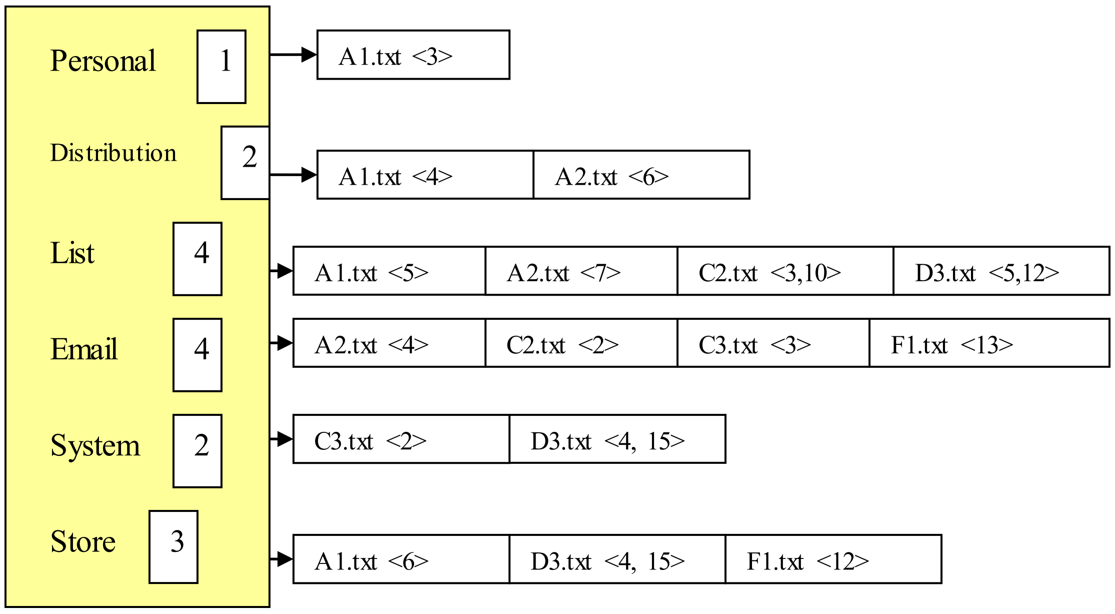


Figure 4.2 Positional index

The positional index stores such information as document name and positions (within document) for each term in the vocabulary (Figure 4.2). Thus, using the positional indexes, the swarm agents can reach every term in a document.
As shown in Figure 4.3, it is possible to reach terms 'personal,' 'distribution,' 'list,' and 'store' from high level document A1.txt.

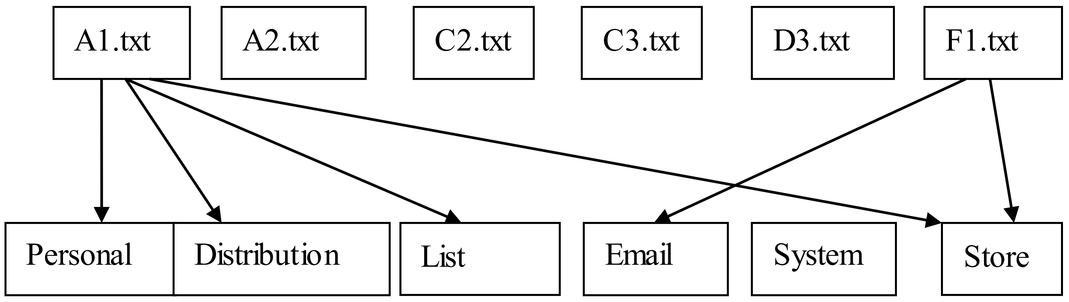


Figure 4.3 Document to terms links inferred from the positional index.

To continue the journey further within the search space (Figure 4.5), the swarm agents reach into low level documents from the vocabulary level via *the inverted index*. The inverted index is built during a preprocessing step performed during the construction of the vocabulary.

First, the documents are parsed, and then undergo term stemming. Words are reduced to their stem such as 'comput-' for 'computer' and 'computing.' Also, stop words such as 'the' and 'of' are removed. Term frequencies for each term in a document are also calculated. The TF-IDF weight is calculated using formula 2.2.5 listed in sec 2.2.2.

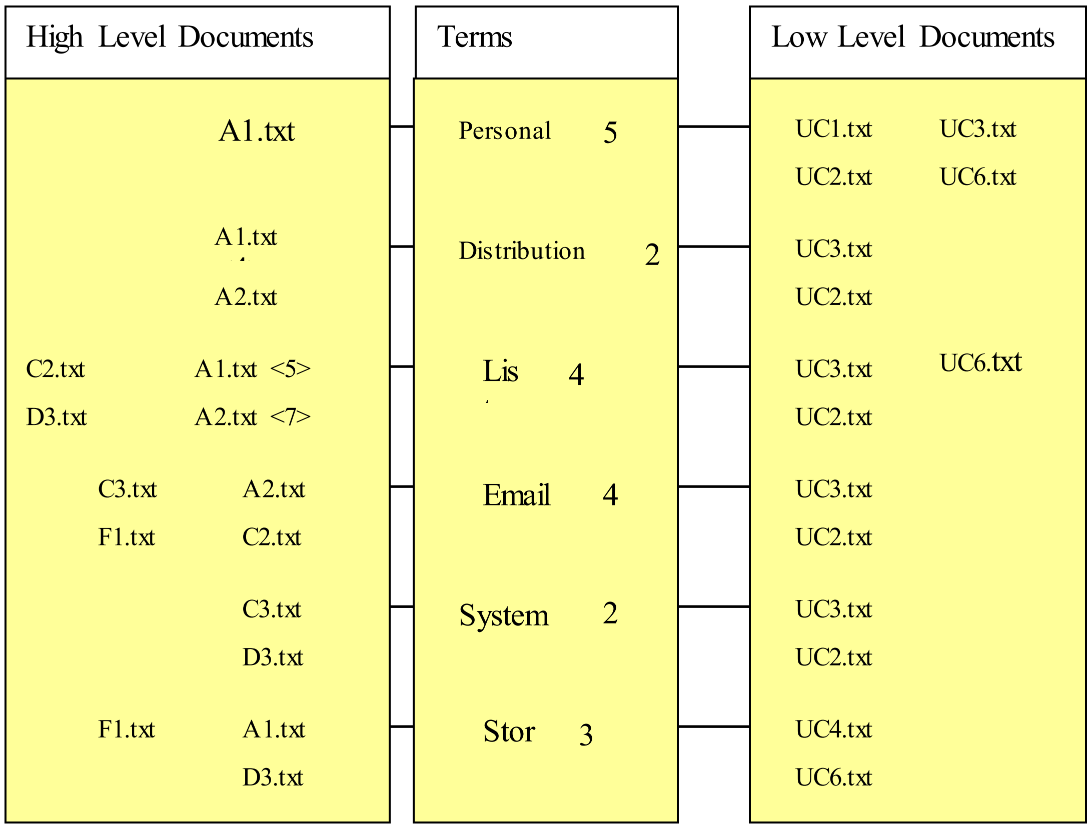


Figure 4.4. Vocabulary with documents compose the search space.

The constructed inverted index indicates not only the textual element associated with a given term, but also the type of the element: high or low. This is necessary for the search processes. The type of element helps the swarm agents to navigate the search space (Figure 4.4). In our model, we direct the swarm agents to go from high level to low level documents.

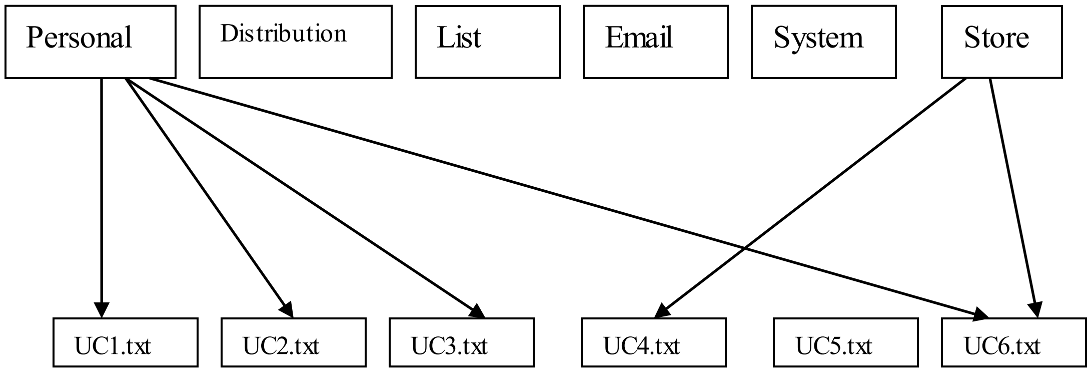


Figure 4.5. Indirect index. Links from terms to documents containing the terms.

The navigation of the search space by the swarm agents is described by the simple swarm algorithm.

4.1 Simple Swarm

The simple swarm technique is described as follows:

```
SIMPLE SWARM TRACELINKS (H, L)
        // Input High and Low level documents H and L
        // Output list of agent count (h,l,n) - from h in l, where n is the count
  1.  For each document h in high level collection H
  2.      // T = {$t_1$,....,$t_n$} sorted terms in doc h
  3.      T ← h.Terms.sortBy(TFIDF)
  4.      For each agent s in swarm S
  5.          i ← Random[1.10]
  6.          t ← T[i]
  7.          // E is a record in the inverted index listing occurrences of
  8.          // term t in low level documents
  9.          E ← Vocabulary[t].LinksToLowLevelDocuments
  10.         E.sortBy(t.TermFrequency)
  11.         j ← Random[1.10]
  12.         e ← E[j]
  13.         e.countAdd(h,l)
  14.     EndFor
  15. EndFor
  16. For each document h in high level collection H
  17.     For each document l in low level collection L
  18.         list agent count from h in l
  19.     EndFor
  20. EndFor
```

Listing 4.1. Pseudo code for simple swarm.

When all agents reach the low-level elements, we can then establish candidate links. To establish and quantify candidate links, we need to count the number of agents that made it to the low-level elements, grouping them by their origin.

The origin is the name of the high-level element from where the agents started their *journey*. If a low-level element B has at least one agent that came from element A, we consider this count of at least one (1) as a potential candidate link between A and B. The candidate links for each high-level element are ordered by the count of the agents at the low-level elements. Agent counts are normalized to a value between 0 and 1, with the top low-level link for each high-level element having a value of 1. Links are filtered out at fixed threshold intervals to calculate recall and precision values at each cutoff threshold.

Figure 4.6 depicts the application of the algorithm to a small example (select terms were chosen for illustrative purposes). Assume that we have high-level requirements Req1.txt and Req2.txt and use cases UC5.txt and UC8.txt:

Req1.txt: "The system shall support personal distribution lists."

Req2.txt: "The system shall be able to add a contact to the address list."
UC5.txt: "User edits personal distribution list by adding new contact."
UC8.txt: "List email contacts."

After pre-processing these elements, we determine that Req1.txt has the terms *personal, distribution,* and *list* and that Req2.txt has the terms *list, address,* and *contact*. Similarly, we know that the low-level element UC5 has the terms *edit*, *personal*, *distribution*, and *list* and that UC8 has the terms *contact*, *list*, and *email*. The inverted dictionary for the collection of all documents is used as the common vocabulary. The terms in the common vocabulary contain links pointing to the documents in which the terms are encountered. The vocabulary term links contain the term frequency count TF and a tag indicating if it is a high or low-level element.

As the algorithm starts, a group of agents is assigned to high level document. The number of agents in the group is greater or equal to the number low-level documents. In the high-level element, the terms are then ordered by the TF-IDF weight of each term in the document. The agent randomly selects a term, for example, the term *personal*. The agent then "positions" itself in the common vocabulary at the term *personal*. The agent inspects the links from the term *personal* to low-level elements. These links are sorted in descending order by term frequency. The agent randomly picks the next link to follow from the top ten or less candidate links. On the last leg of its journey, the agent arrives at the low-level element. At the end of this loop, the resulting composition of agents will have all agents from all high-level documents located at the low-level elements.

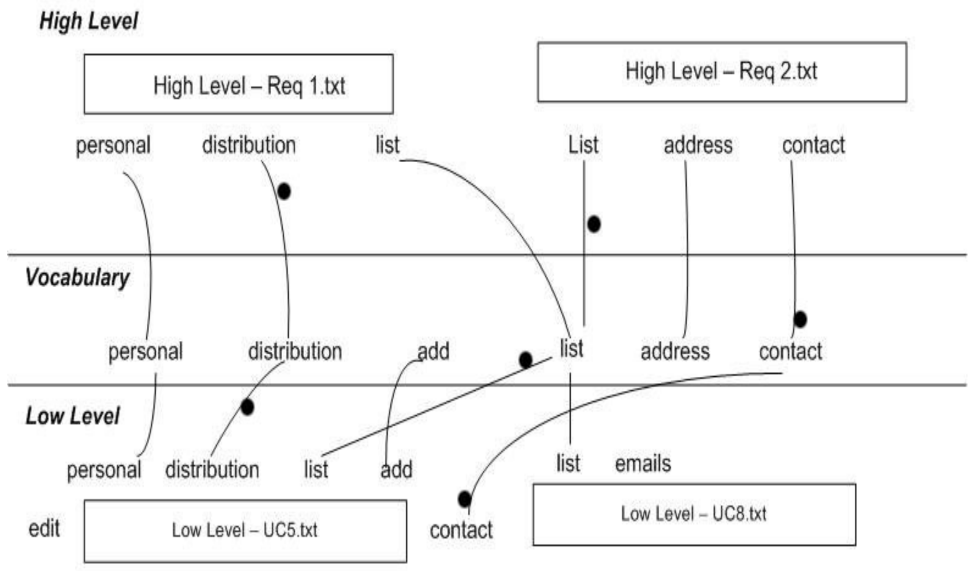


Figure 4.6 Agents tracing links from high-level to low-level elements via vocabulary

An important part of the swarm method algorithm that helps to refine the search results is the threshold filter. For the swarm method, candidate link lists are generated after applying a threshold filter varying from 0.1 to 0.9. The threshold indicates a percentage above which links are considered to be part of the candidate link list. For example, assume that one hundred agents starting from element Req1.txt traverse to documents UC1.txt, UC2.txt, UC3,txt, and UC4.txt, of which 50, 35, 10, 5 agents reach UC1.txt, UC2.txt, UC3.txt, and UC4.txt, respectively. If 0.7 is selected as the threshold, then only UC1.txt and UC2.txt are selected for the candidate link list (normalized values are 1, 0.7, 0.2, and 0.1, respectively).

The simple swarm method we tested used the TF-IDF weight and term frequency as the guiding heuristic for the agents. This version of the algorithm does not use any pheromones. Therefore, formula 2.4.1 is not applicable in its classical sense. This version of the algorithm appears to be a more focused version of TF-IDF. Nevertheless, the simple swarm is a stepping stone for the next method, pheromone swarm.

4.2 Pheromone Swarm

The pheromone swarm method uses the TF-IDF weight amplified by pheromone count on terms and links as the guiding heuristic for the agents. The distinction between the simple swarm method and the swarm with pheromone method lies in the selection of the terms and links by the swarm agents.

A simple swarm agent is driven to consider, select, and focus on the most important terms in the document mostly at random (with some heuristic selection based on TF-IDF value of a term in a document). The agents in the pheromone swarm take into consideration pheromone deposits on the links and terms as they choose the next step of their journey.

In a pheromone swarm, the agents of the swarm search, discover, and guide swarm members to a target location via local interactions in the search space. The agent's decision on what term to select or what path to take is influenced by the presence of pheromone markings on the inspected object, e.g., terms or links.

For example, when an agent starts from a high-level document, the agent has a higher chance of selecting a term if the term has some pheromone markings. The pheromone markings on a term in a high-level document indicate an established fact that this particular term is a neighbor to some other term in a low-level document.

This idea of marking the neighbors and selected terms is based on treating textual documents as collections of *phrases* rather than as *bags of words*. A similar idea was expressed by Zou et al. [27], where the authors focused on "two-word phrases." Our approach is different in this sense; we allow phrases to be loosely defined in a neighborhood of a linking term.

