2018

Adaptive Random Testing in Detecting Layout Faults of Web Applications

Elmin Selay
University of Wollongong, ei978@uowmail.edu.au

Zhi Quan Zhou
University of Wollongong, zhiquan@uow.edu.au

Tsong Yueh Chen
Swinburne University of Technology, tychen@swin.edu.au

Fei-Ching Kuo
Swinburne University of Technology, dkuo@swin.edu.au

Adaptive Random Testing in Detecting Layout Faults of Web Applications

Abstract

As part of a software testing process, output verification poses a challenge when the output is not numeric or textual, such as graphical. The industry practice of using human oracles (testers) to observe and verify the correctness of the actual results is both expensive and error-prone. In particular, this practice is usually unsustainable when developing web applications - the most popular software of our era. This is because web applications change frequently due to the fast-evolving requirements amid popular demand. To improve the cost effectiveness of browser output verification, in this study we design failure-based testing techniques and evaluate the effectiveness and efficiency thereof in the context of web testing. With a novel application of the concept of adaptive random sequence (ARS), our approach leverages peculiar characteristics of failure patterns found in browser layout rendering. An empirical study shows that the use of failure patterns and inclination to guide the testing flow leads to more cost-effective results than other classic methods. This study extends the application of ARSs from the input space of programs to their output space, and also shows that adaptive random testing (ART) can outperform random testing (RT) in both failure detection effectiveness (in terms of F-measure) and failure detection efficiency (in terms of execution time).

Disciplines

Engineering | Science and Technology Studies

Publication Details

Selay, E., Zhou, Z.Q., Chen, T. & Kuo, F. (2018). Adaptive Random Testing in Detecting Layout Faults of Web Applications. International Journal Of Software Engineering And Knowledge Engineering, 28 (10), 1399-1428.

International Journal of Software Engineering and Knowledge Engineering
Vol. 28, No. 10 (2018) 1399–1428 DOI: 10.1142/S0218194018500407

Adaptive Random Testing in Detecting Layout Faults of Web Applications *

Elmin Selay

Institute of Cybersecurity and Cryptology,
School of Computing and Information Technology,
University of Wollongong, Wollongong, NSW 2522, Australia
ei978@uow.edu.au

Zhi Quan Zhou†

Institute of Cybersecurity and Cryptology,
School of Computing and Information Technology,
University of Wollongong, Wollongong, NSW 2522, Australia
zhiquan@uow.edu.au

Tsong Yueh Chen

Department of Computer Science and Software Engineering,
Swinburne University of Technology,
Hawthorn, VIC 3122, Australia
tychen@swin.edu.au

Fei-Ching Kuo

Department of Computer Science and Software Engineering,
Swinburne University of Technology,
Hawthorn, VIC 3122, Australia
dkuo@swin.edu.au

As part of a software testing process, output verification poses a challenge when the output is not numeric or textual, such as graphical. The industry practice of using human oracles (testers) to observe and verify the correctness of the actual results is both expensive and error-prone. In particular, this practice is usually unsustainable when developing web applications – the most popular software of our era. This is because web applications change frequently due to the fast-evolving requirements amid popular demand. To improve the cost effectiveness of browser output verification, in this study we design failure-based testing techniques and evaluate the effectiveness and efficiency thereof in the context of web testing. With a novel application of the concept of adaptive random sequence, our approach leverages peculiar characteristics of failure patterns found in browser layout rendering. An empirical study shows that the use of failure patterns and inclination to guide the testing flow leads to more cost-effective results than

*A preliminary version of this paper appeared in the Proceedings of the 2014 International Conference on Digital Image Computing: Techniques & Applications (DICTA) [1].
†Corresponding author: Tel: (+61-2) 4221 5399

other classic methods. This study extends the application of adaptive random sequences from the input space of programs to their output space, and also shows that adaptive random testing (ART) can outperform random testing (RT) in both failure detection effectiveness (in terms of F-measure) and failure detection efficiency (in terms of execution time).

Keywords: Adaptive random testing; failure-based testing; web testing; adaptive random sequence; graphical output verification; layout fault; failure pattern.

1. INTRODUCTION

With incredibly fast adoption over the two decades, the number of global Internet users stands at 3.2 billion as of 2015 [2]. This has not only reshaped how we interact with each other, but also has irreversibly transformed the way we live. Web applications have become the most produced and used software products, attracting more and more attention from both the industry and the academia.

