Application of Reinforcement Learning to Requirements Engineering: Requirements Tracing

Hakim Sultanov
University of Kentucky
Lexington, KY, USA
hakim.sultanov@uky.edu

Jane Huffman Hayes
University of Kentucky
Lexington, KY, USA
hayes@uky.edu

Abstract— **We posit that machine learning can be applied to effectively address requirements engineering problems. Specifically, we present a requirements traceability method based on the machine learning technique Reinforcement Learning (RL). The RL method demonstrates a rather targeted generation of candidate links between textual requirements artifacts (high level requirements traced to low level requirements, for example). The technique has been validated using two real-world datasets from two problem domains. Our technique demonstrated statistically significant better results than the Information Retrieval technique.**

Index Terms— **machine learning, reinforcement learning, information retrieval; requirements traceability; software engineering; Ubiquitous Grand Challenge, Research Project 2 of Grand Challenges of Traceability**

I. Introduction

The value of tracing requirements through all phases of the Software Development Life Cycle (SDLC) can be appreciated by the customers who request and expect delivery of certain system features and by the developer who failed to deliver the product according to the specifications. Overlooked software requirements can have a profound effect. For example, a software flaw resulted in a product multiplying stock trades which caused Knight Capital Group to lose more than $440 million dollars [1].

To ensure high quality requirements and their proper implementation throughout the phases of the SDLC, a number of steps can be taken. Requirements tracing ensures that requirements are properly addressed in the resulting software artifacts, i.e., use cases, test cases, etc. In recent years, interest in this area has intensified as demonstrated by the amount of work on automating the process of requirement tracing [2][3][4][5]. It should be noted that even the automated traceability techniques require some human analyst involvement. It is important to improve the quality of requirements to ease automation of tracing and other requirements-related techniques as well as to ease the burden of human analysts working with requirements and/or the output of requirements tools.

In the past several years, there has been growing interest in machine learning techniques applied to requirements engineering. Machine learning techniques can help establish knowledge or rules from requirements engineering artifacts [6][7]. The reinforcement learning technique is a machine learning algorithm that is based on computational agents selecting actions to maximize some long term reward. We introduce a reinforcement learning-based method for tracing textual pairs of requirements artifacts. The presented technique has been validated on two sets of software requirements from real projects in two domains, comparing the results to those of a typical information retrieval (IR) tracing technique. We found that our technique demonstrated statistically significant better results than the IR technique.

The paper is organized as follows. Section 2 provides background on reinforcement learning. Section 3 discusses requirements tracing. Section 4 discusses our approach to tracing using the reinforcement learning method. Section 5 presents the validation of the technique on two datasets. Section 6 discusses the results and analysis. Section 7 presents related work. Section 8 concludes and addresses future work.

II. Reinforcement Learning

In reinforcement learning (RL), agents probe and change 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 $a_t \in A(s_t)$, 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, the agent gains reward r_{t+1}, and moves to the state s_{t+1}. Figure 1 displays the interaction between the agent and environment.

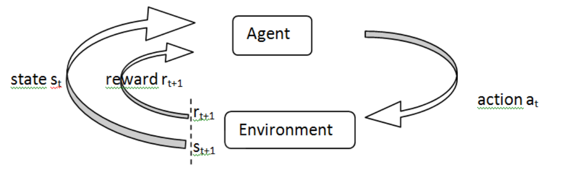

Fig. 1. The interaction of the agent and the environment in reinforcement learning.

As shown in Figure 1, the agent receives the state 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 $A(s_t)$, 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.

The reward the agent collects depends upon the actions it takes. To estimate the desirability of a state, the RL technique uses the notion of a value function. Formally, the value function is represented as:

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

where R_t is the sum of all rewards obtained after time step t. The value $E\pi\{\}$ is the expected reward value given to the agent that follows the policy π. The discount coefficient $\gamma \in [0,1]$ places greater weight on 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 equally.

