Application of Swarm Techniques to Requirements Engineering: Requirements Tracing

Hakim Sultanov, Jane Huffman Hayes
University of Kentucky
Lexington, KY. USA
hakim.sultanov@uky.edu, hayes@cs.uky.edu

Abstract—**We posit that swarm intelligence can be applied to effectively address requirements engineering problems. Specifically, this paper demonstrates the applicability of swarm intelligence to the requirements tracing problem using a simple ant colony algorithm. The technique has been validated using two real-world datasets from two problem domains. The technique can generate requirements traceability matrices (RTMs) between textual requirements artifacts (high level requirements traced to low level requirements, for example) with equivalent or better accuracy than traditional information retrieval techniques.**

Keywords – information retrieval; requirements traceability; swarms; software engineering;

I. Introduction

Software requirements are often an overlooked artifact, far too many practitioners forge ahead with coding before understanding the problem that needs to be solved. There is much evidence to show that the lack of requirements, or of quality requirements, inevitably leads to low software quality [1]. Requirements are even more vital when developing mission- or safety-critical systems, where the consequences of poor quality can be the loss of life or damage to the environment.

A number of activities can be undertaken to improve the quality of requirements. These include, but are not limited to, requirements consistency checking (functional and non-functional), interface requirements consistency checking, requirements reading (to look for fault types such as ambiguous requirements, incomplete requirements, etc.), and requirements tracing (to ensure that requirements are addressed in subsequent artifacts). Some of these activities have been supported by automated techniques. In general, though, techniques are not fully automatic; are not general-purpose; have not been validated on large, real-world systems in numerous domains; still require much effort on the part of the human analyst, etc. Thus, researchers continue to search for new and better techniques to improve the quality of requirements.

Swarm techniques are techniques that apply swarm intelligence – swarm intelligence is a property of a non-centralized group of non-intelligent self-organized agents that collectively behave to perform work [2]. Swarm techniques have been demonstrated to work well on problems in nature such as finding the shortest route to a food source, cooperating in carrying large items, etc. In computer science, swarm techniques have assisted with problems such as network routing/network management, power plant maintenance scheduling, load balancing in distributed systems, image compression, personalized web search, etc. Swarm techniques are useful because they exhibit emerging behavior not found in deterministic techniques, and often overcome issues such as the problem of local minima.

Researchers have successfully applied swarm techniques to a number of problems in software maintenance. For example, Antoniol, Penta, and Harman [3] apply a number of such techniques to the problem of project planning for a large maintenance project. Lam, Xiao, Li [4] apply ant colony optimization (ACO) to the problem of test sequence generation. Reitz [5] examined the use of an ant colony in building a software evolvability component model. Ayari, Bouktif, and Antoniol [6] used ACO to perform test input generation. Such applications are lacking in requirements engineering.

These techniques may be more computationally complex than other techniques that have been widely applied in requirements engineering (such as information retrieval techniques for requirements tracing). However, it is quite possible that the techniques can be adapted in such a way as to still permit practical application to requirements engineering problems. An example of method adaptation can be found in requirements tracing: the application of information retrieval techniques was adapted to take advantage of unique properties of the requirements engineering domain, namely small datasets, queries that may be dependent, queries known in advance, etc.

Similarly, we have examined the characteristics of the requirements engineering domain and have found that they lend themselves well to swarm techniques: small problem size or search space, decomposable problem, a certain degree of non-determinism, etc. Of specific applicability to requirements engineering is the *emergent* behavior feature of swarm intelligence methods. Emergent behavior is a byproduct of the main activity of an agent in a swarm. For example, an ant prefers a path with a stronger pheromone over one with a weaker pheromone. As a result of this behavior of an individual ant, the colony of ants can establish shorter routes.

Porting this idea to requirements engineering, we can model a swarm in such a way that an individual "software ant," with limited reasoning logic, can discover a target item and bring the whole colony to it. Then a group of "ants" can make a decision about the item in a collective manner. For example, consider the problem of classifying requirements (as non-functional, functional; or as high risk, low risk). We can provide some logic for the swarm to cluster the requirements by determining certain important characteristics of what makes a requirement functional/non-functional or low/high risk, thus adapting the agents to target requirements engineering problems of interest. This paper demonstrates such an adaptation to the problem of requirements tracing.

We chose requirements tracing as the first problem on which to demonstrate the swarm techniques as it is well known and there are well established benchmarks for acceptable performance of methods. Also, there is prior work in information retrieval that successfully uses swarm techniques to rank retrieved information [7], an important aspect of requirements tracing.

We introduce a simplified ant colony algorithm for tracing textual pairs of requirements artifacts. We validated the technique on two sets of software requirements from real projects, comparing the results to those of a typical information retrieval (IR) tracing technique. We found that our technique outperformed the IR method for the smallest dataset and outperformed the IR method for the large dataset at higher threshold values.

The paper is organized as follows. Section II provides background on ant colony optimization. Section III discusses requirements tracing. Section IV presents related work. Section V discusses our approach to tracing using the ant colony algorithm. Section VI presents the validation of the technique on two datasets. Section VII presents the results and analysis. Section VIII concludes and discusses future work.

II. Ant Colony Optimization

Insects such as bees and ants, small and simple individually, can accomplish tremendous tasks in a collective effort. Of particular interest to computer scientists is 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 a self-organizing behavior of ant colonies, where ants used pheromone communication [8]. The idea of using pheromone trails as a method of communicating through the environment is the heart of the Ant Colony Optimization (ACO) algorithm [9]. 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. We present the algorithm below.