The swarm with pheromones algorithm is described in Listing 4.2:

```
PHEROMONE SWARM TRACELINKS (H, L)
          // Input High and Low level documents H and L
          // Output list of agent count (h,l,n) - from h in l, where n is the count

1.  For each agent s in swarm S
2.        For each document h in high level collection H
3.                 // T = {t_1,....,t_n} sorted terms in doc h
4.                 T ← h.Terms.sortBy (TFIDF,PheromoneCount)
5.                  i ← Random[1.10]
6.                  t ← T[i]
7.                 E ← Vocabulary[t].LinksToLowLevelDocuments
8.                  E.sortBy (tTermFrequency,PheromoneCount);
9.                 j ← Random[1.10]
10.                 e ← E[j]
11.                N ← l.neighborsOf(t)
12.                For each neighbor n of t
13.                    Vocabulary[n].link[e].addPheromone[h]
14.                    if (Vocabulary[n].links.Contain[h]) and
15.                       (h.Terms[n].isNeighborOf(t)) then
16.                               h.Terms[n].addPheromone()
17.                EndFor
18.       EndFor
19. EndFor
20. For each document h in high level collection H
21.       For each document l in low level collection L
22.            list agents from h in l
23.       EndFor
24. EndFor
```

Listing 4.2 Pseudo code for pheromone swarm

Once all agents reach the low-level elements, they remain there. The pheromone deposit can spread further up the graph to the terms. We use the same methodology to generate candidate links using the cutoff threshold.

As the algorithm starts, a group of agents is assigned to a high level document, for instance Req1.txt. In the high-level element, the terms are then ordered by the product of TF-IDF weight and a linear function of pheromone count in the document. The agent randomly selects a term from the top ten sorted terms. Returning to our original example, the agent picks the term *personal*. The agent then "positions" itself in the common vocabulary at the term *personal*. Then, the agent inspects the links from the term *personal* to low-level elements. In this algorithm the links may contain pheromone deposits.

The pheromone deposits on the link serve as attractors for the agents' path selection. In Listing 4.2, the lines 4,5 and 7,8 use the pheromone counts to select next term and low level document respectively. For example, in line 4 the terms are sorted by pheromone count in descending order. On the line 5, a random term from top 10 terms is selected. A similar two step action takes places in lines 7 and 8: links to low level documents are sorted by pheromone count and a random link is selected from top 10 pheromone marked links. The pheromone deposits on the links indicate that there is another agent at the low-level document that came from a particular high-level document. Furthermore, the residing agent in the low-level document is in the neighborhood of the term *personal*. If the source document of the residing agent is Req1.txt, then our current agent will have a higher probability of selecting this pheromone marked link. Once a link to the low-level document has been selected, the agent crawls down to a low-level element. Once there, the agent diffuses pheromones on the neighbors of the linking term. These pheromone deposits will attract future agents traveling from the Req1.txt high-level document.

To experiment with the size of matching neighborhoods, we indicate how far the pheromones are deposited from the linking term. Swarm agents can also be instructed to deposit pheromones in low-level documents beyond the immediate neighboring documents and terms. To measure how far we allow agents to deposit the pheromones, we introduce a delta value. When delta is equal to one, we deposit the pheromones on the immediate neighbors. When we set the delta to 3, the agents deposit the pheromones up to three neighbors to the left and right of the linking term in the low-level document. When the delta is set to 5, five neighbors on either side of the linking term receive pheromone deposits. If the linking term is at the end or beginning of a document, and there are no "next 3 neighbors" on the right or left, only the present side of the linking term's neighborhood receives pheromone deposits.

The algorithm has to iterate through each swarm agent, each high level document, and sort terms within the high level document by weight and pheromone deposit. If we have A agents, D documents, and T terms, we say N = max(A,D,T). To iterate through every agent, we are bound by N. To iterate through every document, we are bound by N. For every time an agent crosses a document (N x N), the algorithm needs to sort at most N terms in the document (N logN). The pheromone swarm algorithm has a complexity $O(N^3 \log N)$.

5. Validation

This section presents the validation of the research.

5.1 Evaluation Approach

The purpose of this work is to evaluate the performance of the swarm methods to establish better quality candidate links between two sets of textual documents compared to vector space model (VSM) with TF-IDF weighting, referred to as TF-IDF.

We ran TF-IDF and swarm methods on the target set of documents and compared the quality of the candidate links generated by each method. The quality of the links for each set of high level and low level elements was evaluated against the corresponding set of correct links, i.e., the answer set.

For all studies, swarm method results were compared against the TF-IDF method on the Pine and CM1 datasets. Section 5.2 presents our hypotheses. In sections 0 and 5.4.2, we present and evaluate the swarm methods on the Pine dataset using the primary measures of recall, precision, F, and F2 as well as secondary measures for the Pine dataset. In sections 5.4.3 through 5.4.5, we evaluate the swarm methods on the CM1 dataset, along with a discussion about measures. Section 5.8 provides an overall summary of the results. Data points for the figures presented in this section are presented in **Error! Reference source not found.**of the **Error! Reference source not found.**.

5.2 Hypotheses

To validate the performance of each method, we used a one-tailed hypothesis in the form of the following question: *Does "swarm method M" produce better candidate link lists than TF-IDF?*

The independent variable in the study is the method (TFIDF, simple swarm). The dependent variable is MAP. The null hypothesis, **H[method]$_0$**, is:

There is no statistically significant difference in MAP between VSM TF-IDF and Swarm Methods.
$H_0 : MAP_{tf\text{-}idf} = MAP_{swarm}$

The alternative hypothesis, **H[method]$_A$,** can be stated as:

The MAP for Swarm Methods is greater that MAP for VSM TF-IDF.
$H_A : MAP_{swarm} > MAP_{tf\text{-}idf}$

5.3 Statistical Evaluation

The 11-point interpolated precision-recall graph is used to evaluate the statistical significance of the results (sign test). In addition, the Wilcoxon signed-rank test is applied to the MAP results to test for significance at the 0.05 level. In cases where the number of relevant links returned by the queries is different, the Mann-Whitney U Test is used instead of the Wilcoxon test.

5.4 Results to Date

The information that follows previously appeared in a conference [13] and a journal paper [50]. As the baseline, we ran traces on the Pine and CM1 datasets using the TF-IDF method. We treated the results obtained from TF-IDF for recall, precision, DiffAR, and MAP as our reference point. For hypothesis evaluation, we ran ten experiments on the Pine and CM1 datasets.

Due to the fact that agents' heuristics on selecting the "next hop," i.e., term or document, is based on a random choice, we gathered the results from several similar experiments. We made an assumption that ten experiments should be sufficient to observe any trend (if there is any) exhibited by the resulting random behavior of the swarm agents. Furthermore, to validate any statistically significant difference, a set of similar experiments becomes a stronger base for any conclusion. **1 Simple Swarm Applied to the Pine Dataset**

Figure 5.1 presents the 11-point interpolated precision-recall curve for the simple swarm and TF-IDF methods on the Pine dataset. Simple swarm presented higher precision than TF-IDF at 6 out of the 11 recall points, with most of the points near the middle to high end of recall. The difference in precision, however, was not statistically significant using the signed rank test.

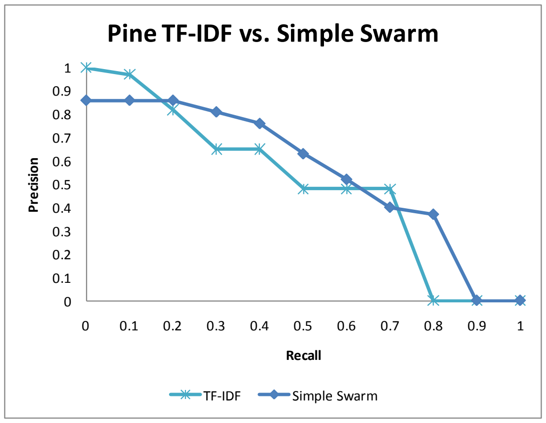


Figure 5.1 11-point Interpolated precision-recall curve for TF-IDF and simple swarm for the Pine dataset

Figure 5.2 depicts the F and F2 measures for both simple swarm and TF-IDF methods on the Pine dataset across the different thresholds. This figure presents a different view of how the two methods performed when threshold filtering was applied. F and F2 values for TF-IDF start off high but degrade as threshold values increased. Simple swarm F and F2 values, on the other hand, did not degrade as quickly as TF-IDF. They performed best between the threshold values of 0.2 and 0.4. Figure 5.2 also shows that simple swarm had a more consistent precision-recall tradeoff compared to TF-IDF when using threshold filtering.

For threshold values less than 0.4, simple swarm showed an increase in performance. The TF-IDF produced a consistent decline in performance as threshold values increased. This behavior can be explained by the fact that agents tend to gather around a smaller subset of elements as threshold values increase. The simple swarm method “directs” each swarm agent to consider and focus on the most important terms in the document, allowing agents to perform a more focused search. After passing an optimum threshold, agents start missing correct targets, e.g., low-level elements that are part of the correct links to the high-level element from which the agents started the journey.

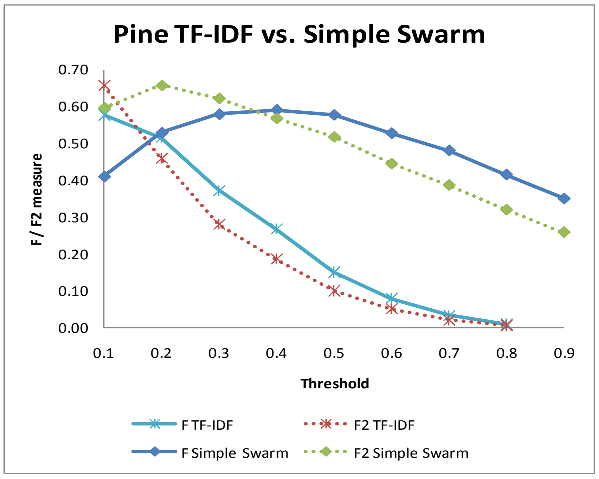


Figure 5.2 F and F2 measures for TF-IDF and simple swarm on the Pine dataset

Another explanation for the difference in F and F2 behavior between TF-IDF and simple swarm is how each link’s weight is calculated. TF-IDF link weights are measures of cosine similarity between the weighted keyword vectors of two documents [17]. For TF-IDF, link weights above 0.8 are uncommon.

Swarm methods, on the other hand, calculate link weights by dividing each link’s agent count by the largest agent count. Using this method, the top-most link always has a weight of 1. The difference in how weights are calculated does not prevent the methods from being compared appropriately as links are filtered using the same threshold values for both methods. The difference in F and F2 behavior indicates that TF-IDF achieves peak scores at lower threshold values compared to swarm. Both methods achieved comparable peak F and F2 values at different threshold values, e.g. TF-IDF at 0.1 and simple swarm at 0.2 and 0.4.

Pheromone swarm precision deteriorated below the 0.2 threshold but still remained near the 0.9 range. Figure 5.3 presents the 11-point interpolated precision-recall curve for the pheromone swarm and TF-IDF methods on the Pine dataset. Pheromone swarm gained a slightly higher precision than TF-IDF at several points for various delta values.

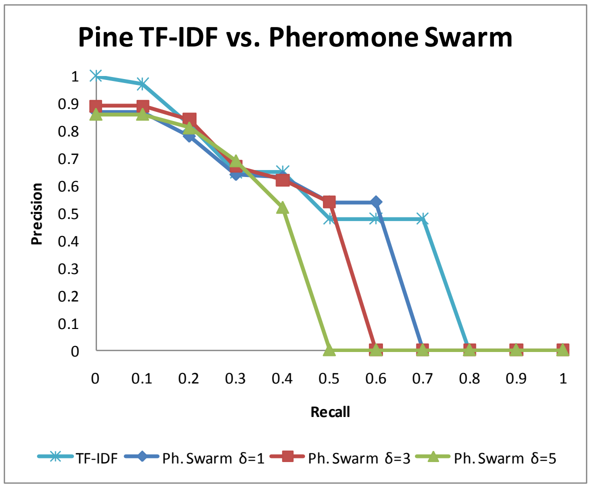


Figure 5.3 11-point interpolated precision-recall curve for pheromone swarm and TF-IDF for the Pine dataset

Figure 5.4 depicts the graph of the F measure for TF-IDF and pheromone swarm. Peak F values for pheromone swarm delta=1 and delta=3 are comparable to the TF-IDF Peak F value, e.g., 0.58, 0.56, 0.58, respectively. Pheromone swarm did not exhibit the same F/F2 trend as simple swarm when threshold values increased. The decrease in F values for the pheromone swarm was still slower than TF-IDF, indicating that the recall/precision tradeoff does not decrease as quickly with each increasing threshold value.

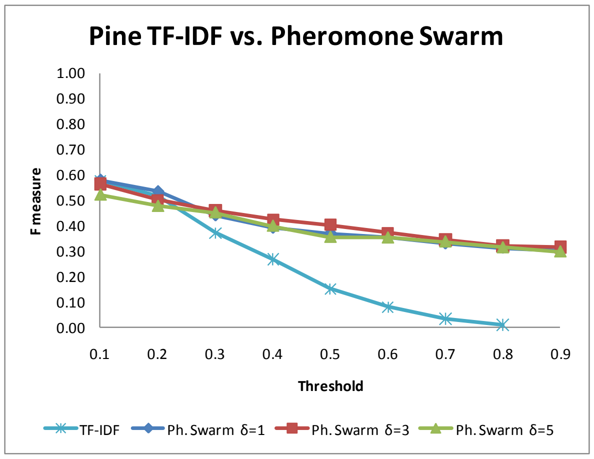


Figure 5.4 F measure for TF-IDF and pheromone swarm for the Pine dataset

Figure 5.5 depicts the graph of the F2 measure for TF-IDF and pheromone swarm for the Pine dataset. The trend in the F2 graph is similar to Figure 5.4, with TF-IDF outperforming pheromone swarm 0.66 to 0.61respectively at the 0.1 threshold. Even so, the recall/precision tradeoff was still slower compared to TF-IDF, implying that the pheromone swarm identified a greater number of relevant candidate links.

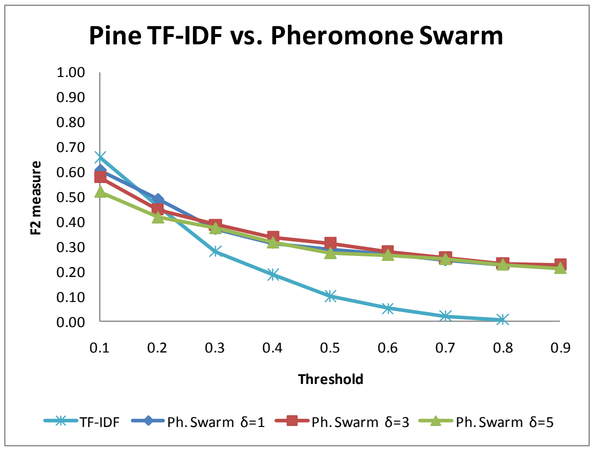


Figure 5.5 F2 measure for TF-IDF and pheromone swarm for the Pine dataset

5.4.2 Secondary measures for the Pine dataset

Figure 5.6 shows DiffAR performance for simple swarm, pheromone swarm, and TF-IDF methods. All swarm methods produced consistently higher DiffAR values compared to TF-IDF. Simple swarm performed the best among all methods, with DiffAR going from 0.41 to 0.93 as threshold values increased. This suggests that link weights from Swarm methods correlate to a greater degree with link correctness. Achieving higher DiffAR represents work that is less frustrating for human analysts, who must ultimately vet all candidate links to form the final traceability matrix.

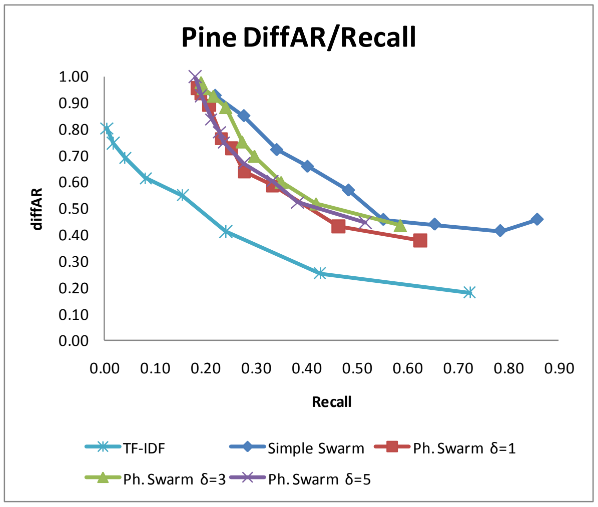


Figure 5.6 DiffAR vs. recall for simple swarm, pheromone swarm, and TF-IDF for the Pine

Figure 5.7 plots MAP vs. recall for the simple swarm, pheromone swarm, and TF-IDF methods. The simple swarm method returned more correct links at higher MAP with the first three thresholds compared to all the other swarm methods. Compared to TF-IDF at the 0.1 threshold, simple swarm achieved 0.76 MAP at 0.86 recall while TF-IDF achieved 0.75 MAP at 0.72 recall.

