A Statistical Approach for Efficient Crawling of Rich Internet Applications[1]

Mustafa Emre Dincturk[1,3], Suryakant Choudhary[1,3], Gregor von Bochmann[1,3], Guy-Vincent Jourdan[1,3], Iosif Viorel Onut[2,3]

[1] EECS, University of Ottawa. 800 King Edward Avenue, K1N 6N5, Ottawa, ON, Canada
[2] Research and Development, IBM® Security AppScan® Enterprise, IBM, 770 Palladium, Ottawa, ON, Canada
[3] IBM Canada CAS Research
{mdinc075, schou062}@uottawa.ca
{bochmann, gvj}@eecs.uottawa.ca
vioonut@ca.ibm.com

Abstract. Modern web technologies, like AJAX result in more responsive and usable web applications, sometimes called Rich Internet Applications (RIAs). Traditional crawling techniques are not sufficient for crawling RIAs. We present a new strategy for crawling RIAs. This new strategy is designed based on the concept of "Model-Based Crawling" introduced in [3] and uses statistics accumulated during the crawl to select what to explore next with a high probability of uncovering some new information. The performance of our strategy is compared with our previous strategy, as well as the classical Breadth-First and Depth-First on two real RIAs and two test RIAs. The results show this new strategy is significantly better than the Breadth-First and the Depth-First strategies (which are widely used to crawl RIAs), and outperforms our previous strategy while being much simpler to implement.

Keywords: Rich Internet Applications, Web Crawling, Web Application Modeling.

1 Introduction

Web applications have been undergoing a significant change in the past decade. Initialy, the web applications were built using simple HTML pages on the client side. Each page had a unique URL to access it. The client (web browser) would send a request for these URLs to the server which in turn would send the corresponding page in response. The client would then entirely replace the previous content with the new information sent by the server.

In the recent years, with the introduction of newer and richer technologies for web application development, web applications have become more useable and interactive. These applications, called Rich Internet Applications (RIAs), changed the traditional web applications in two important aspects: first, client side-scripting languages such

[1] A detailed version can be found at http://ssrg.eecs.uottawa.ca/docs/ICWE2012_long.pdf

as JavaScript have allowed the modification of the web page by updating the Document Object Model (DOM) [5], which represents the client-side "state" of the application. Second, using technologies like AJAX [6] the client can communicate asynchronously with the server, without having the user to wait for the response from the server. In both cases, the URL typically does not change during these client side activities. Consequently, we can now have a quite complex web application addressed by a single URL.

These improvements increased the usability of web applications but on the other hand introduced new challenges. One of the important problems is the difficulty to automatically crawl these websites. Crawling is the process of browsing a web application in a methodical, automated manner or in an orderly fashion. Traditional crawling techniques are not sufficient for RIAs. In traditional web applications, a page is defined by its URL and all the pages reachable from the current page have their URL embedded in the current page. Crawling a traditional web application requires to extract these embedded URLs and traverse them in an effective sequence. But in RIAs, the client-state can also change by executing events which are user actions (or time-outs) that trigger client-side code execution and hence cannot be mapped to a single URL. All these changes mean that traditional crawlers are unable to crawl RIAs, save for a few pages that have distinct URLs.

An important functionality of the web in general is the information it provides. This information can only be made available if the different information sources can be found and indexed. If search engines are not able to crawl websites with new information, they will not be able to index them. Hence a good part of the web in general will be lost. In addition, crawling is also required for any thorough analysis of the web application such as for security and accessibility testing. To our knowledge, none of the current search engines, web application testers and analyzers have the ability to crawl RIAs [1].

In this paper, we introduce a RIA crawling strategy using a statistical model. This strategy is based on the model-based crawling approach introduced in [3] to crawl RIAs efficiently. We evaluate the performances of our statistical model on two real RIAs and two test applications. We further compare our experimental results against other RIA crawling strategies, the Depth-First, the Breadth-First and the Hypercube [3], and we show that the new strategy obtains overall better results.

The paper is organized as follows: In Section 2, we give the basic concepts in RIA crawling. In Section 3, we present the details of the new strategy based on statistical model. In Section 4, we provide experimental results obtained with our prototype. We conclude in Section 5. We omit the related works for space restrictions.

2 Crawling RIAs

A web application can be conceptualized as a Finite State Machine with "states" representing the distinct DOMs that can be reached in the web application and transitions representing event executions. The result of crawling is called a "model" of the application. A model basically contains the states and the possible ways to move from one state to another.

A crawling strategy is an algorithm that decides how the exploration proceeds. In the case of event-based exploration of RIAs, the strategy basically decides which event to explore next. We say that a crawling strategy that is able to find the states of the application early in the crawl is an efficient strategy, since this is the goal of crawling. This is important, since for large RIAs it might not be feasible to wait for the crawler to complete the crawl. In this case, a strategy which discovers a larger portion of the application early on will deliver more data during the alloted time, and thus be more efficient. However, the main problem is that we do not know how the web application has been built and without this prior knowledge of the web application, finding an efficient strategy is difficult.