Consider a graph G = (V, E) and a task of finding the shortest path between two nodes in the graph. For each edge 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, $\tau_{ij}(0)$. Also, a group of «ants» k = 1,...,n is positioned at the source node.

For each iteration, each ant builds a path to the destination node. Also, at each node, each ant decides the next link to take. If ant k is at node i, the probability ***p*** of selecting next node $j \in N_i^k$, which belongs to a set of nodes adjacent to ***i*** [15].

$$p = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)}{\sum_{j \in N_i^k} \tau_{ij}^{a}(t)} & if \quad j \in \mathrm{N}_i^k \\ \quad or & \\ \mathrm{p}_{ij}^{k}(t) = 0 \text{ if } j \notin \mathrm{N}_i^k & \end{cases} \tag{1}$$

In the formula above, « α » is a parameter which amplifies the attractiveness of the pheromone trail.

III. Requirements Tracing

Requirements tracing is defined as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction [10]. A typical process used for requirements tracing of natural language artifacts, manual or automated, generally consists of a number of steps: document parsing, candidate link generation, candidate link evaluation, and traceability analysis [11]. For example, if a requirements document is being traced to a set of use cases, *document parsing* extracts elements from the two artifacts resulting in uniquely identified requirements elements and uniquely identified use case elements. At this point, a human analyst or tool will find relationships or links between the elements, perhaps by selecting one requirement element and then performing string searches (using important words or terms in that element) into the collection of use case elements. If a tool is being used, it may be the case that the human analyst or tool assigns keywords to the requirements and use case elements and then performs keyword matching in order to generate what are called "*candidate links*." Candidate link evaluation deals with assessing the links to ensure that they are correct and *traceability analysis* deals with deciding if the high level artifact has been "*satisfied*" by the lower level artifact (e.g., are there use cases to satisfy a given requirement?). In this work, we concentrate on adapting the swarm technique to the candidate link generation problem.

A. Terminology

First, we define some terminology. The high and low level textual elements are called *documents*. The documents contain *terms*. The collection of all terms from all documents is called the *dictionary* or *vocabulary*. The collection of all terms in a document is called document *corpus*. *The inverted index* is a list of documents listing all documents where a particular term occurs. *Term frequency* $TF_{t,d}$ is the count of how many times a particular term occurs in all documents. *Inverse document frequency*, IDF_t, is a calculated value:

$$\mathrm{IDF_t} = \log\left(\frac{N}{DF_t}\right), \tag{2}$$

where N is the total number of documents in the collection, and DF_t is document frequency, i.e., number of documents where a given term occurs.

To trace high level textual elements (say from a requirements document) to low level textual elements (say from a design document), we use swarm agents that traverse the collection of all documents and the vocabulary shared by the documents.

B. *Measurements*

The tracing results are compared to an answer set of correct or "true" links, that has been prepared by experts, and then evaluated using *Recall* and *Precision* measurements [12].

Recall is evaluated as the total number relevant retrieved documents divided by total number of relevant documents in the whole collection:

$$Recall = \frac{\#of\_relevant\_retrieved}{\#\_relevant\_in\_collection} \tag{3}$$

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

$$Precision = \frac{\#of\_relevant\_retrieved}{\#\_retrieved} \tag{4}$$

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

$$F = \frac{(\beta^2+1)P*R}{\beta^2 P+R} \text{,where } \beta^2 \in [0,\infty) . \tag{5}$$

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

IV. Related Work

We address related work in the areas of traceability link generation and swarm techniques below.

A. *Candidate Link Generation/Text Analysis*

As mentioned in Section III, candidate link generation is concerned with retrieving relevant elements from a given artifact pair. We focus on textual requirements artifacts. Much work has been accomplished in applying information retrieval techniques to the candidate link generation problem. Antoniol, Canfora, Casazza, DeLucia, and Merlo [13] used the vector space model and a probabilistic model to recover traceability from source code modules to man pages and functional requirements. With VSM, they achieved the highest recall (100%) for the Albergate dataset by setting the threshold to 10% of the highest similarity measure, but only achieved precision of 11.98%. Marcus and Maletic [13] applied latent semantic indexing (LSI) to the same datasets used by Antoniol, Canfora, Casazza, DeLucia, and Merlo. They achieved precision of 16.38% at 100% recall for the Albergate dataset. Hayes, Dekhtyar, and Osborne [14] applied the vector space model (VSM) with thesaurus to a dataset and compared this method to manual tracing and to a proprietary tool. They found that manual tracing resulted in higher precision (46%) than the proprietary tool (38.8%) or the VSM + thesaurus method (40.7%), but that the VSM+thesaurus method outperformed the other two approaches in terms of recall (85.4% compared to 43.9% for manual and 63.4% for proprietary tool). Zou, Settimi, and Cleland-Huang [15] examined ways to improve the precision of automated IR traces by using phrasing, obtaining improvements of almost 20% for one dataset when examining the Top 5% of the returned candidate links. Zisman and Spanoudakis [16] examined ways to generate traceability links by applying rules to artifacts that had been tagged with parts of speech.

In general, the above techniques have been able to achieve excellent recall [14] but often at the expense of precision that is not acceptable or is only borderline acceptable. Our work differs in that it uses a greedy algorithm approach to generate candidate link lists, it does not require parts of speech tagging, phrasing, specification of probabilities, the inverted dictionary in addition to links from terms to documents, tags the links with high or low level document identifier. We use the VSM as a baseline against which to compare the performance of our method.