Bellman's equation [8] provides another way to express the value of a state s:

$$V^{*}(s) = \sum_{a} \pi(s,a) \sum_{s`} P_{ss`}^{a} [R_{ss`}^{a} + \gamma V^{\pi}(s`)] \qquad (2)$$

where $P^a_{ss`}$ is the probability of reaching state s\` from s if action a is taken; $R^a_{ss`}$ 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 π^*. Formally, $\pi^* \geq \pi`$, if and only if, $V\pi^*(s) \geq V\pi`(s)$ for any $s \in S$. Alternatively, we can define V^* as

$$V^*(s) = max_{\pi} V^{\pi}(s) \qquad (3)$$

One way to determine an optimal policy is to use the Value Iteration algorithm [8]. 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:

$$V_{k+1}(s) = max_a \sum_{s`} P_{ss`}^{a} [R_{ss`}^{a} + \gamma V_k(s`)] \qquad (4)$$

where P^a_{ss} and $R^a_{ss`}$ bear the same meaning as defined in Equation 2. The value of state s is maximized across all actions a available at s. The pseudo code for the Value Iteration algorithm is shown below:

```
Initialize  V(s) =0, for all s ∈ S
Repeat
    Δ ←0
        For each s ∈ S
        V ←V(s)
        V(s) ←max_a ∑s`Pss` [Rss`+γV(s`)]
        Δ←max(Δ,|v-V(s)|)
        Until Δ<ε (ε a small positive number)
Output a deterministic policy, π, such that
π(s)=argmax_a ∑s`P^a ss`[R^a ss`a+γV(s`)]
```

To apply the reinforcement learning-based approach to the traceability problem, we construct 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.

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" [9]. A typical process used for tracing natural language artifacts, manual or automated, generally consists of a number of steps: document parsing, candidate link generation, candidate link evaluation, and traceability analysis [9]. 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 artifacts (e.g., are there use cases that satisfy a given requirement?). In this work, we concentrate on adapting the RL 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 the 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:

$$IDF_t = \log\left(\frac{N}{DF_t}\right), \quad (5)$$

where N is the total number of documents in the collection and DFt 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 computational agents that traverse the collection of all documents and the vocabulary shared by the documents.

B. Measurements

The tracing results can be evaluated using two measurements: Recall and Precision [10].

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

$$Recall = \frac{\#of\_relevant\_retrieved}{\#\_relevant\_in\_collection}. \quad (6)$$

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

$$Precision = \frac{\#of\_relevant\_retrieved}{\#\_retrieved}. \quad (7)$$

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

$$F = \frac{(\beta^2+1)P*R}{\beta^2 P + R}, \text{ where } \beta^2 \in [0,\infty). \quad (8)$$

When $\beta^2=1$, 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.

IV. Methodology

The search space in a reinforcement learning (RL) model 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.

A. Search Space Navigation

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 2 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, HL
- A term in a high level document, $term_{HL}$
- Low level document, LL
- A term in a low level document, $term_{LL}$
- A synonym term in vocabulary, syn_t

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 2.

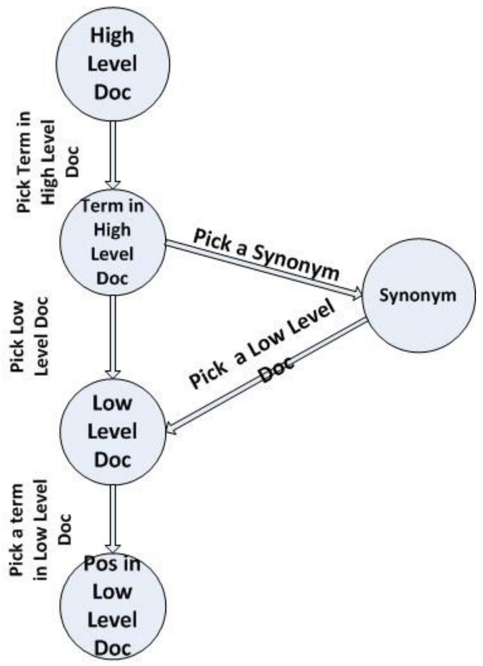

Fig. 2. RL Agent State Transition Diagram.