Primarily motivated by the above goals, we introduced the concept of "Model-Based Crawling" in [3]. Along with that we also introduced a two phase approach. The first phase, "state exploration phase", aims at discovering all the states of the RIA. Once our strategy believes that it has probably found all the reachable states, we proceed to the second phase, the "transition exploration phase" which tries to execute the remaining transitions after state exploration, to confirm that nothing has been overlooked.

In [2], we compiled a list of challenges and assumptions such as state equivalence, user-inputs, server states; which are important to be able to design an efficient crawling strategy and can be handled as separate research efforts.

3 The Probability Strategy

A crawling strategy can be efficient if it is able to predict the results of the event executions with some degree of accuracy. This helps give priority to the events that are more likely to discover new states and hence improve efficiency. Statistics about the past behavior of the event (from different states) can be used to model the future behavior of the event. With this motivation, we introduce a crawling strategy which uses statistical data collected during crawling. The strategy is based on the belief that an event which was often observed to lead to new states in the past will be more likely to lead to new states in the future. We call this new strategy "the Probability strategy".

3.1 Events' Probability of Discovering New States

Let P(e) be the event e's probability of discovering a new state. Remember that the same event "e" can be found in different states (we say that e is "enabled" in these states). The following Bayesian formula, known as the "Rule of Succession" in probability theory, is used to calculate P(e)

$$\mathrm{P(e)} = \frac{\mathrm{S(e)} + \mathrm{p_s}}{\mathrm{N(e)} + \mathrm{p_n}}$$

where

- N(e) is the "execution count" of e, that is, the number of times e has been executed from different states so far.

- S(e) is the "success count" of e, that is, the number of times e discovered a new state out of its N(e) executions.
- p_s and p_n are the terms to represent initial success count and initial execution count respectively. These terms are preset and represent the initial probability of an unexplored event to find a new state.

This Bayesian formula is useful for estimating the probabilities in situations when there are very few observations. To use this formula we assign values to p_s and p_n to set the initial probability. For example, $p_s = 1$ and $p_n = 2$ can be used to set an event's initial probability to 0.5 (note that $N(e) = S(e) = 0$ initially).

Having Bayesian probability instead of using the "classical" probability, $P(e)=S(e)/N(e)$, with some initial values for P(e), avoids in particular have events that get a probability of 0 because no new state were found at their first execution. With our formula, events never have a probability of 0 (or 1) and can always be picked up after a while.

3.2 Choosing the Next Event to Explore

In this section, we describe the logic that helps the strategy decide which event to explore next. We first introduce the notation and definitions used.

- S denotes the set of already discovered states. Initially S contains the initial state.
- $s_{current}$, represents the current state, the state we are at currently in the application while executing the crawl. $s_{current}$ always refers to one of the states in S.
- For a state s, we define the probability of the state, P(s), as the maximum probability of an unexecuted event in s. If s has no unexecuted events then $P(s) = 0$
- d(s, s') is the distance from $s \in S$ to $s' \in S$. It is the length of the shortest path from s to s' in the model of the application discovered so far.

When deciding which event to explore next, the Probability strategy aims to take the action that will maximize the chances of discovering a new state while minimizing the cost (number of event executions). For this reason, starting from the current state $s_{current}$, we search for a state s_{chosen} such that exploring the event with probability $P(s_{chosen})$ in s_{chosen} achieves this goal.

All the states that still have unexplored events are candidates to be s_{chosen}. However we have to take into account the distance from the $s_{current}$ to the s_{chosen} in addition to the raw probabilities when deciding s_{chosen}. Note that from $s_{current}$ reaching to any other state in S means following a known path (consisting of already explored events). Between two states that are at different distances from $s_{current}$, we may consider reaching the one that is farther away because of its higher probability. However, the time to execute the extra events in this path could actually be used for exploration if the closer state is chosen. To make decisions in such situations we need to balance the cost of executing known events with the higher probability of the farther state.