There are a number of researchers who have applied the Particle Swarm Optimization (PSO) algorithm to the problem of analyzing textual documents. PSO is a direct search method for some 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 pace. The swarm agents are modeled to have a position in a search space and a velocity. The agents iteratively evaluate some fitness function. Typically, the agents' position and velocity are used as input parameters for the fitness function. The agents operate on the premise of their own "best" position and the swarm's and the neighbors' "best" position. The "best" implies a point in the search space where the fitness function has reached some optimal value [17].

B. *Swarm techniques*

Merwe and Engelbrecht applied data clustering using PSO on six different classification problems [18] . The problems ranged from 400 vectors randomly created in 2 dimensional space to Wisconsin breast cancer database, with the objective to classify data as benign or malignant tumors. Diaz-Aviles and Wolfgang proposed a swarm ranking method for Information Retrieval using Particle Swarm Optimization on benchmark database LETOR. The the swarm were put first through learning to rank IR results [7]. Cui, Potok, and Palathingal used PSO to cluster text documents [19].

In the above work, the researchers model the search space as a hyperspace of words or terms. The fitness function is, in some form or fashion, a Euclidian distance in the vector space of terms between the multidimensional points. The proposed vector space model (VSM) treats each term as a dimension of the multidimensional space. For example, for data clustering Merwe and Engelbrecht [19] used a variation of a distance vector to randomly seed centroid vectors, i.e., to seed some starting search points in the search space. The drawback of a VSM approach, in general, is that it treats terms as separate dimensions of the search space. Each new term increases the vector space dimension size and hence increases the complexity and number of necessary computations.

Also, Diaz-Aviles and Wolfgang [7] used training (learning) for a collection of queries and resulting retrieved documents. They used a training set as well as a validation set to attempt to reduce overfitting. They optimized a standard IR measure called Mean Average Precision or MAP. They found that the approach significantly outperformed standard approaches. Our method is similar in that we use a swarm algorithm to rank retrieved requirements 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. Also, we use a simple swarm algorithm, discussed further below.

Aghdam, Ghasem-Aghaee, and Basiri used ACO to select text features [20]. In this work, the ACO algorithm did not have any a priori knowledge about the text features. Our method is similar in that it does not involve "learning" and in that the agents do not have predetermined knowledge about the space they traverse. Our method is a *simple* version of ACO. We use the phrase "*simple*" because there is no actual use of pheromones. The search and discover phase of the algorithm is "random roulette" and is greedy. The term and document frequencies of the text collection are used as guiding heuristics for agent behavior. Technically the algorithm is still a swarm, 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.

V. METHODOLOGY

A swarm agent starts from a high level textual document and follows a word or term that is present in the high level document via the common vocabulary. In our setup, the common vocabulary is also the inverted index for the collection of documents. From the term in the inverted index, the agent follows a link down to a low level element.

First, the documents are parsed, and then undergo term stemming (words are reduced to their stem such as 'comput-' for 'computer' and 'computing'), and stop word removal (words such as 'the' and 'of' are removed). During this preprocessing, term frequencies for each term in a document are calculated. The TF-IDF weight is calculated using the following formula:

$$\text{TF-IDF}_{t,d} = \text{TF}_{t,d} * \text{IDF}_t \qquad (6)$$

The documents are tagged as high or low level elements. The preprocessing also builds the inverted index. 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. In applying the swarm of agents, each high level element is assigned a fixed number of agents roughly greater or equal to the number of low level elements.

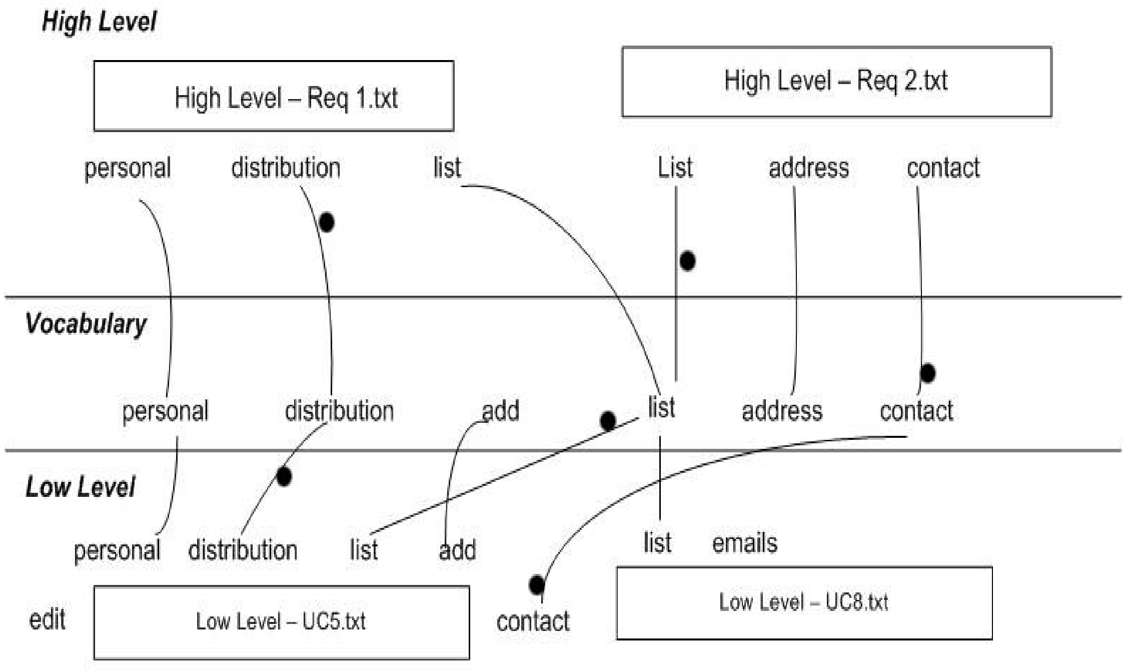

Figure 1. Agents tracing links from high level to low level elements via vocabulary.