The sole purpose of a webpage is no longer to deliver information in a simple format as decades ago. Driven by the rapid growth of the demand for complex aesthetics and functionality, web applications are much harder to develop, maintain and test[3, 4]. As an important component of the software life cycle, the main purpose of testing is to make sure that the target web application offers consistent look and feel for the target audience.

During both development and maintenance, testing is conducted to assure quality and reliability by detecting failures so that they are corrected before causing usability problems for end users. Layout issues are not usually easy to identify, especially when this process is automated. Human testers must inspect each webpage by eye to verify correctness even in automated scenarios. Unlike other software products, the standard-compliant code of the web application does not guarantee its correct look and feel. Obviously, manual work of this type is time-consuming and therefore expensive. Additionally, such work multiplies over time given that we have to do testing again (regression testing) when we introduce a new enhancement, patch or fix to an application that was already tested[5, 6].

The test suite grows and becomes more complex after each phase because the tester would ideally want to run all previous test cases besides new test cases. This is to make sure the new feature or modification does not affect previously working functionality.

One of the major directions in attempts to verify correctness of the output in web applications has been to achieve an effective visual analysis of layout images captured from different browsers by automated testing tools such as Selenium [7]. A common testing approach is to conduct pixel-level comparison between a screenshot of the observed output and a previously taken reference image [8]. Since such screenshots are large in size and expected to be different to certain extent (such as differing in a few stand-alone pixels) even when no faults exist, it is impractical to use standard image matching, pattern recognition, keypoint matching, histogram similarity/difference analysis, perceptual comparison, hashing, or other advanced

methods as these methods are either expensive or too difference-sensitive. In this paper, our focus is to discover the presence and location of structural layout faults (significantly visible blocks), if any, efficiently at an affordable computational cost.

In this paper, we propose a *failure-based testing* approach [9] to leverage the characteristics of browser layout rendering to curb testing complexity and improve cost effectiveness. Our approach is based on the knowledge about web application layout failure patterns. We apply varieties of *adaptive random testing* (ART) to achieve an even distribution of reference points in the *output space* although in the past ART has only been used for test case selection in the *input space* [9, 10]. We evaluate the effectiveness and efficiency of the proposed approach in the context of an enterprise-level case study for automated testing of selected real-world web applications. The empirical results show that our approach is more effective and more efficient compared with conventional approaches.

The rest of this paper is presented as follows: Section 2 explains the background to the problem and reviews related work in the field of web application testing. Section 3 presents our approach and its internal working mechanism. Section 4 gives the implementation and configuration details of our testing framework. Section 5 presents an empirical study and its results. Section 6 summarizes the contributions of this work and concludes the paper.

2. BACKGROUND AND RELATED WORK

2.1. *Web browsers*

A web browser allows running web application on the user's machine which can be any Internet-enabled devices, including but not limited to mobile phones, tablets, laptops and desktop computers. The primary function of a browser is to visually render the output by interpreting markup directives, auxiliary style and control instructions the accessed web resource provides [11]. As soon as the commercialization of the Internet emerged in early 1990s, web browsers started to differ with their own extended features and layout rendering engines. Established in 1994, the World Wide Web Consortium (W3C) undertook to regulate interoperability and compliance among browsers. However, despite all endeavours over the past two decades, the full standardization and interoperability have yet to be seen in this area. Assuming the application developers do not impose a restriction on browsers or devices, they must do significant amounts of cross-browser and cross-device testing to achieve consistency. To target particular browsers, developers use conditional statements or certain directives which are only read by specific browsers. As mobile and hand-held devices started to become more popular, dedicated mobile sites in addition to desktop versions started to emerge. However, increased cost and the need to improve user experience across all devices forced designers and developers to shift to responsive design patterns where the application adapts to device screen size [12].