For our analysis it is necessary to have an estimation of discovering a state by exploring an event from an "average" state in S. For this purpose, we will use the average probability P_{avg} that is defined as follows.

$$P_{avg} = (\Sigma_{s \in S} P(s)) / |S|$$

To select a state that maximizes the probability while minimizing the cost, we need a mechanism that compares two states and decides which is more preferable. Let's say we want to compare s and s'. If both are at the same distance from $s_{current}$ then the one with the higher probability is obviously a better choice. But if the cost of reaching one of the states, is higher than the other, say $d(s_{current}, s) < d(s_{current}, s')$) then there can be two cases. If $P(s) \geq P(s')$ then s is clearly a better choice. But if $P(s) < P(s')$ then the fact that reaching s' is costlier than reaching s should be reflected in the comparison mechanism. To make up for the difference in the cost, we should allow the exploration of a sequence of $k = d(s_{current}, s') - d(s_{current}, s)$ extra events after executing the event with probability P(s) from s. Thus we use the probability of discovering a new state after executing the event from S and executing k more unexecuted events (each with a probability of P_{avg} to discover new state). This is given by the following formula

$$1 - (1 - P(s))(1 - P_{avg})^{d(s_{current},\ s) - d(s_{current},\ s')} \qquad (1)$$

Now we can compare this value with P(s') and choose the option with higher probability.

Summarizing, the s_{chosen} that we are looking for is the state, $s \in S$ that satisfies the following condition

$\forall\ s' \in S$

- if $d(s_{current},\ s) = d(s_{current},\ s'),\ P(s') \leq P(s)$
- if $d(s_{current},\ s) < d(s_{current},\ s'),\ 1 - (1 - P(s))(1 - P_{avg})^{d(s_{current},\ s') - d(s_{current},\ s)} \geq P(s')$
- if $d(s_{current},\ s) > d(s_{current},\ s'),\ 1 - (1 - P(s'))(1 - P_{avg})^{d(s_{current},\ s) - d(s_{current},\ s')} < P(s)$

3.3 The Algorithm

In this section we give an algorithm that picks an s_{chosen} from S. The algorithm initializes the variable s_{chosen} to the $s_{current}$ and proceeds in iterations. At iteration i the states at a distance i from the $s_{current}$ are compared against the current s_{chosen}. We check if any of them is more preferable to $s_{chosen.}$

We optimize the search by exploiting the fact that we do not necessarily need to explore all the states in S to find s_{chosen} .The search can be stopped the moment we detect that it is not possible to find any state further away with a higher probability. This is possible since we take into account the cost of distance while comparing the probability of states. We only need to know P_{best}, the probability of the state with maximum probability in S.

Then the maximum distance that needs to be considered from s_{chosen} (noted as maxDistanceToCheckFrom(s_{chosen})) is the value of smallest d that satisfies

$$1 - (1 - P(s_{chosen}))(1 - P_{avg})^{d} \geq P_{best} \qquad (2)$$

When the left hand side of (2) becomes as large as P_{best} then it is not required to look further since even the states that might have the maximum probability, P_{best}, will not be preferable anymore to s_{chosen} due to the distance factor.

```
Algorithm ChooseStateToExplore
$s_{chosen}$ := $s_{current}$; i := 1; distanceToCheck := maxDistanceToCheckFrom($s_{chosen}$);
while ( i < distanceToCheck) {
    Let s be the state with max probability at distance i from $s_{current}$;
    if (s is preferable to $s_{chosen}$) {
        $s_{chosen}$ := s;
        distanceToCheck += maxDistanceToCheckFrom($s_{chosen}$);
    }
    i++;
}
return $s_{chosen}$;
```

4 Experimental Results

In this section, we evaluate the performance of the Probability strategy on two real RIAs and two test RIAs. We have used the following metrics for performance evaluation.
(1) Number of events and resets required to discover all states
(2) Number of events and resets required to explore all transitions

A reset is loading the URL of the application to go back to the initial state. Resets are typically costlier (in terms of time of execution) than event execution. For simplicity we have combined the events and resets required for state exploration and transition exploration into a single cost factor. For this purpose, we have expressed the cost of resets in terms of number of event execution (the actual value used is application dependent). We believe that number of events execution is a good metrics for performance evaluation, since the time to crawl is proportional to the number of events executed during the crawl.

We compare the performance of our strategy with the Breadth-First and the Depth-First strategies and our existing Hypercube strategy. We also present, for each application the optimal number of events executions to explore all the states of the application. It is important to understand that this optimal value is calculated after the fact, once the model of the application is obtained. This number is found by an Asymmetric Traveling Salesman Problem (ATSP) solver [4] on the graph instance obtained for the application.

In an effort to minimize any influence that may be caused by considering events in a specific order, the events at each state are randomly ordered for each crawl. Also, each application is crawled 5 times with each method and the average cost of these 5 runs is used for comparison.

The first real RIA we consider is an AJAX-based periodic table[2]. In total 240 states and 29034 transitions are identified by our crawler and the reset cost is 8. The second real application considered is Clipmarks[3]. For this experimental study we have used a partial local copy of the website. It consists of 129 states and 10580 transitions and

[2] http://code.jalenack.com/periodic (Local version: http://ssrg.eecs.uottawa.ca/periodic/)
[3] http://www.clipmarks.com/ (Local version: http://ssrg.eecs.uottawa.ca/clipmarks/)

the reset cost is 18. The third application, TestRIA is a test application that we developed using AJAX[4]. It has 39 states and 305 transitions and a reset cost of 2. The fourth application is a demo web application maintained by the IBM® AppScan® Team[5]. We have used the AJAX-fied version of the website. The application has 45 states and 1210 transitions and a reset cost of 2.