A loop through all high level elements is then executed that undertakes the following:

- a swarm agent is assigned to each high level element;

- the terms in that high level element are ordered by TF-IDF weight;

- the agent randomly selects a term from the top 10 or less terms (per the corpus for the whole document);

- using the selected term, the agent "crawls" from the high level element to the selected term in the inverted dictionary, i.e., the vocabulary space;

- once the agent descends to the level of the inverse dictionary, the agent again randomly selects from among the top 10 documents ordered by the term frequency; and once a low level element is picked, the agent moves down to that low level element.

When all agents reach the low level elements, we can determine 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. A count of the agents at the low level element above a preset threshold indicates a candidate link. By establishing a cutoff percentage for the top candidate link count, we obtain certain precision and recall values. We may consider, for example the top 10 or the top 20%. Section VII lists the recall and precision values with regard to the cutoff threshold.

Fig. 1 depicts the application of the algorithm to a small example. Assume that we have two high level requirements Req1 and Req2:

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

Req2: "The system shall be able to add a contact to the address list."

After pre-processing of these elements, we determine that Req1 has the terms *personal, distribution,* and *list* and that Req2 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 *edit*, *contact*, *list*, and *email*. We also develop the common vocabulary by creating the inverted dictionary for the collection of all documents. 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 a high or low level element.

As the algorithm starts, a group of agents greater than or equal to the number of low level elements is assigned to Req1.txt. In the high level element, the terms are then ordered by product of TF-IDF weight and term frequency in the document. The agent randomly selects a term from the, say, top ten sorted terms. For example, the agent may pick 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. These links are also sorted in descending order by term frequency. The agent picks the next link to follow randomly from the top 10 or less candidate links. At the last leg of its journey, the agent arrives at a low level element. At the end of this loop, the result composition will have all agents from all high level documents located at the low level elements. The count of the agents coming from the same high level element greater than a certain threshold indicates a candidate link between the high and low elements.

A discussion of threshold is in order. For the swarm method, the measurements were calculated based on the threshold filter varying from 0.1 to 0.9. The threshold indicates a percentage above which the suggested candidate links are considered for recall and precision estimation. For example, assume that one hundred agents starting from element A1.txt traverse to documents UC1.txt, UC2.txt, UC3,txt, and UC4.txt. Further, assume that 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 (as we discard low level elements with counts below 30).

In order to validate the approach, we applied it to two sets of requirements from software systems. The study design and threats to validity are presented below.

VI. Validation

This section will present the design of the study as well as threats to validity.

A. Study Design

Requirements from two projects were used for the study. The first project is Pine [21], a text-based email system developed by the University of Washington. This project consists of a requirements document (49 requirements) and a set of textual use cases (133). There are 247 true links in the answer set. Development of answer sets will be discussed further under Threats to Validity. The second project is CM-1, a NASA scientific instrument [22]. The project consists of a complete requirements document (235 requirements) and a complete design document (220 design elements). There are 361 true links in the answer set.

The study was conducted as follows. Each project was first pre-processed. Stop words such as "the," "and", "of" were removed. Words were stemmed to their root using Porter's algorithm [23]. Next, the swarm dictionary was built. Each high level element was selected one at a time and the ant colony algorithm was applied. The output was captured as a candidate RTM and was compared to the answer set. We captured information on the number of true links identified as well as the number of false positives. From this, we calculated recall and precision (as discussed in Section III). The F and F2 measures were then calculated. We then compared these measures to those obtained for the same datasets using the Vector Space Model (VSM) information retrieval technique. The results are discussed in Section VII.

B. *Threats to Validity*

Threats to external validity deal with whether the results can be generalized. The greatest threat to external validity is that a small number of datasets were used for validation and that one of the datasets was relatively small. The results, therefore, cannot be generalized for all projects in all domains.

We reduced internal threats to validity by using datasets for which answer sets have been independently verified by more than one analyst and in some cases more than one research group (CM-1). There was a potential for bias though in that the answer sets were created by human analysts that are familiar with the traceability research domain. We also used a vetted tool, RETRO, and adapted it in order to implement the ant colony technique.

We reduced threats to construct validity by using standard information retrieval measures to evaluate effectiveness.

VII. RESULTS AND ANALYSIS

In this section, we present the results for the simple swarm and TF-IDF retrieval methods on the Pine and CM1 datasets.

A. *Pine Dataset*

Table 1 lists the recall and precision for the swarm method on the Pine dataset. The first column lists the threshold; columns two and three list recall and precision, respectively; column four lists the F-measure; and column five lists the F2 measure. For example, the swarm technique yielded a recall of 71%, precision of 71%, F of 0.71 and F2 of 0.71 for a threshold of 0.4. With the threshold equal to 0.9 the precision reached 88%, driving the recall down to 39%.

Table I shows that for swarms, recall starts high at 91% with precision at 30% for a "selective" threshold of 0.1. After crossing the threshold of 0.4, the recall drops faster than precision gains: from 71% down to 60% for recall vs. a small gain from 71% to 74% for precision. With further losses in recall, the gains in precision were not drastic; on average for every 2% loss in recall, there was only a 1% gain in precision.