2.2. *Testing a web application*

Testing web applications inherits most of the standard testing methods applicable to software as a whole. Being an intangible component of computers, software is developed by programmers, who translate the requirements specification into machine-understandable instructions in the high level. Obviously, the code written by the programmer may not necessarily be a correct translation of the requirements, and may not necessarily cover all items in the specification document. We are often left with only one option: testing the end product. There exist many types of software testing. Regression testing plays a significant role in software maintenance to ensure that new enhancements added to the existing application do not break any previously tested components or the end-product as a whole [5, 6, 13]. Two key test components are required in the conduct of testing: test data (or a suite of test cases or selection of reference points in our case) and test oracle. Test data mainly refer to the inputs given to the application while the test oracle is used to describe the mechanism against which the result is checked to determine whether it is correct [14, 15]. In practice, human oracles observe and verify the correctness of the actual result when the quantity is at a reasonable level. However, this is not the case when it comes to enterprise-scale applications. Oracles for automated software testing are considered as expensive and difficult to develop since this process involves elicting an oracle from requirements specification, program simulation or a trusted implementation of the product [3, 14]. Differently from other software products, web applications produce their output in visual form in a browser screen. The visual output can differ for users using different devices, browsers or a combination of them, further complicating the verification by non-human test components. This paper mostly focuses on the effective strategies to facilitate such initial verification by non-human actors in test-driven environments or during the maintenance cycle in the face of changing requirements.

2.3. *Motivating example and problem statement*

We presented a simple example in Figure 1 to illustrate the relevance of the problem and motivation behind the automated approach to web application testing. The following HTML and CSS snippet which is validated as standard-complaint by the W3C Validation Service renders inconsistent visual output in different browsers running on the same machine, as shown in Figures 2 to 6.

This example illustrates that standard-compliant code does not necessarily guarantee the intended look and feel in the web applications unlike other software products where the main focus is on correct implementation. In such cases, testing tools used to verify correctness of markup, object representation and structure cannot warrant the consistency of the rendered visual output across multiple platforms and browsers (and their different versions). Modifying the same markup to achieve reasonably consistent or similar output across all browsers requires a large amount of development and testing effort. Given the scale of code in an average industrial web

```
   <!DOCTYPE html>
<html lang="en-AU">
<head>
    <style type="text/css">
     body{
       padding:10px;
     }
    .box {
       width:250px;
       height:250px;
       float:left;
       border-radius:50%;
       overflow:hidden;
       transform:skew(10deg,15deg);
       border:solid 4px black;
       box-shadow: 0 0 0 8px #ff0030;
       color:black;
       text-align:center;
       font-size:2.2rem;
       line-height:250px;
     }
    .box.blue-red {
        background:
        linear-gradient(red, yellow, blue);
        content: "Test";
        transform: rotate(120deg);
        letter-spacing: 4px;
     }
    .box.blue-red span:after{
      content:"Run";
    }
    .box:first-child {
       margin-right:40px;
    }
    img {
       height: 100%;
       width: 100%;
       object-fit: contain;
     }
    </style>
</head>
<body>
    <div id="main">
        <div class="box">
<img src="//cdn.yoomroo.com
/assets/test/paris-test.jpg" />
        </div>
        <div class="box blue-red">
          <span>Test</span>
          </div>
    </div>
</body>
</html>
```

Fig. 1: A simple CSS AND HTML example.

application, it is not difficult to imagine the amount of efforts and costs required. Additionally as web applications tend to change more frequently, each enhancement added to the previously tested application requires regression testing to ensure that the new feature does not cause problems with any previously working components. This motivation example demonstrates the need for effective detection of inconsistencies in web interfaces to minimize human oracles - involvement and judgement

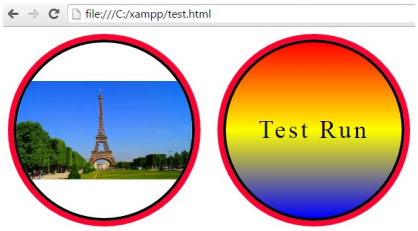

Fig. 2: Google Chrome, version 35.0.1916.114 m

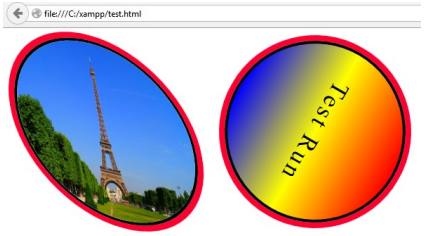

Fig. 3: Mozilla Firefox, version 29.0.1

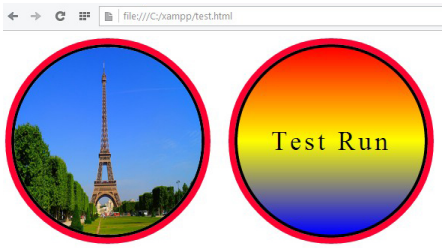

Fig. 4: Opera, version 17.0.1241.45

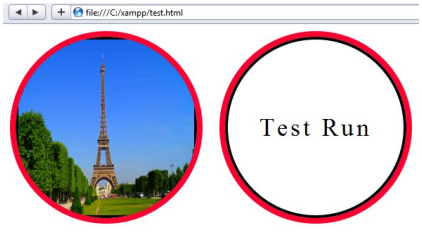

Fig. 5: Safari, version 5.1.7 (7534.57.2)