As we can see in Figure 2, when an agent is positioned at a high level document, state 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 $term_{HL}$. From the state $term_{HL}$ 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 LL. From the state LL the agent should select a term in the low level document. If the low level document contains the term term$_{HL}$ in several positions, the agent needs to select a position term$_{LL}$ 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$_{HL}$, 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 synonym term. Possible actions at states and transitions between the states are summed up in Table I

TABLE I. AGENT ACTIONS

From	Action	To
Top Level Doc HL	Select a term	term$_{HL}$
term$_{HL}$	Select low level doc or synonym	LL
term$_{HL}$	Select low level doc or synonym	syn
LL	Select a term	term$_{LL}$
Syn	Select low level doc	LL

Each action listed in Table I 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:

$$\Pr(\text{State}_i) = \frac{1}{N} \quad , \quad (9)$$

where S$_i$ is a reachable state and N is the number of all reachable states. For example, if the agent is in a "term- high-level" state t$_{HL}$ and has ten possible low level documents, i.e., ten reachable 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

$$\Pr(\text{State}_i) = \frac{Value(S_i)}{\sum_{j \in possible\ states}(S_j)} \quad , \quad (10)$$

The probability of transitioning into the state Si 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 LL states from the state t$_{HL}$ had the following values associated with them: {20, 50, 30, 0, 0, 0, 0, 0, 0, 0}, the probability of transitioning into the first LL state is 0.2, into the second 0.5, the third 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:

$$\Pr(\text{State}_i) = \frac{Value(S_i * S_i)}{\sum_{j \in possible\ states}(S_j * S_j)} \quad . \quad (11)$$

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.

Consider the term 'list.' In the course of the reinforcement learning algorithm, the state "A3-list" received the value 1.55. 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.

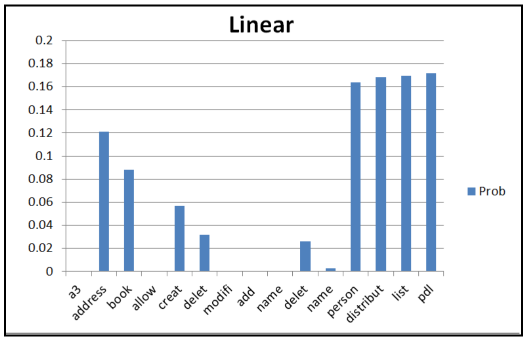

Fig. 3. Term selection probability based on linear selection behavior.

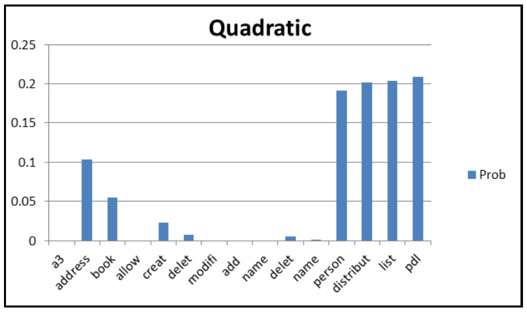

Fig. 4. Term selection probability based on quadratic selection behavior.

It can be visually seen that the term 'delet' in Figure 4 is significantly smaller compared to the terms 'distribut,' 'list,' and 'pld.' It is also worth mentioning that the proposed 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" term$_{HL}$ can have a reward. This is a numeric value associated with transitioning into the term$_{HL}$ 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:

$$\sum_{w_1 \in H} \sum_{w_2 \in L} \delta(t_1, t_2)\left[tfidf_{w_1} * C_{high} + tfidf_{w_2} * C_{low}\right] \quad , \qquad (12)$$

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 $\delta(w_1,w_2)$ is calculated as follows:

$$\delta(w_1, w_2) = \begin{cases} 1 \; if \; w_1 = w_2 \\ 0 \; if \; w_1 \neq w_2 \end{cases} \quad . \; (13)$$