TABLE I. RECALL – PRECISION FOR SWARMS ON PINE DATASET

Threshold	Recall	Precision	F	F2
0.1	91%	30%	0.46	0.65
0.2	82%	45%	0.58	0.71
0.3	73%	55%	0.63	0.68
0.4	71%	71%	0.71	0.71
0.5	60%	74%	0.66	0.62
0.6	51%	80%	0.62	0.54
0.7	45%	81%	0.58	0.49
0.8	42%	83%	0.56	0.47
0.9	39%	88%	0.54	0.44

From the swarm population point of view, the lowest value for recall of 39% indicates that, on average, a small group of swarm agents correctly identified 39% of all correct candidate links, i.e., they "guessed correctly" at threshold value of 0.9, i.e., in 9 out of 10 agents. The 91% recall value for 0.1 threshold indicates that, on average, 90% of the agents departing for a single high level element ended up correctly identifying only 30% of all correct links. Another way to say this is that, on average, 3 out of 10 agents ended up at the correct destination element. Recall that the agents traverse the search space from high level elements to low level elements via common terms.

Fig. 2 presents the precision-recall curve for the swarm and TF-IDF methods on the pine dataset. Note that swarms exhibited a visible advantage for this dataset. Even though TF-IDF reached 100% precision, recall became 0.32%, rounded down to 0% in Table II. For the swam method, the highest precision was 88%, but recall was merely 39%; the sharp drop in recall is visible at the 60% - with precision gaining 1% from 80% to 81%, the recall dropped from 51% to 45%.

Fig. 3 depicts the graph of the F and F2 measures from Table I. Note that F and F2 reached the maximum value of 0.71 at 0.4 threshold.

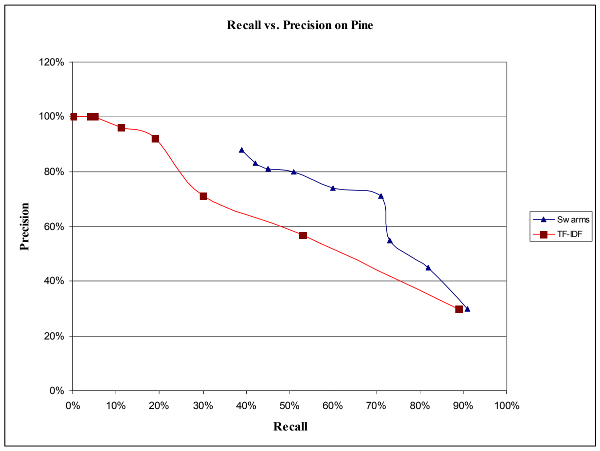

Figure 2. Precision-Recall curve for swarms and TF-IDF for Pine Dataset.

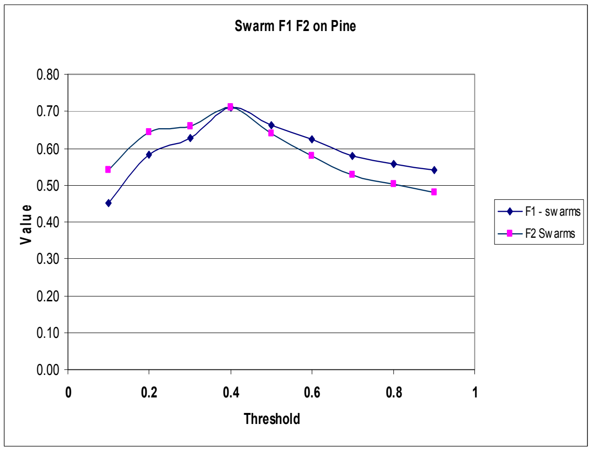

Figure 3. F and F2 for Swarms for Pine Dataset.

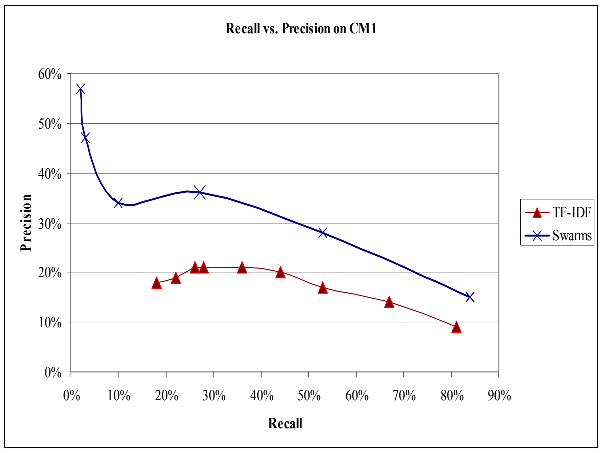

Figure 4. Recall vs. Precision swarms and TF-IDF CM1 Dataset.

All F and F2 values stayed above 0.40 for the swarm method for the Pine dataset. The steep increase in F and F2 values as threshold is increased from 0.1 to 0.4 indicates that as we narrow down the count of "considered" swarm agents starting for a single element based on the grouping count, we observe that the agents tend to gather in a small subset of elements. By increasing the threshold value, we see that the agents start missing correct targets, i.e., low level elements that are part of the correct links to the high level element from which the agents started the journey.

Table II presents the recall and precision as well as F and F2 values for the TF-IDF retrieval method applied to the Pine dataset. The link weight indicates a cosine similarity, ranging from 0 to 1, between the weighted keyword vector of two documents [12]. From examining Fig. 4, it can be noted that there is a much clearer trade-off between recall and precision for TF-IDF than for swarms. Recall starts very high at 89% and then quickly drops to below 50% (filtering at a link weight of 0.3) with precision starting at 30% and increasing to 100% at link weight 0.6. It is apparent that increases in precision are at the expense of recall.