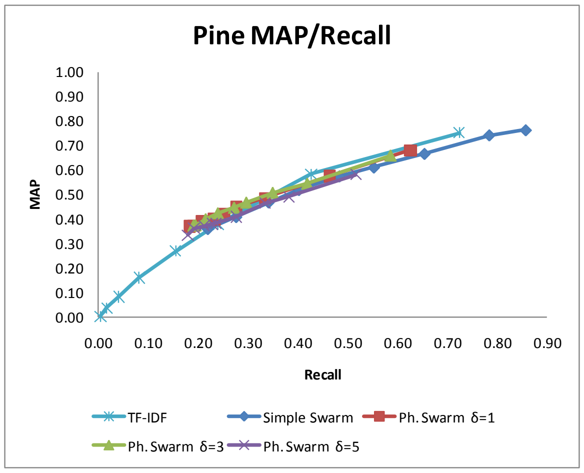


Figure 5.7 MAP vs. recall for simple swarm, pheromone swarm, and TF-IDF for the Pine dataset

5.4.3 Simple Swarm on CM1 dataset

Next, we examined the results for the CM1 dataset. Figure 5.8 shows the recall/precision graph for the simple swarm and TF-IDF methods. Note that precision values for this dataset are significantly lower than Pine due to the larger size of the dataset. This is a common phenomenon for IR methods that larger datasets yield smaller precision values.

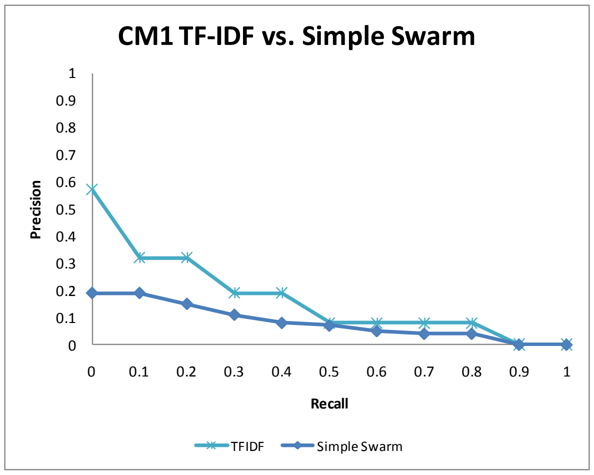


Figure 5.8 11-point interpolated precision-recall curve for the simple swarm and TF-IDF methods on the CM1 dataset

The recall/precision tradeoff between the two methods is slightly different than the tradeoff seen in the Pine dataset. Precision increased slowly when recall decreased, e.g., for simple swarm, precision only increased from 0.04 to 0.07[3] while recall dropped from 0.8 to 0.5. This indicates that simple swarm agents were not picking the correct low-level elements as threshold values increased. It is apparent that the search options given to the swarm agents restricted their options to explore and directed them to an overly limited number of low-level elements.

[3] We acknowledge that this is not acceptable precision.

Figure 5.9 shows the F and F2 measures for the simple swarm and TF-IDF methods. The F and F2 measurement for simple swarm on CM1 did not exceed 0.25. Note that the F measure for simple swarm did not change significantly, varying from 0.15 to 0.24.

TF-IDF achieved a peak F value of 0.28 and peak F2 value of 0.37, significantly outperforming simple swarm. For CM1, the TF-IDF method performed better than simple swarm for both F and F2 measurements. TF-IDF performed best at the 0.2 threshold value while simple swarm performed best at the 0.8 threshold for F and the 0.5 threshold for F2. Precision for simple swarm ranged from 0.04 to 0.19, contributing to the low F/F2 values and indicating that the two document levels contained many "coincidental matches," that is to say, even if the elements contained many similar terms, they were not necessarily classified as true links in the answer set.

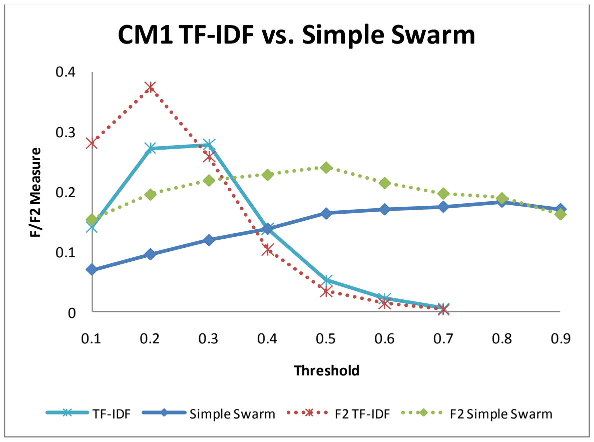


Figure 5.9 F and F2 for the simple swarm and TF-IDF methods on the CM1 dataset

5.4.4 Pheromone Swarm on the CM1 dataset

Figure 5.10 shows the precision-recall curve for the pheromone swarm and TF-IDF methods where agents deposited pheromones up to 1, 3, and 5 neighbors away, e.g., delta=1, delta = 3, and delta = 5. The pheromone swarm method performed worse at almost all recall points except for 0.5 recall, where pheromone swarm delta=1 and delta=3 tied with TF-IDF. Note that delta does not have much of an effect on precision for most of the recall points. That implies that the size of a neighborhood does influence the precision on CM1.

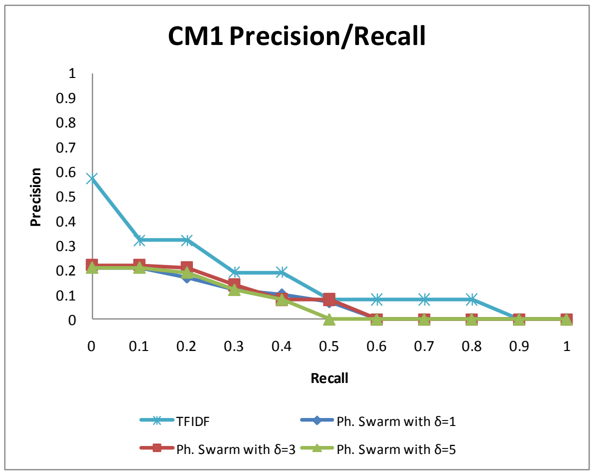


Figure 5.10 11-point interpolated precision-recall curve for pheromone swarm, delta = 1, 3, 5, and the TF-IDF methods for the CM1 dataset

Figure 5.11 shows the F and F2 measures for the pheromone swarm and TF-IDF methods. The F measurement stayed under 0.19. At the same time, F2 reached 0.26 at the threshold value of 0.3. Note that the F measure remained in the narrow "corridor" between 0.12 and 0.19 for the most part. The "corridor" of F2 values was between 0.17 and 0.26 in the CM1 dataset. TF-IDF outperformed pheromone swarm, with similar results compared to simple swarm, although the peak F2 value for pheromone swarm was at the 0.2 threshold.

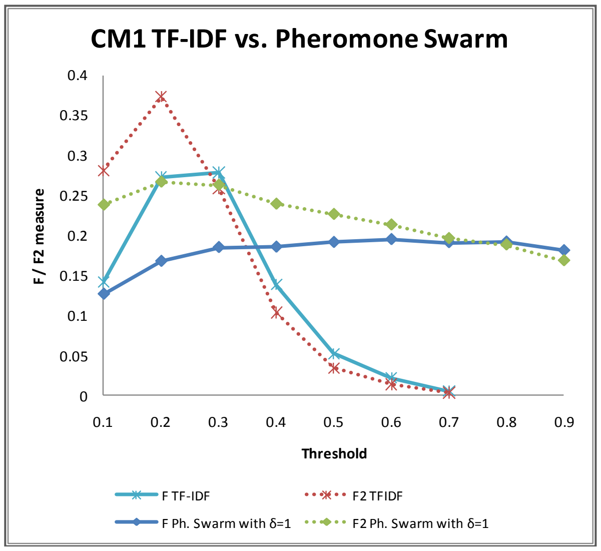


Figure 5.11 F and F2 measures for pheromone swarm, delta=1, and TF-IDF methods for the CM1 dataset

Figure 5.12 and Figure 5.13 show the F and F2 measures for the TF-IDF and pheromone swarm methods for CM1. F measure for pheromone swarm increased slowly with each threshold increase, while F2 measure slowly decreased instead. Pheromone swarm with delta = 3 seemed to perform better than the other two delta values, achieving peak F value of 0.20 and peak F2 value of 0.25. Therefore, expanding the pheromone affected neighborhood did not seem to improve the performance of the method.

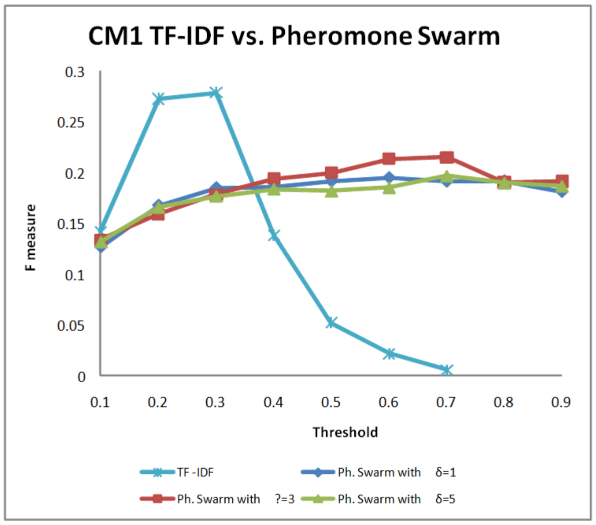


Figure 5.12 F measure for the pheromone swarm, delta = 1, 3, 5, and the TF-IDF methods for the CM1 dataset

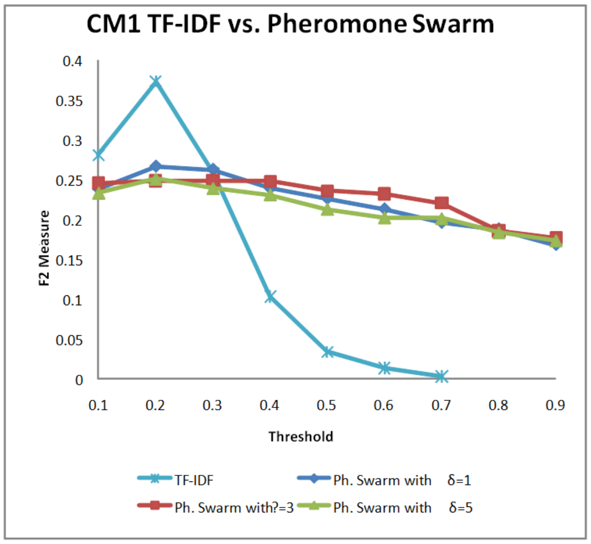


Figure 5.13 F2 measure for the pheromone swarm, delta = 1, 3, 5, and the TF-IDF methods for the CM1 dataset

5.4.5 Secondary measures for the CM1 dataset

Figure 5.14 shows DiffAR performance for simple swarm, pheromone swarm, and TF-IDF methods. Similar to Pine, all swarm methods had higher DiffAR values compared to TF-IDF. All Swarm methods performed about the same, with simple swarm performing worse between threshold values of 0.1 to 0.3.

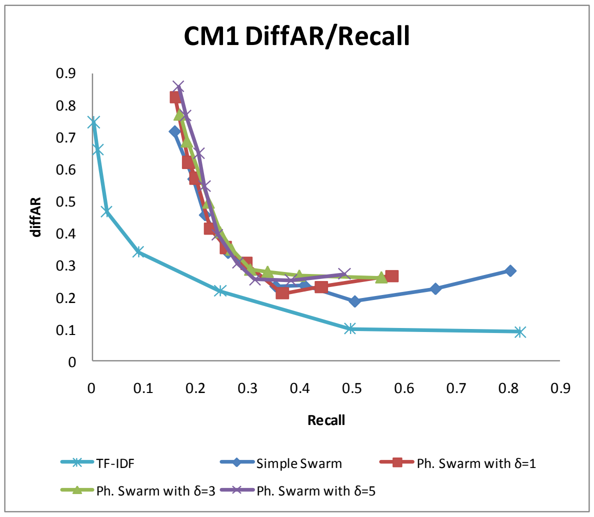


Figure 5.14 DiffAR vs. recall for simple swarm, pheromone swarm, and TF-IDF methods on the CM1 dataset

Figure 5.15 plots MAP vs. recall for the simple swarm, pheromone swarm, and TF-IDF methods. Simple swarm performed better than TF-IDF at the 0.1 to 0.3 threshold. Pheromone swarm with delta = 3 also performed better than TF-IDF at the 0.1 threshold. Pheromone swarm with delta = 1 performed worse than TF-IDF, but as delta increased, performance was comparable to TF-IDF. Note, however, that MAP was still quite low at 0.23, indicating that, on average, each document (high-level element) has an average precision of 23%.

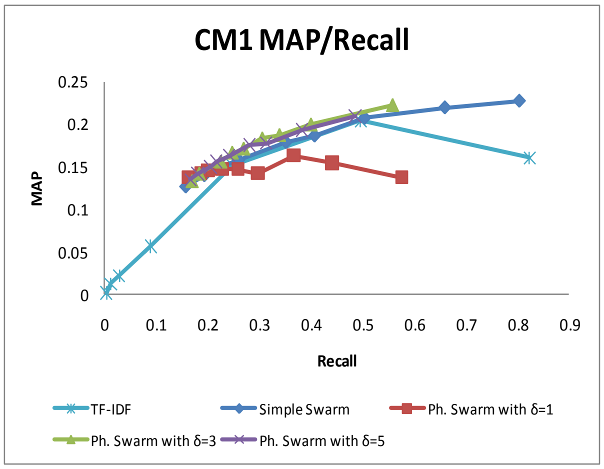


Figure 5.15 MAP vs. recall for the simple swarm, pheromone swarm, and TF-IDF methods on CM1

5.5 Statistical Analysis

Table 5.1 shows the values for MAP and DiffAR for TF-IDF, simple swarm, and pheromone swarm methods. The lower value of MAP implies better results; the higher value of DiffAR indicates a higher quality of the candidate links. As we can see, the MAP values for Pine were better in the experiments run with Pheromone swarm with

delta size 1, 3 and 5. For example, the Pheromone method with delta =1 produced MAP of 0.68 for Pine Dataset. The Wilcoxon-Signed Rank test indicated signed rank statistic $W_+ = 397$ and $W_- = 164$ with sample size of 33, and statistical significance of $p <= 0.03$.
In Wilcoxon-Signed Rank test, the high value for the "positive" sum (i.e., 397 vs 164 in the case of delta =1 at threshold 0.1) implies that we can reject the null hypothesis. We see mixed results for MAP and DiffAR produced by both methods (simple swarm and pheromone swarm).