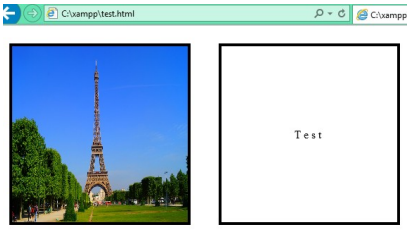

Fig. 6: Microsoft Internet Explorer 7 (Mode)

of a human tester. Since regression testing involves running previous test cases, it is possible to collect sufficient results (e.g. screenshots) previously obtained on the same browsers / platforms or on different browsers / platforms to compare against (as an oracle) in the future tests. This is useful particularly in the case of testing web applications since there are frequent changes during or after the development, usually without comprehensive requirements specifications.

2.4. *Related Work*

Over the past decades significant efforts have been made by the academia and the industry towards testing and automated testing of graphical user interfaces (GUI)[16]. However, much of the work primarily deals with the verification of functionality rather than look-and-feel. The state of the art prevalent in the industry involves a manual and visual inspection of a webpage to decide whether the output rendered is expected or acceptable. There are automated testing tools but they either consequently depend heavily on human oracles or lead to unnecessary and high computational overhead in the verification of the output. Two popular solutions to automating web application testing in the past decade were Microsoft Expression Web [17] and Adobes Browser Lab [18]. The former was intended to validate the standards-compliance of the markup without any feedback about its look and behaviour in different browsers or screen sizes, while the latter, which was discontinued by Adobe after May 2013, provides diagnostic tools and screenshots, leaving it to the web developer to make a judgement. Selenium has become the most used tool to automate the process of programmatically capturing screenshots. These tools require significant developer time and costs in visually observing and evaluating what the issue is. There have been dozens of similar commercial tools or online testing platforms available with no easy-to-use configuration options. The main challenge facing both the research community and industry has been testing oracle automation: how to automatically verify pass-fail states. Literature before mid-2000s mostly dealt with the white-box testing of web applications. During this time web testing was a newborn notion and Microsoft Internet Explorer had a dominant position in the browser space, leaving little concern of multi-browser compatibility issues. However, the landscape has changed rapidly amid the surging global Internet access rates.

One of the first attempts to test dynamic web applications was the VeriWeb project which targeted functional testing of web application [19].

Ricca and Tonella (2002) introduced a concept relying on two distinct models of the application: a navigation model and a control flow model [20]. Their strategy covered white-box testing of the application under test. A roadmap was presented by the same authors to include function testing of web applications in 2005 [21]. Their testing approach covered functional testing, code coverage testing and model based testing.

Bedi and Schroeder (2004) detailed challenges of web application testing, pre-

senting interesting observations about the problems of testing an e-commerce application [22]. Di Lucca and Fasolino (2006) outlined that the problem of web applications testing must be carried out at different levels and from various aspects due to the complexity involved [23]. A new fault taxonomy was presented to embrace the probable fault-causing particularities of web applications [24]. Eaton and Memon (2007) proposed a compliance evaluation technique, where developers would manually supply examples of both correct and incorrect webpages for the calculation of probability of faultiness for the next tag [25]. A constraint of this solution was that it ignored Cascading Style Sheets (CSS), which defines layout and style of an HTML document.

As web technologies began to become more popular in the light of social media boom and soaring Internet penetration rate across the globe, the concept of automating web testing and oracle generation drew interest from the academia and industry with initial approaches primarily focused on HTML document verification. Oracle automation using screenshot comparison, which was explored as a possible solution years ago [26], started to gain renewed attention in the field of web application testing. Choudhary et al. presented a tool named WebDiff to automate the identification of cross-browser discrepancies [3]. Their proposed approach included using the Document Object Model (DOM) information and histogram-based image matching of screenshots. With its novelty considered, this approach was one of the most notable steps taken towards automated web application testing.

Mesbah and Prasad came up with a new automation method in this field [27]. Their strategy included using an Ajax crawler Crawljax (open-source) to capture and store observed behaviour of the webpage for each browser as a what they called a "finite-state machine navigation model". In the next step, they suggested comparing generated models to determine where they differ. The equivalence checking is executed by producing state graphs out of the generated navigation models and comparing their edges and nodes. Each node represented an abstraction of the DOM tree instance. Although this was a step towards an automated testing process, the data collected and the comparisons are done at the DOM level.