TABLE II. RECALL-PRECISION FOR TF-IDF ON PINE DATASET

Link Weight	Recall	Precision	F	F2
0.1	89%	30%	0.45	0.64
0.2	53%	57%	0.55	0.53
0.3	30%	71%	0.43	0.34
0.4	19%	92%	0.32	0.23
0.5	11%	96%	0.20	0.13
0.6	5%	100%	0.10	0.06
0.7	4%	100%	0.07	0.05
0.8	0%	100%	0.01	0.01

As can be seen from Fig. 5, the top values for F and F2 are achieved at link weight 0.2. For the swarm method, the highest value for both F and F2 was 0.71. Note that the highest F and F2 values achieved by TF-IDF were 0.58 and 0.66, respectively. The F and F2 measurement exhibited a sharp increase when the weight link changed from 0.1 to 0.2. F and F2 demonstrated a steady and steep decline as the link weight filter grew from 0.2 to 0.8.

We do not see recall and precision above 0.8 weight link threshold. The set did not produce elements generating a "cosine" similarity greater than 0.8. It is logical to expect that we will not see any documents retrieved with this high weight link value.

For both swarms and TF-IDF, recall grew smaller as the filter value (link weight and threshold) increased. Overall, the swarm method displayed better results than TF-IDF. At the highest precision of 88% for swarms, the recall was 39%; for TF-IDF the recall dropped down to 5% at the highest precision of 100%.

The swarm method we tested used the TF-IDF weight and term frequency as the guiding heuristic for the agents.

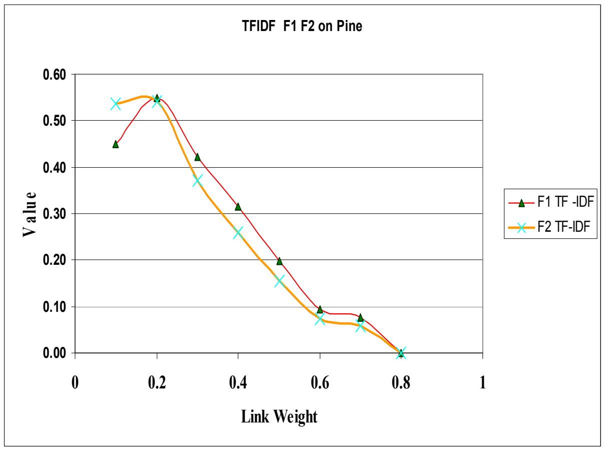

Figure 5. F and F2 for TF-IDF for Pine Dataset.

The distinction between the swarm method and pure TF-IDF comes from the fact that we "direct" the swarm agent to consider and focus on the most important terms in the document, by sorting the terms using weights and frequency counts. This allowed the swarm to perform a more focused search.

B. CM-1 Dataset

Next, we examine the results for the CM1 dataset. Table III shows the recall and precision values for the swarm method on the CM1 dataset. Note that recall starts off at about the same high level as for the smaller Pine dataset, but that precision starts off much lower (at 9% versus 30%). This is a common phenomenon for IR methods also (that larger datasets yield smaller precision values).

It can also be seen from the table that as the threshold increased, the recall decreased. Even so, the precision was not improving significantly as recall dropped. Even with recall as low as 36%, precision only reached 21%. It became apparent that the swarm agents were not picking the correct low level elements as we increased the value of the threshold. The 0.9 threshold is the top 10 % of all groupings of agent counts at the low level element. The 18% recall indicates that, on average, 90% of the agents starting from a high level element reached and grouped around only 18% of the correct low level elements. It became apparent that search options given to the swarm agents restricted their options to explore and directed them to a limited number of low level elements.

Fig. 6 shows the F and F2 measures for the swarm method on the CM-1 dataset. As shown by the values in Table III, the F and F2 measurement for swarms on CM1 did not exceed 0.35. Note that the F measure did not change significantly, ranging from 0.17 to 0.28. The "corridor" of F, F2 values between 0.17 and 0.33 indicates that, on average, between 1 out of 7 agents (0.17) and 1 out of 4 agents (0.28) reach correct low level elements in the CM1 dataset. The F2 reached its peak at 0.31, indicating that the swarms did perform better in terms of recall than precision. Both F and F2 values drop as the threshold approaches 1.

TABLE III. RECALL-PRECISION FOR SWARMS ON CM1 DATASET

Threshold	Recall	Precision	F	F2
0.1	81%	9%	0.16	0.22
0.2	67%	14%	0.23	0.30
0.3	53%	17%	0.26	0.31
0.4	44%	20%	0.28	0.31
0.5	36%	21%	0.27	0.29
0.6	28%	21%	0.24	0.25
0.7	26%	21%	0.23	0.24
0.8	22%	19%	0.20	0.21
0.9	18%	18%	0.18	0.18

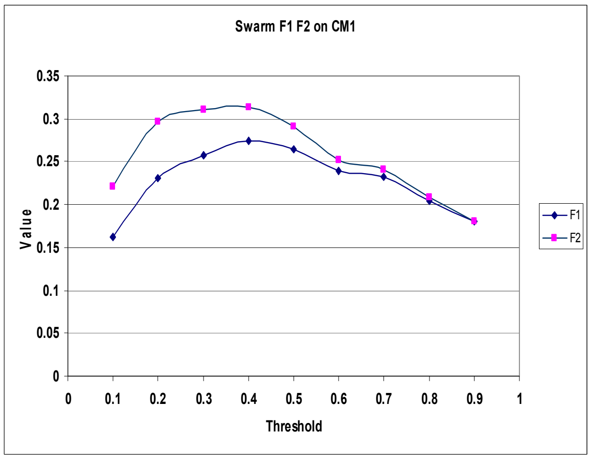

Figure 6. F and F2 for Swarms for CM1 Dataset.