Table 5.1 Statistical Analysis for the TF-IDF, simple swarm and pheromone swarm methods

		MAP		DiffAR	
CM1	tfidf@0.2	0.204	Wilcoxon Signed-Rank (tfidf vs swarm)	0.101	Mann-Whitney (tfidf vs swarm)
	ss@0.3	0.227	W+ = 4806, W- = 6519, N = 150, p <= 0.1083	0.281	**U = 9839, z = -4.65, p < 0.0001**
	delta1@0.3	0.163	**W+ = 4191, W- = 2479, N = 115, p <= 0.01696**	0.212	**U = 5979, z = -1.99, p < 0.0466**
	delta3@0.1	0.222	W+ = 6117, W- = 6924, N = 161, p <= 0.4964	0.261	**U = 7989, z = -3.2, p < 0.0014**
	delta5@0.1	0.209	W+ = 4297.50, W- = 4747.50, N = 134, p <= 0.6181	0.272	**U = 7722, z = -2.92, p < 0.0035**
Pine	tfidf@0.1	0.75	Wilcoxon Signed-Rank (tfidf vs swarm)	0.179	Wilcoxon Signed-Rank (tfidf vs swarm)
	ss@0.1	0.76	W+ = 272.50, W- = 322.50, N = 34, p <= 0.6753	0.456	**W+ = 40, W- = 1088, N = 47, p <= 3.037e-08**
	delta1@0.1	0.68	**W+ = 397, W- = 164, N = 33, p <= 0.0382**	0.377	**W+ = 232, W- = 896, N = 47, p <= 0.0004516**
	delta3@0.1	0.66	**W+ = 425, W- = 170, N = 34, p <= 0.0299**	0.436	**W+ = 162, W- = 966, N = 47, p <= 2.151e-05**
	delta5@0.1	0.58	**W+ = 591, W- = 112, N = 37, p <= 0.0003**	0.445	**W+ = 151, W- = 977, N = 47, p <= 1.271e-05**

5.6 Threats to Validity

The lines in bold in Table 5.1 imply we can reject the null hypotheses in favor of the alternative. Yet, not every experiment indicated that the swarm methods outperformed the benchmark method (VSM TF-IDF).

5.7 Threats to Validity

Threats to **conclusion validity** threaten the ability to draw correct conclusions from the study results. By using two datasets and applying similar treatments, we addressed the reliability of the treatment implementation. We used standard information retrieval measures to evaluate effectiveness, such as MAP. Both datasets were analyzed using the TF-IDF, simple swarm, and pheromone swarm methods.

There was a possible threat to **internal validity** due to experimenter bias. The answer sets were created by human analysts that are familiar with the traceability research domain. We reduced this threat by using datasets for which answer sets had been independently verified by more than one analyst, and in some cases more than one research group (CM1). We also used a vetted tool, RETRO.NET [9], and adapted it in order to implement the swarm techniques.

There was another possible threat to **internal validity** due to stochastic agent behavior. The swarm methods randomly select links to follow. To mitigate this threat, we ran each method ten times and examined the mean recall and precision values. In future experiments, we plan to execute the same methods at least ten times.

Threats to **construct validity** undermine how the experiment settings and measurements truly determine the correct desired properties. In our experiments we decided to use the agent count in low level documents as a measurement for candidate links. We reduced threats to **construct validity** by using a relative agent count (out of total swarm size) rather than absolute count to indicate a candidate link.

Threats to **external validity** deal with whether the results can be generalized. Results to date used two datasets from two different domains for validation. Though both datasets are real projects (not student projects), one of them is relatively small (49 x 51). Therefore, it is not possible to state that the study sufficiently validated all domains or all projects [9].

5.8 Overall Summary

Though the swarm agent counts and TF-IDF links weights are not calculated in the same manner, they serve a similar role; they are used for filtering the candidate links. The higher the filter value (a close cosine similarity in documents in TF-IDF or a higher agent count in swarm methods), the more the F values decreased for TF-IDF and Swarm methods on both datasets.

Figure 5.1 shows that F values for TF-IDF perform better than simple swarm below threshold values of 0.2 on the Pine dataset. After the threshold is increased, the swarm's F values (with/without pheromones) were better than TF-IDF as seen in Figure 5.2 and Figure 5.4. Furthermore, TF-IDF exhibited a steep decline in F and F2 as threshold values increased. Swarms demonstrated better values for F measurements for higher threshold values. The higher threshold implies that an analyst has to review fewer links of higher quality.

Figure 5.9 showed better performance for the TF-IDF method than simple swarm on the CM1 dataset, achieving 0.28 for F and 0.37 for F2. Simple swarm performed better than TF-IDF past the 0.4 threshold.

Pheromone swarm, in general, performed better than simple swarm on the CM1 dataset. Pheromone swarm with delta=3 reached the highest value for F of 0.21 at the 0.6 threshold. Furthermore, pheromone swarm exhibited a gradual increase in F value as the threshold increased. TF-IDF reached its peak F value of 0.28 at the 0.3 threshold and then declined sharply as threshold values increased.

The same trend was observed with TF-IDF in the CM1 and Pine datasets for the F measurement. The F2 values for the swarm methods exhibited a slightly different behavior. F2 values slowly declined as the threshold increased for all swarm methods. Even in these instances, the swarms displayed a more gradual change in performance as the threshold increased. Pheromone swarm F2 values gradually decreased from 0.25 to 0.18.

In summary, the simple swarm approach showed some advantage over the TF-IDF method on the Pine dataset, yet it did not fare as well on CM1. At the same time, with pheromone swarm, any advantage indicated on the Pine dataset was lost. Pheromone swarm performance on CM1 improved over simple swarm, but still underperformed TF-IDF.

A possible explanation for this is the way that the high and low elements are connected. The Pine dataset contains 49 high-level and 51 low-level elements, with 2,499 possible links. The CM1 dataset contains 235 high and 220 low elements, creating a search space of 51,700 possible candidate links. The answer set for the Pine dataset has 246 links, about 10% of all possible links. In the CM1 dataset, the ratio of true links over possible links goes down to less than 1% (361 true links divided by 51,700). CM1 also uses a significant amount of technical terms and acronyms, causing the swarm agents to end up at incorrect low-level elements. One can draw a logical conclusion to utilize a thesaurus when a dataset contains many acronyms.

It appears that in a compact dataset such as Pine, the pheromones made the agents "over choose" certain links. This led to lower starting recall and higher precision as seen in Figure 5.3. On the other hand, for CM1, pheromone swarm delivered better precision than simple swarm with more focused selection in a sparsely linked set. Agents got to pick proper links based on the pheromone markings previously deposited by other agents.

For the CM1 dataset, the MAP measurements exhibited some variance with regard to the pheromone swarm method. Pheromone swarm at delta = 3 performed just "above" TF-IDF and all other swarm methods. As we saw earlier for the CM1 dataset, increasing the size of the affected neighborhood delivered some performance gains. Simple swarm had better MAP at lower thresholds for both datasets.

Another interesting result we observed was related to the size of the neighborhood of a linking term. When we increased the delta from 3 to 5 for the pheromone swarm, we noticed a slight drop in performance across just recall measurements and both datasets. Apparently, by depositing pheromones on neighbors that are "too remote," the agent introduces too much noise for future agents. For example, on the CM1 dataset with delta = 3 the starting recall and precision values were 56% and 8%, respectively. When we increase the delta to 5, i.e., five neighboring terms on either side of the linking term received deposits, the starting recall and precision became 48% and 8%, respectively.

Maarek et al. [35] and Niu and Easterbrook [36] experimented with a neighborhood of size five (5) using 'lexical affinities.' Our work differs from 'lexical affinities' in several ways. Unlike 'lexical affinities,' the swarms consider neighbors that may cross sentence boundaries. 'Lexical affinities' pick up two word units, whereas the swarm considers all terms within the limits of the inspected neighborhood. This difference may explain why we obtained an optimal neighborhood of three (3) as opposed to five (as in 'lexical affinities').

To achieve high recall and precision results for the CM1 dataset, the swarm agents have to conduct the search with a more narrow focus. The use of a thesaurus might have directed the swarm agents to the proper document. In addition, a method of handling acronyms might be of significant assistance. In this case, the thesaurus may become project specific.

In the case of TF-IDF at low threshold values, the method considered a greater number of the low-level elements as possible candidate links, thus yielding higher recall at the cost of precision. The Swarm method, a more focused approach than TF-IDF, limited the "discovery horizon" for the agents by focusing on the top terms in a textual element, hence limiting the possible search alternatives. Both methods increased recall at the expense of precision.

6 Reinforcement Learning Model

This section presents the Reinforcement Learning method overview and observations made based on empirical results.

6.1 Overview

The search space in a reinforcement learning (RL) model is similar to the search space described in the swarm technique. It has three layers of data. The top level consists of the high level documents. The middle level consists of all terms in all documents. The bottom level consists of the low-level documents.

The agents traverse the search space starting from the top level documents down to the low level documents by selecting the terms in the middle layer that are common between the selected documents. The main idea of the algorithm is to equip the agents with some heuristics to navigate the search space and choose the correct candidate links between the high and low level documents.

To define a search space in terms of the RL model, we need to define states, actions, transitions, and rewards. Figure 6.1 lists states and the transitions between them.

States are defined by the agent's position in the data space. The agents can be in any of the following states:

- Top level document, $\mathcal{HL}$
- A term in a high level document, $t_{\mathcal{HL}}$
- Low level document, $\mathcal{LL}$
- A term in a low level document, $t_{\mathcal{LL}}$
- A synonym term in vocabulary, st.

The agent's states are the positions in the search space where the agent can be located. The action the agent selects determines the states in the search space to which the agent will transition. Possible actions at states and transitions between the states are shown in the Agent State Transition Diagram, Figure 6.1.

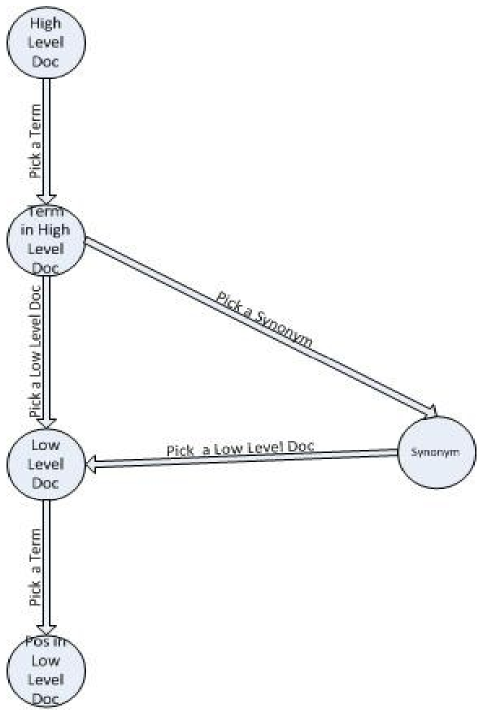


Figure 6.1. RL Agent state transition diagram.

As we can see in Figure 6.1, when an agent is positioned at a high level document, state $\mathcal{HL}$, the agent starts by selecting a term as the starting point for its journey (the heuristic of selecting terms is described below). By selecting a term in a high level document, the agent transitions to the state $t_{\mathcal{HL}}$. From the state $t_{\mathcal{HL}}$, the agent can choose a low level document that contains either the term or a synonym of the term.

By choosing a low level document, the agent transitions to the state $\mathcal{LL}$. From the state $\mathcal{LL}$, the agent should select a term in the low level document. If the low level document contains the term $t_{\mathcal{HL}}$ in several positions, the agent needs to select a position $t_{\mathcal{LL}}$ within the low level document to maximize the match between the neighborhoods in the high and low level documents. A neighborhood is a textual segment located around a linking term.

Alternatively, from state $t_{\mathcal{HL}}$, the agent can also choose to explore the synonyms of the term. If the agent selects a synonym, it transitions to the state s_t. From the state s_t, the agent can only choose a low level document containing the synonymus term. Possible actions at states and transitions between the states are summed up in Table 6.1.

Table 6.1. Agent actions.

From		Action*	To	
Top Level Document	HL	Select a term	Term ($_{\text{High Level Doc}}$)	t_{HL}
Term($_{\text{High Level Doc}}$)	t_{HL}	Select Low Level or Synonym	Low Level Document	LL
Term($_{\text{High Level Doc}}$)	t_{HL}	Select Low Level or Synonym	Synonym	s_t
Low Level Document	LL	Select a term	Term($_{\text{Low Level Doc}}$)	t_{LL}
A synonym term in vocabulary	s_t	Select Low Level	Low Level document	t_{LL}

Each action listed in Table 6.1 can be in one of the three behaviors: random, linear, or quadratic.

In random behavior the agent has an equal probability of transitioning in any of the available next states. The formula for the random behavior is as follows:

$$\Pr(S_i) = \frac{1}{N} \quad , \tag{6.1}$$

where S_i is a reachable state and N is the number all reachable states. For example, if the agent is in a "term- high-level" state $t_{\mathcal{HL}}$ and has ten possible low level documents, i.e. ten reachable $\mathcal{LL}$ states, the probability of transitioning into each of the reachable states is only 0.1.

Linear behavior allocates the transitional probabilities to the reachable states proportional to the numeric values or rewards the reachable states possess. The formula for linear behavior is as follows:

$$\Pr(\text{State}_i) = \frac{Value\,(S_i)}{\sum_{j\,\epsilon possible\ states}(S_j\,)} \quad , \tag{6.2}$$

The probability of transitioning into the state S_i is proportional to the value in the state S_i, divided by the sum of values of possible transition states. For example, if the ten reachable $\mathcal{LL}$ states from the state $t_{\mathcal{HL}}$ had the following values associated with them: {20, 50, 30, 0, 0, 0, 0, 0, 0, 0}, the probability of transitioning into the first $\mathcal{LL}$ state is

0.2, into the second 0.5, the third is 0.3. The remaining reachable states would receive 0 transition probability. We describe the numeric state values and rewards later.

When the agent selects the quadratic behavior, the transition probabilities from the example above would be distributed based on the following formula:

$$\Pr(\text{State}_i) = \frac{Value(S_i * S_i)}{\sum_{j \in possible\ states}(S_j * S_j)} \tag{6.3}$$

The probability of transitioning into the state S_i is proportional to the squared value in the state S_i, divided by the sum of squared values of possible transition states.

Table 6.2 shows state values and associated behavioral probabilities for an agent inspecting the A3.txt high level document in the Pine dataset. The probabilities depend upon the term selection behavior the agent may choose.

Table 6.2. Term selection probability based on the transition values and selection behavior.

Term	Value	Random	Linear	Quadratic
a3	0	0.067	0.00	0.00
address	1.106	0.067	0.12	0.10
book	0.808	0.067	0.09	0.06
allow	0	0.067	0.00	0.00
creat	0.52	0.067	0.06	0.02
delet	0.29	0.067	0.03	0.01
modifi	0	0.067	0.00	0.00
add	0	0.067	0.00	0.00
name	0	0.067	0.00	0.00
delet	0.24	0.067	0.03	0.00
name	0.026	0.067	0.00	0.00
person	1.5	0.067	0.16	0.19
distribut	1.54	0.067	0.17	0.20
list	1.55	0.067	0.17	0.20
pdl	1.57	0.067	0.17	0.21

Consider the term 'list.' In the course of the Reinforcement Learning algorithm, the state "A3-list" received the value 1.55. The random selection behavior estimates the probability of transitioning into "A3-list" state from "A3" state as 0.067. The linear selection behavior raises the probability of such transition to 0.17. The quadratic selection assigns the transition from ‘A3’ to ‘A3-list’ the highest probability, 0.21.

Figure 6.1 and Figure 6.2 display the transitional probabilities for linear and quadratic selection behaviors based on Table 6.2

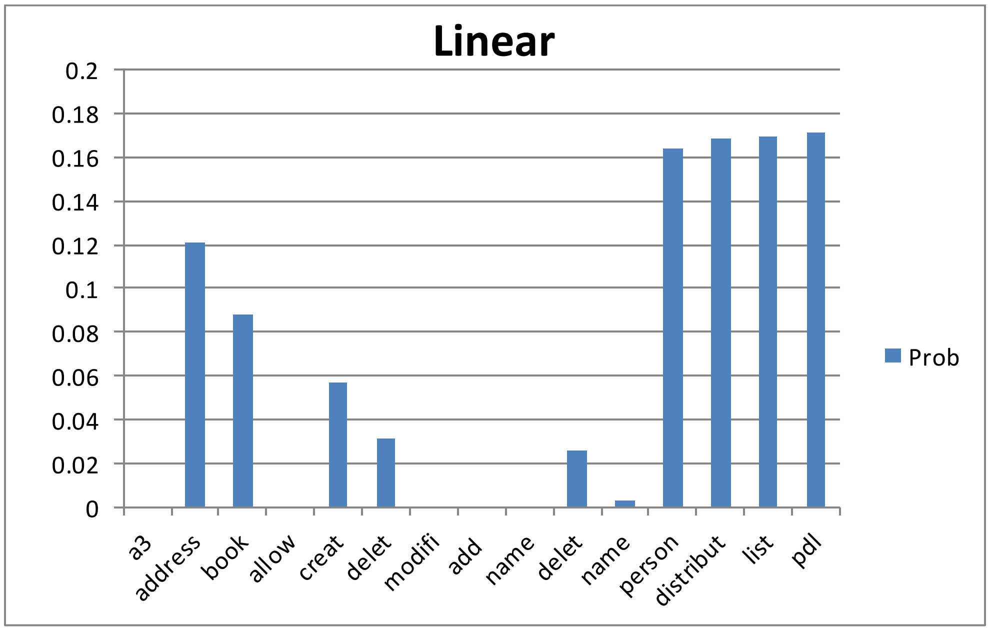


Figure 6.2. Term selection probability based on linear selection behavior.