Another recent addition to the toolset has been WebMate, a tool which primarily focuses on enhancing the coverage and automated crawling rather than output verification [28].

There have recently emerged a number of other tools designed to streamline regression testing of web applications but they leave judgement to human testers [29, 30, 31].

3. OUR APPROACH AND CONCEPTUAL FRAMEWORK

3.1. *An insight*

Since our aim is not to propose a new strategy for testing web applications but is to present a new cost-effective output verification approach, the rest of the paper mainly focuses on this direction. As outlined in Section 2.4, the current state of the

art of web application testing involves automated crawling and image-comparison-based output verification in addition to DOM-based testing. In the literature, image matching algorithms have been designed to either compare the images for difference, similarity or identity, or for use with sophisticated approaches such as perception metrics or pattern recognition. However, these comparison algorithms do not satisfy the needs of web application testing in relation to screenshot comparison. As web applications are dynamic, due to the expected presence of dynamic regions such as advertisement banners, images to be compared are expected to be different and similar to a certain degree. In addition to inclusion and exclusion regions, given the scale and frequency of regression testing the comparison method should be efficient in terms of execution time.

Comparing screenshots by their histograms which in the form of data represent the distribution of the feature in the image results in a bin-to-bin or cross-bin comparison. An approach of this kind has previously been applied by Choudhary et al. for the purpose of comparing web application images [3]. To deal with the false positive instances of detection in a basic histogram comparison, Choudhary et al. resorted to the Earth Movers Distance (EMD). Being an advanced effective method, EMD is used in the field of computer vision to effectively compare discrete distributions, where there is a need to neglect small-sized variations in the conduct of shape matching, texture analysis, image retrieval and object recognition. This is achieved by measuring the given distance between two probability distributions over a region corresponding to Wasserstein metric [32, 33, 34]. For the purpose of histogram measurements, the time required for calculating the dissimilarity between two non-identical histograms is proportional to the given size of the histograms. This approach has also been used in different domains of the industry. For example, to evaluate similarity of colour histograms, a new class of quadratic form distance functions is successfully used in IBM's QBIC image retrieval system [33]. Among other factors such as implementation, its time complexity stands out as the main concern of this solution.

It is to be noted that, due to special characteristics, browser screenshot comparison requires a low-cost, easy-to-implement, and still cost-effective method. For instance, identification of objects or patterns with precision is not required. Likewise, owing to the demand for fast, in-house and customizable solutions, sophisticated methods for keypoint or feature recognition are not appropriate for a test-driven web development process.

Bitmap-based image comparison is easy to use, flexible and widely available in the industry. This strategy includes pairwise comparison of pixels between the reference image (oracle) and the compared image (pixels in the exclusion zones are neglected). The intuitive approach would be to iterate pixel by pixel over the image and retrieve the colour of pixels that have the same coordinates and compare them for consistency. However, this is a non-trivial task given that large screenshots can contain millions of pixels (reference points).

In order to provide background information why it may not be necessary to

compare all pixels for the identification of browser layout inconsistencies, we have to look at failure patterns in the next section.

3.2. *Failure patterns and failure rate*

Certain common patterns have been observed by the researchers on the geometry of failures within the input space [35, 36, 37, 38, 39]. Boyer et al. found that input data at or near intersections of the domain regions are more likely to cause failure [35]. White and Cohen demonstrated that failures tend to clump closer to the input domain boundaries, forming hyperplanes [36]. Ammann and Knight showed that failure regions happen to manifest themselves as locally continuous and their geometry provide information about the possible performance of test cases selected in these regions [37]. Finelli noted that continuous regions across the input domain have so-called error crystals which lead to program failures [38]. Bishop observed contiguous failure and success regions across the input space, concluding that contiguous failure regions often form angular and elongated shapes, as opposed to the theorized "blobs" [39]. In a pioneering work by Chen et al., it was further explained that in terms of their distribution and formation, failure-causing inputs create three common patterns, namely, block pattern, strip pattern, and point pattern [40] as in Figure 7, where the non-point patterns are more common in real-world applications.

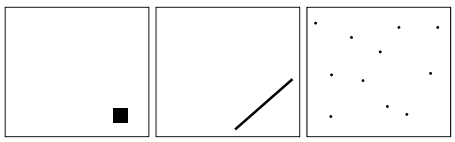

Fig. 7: Failure patterns: block pattern (left), strip pattern (middle), and point pattern (right).