Table IV lists the recall, precision with F and F2 values for the TF-IDF method for CM1. In this case, the TF-IDF method performed better. The recall and precision started at higher values of 84% and 15%, respectively vs. 81% and 9% for swarms. The highest values for F and F2 measurements for TF-IDF on the CM1 dataset were 0.37 and 0.33, respectively. As the weight link reached 0.7, only 50% of the retrieved elements were relevant (precision). From this we can deduce 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.

Fig. 7 shows the F and F2 measures for CM-1. It shows a sharp drop in F and F2 measures after the link weight exceeds 40%. This drop indicates that precision did not compensate for the drop in recall. As the link weight filter approaches 1, both the F and F2 values drop below 0.1. With the weight link value at 0.7, we retrieved only one relevant document out of 332 that were truly relevant (according to the answer set). One out of 332 returned yields recall equal to 0.32%.

TABLE IV. RECALL-PRECISION FOR TF-IDF ON CM1 DATASET

Link Weight	Recall	Precision	F	F2
0.1	84%	15%	0.25	0.21
0.2	53%	28%	0.25	0.21
0.3	27%	36%	0.37	0.33
0.4	10%	34%	0.31	0.32
0.5	3%	47%	0.15	0.19
0.6	2%	57%	0.06	0.08
0.7	0%	50%	0.04	0.06

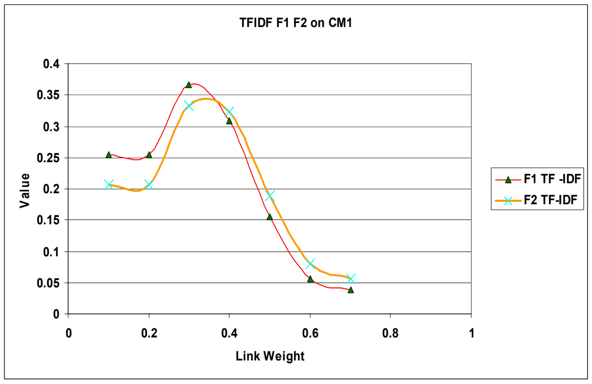

Figure 7. TF-IDF F and F2 graph for CM1 Dataset.

C. *Comparision*

Fig. 8 and Fig. 9 show comparative graphs for swam performance vs. that of TF-IFD. Even though the threshold value in swarms and weight link in TF-IDF are not calculated the same way, they play a similar role, i.e., they serve as a filter for the candidate links. The higher the filter, a close cosine similarity in documents in TF-IDF or a higher agent count in swarms, the more the F values decrease for TF-IDF and swarms on both datasets.

Fig. 8 shows that swarms performed better on the Pine dataset, reaching a maximum value for F and F2 of 0.71. At the same time, the TF-IDF achieved its highest values for F and F2 of only 0.55 and 0.64, respectively. Furthermore, TF-IDF exhibits a steep decline in F and F2 as the filter values for threshold and weight links increase. As we can see in Fig. 8, the swarms have an apparent advantage over TF-IDF on the Pine dataset.

Fig. 9 indicates a better performance for the TF-IDF method on the CM1 dataset, achieving 0.37 for F and 0.33 for F2, respectively. TF-IDF exhibits a clear increase in these values around the 0.20 threshold value.

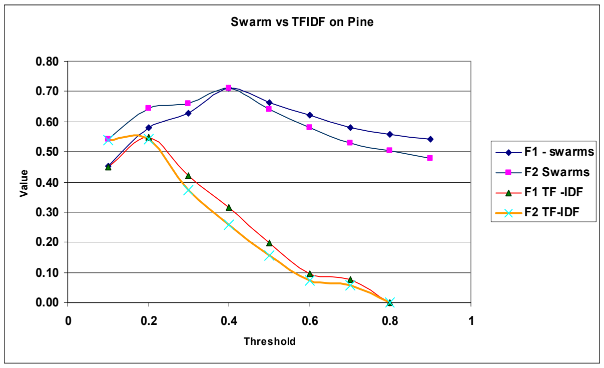

Figure 8. Comparative graph for swarm and TF-IDF F and F2 on Pine Dataset.

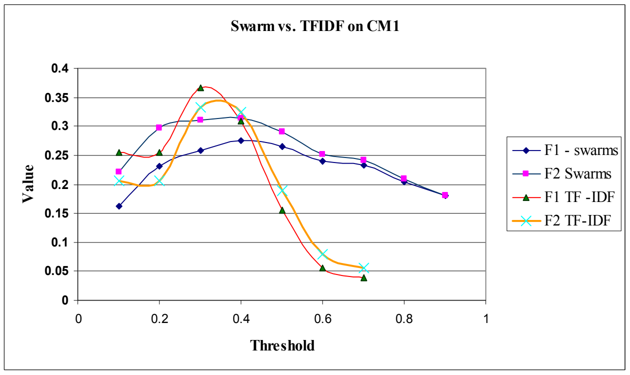

Figure 9. Comparative graph for swarm and TF-IDF F and F2 on CM1 Dataset.

Though TF-IDF reached F, F2 values of 0.37, 0.32 vs. 0.27 and 0.31 for swarms, the TF-IDF advantage is present for all filter values of link weight and threshold. Both methods performed better for the CM1 dataset when filtering was between 0.2 and 0.4.