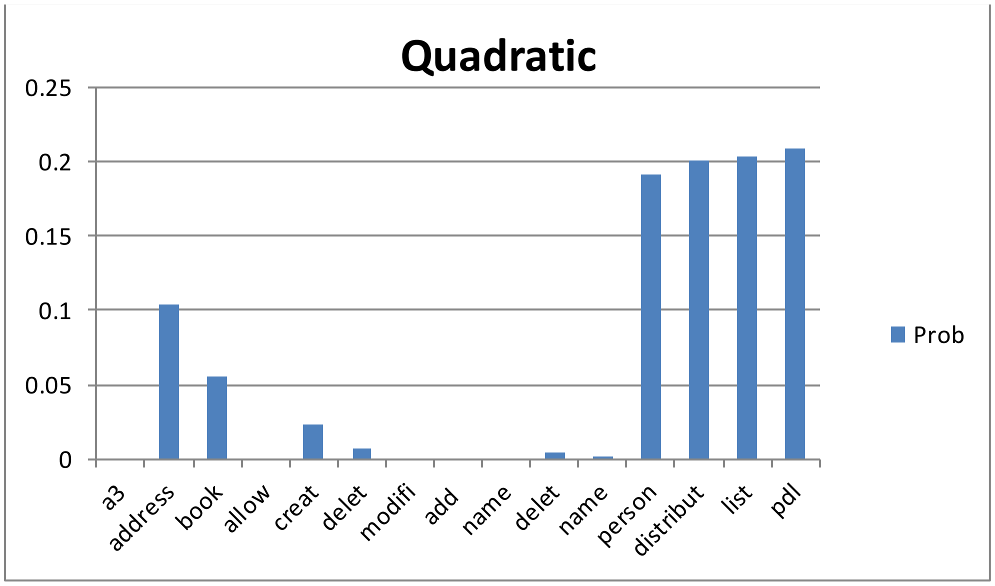


Figure 6.3. Term selection probability based on quadratic selection behavior.

It can be visually seen that the term 'delet' in Figure 6.3 is significantly smaller compared to the terms 'distribut,' 'list,' and 'pld.' It is also worth mentioning that our RL algorithm differentiates between the different positions of a term in a document. For example, the term 'delet' appears in document A3 in two positions. Each position, or state "A3-pos,' receives different values based on values calculated during the Value Iteration algorithm. Therefore, the positions receive different transition probabilities.

A state "a term in low level document" $t_{\mathcal{HL}}$ can have a reward. This is a numeric value associated with transitioning into the $t_{\mathcal{HL}}$ state. The reward is calculated by comparing the text segments in two neighborhoods: in high and low level documents. The comparison evaluates how many common terms the two segments share. The reward is estimated using the following formula:

$$\sum_{w_1 \in H} \sum_{w_2 \in L} \delta(w_1, w_2)\left[\, tfidf_{w_1} * C_{high} + \; tfidf_{w_2} * C_{low}\right] \quad , \tag{6.4}$$

where H is the collection of high level documents, L is the collection of low level documents, and w_1, w_2 are the terms in H and L documents respectively.

The function $\delta(w_1, w)$ is calculated as follows:

$$\delta(w_1, w_2) = \begin{cases} 1 \; if \; w_1 = w_2 \\ 0 \; if \; w_1 \neq w_2 \end{cases} . \tag{6.5}$$

The multiplication coefficients range from 1 to 10: $C_{high}, C_{low} \in \{1{,}10\}$. The range of multiplication coefficients is a calculated estimate on the similarity of the textual neighborhoods. The higher values for the $C_{high}\; and\; C_{low}$ coefficients imply that the matching terms are close to each other in the neighborhood of the linking term.

The reward associated with transitioning into a position in a low level document is propagated back to the high level documents through the common linking terms. As described in the background section (Section 2.3), the agents choose the behavior in the RL model, i.e., the search space navigation policy, to maximize expected return. The expected return is calculated by the formula:

$$R = \sum_{t=0}^{N-1} r_t , \tag{6.6}$$

where r_t is the reward received after $\boldsymbol{t}$-th transition action.

The Reinforcement Learning algorithm for requirements traceability is described in Listing 6.1:

REINFORCEMENT LEARNING TRACELINKS (H, L)

// Input High and Low level documents H and L

*// Output list of agent count **(h,l,n)** - from **h** in **l**, where **n** is the count*

```
// Creates State Space
 For each document  hl in high level collection H
     States.Add(NewState(hl))
    For each term  t in high level document h
             i $\leftarrow$ position of t in hl
             States.Add(newState(hl_t))
             // Iterate through low level documents linked via term t
             For each document ld in  Vocabulary.GetDocsByTerm (t)
                 If ld is lowLevelDocument
                      For each position j  of term t in ld
                          lowLevelDocState $\leftarrow$ newState(ht_t_ld_posj)
                          Value = EstimateMatchingValue(hl,ld,i,j)
                          lowLevelDocState.Value $\leftarrow$ Value
                     End For
                Else  // ld is a synonym
                      ld_2 $\leftarrow$ Vocabulary.GetDocsByTerm (ld)
                      For each position j  of term ld in ld_2
                           lowLevelDocState $\leftarrow$ newState(ht_t_ld_posj)
                           Value = EstimateMatchingValue(hl,ld_2,i,j)
                          lowLevelDocState.Value $\leftarrow$ Value
                     End For
                End if
             End For
  End For


 // Calculate state values
 For cycle 1 to 5
    For each state s in States
        argMaxValue $\leftarrow$ 0
        possibleSates $\leftarrow$ s.Transitions
        For each action a in Actions
             possibleStates.TransitionProbabilitiesForAction(a)
            For each ps in possibleStates
                possibleValue $\leftarrow$ possibleValue + ps.Probability* ps.Value
              If possibleValue > argMaxValue
                 bestAction $\leftarrow$ a
                 s.Policy $\leftarrow$ a
```

```
          End if
        End For //possible states
        argMaxValue ← Max(possibleValue, argMaxValue)
      End For // Actions
      s.Value←s.TransReward + argMaxValue
    End For // states

// Traverse the Search Space
For each top level document hl
    For each agent ant in colony
      currentState ← States(hl)
      While CurrentState != low level document
          //using currentState.Policy and currentStates.Transitions
          nextState ← currentState.SelectNextState
          currentState ←nextState
      End While
    End For
End For
```

Listing 6.1 Pseudo code for requirements traceablity reinforcement learning

The Reinforcement Learning algorithm determines an optimal transition policy for each state by maximizing the expected return. The transition policy will become the guiding heuristic for the agents to traverse the search space.

6.1.1 Path Saturation

The agents choose to select certain states based on the space traversal policy. When an agent is presented a choice of possible next states $\boldsymbol{S} = \{s_1, s_2, \ldots , s_k\}$, the probability of transitioning into the next state depends on the value the next state holds. It is possible for one of the next states to have a value which is much higher than the values of other possible next states. In this case, the probability of transitioning into $\boldsymbol{S}_i$ is higher than the probability of transitioning into any other state:

$$\Pr(s_i) >> \Pr(s_j), \text{ where } s_i, s_j \in \{s_1, s_2, \ldots , s_k\} \text{ and } i \neq j, \tag{6.7}$$

It is possible to have a situation where a majority of agents always select the state with the transition probability much higher than other possible states. This scenario may limit the search only to the states with high values. To address this situation, we introduce the notion of path saturation.

Path saturation is a value added to define the number of agents transitioning from state S_A to state S_B. As the saturation value gets higher, the probability of transitioning from S_A to

S_B becomes smaller. The saturation value from S_A to S_B on the transition path has the inverse effect on the transition probability from S_A to S_B.

In a manner similar to the Swarm Algorithm, the candidate links are estimated by the agent count gathered in the low level documents. A candidate link between high level document HL_{doc} and the low level document LL_{doc} gets a count of one if an agent starting from HL_{doc} has reached the low level document LL_{doc}. After all counts on the candidate links have been calculated, the candidate links are ranked by the agent count.

Having defined the search space and the search space traversal heuristic, the next step is to outline the experiment design.

6.1.2 Experiment Design

In order to validate the RL technique, it was applied to two datasets. The datasets are the same as used for the swarm algorithm. The first project consisted of 49 textual requirements and 51 textual use cases. The dataset is a text-based email system Pine developed by the University of Washington [51].

The Pine dataset contains 246 true links. These links form the answer set, i.e. a collection of links against which we can validate our findings.

The second project consisted of 22 requirements documents and 53 design documents in the NASA scientific instrument project CM1SUB [52]. The project has 45 true links in the answer set.

The experiments were conducted using a Vector Space Model using TF-IDF weighting (TF-IDF hereafter) and the Reinforcement Learning (RL) method. The independent variable in the study was the method (TFIDF, Reinforcement Learning). The dependent variables were recall and precision. The precision-recall graph and statistical analysis were used to evaluate the results.

All textual documents were pre-processed, the agents selected each high-level element one at a time and the agents used the search space navigation heuristics established by the RL method. The output was captured in the form of candidate RTM. The results were compared to the answer set to calculate recall and precision (formulas 2.1.1 and 2.1.2) defined earlier.

To eliminate any possible threats to the validity of the experiment, several controls were implemented.

Internal threats to validity included possibly indicating a relationship between the treatment methods and the outcome, when in reality there was no relationship. First, in our controlled experiment, we used the same datasets in the same environment. This was done to provide a fixed environment where it was possible to observe the differences in the outcome only where the treatments are different, i.e., where we apply different candidate link generating algorithms.

To address the possible threat to internal validity due to repeated testing, each method was run ten times and examined using the mean recall and precision values. Each method produced average recall and precision values with variances ranging from 0.003 to 0.06.

To protect the ability to draw valid conclusions from the study, the same two datasets were analyzed using similar treatments. In this experiment, both datasets were analyzed using the TF-IDF and the RL methods.

Another possible threat identified was the effect of experimenter bias on the ability to reach valid conclusions based on the data. This threat was reduced by using datasets where the answer sets were independently verified by more than one analyst. In the case of CM1SUB dataset, more than one research group was used.

There was additional potential for bias in that the answer sets created by human analysts familiar with the traceability research domain. The vetted tool, RETRO.NET [36], was used and adapted in order to properly implement the RL technique. The threats to validity were also reduced by using the standard information retrieval measures of recall and precision to evaluate effectiveness.

In addition to the internal threats to validity, threats to external validity and the ability to properly generalize the results were addressed by using two datasets for validation. Though both datasets are real projects (not student projects), they are small size datasets.

Also, though the datasets do represent two different domains, it is not possible to state that the study sufficiently validated all domains or all projects.

6.2 Results

Following the completion of the experiments, the RL method and TF-IDF method were evaluated for the Pine and CM1SUB datasets using the primary measures of recall and precision. Section 6.2.1 presents the results and observations made during the initial stage of the RL algorithm development. The RL results for Pine are shown in in Section 6.2.2. In section 6.2.3, we evaluate the RL method for the CM1SUB dataset. Section 6.2.4 provides an overall summary of the results and directions for future possible work. Data points for the figures in this section are presented in Table 2 of Appendix A.

6.2.1 Reinforcement Learning Initial Results and Points of Interest

The initial development phase for the RL algorithm yielded results that were less than impressive. For both datasets, Pine and CM1SUB, the precision-recall curves for the RL method were below the precision-recall curves for the TF-IDF method. Figure 6.4 presents precision-recall curve for Pine obtained using an early version of the RL method. It is clearly visible that the RL method at that phase underperformed the TF-IDF method.

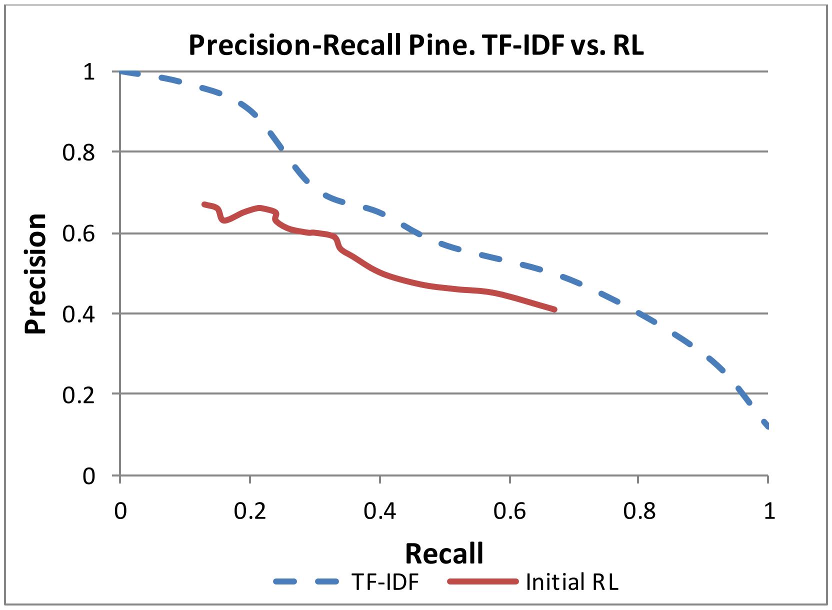


Figure 6.4 Precision-recall curves for TF-IDF and initial phase of reinforcement learning methods for the Pine dataset

Compared to the TF-IDF method, the initial version of the RL algorithm showed lower precision values for the same values of recall. The highest value of precision for the RL method was 0.67 at recall 0.13. At the same time, for TF-IDF at recall 0.13 the precision was 0.95. We observe a similar situation for the CM1SUB dataset on the initial version of the RL algorithm, shown in Figure 6.5.

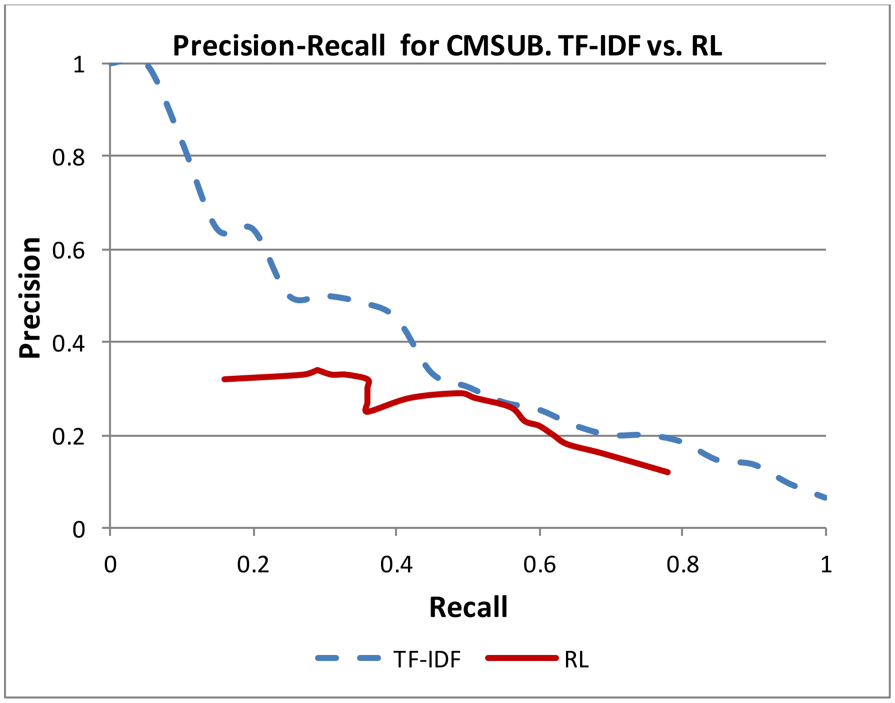


Figure 6.5 Precision-recall curves for TF-IDF and initial phase of reinforcement learning methods for the CM1SUB Dataset

For the CM1SUB dataset, the initial version of the RL algorithm showed lower precision values for the same values of recall. The highest value of precision for the RL method was 0.34 at recall 0.29. The TF-IDF method achieved precision of 0.5 at recall 0.29.