The multiplication coefficients range from 1 to 10: C_{high},C_{low} $\in\{1,10\}$. The range of multiplication coefficients is a calculated estimate of 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 II), 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:

$$R = \sum_{t=0}^{N-1} r_t \quad , \; (14)$$

where r_t is the reward received after t-th transition action. The reinforcement learning algorithm for requirements traceability listed below:

```
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
1. // Create State Space
2. For each doc hl in high level collection H
3.    States.Add(NewState(hl))
4.     For each term t in high level doc h
5.       i ← position of t in hl
6.       States.Add(newState(hl_t))
7.   // Iterate through low level doc linked via term t
8.  For each doc ld in Vocabulary.GetDocsByTerm (t)
9.        If ld is lowLevelDocument
10.         For each position j of term t in ld
11.              lLevelDocState← newState(ht_t_ld_posj)
12.              Value=MatchingValue(hl,ld,i,j)
13.               lowLevelDocState.Value□Value
14.          End For
15.        Else // ld is a synonym
16.          ld_2← Vocabulary.GetDocsByTerm (ld)
17.          For each position j of term ld in ld_2
18.               lLevelDocState←newState(ht_t_ld_posj)
19.               Value=MatchingValue(hl,ld_2,i,j)
20.               lowLevelDocState.Value□Value
21.           End For
22.         End if
23.  End For
24. End For
25.  // Calculate state values
26. For cycle 1 to 5
27.     For each state s in States
28.          argMaxValue ← 0
29.          possibleSates ← s.Transitions
30.           For each action a in Actions
31.                possibleStates.TransitionProb(a)
32.               For each ps in possibleStates
33.                   pValue← pValue+ps.Prob*ps.Value
34.                   If pValue > argMaxValue
35.                        bestAction ← a
36.                        s.Policy ← a
37.                    End if
38.               End For //possible states
39.                 maxValue←Max(pValue, maxValue)
40.             End For // Actions
41.             s.Value← s.TransReward + maxValue
42.        End For // states
43.     // Traverse the Search Space
44.     For each top level document hl
45.            For each agent ant in colony
46.             currentState ← States(hl)
47.             While CurrentState != low level document
48.               //use curState.Policy
49.                   //and curStates.Transitions
50.               nextState ← currentState.SelectNextState
51.               currentState← nextState
52.             End While
53.         End For
54.     End For
```

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.

B. 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 S= $\{s_1, s_2,\dots, s_k\}$, 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 s_i is higher than the probability of transitioning into any other state:

$$Pr(si) >> Pr(sj), \text{ where } s_i, s_j \in \{s_1, s_2,\dots, s_k\} \quad i \neq j. \; (15)$$

It is possible to have a situation where the 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.

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. 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

V. Validation

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

A. Study Design

In order to validate the proposed approach, the RL based technique was applied to two datasets. The first consists of 49 textual requirements and 51 textual use cases. The dataset is a text-based email system, Pine, developed by the University of Washington [10]. 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 consists of 22 requirements documents and 53 design documents in the NASA scientific instrument project CM1SUB [11]. The project has 45 true links in the answer set.

The experiments were conducted using a Vector Space Model with TF-IDF weighting (TF-IDF hereafter) and the reinforcement learning (RL) method. The independent variable in the study is the method (TFIDF, reinforcement learning). The dependent variables are 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 based method. The output was captured in the form of a candidate TM. The results were compared to the answer set to calculate recall and precision (Equations (6) and (7)) defined earlier.

To eliminate any possible threats to the validity of the experiment several controls were implemented. Internal threats to validity include possibly indicating a relationship between the treatment methods and the outcome when in reality there is 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 were 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 techniques. The threats to validity were also reduced by using standard information retrieval measures 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 in size. 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 [36]. The results are discussed below.

VI. Results and Analysis

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. Subsection A presents the RL results for Pine. Subsection B presents results for CM1SUB dataset. We share our observations in Subsection C.

A. Pine Dataset

Figure 5 presents the precision-recall curve for the RL and TF-IDF methods for the Pine dataset.

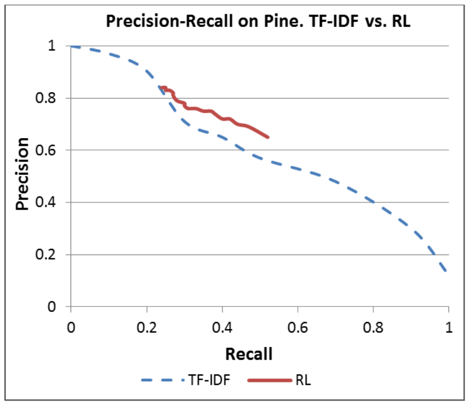

Fig. 5. Precision-Recall curves for TF-IDF and Reinforcement Learning methods for the Pine Dataset.