4.1 State Exploration

For compactness we have used boxplots: the top of vertical lines show the maximum number required to discover all the states.The lower edge of the box, the line in the box and the higher edge of the box indicate the number required to discover a quarter, half and 3 quarters of all the states in the application, respectively.

For all applications, Probability strategy has performed significantly better than the Breadth-First and the Depth-First strategies. The paper [3] proved the efficiency of the Hypercube strategy compared to the current state of the art commercial products and other research tools. Probability strategy also showed better performance than the Hypercube strategy. The box plots are drawn in logarithmic scale.

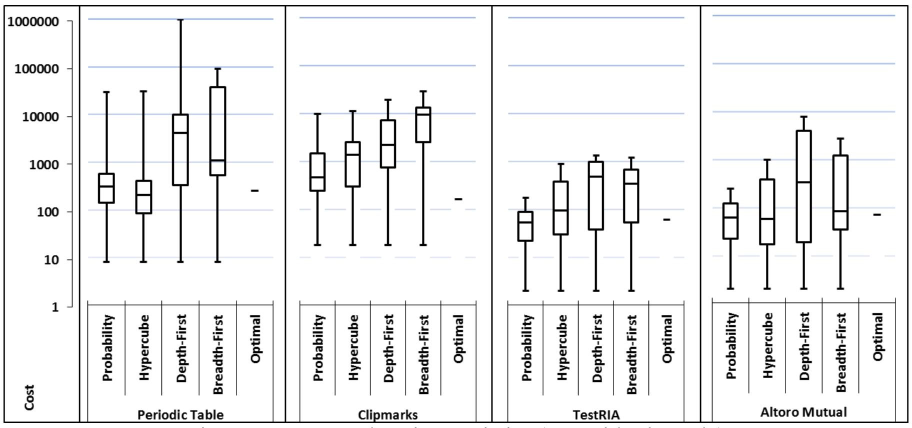

Figure 1: State exploration statistics (Logarithmic scale)

4.2 Transition Exploration

Table 1: Transition Exploration Statistics

	Periodic Table	Clipmarks	TestRIA	Altoro Mutual
Probability	31403	12344	979	2526
Hypercube	31810	12356	996	2542
Depth-First	983582	32026	1345	7693
Breadth-First	181924	19914	1324	3744

[4] http://ssrg.eecs.uottawa.ca/TestRIA/

[5] http://www.altoromutual.com/

Table 1 presents the overall cost of crawling. For all applications, the cost required by the Probability strategy is better than or comparable to the Hypercube strategy but it exceeds the Depth-First and the Breadth-First strategies by a significant margin.

5 Conclusion

We have presented a new crawling strategy based on the idea of model-based crawling introduced in [3]. Experimental results show that this strategy outperforms the standard crawling strategies by a significant margin. Further, it also outperforms the Hypercube strategy in most cases and it performs comparably in the least favorable example, while being very much simpler to understand and to implement. This makes Probability a good choice for general purpose crawling. When compared to the optimal solution, there is still some room for improvement. However, the optimal solution is calculated after the website model is known, and thus can only be used as a benchmark, not to actually crawl an unknown web application.

Acknowledgments. This work is supported in part by IBM and the Natural Science and Engineering Research Council of Canada.

References

1. Bau, J., Bursztein, E., Gupta, D., and Mitchell, J.C.: State of the Art: Automated Black-Box Web Application Vulnerability Testing. In Proc. IEEE Symposium on Security and Privacy. (2010).
2. Benjamin, K., Bochmann, G.v., Jourdan, G.V., and Onut, I.V.: Some Modeling Challenges when Testing Rich Internet Applications for Security. In First International workshop on modeling and detection of vulnerabilities. Paris, France. (2010).
3. Benjamin, K., Bochmann, G., Dincturk, M.E., Jourdan, G.-V. AND Onut, I.V., A Strategy for Efficient Crawling of Rich Internet Applications. In S. Auer, O. Díaz & G. Papadopoulos, eds. Web Engineering: 11th International Conference, ICWE 2011,Paphos, Cyprus. Springer Berlin / Heidelberg. 74-89. (2011)
4. Carpento, G., Dell'amico, M. and Toth, P., 1995. Exact solution of large-scale, asymmetric traveling salesman problems. ACM Trans. Math. Softw., 21(4)
5. W3C. Document Object Model (DOM). http://www.w3.org/DOM/ (2005)
6. Jesse James Garrett. Adaptive Path. http://www.adaptivepath.com/publications/essays/archives/000385.php (2005)