Another observation we made was that on both datasets, the initial RL method demonstrated a narrower corridor of precision-recall values. For Pine, the precision ranged from 0.41 up to 0.67; for CM1SUB, the precision ranged from 0.12 up to 0.32. This led us to believe that the RL method maintains a "narrower focus" compared to TF-IDF. Further investigation of the algorithm accentuated the importance of the "matching neighborhood function." We observed the direct effect of the way the textual neighborhoods are matched on the quality of candidate links. The pseudo code presented in Listing 6.1 displays, in line 12, the call to the neighborhood matching function.

After careful consideration and analysis, we decided to utilize *tf-idf* weight of the terms in establishing matches between textual neighborhoods (Formula 6.4). Intuitively, it makes sense that two textual segments, i.e., neighborhoods, sharing a number of terms with high *tf-idf* weight, may in fact have a strong link between them. The results for the improved Reinforcement Learning algorithm are presented in the next sections.

6.2.2 Reinforcement Learning on Pine

Figure 6.6 presents the precision-recall curve for the RL and TF-IDF methods for the Pine dataset.

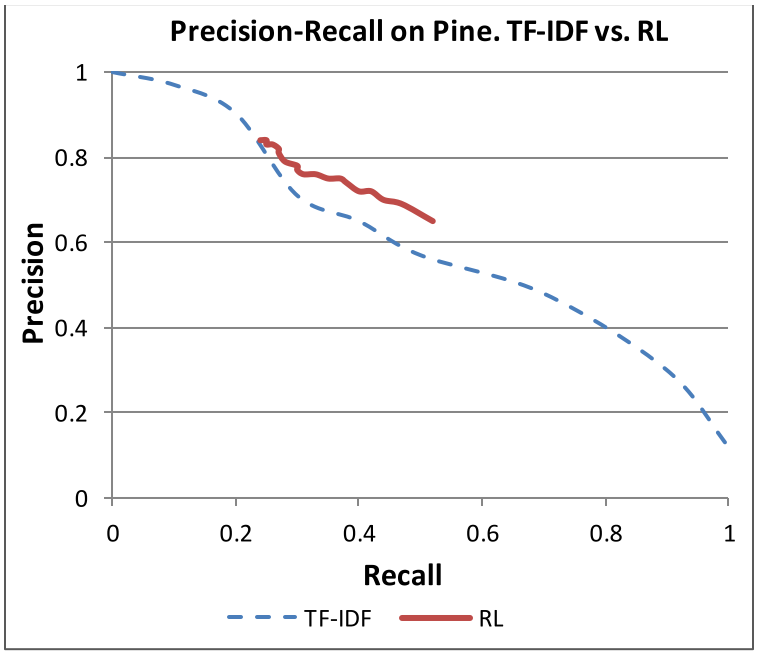


Figure 6.6 Precision-recall curves for TF-IDF and reinforcement learning methods for the Pine method

The RL method demonstrates higher precision values than TF-IDF for the same values of recall. The highest precision for RL method is 0.84 at recall 0.24. As we can see in Figure 6.6, the highest precision-recall value in RL is at the same position as in TF-IDF.

By inspecting other values of the precision-recall graph, we see the RL method produced a more focused result. The lowest precision returned by the RL method is 0.65 at recall 0.52. The comparable result for TF-IDF achieves precision 0.65 at recall 0.4. The quality of candidate links produced by the RL method is better; the RL achieves higher precision than TF-IDF for the same recall values.

For the Pine dataset, at recall of 0.42 the RL method achieves precision of 0.73. As we can see from Table 2 in Appendix A, the RL method filtered at 0.25 suggested 141 links. The number of correctly identified links was 103. The total number of correct candidate links for the Pine dataset is 248. The 103 correctly suggested links out of a total of 248 equates to 0.42 recall.

The TF-IDF method at 0.20 filtering on the Pine dataset suggests 162 links; 106 links are correctly identified. 106 out 248 is 0.42 recall. Having similar recall values, the two methods achieved different precision: the TF-IDF method achieves 0.65 (0.65= 106/162); the RL method achieves 0.73 (103/141). The RL method retrieves a higher number of relevant documents compared to the TF-IDF method.

To evaluate any statistical difference between the two methods, the recall and precision numbers were compared on the overlapping recall value range. For the Pine dataset, the TF-IDF method covered recall values from 0 to 1, while the RL method covered recall values from 0.23 to 0.52. Using the recall point from the RL method, the precision values were interpolated for the TF-IDF method. Twenty recall values and twenty precision values for TF-IDF and RL were used to define the null hypothesis and alternative hypotheses for the results:

> **H_0**: There is no difference between the precision values of the TF-IDF interpolated precision-recall graph compared to the precision values for the RL method's precision-recall graph.
>
> **H_1**: There is a difference between the precision values of the TF-IDF interpolated precision-recall graph compared to the precision values for the RL method's precision-recall graph.

The Wilcoxon Signed Ranked method was used to evaluate the null hypothesis. The critical value for $Z_{critical}$ test was ±1.96 at confidence level $\alpha = 0.05$. The results of the calculations produced the following values:

- $W_- = -205$,
- $W_+ = 20$,
- $Z = -3.82$.

Since $Z < Z_{critical}$, the null hypothesis was rejected. This left the conclusion that there is a statistically significant difference between the precision values of the two methods.

6.2.3 Reinforcement Learning on CM1SUB

The RL method applied on the CM1SUB dataset produced results similar to the results obtained on the Pine dataset. Figure 6.7 shows the precision-recall values for the RL method compared to the precision-recall values for the TF-IDF method using the CM1SUB dataset.

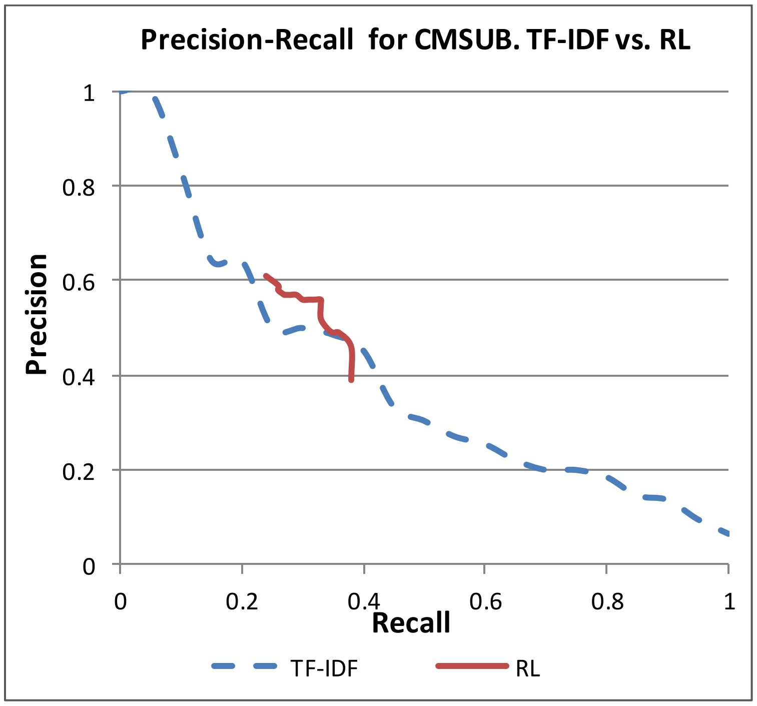


Figure 6.7 Precision-recall curves for TF-IDF and reinforcement learning methods for the CM1SUB

As shown in Figure 6.7, the points in the Precision-recall plane for the RL method have higher precision values than the points for the TF-IDF method. The RL method reaches a precision of 0.61 at recall of 0.24. The TF-IDF method reaches a precision of only 0.5 at a 0.24 recall value.

When comparing recall and precision values for the RL method, recall values grow to 0.38 as precision drops to 0.39. The RL method results also cluster in the area from recall 0.39 and precision 0.39 up to precision value 0.61 at recall 0.24. These values in dicate the RL method does target the relevant candidate links.

For the CM1SUB dataset, the recall and precision numbers were compared between the two overlapping recalls to confirm any statistical difference between the two methods. With values similar to those for the Pine dataset, the RL method covers a limited range of recall values 0.28 to 0.34.

The precision values for the TF-IDF method were interpolated using 20 recall values and 20 precision values for TF-IDF and RL. The null hypothesis and alternative hypotheses were defined as follows:

> **H_0**: There is no difference between the precision values of the TF-IDF interpolated precision-recall graph compared to the precision values for the RL method's precision-recall graph.
>
> **H_1**: There is a difference between the precision values of the TF-IDF interpolated precision-recall graph compared to the precision values for the RL method's precision-recall graph.

The Wilcoxon Signed Ranked method was also used to evaluate the null hypothesis as was done previously for the Pine dataset. The critical value for $Z_{critical}$ test was found to be ±1.96 at confidence level $\alpha = 0.05$. The calculations produced the following values for W_-, W_+ and Z :

- $W_- = -153$,
- $W_+ = 18.5$,
- $Z = -3.07$.

Since our $Z < Z_{critical,}$ as found previously for the Pine dataset, the null hypothesis must also be rejected. This left us to conclude that there is a statistically significant difference between the precision values of the TF-IDF and RL methods on CM1SUB.

6.2.4 Observations

In light of the results obtained from the experiments, we were able to make several interesting observations.

Typically, when we consider a precision-recall curve, we observe three possible outcomes: high recall values and low precision; high precision and low recall; and values in between these two extremes [53], [15],[9].

High precision and low recall implies that we accurately retrieved a small fraction of the required documents, but not most of them. Low precision and high recall implies that we retrieved most of the required documents, but at the same time, we retrieved more unrelated documents as well.

Ideally, when we issue a query we would like to retrieve all the correct documents and no unrelated items. This ideal scenario should provide high recall and high precision values; our precision-recall curve should reside in the upper right area of the graph as shown in Figure 6.8. We would like our precision recall curve to resemble the ideal shape, i.e. move the top right corner of the precision recall graph and raise the lower boundaries on recall and precision values.

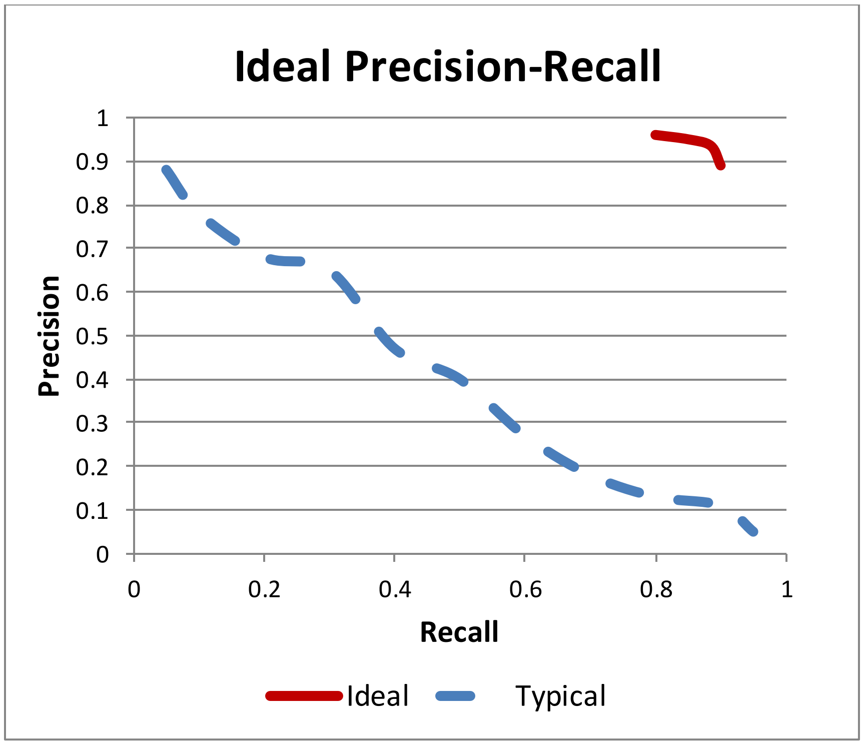


Figure 6.8 Precision-recall curves ideal vs. typical

For both datasets, the RL method demonstrated higher precision values than the TF-IDF method for the same recall values. For the Pine dataset, the RL method reached precision value 0.65 at recall 0.52. The TF-IDF method only reached precision value 0.52 at recall 0.52.

We observed a similar difference in precision between the RL and TF-IDF methods using the CM1SUB dataset. The RL method reached precision 0.61 at recall of 0.24, while TF-IDF reached precision 0.5 at recall of 0.24.

It should be noted that the RL method did not cover the whole spectrum of recall or precision values. The minimum recall for RL on Pine is 0.23; the maximum recall for RL on Pine is 0.52. The minimum precision for RL on Pine is 0.65; the maximum precision for RL on Pine is 0.84.

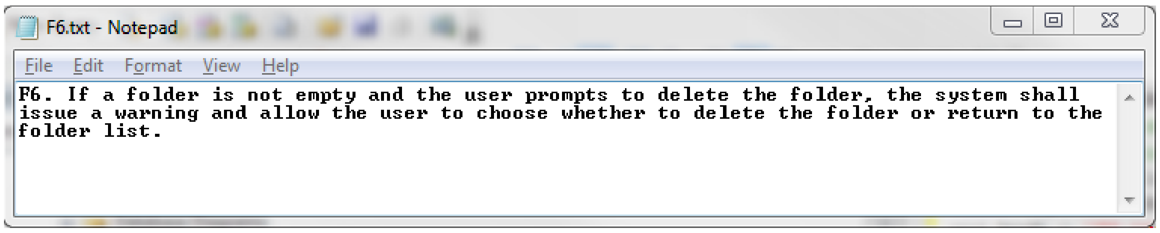
A precision-recall curve for the RL method using the CM1SUB dataset was also limited by min/max values in recall and precision. For CM1SUB, the minimum recall value for RL is 0.24; the maximum recall was 0.38. The minimum precision value for RL was 0.39, the maximum was 0.61.

The precision-recall data points for the RL method on both datasets exhibited a more focused result, compared to the TF-IDF method. However, the TF-IDF method did reach values close to 1 in recall and precision.

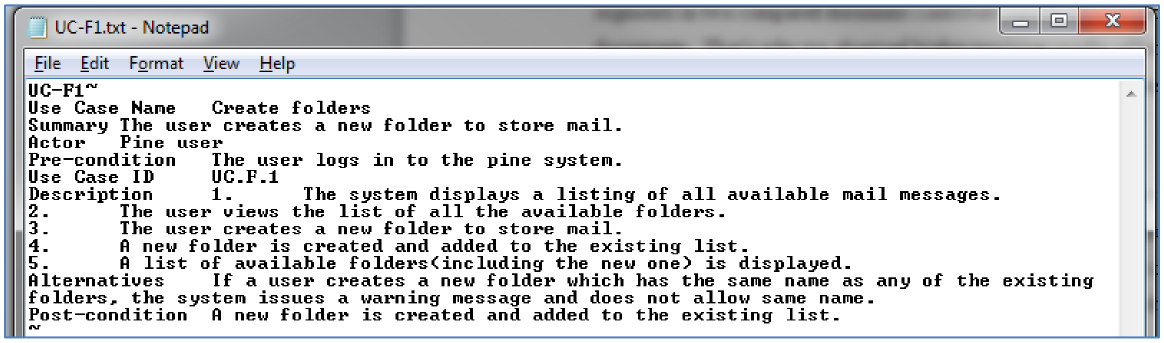
At the same time, when TF-IDF recall reaches to 1, precision drops to almost 0. The same is true for precision: when precision reaches 1, the recall drops close to 0. The RL method recall does not drop below 0.23 for Pine and produces recall higher than 0.24. Also, the lower boundaries for precision on the RL method for the Pine and CM1SUB datasets were 0.37 and 0.39, respectively.

One explanation for the observed trends using the RL method is that the common textual segments in two compared documents contribute significantly to promoting a possible link between the two documents. In other words, the candidate links suggested by the RL method shared common textual segments. This is why the higher precision results are produced in the RL method on both datasets.