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 5, 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 Zcritical 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 < Zcritical, the null hypothesis was rejected. This left the conclusion that there is a statistically significant difference between the precision values of the two methods.

B. *CM1SUB Dataset*

The RL method applied on the CM1SUB dataset produced results similar to the results obtained on the Pine dataset. Figure 6 shows the precision-recall values for the RL method compared to the precision-recall values for the TF-IDF method using the CM1SUB dataset.

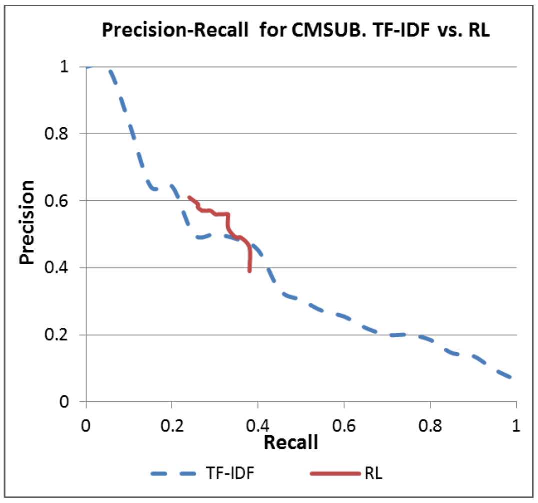

Fig. 6. Precision-Recall curves for TF-IDF and Reinforcement Learning methods for the CM1SUB Dataset.

As shown in Figure 6, 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. The RL method does target the relevant candidate links.

For the CM1SUB dataset, the recall and precision numbers were compared between the two overlapping recall 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 Zcritical 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 < Zcritical, 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.

C. Observations

In light of the results obtained from the experiments, we make the following observations.

Typically, when we consider a precision-recall curve, we observe: high recall values and low precision; high precision and low recall; and values in between these two extremes [12], [13], [2]. 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 7. 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 of recall and precision values. The closer we can get to the ideal shape of the precision-recall curve, the fewer links a human analyst will have to inspect.

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.

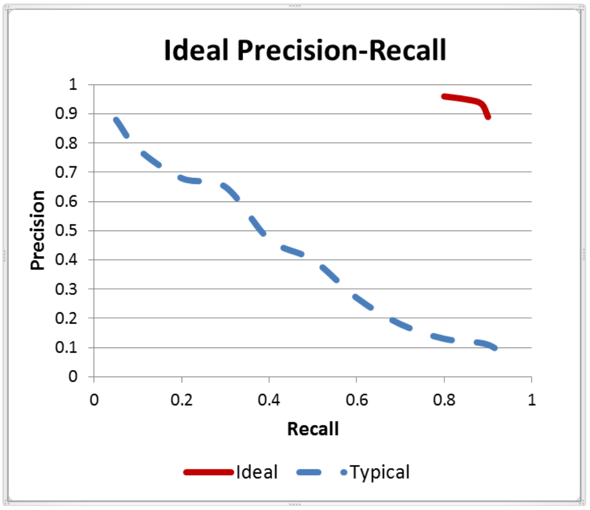

Fig. 7. Precision-Recall curves, Ideal vs. Typical.

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 for both datasets exhibited a more focused result in producing candidate links 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 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 for 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. Even though in many cases 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 no t 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. Next, we compare our work to prior art.

VII. Related Work

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

A. Candidate Link Generation/Text Analysis

A number of researchers have successfully applied information retrieval techniques to the candidate link generation problem. Most have applied the vector space model to generate traceability between various artifact pairs (such as source code modules and manual pages, functional requirements and source code) [11,13]. In general, high values of recall are achieved (90 – 100%) with low precision (2 – 12%) [11,2,3]. As a result, work has focused on improving precision. Approaches have ranged from phrasing to applying rules to tagged artifacts [9, 13,14].