In summary, the swarm approach showed some advantage over the TF-IDF method on the Pine dataset, yet it did not fare as well on CM1. 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 a total number of possible links of about 1250 (half of the traceability matrix). The CM1 dataset contains 235 high and 220 low elements, thus producing about 26,000 possible candidate links for the search space. The answer set for the Pine dataset has 248 links, about 20% of all possible links. In the CM1 dataset, this ratio of existing links vs. possible is less than 2% (361 divided by 26000). Also, the CM1 dataset uses much technical terminology and acronyms, meaning that swarm agents often ended up at incorrect low level elements.

To get high recall and precision results on the CM1 dataset, the collection of candidate links has to be tightly focused and highly precise. The use of a thesaurus might have directed the swarm agents to the proper document. In the case of TF-IDF at low filter values, the method considers a greater number of the low level elements as possible candidate links, thus yielding some incorrect results at the cost of recall. The swarm method, a lighter weight approach than TF-IDF, limited the "discovery horizon" for the agents by focusing on the top terms in a textual element, hence limiting possible search alternatives.

VIII. Conclusions and Future Work

The paper presents a simple swarm method in support of requirements traceability. The method showed promising results on the Pine dataset, achieving recall of 71% with precision of 71% and outperforming TF-IDF. At the same time, the limitations of the method were discovered on the CM1 dataset. As mentioned earlier in the paper, the shortcomings of the method may be a result of the limitations imposed on the swarm agent search behavior.

An interesting observation about the performance of swarms and TF-IDF on Pine and CM1 datasets can be drawn from the "pick" humps between link weight values 0.2 and 0.4 on Figs. 8 and 9. As the TF-IDF method compares the documents based on the term similarity, the method treats a document as a "bag of words" compared to another document, i.e., another "bag of words." The similarity up to 0.4 of the terms helps to establish true links with maximum performance measured in F and F2 harmonics. The observation reiterates the reasoning for the idea of enabling the swarms to concentrate on the top terms in a document, rather than exploring all of them.

We plan to expand this work further by using a thesaurus, permitting the agents to discover links not only through a single term but through term synonyms. We also plan to address the use of term proximity as a possible attributing factor for agent search behavior. In light of the observations made that between 20 and 40 % of common terms contribute the highest level of true candidate links, it would be interesting to see if we can achieve the same results using a smaller swarm population.

Acknowledgment

This work is funded in part by the National Science Foundation under NSF grant CCF-0811140. We thank Wei-Keat Kong for helpful advice on the swarm method and on both datasets.

References

[1] T.T. Tun, M. Jackson, R. Laney, B. Nuseibeh, and Y. Yu, "Are Your Lights Off? Using Problem Frames to Diagnose System Failures," *Requirements Engineering, IEEE International Conference on*, Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 343-348.

[2] "Amazon.com Link."

[3] G. Antoniol, M.D. Penta, and M. Harman, "Search-Based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project," *Proceedings of the 21st IEEE International Conference on Software Maintenance*, IEEE Computer Society, 2005, pp. 240-249.

[4] C.P. Lam, J. Xiao, and H. Li, "Ant colony optimisation for generation of conformance testing sequences using a characterising set," *Proceedings of the third conference on IASTED International Conference: Advances in Computer Science and Technology*, Phuket, Thailand: ACTA Press, 2007, pp. 140-146.

[5] M. Reitz, "Software Evolvability by Component-Orientation," *Proceedings of the Second International IEEE Workshop on Software Evolvability*, IEEE Computer Society, 2006, pp. 66-73.

[6] K. Ayari, S. Bouktif, and G. Antoniol, "Automatic mutation test input data generation via ant colony," *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, London, England: ACM, 2007, pp. 1074-1081.

[7] E. Diaz-Aviles, W. Nejdl, and L. Schmidt-Thieme, "Swarming to rank for information retrieval," *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, Montreal, Québec, Canada: ACM, 2009, pp. 9-16.

[8] Deneubourg, S. Aron, S. Goss, and J. Pasteels, "The self-organizing exploratory pattern of the argentine ant," *Journal of Insect Behavior*, vol. 3, Mar. 1990, pp. 168, 159.

[9] A.P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*, Wiley, 2006.

[10] O. Gotel and A. Finkelstein, "Extended Requirements Traceability: Results of an Industrial Case Study," IEEE Computer Society, 1997, p. 169.

[11] J.H. Hayes, A. Dekhtyar, S. Sundaram, and S. Howard, "Helping Analysts Trace Requirements: An Objective Look," 2004.

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

[13] 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, 2002, pp. 970-983.

[14] J.H. Hayes, A. Dekhtyar, and J. Osborne, "Improving Requirements Tracing via Information Retrieval," 2003, pp. 151-161.

[15] Xuchang Zou, R. Settimi, and J. Cleland-Huang, "Phrasing in Dynamic Requirements Trace Retrieva," 2006, pp. 265-272.

[16] G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause, "Rule-Based Generation of Requirements Traceability Relations," *Journal of Systems and Software*, vol. 72, 2004, pp. 105-127.

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

[18] D. van der Merwe and A. Engelbrecht, "Data clustering using particle swarm optimization," *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, Canberra, ACT, Australia: , pp. 215-220.

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

[20] M.H. Aghdam, N. Ghasem-Aghaee, and M.E. Basiri, "Text feature selection using ant colony optimization," *Expert Syst. Appl.*, vol. 36, 2009, pp. 6843-6853.

[21] "Pine Email System."

[22] "NASA IV&V Facility Metrics Data Program - GLOSSARY AND DEFINITIONS."

[23] M. Porter, "An algorithm for suffix stripping," *PROGRAM*, vol. 14, 1980, pp. 130-137.