The upper boundary on precision for RL for both Pine and CM1SUB datasets is 0.84 and 0.61, respectively. This indicates that having common segments between textual documents is not enough to establish a true link between them. If the RL method links the documents with common segments, the upper boundary on the precision indicates that some documents sharing textual segments may not have a logical link between them. For example, the RL method suggested a link between high level document F6.txt and low level UC-F1.txt as shown in Figure 6.9.



```
F6. If a folder is not empty and the user prompts to delete the folder, the system shall
issue a warning and allow the user to choose whether to delete the folder or return to the
folder list.
```



```
UC-F1~
Use Case Name   Create folders
Summary The user creates a new folder to store mail.
Actor   Pine user
Pre-condition   The user logs in to the pine system.
Use Case ID     UC.F.1
Description     1.      The system displays a listing of all available mail messages.
2.      The user views the list of all the available folders.
3.      The user creates a new folder to store mail.
4.      A new folder is created and added to the existing list.
5.      A list of available folders(including the new one) is displayed.
Alternatives    If a user creates a new folder which has the same name as any of the existing
folders, the system issues a warning message and does not allow same name.
Post-condition  A new folder is created and added to the existing list.
~
```

Figure 6.9 Two documents sharing common segments

By tracing the agents, we can see that the suggested link came as the result of a common segment in F6.txt and UC-F1.txt: “the system shall issue a warning.”

Even though the wording of the segment is the same in both documents, the information carried by this common segment is not sufficient to link the documents. This suggests that not all common textual segments are “created equal.”

At the same time, the lower boundary on the RL method's precision for Pine and CM1SUB datasets does not fall below 0.65 and 0.39, respectively. This fact suggests that the common segments play an important role in identifying correct candidate links between high and low level documents. The portion of the relevant documents returned by the RL method did not fall below 0.65 and 0.39 for Pine and CM1SUB datasets respectively.

With the lower boundaries on precision, the RL method reaches the upper boundaries for recall (0.52 and 0.38). This indicates that the common textual segments may not necessarily uncover all possible ways of linking the documents.

6.2.5 Hard Traces

To evaluate the value of discovering common textual segments using the RL method, we compared the quality of candidate links on the Web Archive tool (WARC) dataset [54]. Figure 6.10 Precision-recall curve for WARC. shows precision-recall curves for the TF-IDF and RL methods on the WARC dataset.

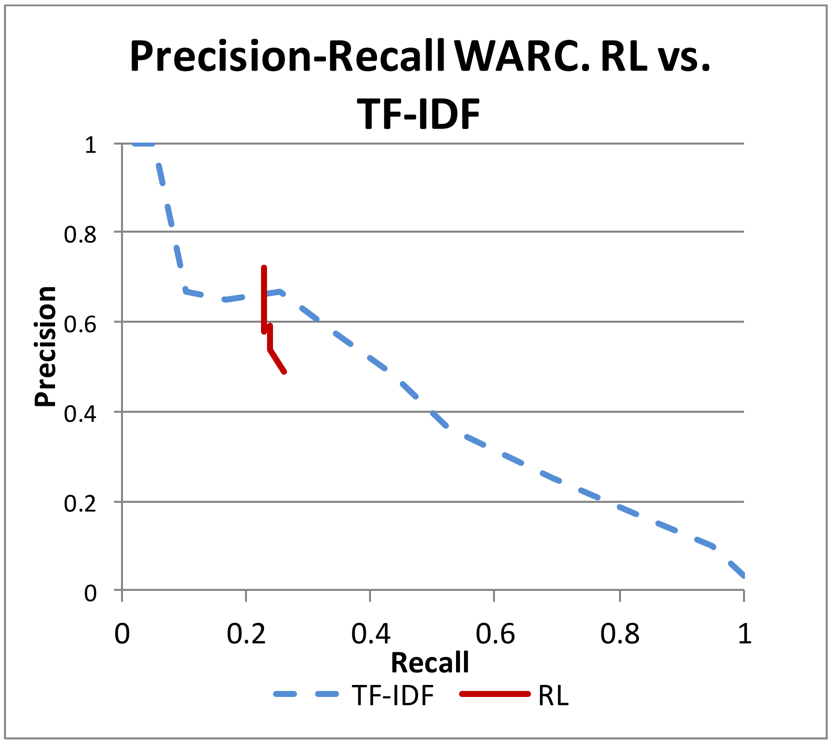
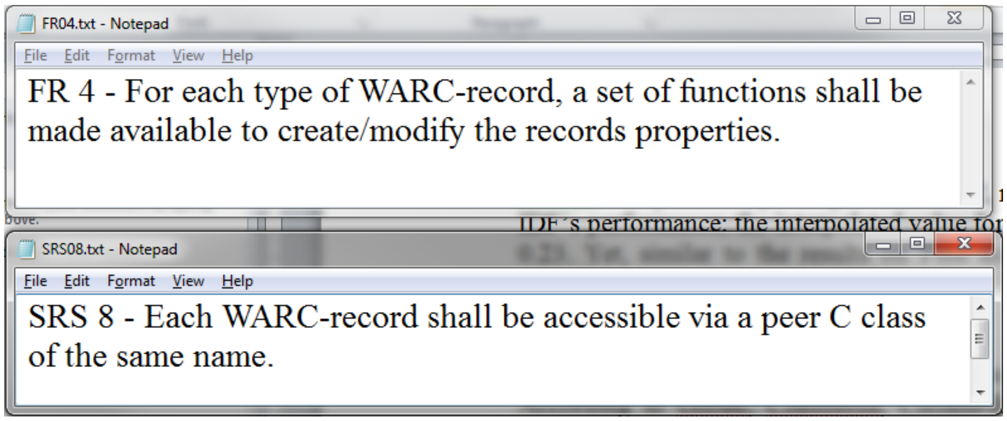


Figure 6.10 Precision-recall curve for WARC.

As shown in Figure 6.10, the points in the precision-recall curve for the RL method are gathered around the recall value of 0.23 and the precision varies from 0.49 to 0.72. The RL method did not demonstrate a significant performance gain with respect to the TF-IDF method. Only on one point (precision 0.72, recall 0.23) did the RL curve exceed TF-IDF's performance; the interpolated value for TF-IDF there is precision of 0.70 at recall of 0.23. Yet, similar to the results on Pine and CM1SUB, the RL method demonstrated focused results: the recall ranged from 0.23 to 0.26 and precision ranged from 0.49 to 0.72.

What interested us in this set of results was the performance on the "hard traces." According to Gibiec, Czauderna, Cleland-Huang, the hard traces exhibit average precision less than 10% and occur when documents do not share common terms or synonyms in a thesaurus [55]. The RL method was able to locate some of the hard traces with 100% recall and precision. For example, table A.3 in the appendix shows 100% recall and precision for FR30.txt, FR33.txt, and FR38.txt. Yet, the RL method completely missed some of the hard traces. If we analyze the "completely" missed links, we can see that the documents comprising the link shared very few common terms. An example of such documents is F04.txt which has the following low level documents in the answer set: SRS08.txt, SRS09, and SRS10.txt. By zooming further into the content of the documents (Figure 6.11), we see only a single pair of terms common between the two documents.

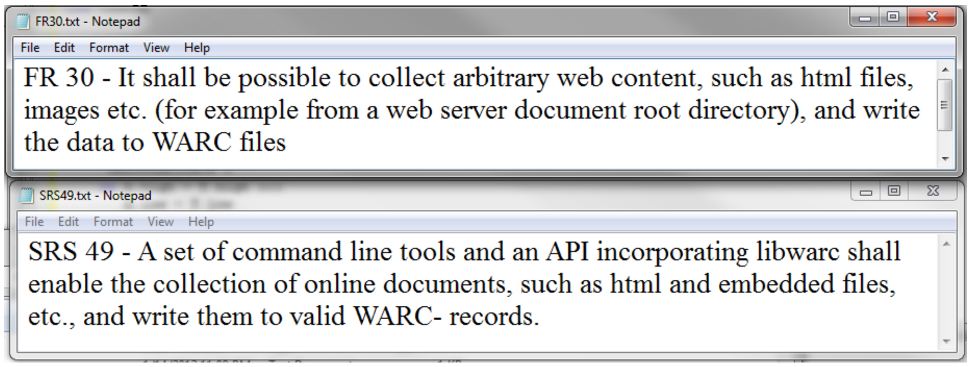


FR 4 - For each type of WARC-record, a set of functions shall be made available to create/modify the records properties.

SRS 8 - Each WARC-record shall be accessible via a peer C class of the same name.

Figure 6.11 Two hard trace documents from WARC dataset comprising missed link

At the same time, the RL method did pick up the documents that comprise the hard to trace links and share common textual segments. Figure 6.12 shows documents FR30.txt and SRS49.txt sharing several common textual segments.



FR 30 - It shall be possible to collect arbitrary web content, such as html files, images etc. (for example from a web server document root directory), and write the data to WARC files

SRS 49 - A set of command line tools and an API incorporating libwarc shall enable the collection of online documents, such as html and embedded files, etc., and write them to valid WARC- records.

Figure 6.12 Two hard trace documents from WARC dataset comprising link discovered by the RL method.

6.3 Future Work

Comparing RL to TF-IDF, which links the documents based on all common terms and their weight, the RL method promotes the links between documents with common terms located close to each other. In other words, the RL method identifies common textual segments between documents, and suggests links between such documents. By doing so, the RL method outperforms the TF-IDF method for the same recall values. RL's higher precision at the same recall rate provides a human analyst with a more compact and focused collection of candidate links.

Considering the encouraging results from the RL method, future work can be directed to incorporate the advantages that the RL method offers. Future work will incorporate a feedback mechanism similar to the one in Mencer's work [49]. Feedback may improve the accuracy of the generated candidate links.

Also, combining the feedback with personalized filtering, similar to Seo's work [45], should definitely affect the accuracy of the candidate links. A part of speech tagging or noun-verb phrasing technique [27] shall also be considered in future work. By classifying terms in textual documents, we can amplify the importance of one type of textual segment over others.

Appendix

Table 0.1. Detailed results for the TF-IDF and pheromone swarm methods on the Pine and CM1 datasets

Pine

TF-IDF

Threshold	Recall	Precision	F	F2	DiffAR	MAP
0.1	0.72	0.48	0.58	0.66	0.18	0.75
0.2	0.43	0.65	0.51	0.46	0.25	0.58
0.3	0.24	0.82	0.37	0.28	0.41	0.38
0.4	0.15	0.97	0.27	0.19	0.55	0.27
0.5	0.08	0.95	0.15	0.10	0.61	0.16
0.6	0.04	1.00	0.08	0.05	0.69	0.08
0.7	0.02	1.00	0.03	0.02	0.75	0.04
0.8	0.00	1.00	0.01	0.01	0.80	0.00

Simple Swarm

Threshold	Recall	Precision	F	F2	DiffAR	MAP
0.1	0.86	0.27	0.41	0.60	0.46	0.76
0.2	0.78	0.40	0.53	0.66	0.41	0.74
0.3	0.65	0.52	0.58	0.62	0.44	0.67
0.4	0.55	0.63	0.59	0.57	0.46	0.61
0.5	0.48	0.71	0.58	0.52	0.57	0.57
0.6	0.40	0.76	0.53	0.44	0.66	0.52
0.7	0.34	0.81	0.48	0.39	0.72	0.47
0.8	0.28	0.83	0.41	0.32	0.85	0.41
0.9	0.22	0.86	0.35	0.26	0.93	0.36

Pheromone Swarm δ=1

Threshold	Recall	Precision	F	F2	DiffAR	MAP
0.1	0.63	0.54	0.58	0.61	0.38	0.68
0.2	0.46	0.63	0.53	0.49	0.43	0.57
0.3	0.33	0.64	0.44	0.37	0.58	0.48
0.4	0.28	0.66	0.39	0.31	0.64	0.45
0.5	0.25	0.69	0.37	0.29	0.73	0.42
0.6	0.23	0.73	0.35	0.27	0.76	0.40
0.7	0.21	0.78	0.33	0.24	0.89	0.39
0.8	0.19	0.84	0.31	0.23	0.93	0.37
0.9	0.18	0.87	0.30	0.22	0.95	0.37

Pheromone Swarm δ=3

Threshold	Recall	Precision	F	F2	DiffAR	MAP
0.1	0.59	0.54	0.56	0.58	0.44	0.66
0.2	0.42	0.62	0.50	0.45	0.52	0.54
0.3	0.35	0.67	0.46	0.39	0.60	0.51
0.4	0.30	0.74	0.42	0.34	0.70	0.47
0.5	0.27	0.77	0.40	0.31	0.75	0.45
0.6	0.24	0.81	0.37	0.28	0.88	0.42
0.7	0.22	0.84	0.34	0.25	0.93	0.40
0.8	0.20	0.87	0.32	0.23	0.95	0.37
0.9	0.19	0.89	0.31	0.23	0.98	0.37

Pheromone Swarm δ=5

Threshold	Recall	Precision	F	F2	DiffAR	MAP
0.1	0.52	0.52	0.52	0.52	0.45	0.58
0.2	0.38	0.63	0.48	0.41	0.52	0.49
0.3	0.33	0.69	0.45	0.37	0.60	0.46
0.4	0.28	0.71	0.40	0.31	0.67	0.41
0.5	0.24	0.71	0.35	0.27	0.75	0.38
0.6	0.23	0.77	0.35	0.26	0.79	0.38
0.7	0.21	0.81	0.34	0.25	0.84	0.37
0.8	0.19	0.85	0.31	0.23	0.93	0.36
0.9	0.18	0.86	0.30	0.21	1.00	0.33

CM-1

TF-IDF

Threshold	Recall	Precision	F	F2	DiffAR	MAP
0.1	0.82	0.08	0.14	0.28	0.09	0.16
0.2	0.50	0.19	0.27	0.37	0.10	0.20
0.3	0.25	0.32	0.28	0.26	0.22	0.15
0.4	0.09	0.31	0.14	0.10	0.34	0.06
0.5	0.03	0.45	0.05	0.03	0.47	0.02
0.6	0.01	0.57	0.02	0.01	0.66	0.01
0.7	0.00	0.50	0.01	0.00	0.75	0.00

Simple Swarm

Threshold	Recall	Precision	F	F2	DiffAR	MAP
0.1	0.80	0.04	0.07	0.15	0.28	0.23
0.2	0.66	0.05	0.10	0.20	0.23	0.22
0.3	0.50	0.07	0.12	0.22	0.19	0.21
0.4	0.41	0.08	0.14	0.23	0.24	0.19
0.5	0.35	0.11	0.16	0.24	0.23	0.18
0.6	0.26	0.13	0.17	0.22	0.34	0.16
0.7	0.22	0.15	0.17	0.20	0.46	0.15
0.8	0.19	0.17	0.18	0.19	0.57	0.14
0.9	0.16	0.19	0.17	0.16	0.72	0.13

Pheromone Swarm with δ=1

Threshold	Recall	Precision	F	F2	DiffAR	MAP
0.1	0.58	0.07	0.13	0.24	0.27	0.14
0.2	0.44	0.10	0.17	0.27	0.23	0.15
0.3	0.37	0.12	0.18	0.26	0.21	0.16
0.4	0.30	0.13	0.19	0.24	0.31	0.14
0.5	0.26	0.15	0.19	0.23	0.35	0.15
0.6	0.23	0.17	0.19	0.21	0.41	0.15
0.7	0.20	0.18	0.19	0.20	0.57	0.15
0.8	0.19	0.20	0.19	0.19	0.62	0.14
0.9	0.16	0.21	0.18	0.17	0.82	0.14

Pheromone Swarm with δ=3

Threshold	Recall	Precision	F	F2	DiffAR	MAP
0.1	0.56	0.08	0.13	0.25	0.26	0.22
0.2	0.40	0.10	0.16	0.25	0.27	0.20
0.3	0.34	0.12	0.18	0.25	0.28	0.19
0.4	0.30	0.14	0.19	0.25	0.29	0.18
0.5	0.27	0.16	0.20	0.24	0.35	0.17
0.6	0.25	0.19	0.21	0.23	0.40	0.17
0.7	0.22	0.21	0.21	0.22	0.50	0.16
0.8	0.18	0.20	0.19	0.19	0.69	0.14
0.9	0.17	0.22	0.19	0.18	0.77	0.13