Mader and Gotel [3] presented an approach for automated update of traceability relations between analysis and design artifacts presented in UML. To update the relations, the method needs to have a set of pre-established relations. The method offers the recognition of the changes in UML diagrams and then updates the relations of the changed diagram elements.

In general, the above techniques have been able to achieve excellent recall [12] but often at the expense of precision that is not acceptable or is only borderline acceptable. In our work, we aim to keep both recall and precision high. The RL agents navigate the search space to link documents. The agents' traversal heuristic links high level documents to related low level counterparts. We use the VSM as a baseline against which to compare the performance of our method.

Ziftci and Krueger [14] correctly point out one of the weaknesses of the traditional IR method: low precision at a high recall. The authors use a notion called "feature marker" to establish traces between functional requirements and test cases. The proposed method does achieve precision-recall values above 90%. The extra burden for the method is due to the use of an execution tracing tool, aka a profiler. The profiler lists the methods and classes called during execution of test cases. By articulating the aim of achieving precision-recall above 90%, the authors emphasize the target goals for requirements traceability research. On the other hand, the shortcoming of their proposed method is the restricted range of options: the execution traces can be obtained only after the source code has been compiled. The method cannot be applied when the requirements need to be traced to design elements.

B. Machine Learning Techniques

Cleland-Huang, Czauderna, Gibiec, and Emenecker present two machine learning approaches to improve traces between regulatory codes and product requirements [15]. 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.

Establishing links between documents can 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 [16]. 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 terms and the terms located close to linking term in the text.

Using the similarities between textual documents, i.e., common textual segments, and establishing the logical links based on these segments is the main focus of our research work. The work presented by Menczer and Belew lists many features similar to our work [17]. 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.

An agent in Mencer and Belew's work senses the "current neighborhood" by analyzing the text where the agent is situated. That matching feature is similar to the concept of term neighborhood that we use. The agents in Mencer 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 agents 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. The links between documents in the work of Menczer and Belew's are web links. In our work, the links between documents are established via common terms. The agents of Menczer and Belew receive user feedback on the suggested links; in our work the agents do not receive feedback. The agents in Menczer and Belew's work are created with "initial reservoir of 'energy' [17]." The agents in our research do not utilize any energy measurements for the search space traversal.

Summing up related work, we can state the following:

- It is useful to link documents by treating them as a collection of phrases, not a bag of words [16].
- Small textual segments and their similarity can be evaluated based on semantic distance [17].
- Text around the linking term provides good location data on compared textual segments [16] [17].

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 [16] [17] [18].

VIII. Conclusions and 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 [17]. Feedback may improve the accuracy of the generated candidate links.

Also, a parts of speech tagging or noun-verb phrase technique [19] can be considered in future work. By classifying terms in textual documents, we can amplify the importance of one type of textual segment over the others.

Evaluating how the human analyst can use the focused results suggested by the RL algorithm is also a future work direction.

ACKNOWLEDGMENT

This work was funded in part by the National Science Foundation under NSF grant CCF-0811140. We thank Dr. Judy Goldsmith for advice on the RL method.

REFERENCES

[1] S. Ruhle, C. Harper, and N. Mehta, "Knight Trading Loss Said to Be Linked to Dormant Software," Bloomberg. [Online]. Available: http://www.bloomberg.com/news/2012-08-14/knight-software.html. [Accessed: 11-Jan-2013].

[2] J. 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.

[3] P. MäDer and O. Gotel, "Controversy Corner: Towards automated traceability maintenance," J. Syst. Softw., vol. 85, no. 10, pp. 2205–2227, Oct. 2012.

[4] 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.

[5] E.-V. Chioasca, "Using machine learning to enhance automated requirements model transformation," in 2012 34th International Conference on Software Engineering (ICSE), 2012, pp. 1487 – 1490.

[6] 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.

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

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

[9] J. Matthias, "Requirements Tracing," Communications of the ACM, vol. 41, no. 12, 1998.

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

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

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

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

[14] C. Ziftci and I. Krueger, "Tracing requirements to tests with high precision and recall," in Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, Washington, DC, USA, 2011, pp. 472–475.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.