Pheromone Swarm with δ=5

Threshold	Recall	Precision	F	F2	DiffAR	MAP
0.1	0.48	0.08	0.13	0.23	0.27	0.21
0.2	0.38	0.11	0.17	0.25	0.25	0.19
0.3	0.31	0.12	0.18	0.24	0.25	0.18
0.4	0.28	0.14	0.18	0.23	0.31	0.18
0.5	0.24	0.15	0.18	0.21	0.39	0.16
0.6	0.22	0.16	0.18	0.20	0.55	0.16
0.7	0.20	0.19	0.20	0.20	0.65	0.15
0.8	0.18	0.20	0.19	0.18	0.77	0.14
0.9	0.17	0.21	0.19	0.17	0.86	0.13

Table 0.2. Detailed results for the TF-IDF and RL methods on the Pine and CM1SUB datasets

Pine

TF-IDF

Threshold	Recall	Precision	Returned	Relevant	Answerset
0	0.992	0.115	2132	246	248
0.05	0.891	0.304	728	221	248
0.1	0.722	0.481	372	179	248
0.15	0.524	0.565	230	130	248
0.2	0.427	0.654	162	106	248
0.25	0.302	0.714	105	75	248
0.3	0.238	0.819	72	59	248
0.35	0.19	0.922	51	47	248
0.4	0.153	0.974	39	38	248
0.45	0.109	0.964	28	27	248
0.5	0.081	0.952	21	20	248
0.55	0.052	1	13	13	248
0.6	0.04	1	10	10	248
0.65	0.036	1	9	9	248
0.7	0.016	1	4	4	248
0.75	0.004	1	1	1	248
0.8	0.004	1	1	1	248
0.85	0	0	0	0	248
0.9	0	0	0	0	248
0.95	0	0	0	0	248

CMSUB

TF-IDF

Threshold	Recall	Precision	Returned	Relevant	Answerset
0	1	0.05	893	45	45
0.05	0.933	0.101	414	42	45
0.1	0.822	0.18	206	37	45
0.15	0.622	0.255	110	28	45
0.2	0.422	0.365	52	19	45
0.25	0.311	0.5	28	14	45
0.3	0.222	0.556	18	10	45
0.35	0.2	0.643	14	9	45
0.4	0.111	0.833	6	5	45
0.45	0.067	1	3	3	45
0.5	0.044	1	2	2	45
0.55	0	0	0	0	45
0.6	0	0	0	0	45
0.65	0	0	0	0	45
0.7	0	0	0	0	45
0.75	0	0	0	0	45
0.8	0	0	0	0	45
0.85	0	0	0	0	45
0.9	0	0	0	0	45
0.95	0	0	0	0	45

RL

Threshold	Recall	Precision	Returned	Relevant	Answerset
0.05	0.51	0.64	196	126	248
0.1	0.46	0.69	166	115	248
0.15	0.44	0.71	155	110	248
0.2	0.43	0.72	147	106	248
0.25	0.42	0.73	141	103	248
0.3	0.38	0.74	129	95	248
0.35	0.37	0.75	122	92	248
0.4	0.35	0.74	117	87	248
0.45	0.34	0.74	114	84	248
0.5	0.31	0.74	106	78	248
0.55	0.3	0.74	101	75	248
0.6	0.29	0.76	94	71	248
0.65	0.28	0.78	89	69	248
0.7	0.27	0.8	85	68	248
0.75	0.27	0.8	84	67	248
0.8	0.26	0.82	79	65	248
0.85	0.25	0.83	76	63	248
0.9	0.25	0.83	76	63	248
0.95	0.24	0.82	73	60	248
1	0.23	0.84	69	58	248

RL

Threshold	Recall	Precision	Returned	Relevant	Answerset
0.05	0.38	0.39	44	17	45
0.1	0.38	0.46	37	17	45
0.15	0.36	0.5	32	16	45
0.2	0.33	0.5	30	15	45
0.25	0.33	0.52	29	15	45
0.3	0.33	0.54	28	15	45
0.35	0.33	0.56	27	15	45
0.4	0.33	0.56	27	15	45
0.45	0.33	0.58	26	15	45
0.5	0.31	0.56	25	14	45
0.55	0.29	0.54	24	13	45
0.6	0.29	0.57	23	13	45
0.65	0.27	0.57	21	12	45
0.7	0.27	0.57	21	12	45
0.75	0.27	0.57	21	12	45
0.8	0.27	0.57	21	12	45
0.85	0.27	0.57	21	12	45
0.9	0.27	0.6	20	12	45
0.95	0.27	0.63	19	12	45
1	0.27	0.63	19	12	45

Table 0.3. Detailed results for the TF-IDF and RL methods on the Pine and CM1SUB datasets

TF IDF

High Level Doc	Average Recall	Average Precision
FR03.txt	0.30	0.07
FR04.txt	0.10	0.01
FR05.txt	0.08	0.00
FR06.txt	0.22	0.06
FR08.txt	0.20	0.02
FR11.txt	0.10	0.01
FR12.txt	0.15	0.01
FR15.txt	0.30	0.08
FR18.txt	0.20	0.03
FR20.txt	0.13	0.01
FR27.txt	0.23	0.05
FR28.txt	0.20	0.03
FR29.txt	0.20	0.03
FR30.txt	0.35	0.09
FR33.txt	0.40	0.10
FR34.txt	0.30	0.08
FR38.txt	0.40	0.09

RL

High Level Doc	Average Recall	Average Precision
FR03.txt	1.00	0.75
FR04.txt	0.00	0.00
FR05.txt	0.00	0.00
FR06.txt	0.00	0.00
FR08.txt	0.00	0.00
FR11.txt	0.00	0.00
FR12.txt	0.00	0.00
FR15.txt	0.00	0.00
FR18.txt	0.00	0.00
FR20.txt	0.00	0.00
FR27.txt	0.00	0.00
FR28.txt	0.00	0.00
FR29.txt	0.00	0.00
FR30.txt	1.00	1.00
FR33.txt	1.00	1.00
FR34.txt	0.21	0.11
FR38.txt	1.00	1.00

References

[1] M. Dowson, “The Ariane 5 software failure,” *SIGSOFT Softw. Eng. Notes*, vol. 22, no. 2, p. 84, 1997.

[2] R. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. McGraw-Hill Science/Engineering/Math, 2009.

[3] C. Jones, *Patterns of Software System Failure and Success*. Intl Thomson Computer Pr (Sd), 1995.

[4] B. Boehm and V. R. Basili, “Software Defect Reduction Top 10 List,” *Computer*, vol. 34, no. 1, pp. 135–137, 2001.

[5] B. W. Boehm, *Software Engineering Economics*. Prentice Hall, 1981.

[6] “SWEBOK Guide - Chapter 2,” 20:14:23. [Online]. Available: http://www.computer.org/portal/web/swebok/html/ch2. [Accessed: 31-Aug-2010].

[7] IEEE, *IEEE Standard 610.12-90 IEEE Standard Glossary of Software Engineering Terminology*. IEEE, 1990.

[8] J. H. Hayes, A. Dekhtyar, and S. Sundaram, “Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods,” *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 4 – 19, 2006.

[9] J. H. Hayes, A. Dekhtyar, S. Sundaram, E. Holbrook, S. Vadlamudi, and A. April, “REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery,” *Innovations in Systems and Software Engineering*, vol. 3, no. 3, pp. 193–202, 2007.

[10] T. Mundie and F. Hallsworth, “Requirements analysis using SuperTrace PC,” 1995.

[11] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice Hall, 2002.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[13] H. Sultanov and J. H. Hayes, “Application of Swarm Techniques to Requirements Engineering: Requirements Tracing,” presented at the 18th Intl. Requirement Engineering Conference, Sydney, Australia, 2010.

[14] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, 1st ed. Cambridge University Press, 2008.

[15] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press, Addison-Wesley, 1999.

[16] J. Matthias, “Requirements Tracing,” *Communications of the ACM*, vol. 41, no. 12, 1998.

[17] Deneubourg, S. Aron, S. Goss, and J. Pasteels, “The self-organizing exploratory pattern of the argentine ant,” *Journal of Insect Behavior*, vol. 3, no. 2, pp. 168, 159, Mar. 1990.

[18] O. C. Z. Gotel and A. C. W. Finkelstein, “An Analysis of the Requirements Traceability Problem,” pp. 94–101, 1994.

[19] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, “Recovering Traceability Links between Code and Documentation,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 970–983, 2002.

[20] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhan, and S. Christina, “Goal-Centric Traceability for Managing Non-Functional Requirements,” 2005.

[21] P. Lago, H. Muccini, and H. van Vliet, "A scoped approach to traceability management," *Journal of Systems and Software*, vol. 82, no. 1, pp. 168–182, Jan. 2009.

[22] M. C. Panis, "Successful Deployment of Requirements Traceability in a Commercial Engineering Organization...Really," Sydney, Australia, 2010.

[23] A. Egyed, P. Grünbacher, and F. Graf, "Effort and Quality of Recovering Requirements-to-Code Traces: Two Exploratory Experiments," Sydney, Australia, 2010.

[24] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, p. 13–es, Sep. 2007.

[25] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," Washington, DC, USA, 2003, pp. 125–135.

[26] J. H. Hayes, A. Dekhtyar, and J. Osborne, "Improving Requirements Tracing via Information Retrieval," 2003, p. 138.

[27] X. Zou, R. Settimi, and J. Cleland-Huang, "Improving automated requirements trace retrieval: a study of term-based enhancement methods," *Empir Software Eng*, vol. 15, no. 2, pp. 119–146, Jul. 2009.

[28] Y. S. Maarek, D. M. Berry, and G. E. Kaiser, "An Information Retrieval Approach for Automatically Constructing Software Libraries," *IEEE Transactions on Software Engineering*, vol. 17, pp. 800–813, 1991.

[29] N. Niu and S. Easterbrook, "Extracting and Modeling Product Line Functional Requirements," Los Alamitos, CA, USA, 2008, pp. 155–164.

[30] A. Zisman, G. Spanoudakis, E. Perez-Minana, and P. Krause, "Towards a Traceability Approach for Product Family Requirements," 2002.

[31] S. Sundaram, J. H. Hayes, A. Dekhtyar, and A. Holbrook, "Assessing traceability of software engineering artifacts," *Requirements Engineering Journal*, vol. early view, Jan. 2010.

[32] G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause, "Rule-Based Generation of Requirements Traceability Relations," *Journal of Systems and Software*, vol. 72, no. 2, pp. 105–127, 2004.

[33] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, 1st ed. Oxford University Press, USA, 1999.

[34] D. W. van der Merwe and A. P. Engelbrecht, "Data clustering using particle swarm optimization," presented at the 2003 Congress on Evolutionary Computation, Canberra, ACT, Australia, 03:04:17, pp. 215–220.

[35] X. Cui, T. E. Potok, and P. Palathingal, "Document Clustering using Particle Swarm Optimization," *IEEE SWARM INTELLIGENCE SYMPOSIUM, THE WESTIN*, 2005.

[36] E. Diaz-Aviles, W. Nejdl, and L. Schmidt-Thieme, "Swarming to rank for information retrieval," Montreal, Québec, Canada, 2009, pp. 9–16.

[37] M. H. Aghdam, N. Ghasem-Aghaee, and M. E. Basiri, "Text feature selection using ant colony optimization," *Expert Syst. Appl.*, vol. 36, no. 3, pp. 6843–6853, 2009.

[38] H. Azzag, N. Monmarche, M. Slimane, and G. Venturini, "AntTree: a new model for clustering with artificial ants," presented at the Evolutionary Computation, 2003. CEC '03. The 2003 Congress on, 2003, vol. 4, pp. 2642–2647 Vol.4.

[39] A. Abraham and V. Ramos, “Web usage mining using artificial ant colony clustering and linear genetic programming,” presented at the Evolutionary Computation, 2003. CEC ’03. The 2003 Congress on, 2003, vol. 2, pp. 1384–1391 Vol.2.

[40] H. Li and C. P. Lam, “Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams,” in *Testing of Communicating Systems*, vol. 3502, F. Khendek and R. Dssouli, Eds. Springer Berlin / Heidelberg, 2005, pp. 405–405.

[41] G. Spanoudakis, A. d’ Avila-Garces, and A. Zisman, *Revising Rules to Capture Requirements Traceability Relations: A Machine Learning Approach*. 2003.

[42] A. S. d’ Avila Garcez, A. Russo, B. Nuseibeh, and J. Kramer, “Combining abductive reasoning and inductive learning to evolve requirements specifications,” *Software, IEE Proceedings -*, vol. 150, no. 1, pp. 25 – 38, Feb. 2003.

[43] J. Hawthorne, “Inductive Logic,” in *The Stanford Encyclopedia of Philosophy*, Winter 2012., E. N. Zalta, Ed. 2012.

[44] P. A. Flach and A. C. Kakas, Eds., *Abduction and Induction: Essays on their Relation and Integration*, Softcover reprint of hardcover 1st ed. 2000. Springer, 2010.

[45] Y.-W. Seo and B.-T. Zhang, “A reinforcement learning agent for personalized information filtering,” in *Proceedings of the 5th international conference on Intelligent user interfaces*, New York, NY, USA, 2000, pp. 248–251.

[46] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, “A machine learning approach for tracing regulatory codes to product specific requirements,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, New York, NY, USA, 2010, pp. 155–164.

[47] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, “Software traceability with topic modeling,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, New York, NY, USA, 2010, pp. 95–104.

[48] V. Hatzivassiloglou, J. L. Klavans, and E. Eskin, “Detecting Text Similarity over Short Passages: Exploring Linguistic Feature Combinations via Machine Learning,” in *IN PROCEEDINGS OF THE 1999 JOINT SIGDAT CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING AND VERY LARGE CORPORA*, 1999, pp. 203–212.

[49] F. Menczer and R. K. Belew, “Adaptive information agents in distributed textual environments,” in *Proceedings of the second international conference on Autonomous agents*, New York, NY, USA, 1998, pp. 157–164.

[50] H. Sultanov, J. H. Hayes, and W.-K. Kong, “Application of swarm techniques to requirements tracing,” *Requirements Eng*, Jun. 2011.

[51] “Pine Email System,” 19:03:51. [Online]. Available: http://www.washington.edu/pine/. [Accessed: 19-Feb-2010].

[52] M. D. P. Website, *CM-1 Project*. .

[53] S. T. Dumais, G. W. Furnas, T. K. Landauer, and S. Deerwester, “Using latent semantic analysis to improve information retrieval,” 1988, pp. 281–285.

[54] “WARC, Web Archive Eighteen (18),” 19:03:51. [Online]. Available: http://code.google.com/p/warc-tools/. [Accessed: 19-Feb-2010].

[55] M. Gibiec, A. Czauderna, and J. Cleland-Huang, “Towards mining replacement queries for hard-to-retrieve traces,” in *Proceedings of the IEEE/ACM international*

conference on Automated software engineering, New York, NY, USA, 2010, pp. 245–254.

Vita

Date and Place of Birth:
Tashkent, USSR

Education:
Tashkent State University, May 1995
Diploma. Applied Mathematics and Mechanics

Rochester Institute of Technology
Bachelor of Science in Applied Mathematics, August 1994

Oklahoma State University
Master in Computer Science. August 1997

Professional Positions Held:
Citibank, Technology Head
Almaty, Kazakhstan
December 1997-October 2002

SMC, Programmer
Lexington, Kentucky
April 2004 – August 2008

ACS, Software Developer
Lexington, KY
August 2008 – Dec 2011

Kinemetrix, Project Engineer
Lexington, KY
Jan 2012 – present

Professional Publications:
H. Sultanov and J.H. Hayes, "Application of Swarm Techniques to Requirements Engineering: Requirements Tracing," Sydney, Australia: 2010.

H. Sultanov, J.H. Hayes, and W.-K. Kong, "Application of swarm techniques to requirements tracing," Requirements Engineering, Jun. 2011

H. Sultanov, W.-K. Kong, Jane Hayes, W Li, "Software Verification and Validation Research Laboratory (SVVRL) of the University of Kentucky: Traceability Challenge 2011: Language Translation," TEFSE 2011, ICSE workshop

H. Sultanov, J.H. Hayes, "Application of Reinforcement Learning Requirements Engineering: Requirements tracing," Requirements Engineering Conference, Jul. 2013 (accepted)