According to this observation, block and strip patterns represent contiguous failure regions, or certain segments of the input space where failure-causing inputs happen to form clusters. Using the failure pattern information, the testing process flow can be guided to improve failure-detection effectiveness of the test cases.

One of the principal attribute associated with software under test is its failure rate, defined as the ratio of total failure area to the input domain area [41, 42]. The failure rate is fixed, but unknown prior to testing.

3.3. *Our observation: Failure pattern in web layouts*

As specified in W3C standards [43], elements in the web document are two-dimensional squares laid out one after another or nested inside each other. This type of arrangement is referred to as a flow. The flow denotes "by default, elements

flow one after another in the same order as they appear in the HTML source, with each element having a size and position that depends on the type of element, the contents of the element, and the display context for the element as it will render on the page" [44]. Within the rectangular space occupied by the elements, different shapes of objects can be rendered by masking them or changing their border radius from inside, as in Figure 8.

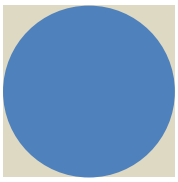

Fig. 8: A visually round object takes a rectangular space.

Taking into account the aforementioned statement, we can intuitively assume that the failure pattern the faulty elements can potentially form is to fall under *non-point failure patterns*. We further observe that, as elements flow in sequence, a faulty element will drive the subsequent element to flow, still forming block failure patterns in a *domino effect*. They may overflow to the next row, further limiting the chance of strip and point pattern formations. The flow of blocks and grids have especially become more noticeable with the emergence of responsive web design trends in the industry. To allow web applications to adjust to the size of the device screen, the width of grids are set in percentage (relative unit) to create a "fluid layout," so the webpage can stretch and contract relative to the user's screen size. As device has a fixed viewport, the grids at the end of the row are pushed to the next row.

In short, we postulate the following failure pattern in web layouts: *The flow behaviour adds special interest to the right and bottom boundaries of the output space because of the likely fault inclination.*

3.4. *Adaptive random testing (ART)*

Software testing process with the black-box approach involves executing test cases to determine whether the output is in line with the expected behaviour defined in the specification. As test cases are to selected from the input domain, the number of possible test cases can be extremely large, making exhaustive testing impractical, if not impossible. Due to time and resource limitations, a test case selection strategy is employed to identify test cases which have higher potential to reveal failures [45, 40].

One of the test case selection methods is random testing, in which samples are selected randomly from the input domain as per the sampling distribution to feed the software in test [46]. Generally, random testing is simple to implement and cheap

to utilize while effective to detect failures. It allows generating input in unexpected ways and ranges to discover the fault when it is not possible or affordable to run all test cases in the input domain [41, 42, 47]. However, random testing is not tailored with the available knowledge for the testing requirements.

To improve this front while keeping the desired benefit of randomness, an ART strategy has been developed [9, 10]. Unlike classic random testing, ART makes use of the knowledge available of previously executed test cases to ensure the next test cases can be selected in such a way that they are more evenly distributed over the input domain. This notion of even distribution here leverages the fact of common contiguous failure patterns to reduce the number of test cases needed to detect the first failure. It has been observed that ART is more suitable for non-point failure patterns [10].

The first widely adopted ART method is the Fixed Size Candidate Set ART (FSCS-ART) [10], which ensures that the qualifying candidate is at the farthest distance from its closest neighbour in the already applied test cases, as further explained in Section 4.5. Compared with classic random testing, test cases generated using FSCS-ART are farther apart from each other and they still preserve the desirable property of randomness. The FSCS-ART algorithm sports an $O(n^2)$ time complexity, where n is the number of test cases generated.

There have been a series of new methods proposed to curb the overhead towards O(n) while preserving the core principle of ART – even spread of the test cases [48, 49, 50, 51, 52, 53]. One such solution is known as "forgetting test cases" [49]. Instead of remembering the entire set of the previously executed test cases, a "forgetting" algorithm chooses to remember only a small and fixed number of previously executed test cases. In this way, an FSCS-ART-with-Forgetting algorithm boasts a linear time complexity of O(n), which is in the same order of complexity as random testing.

In addition, ART's test case generation time can further be reduced by means of the mirroring technique [48], which reflects or translates the test cases originally generated in one subdomain of the input domain to other subdomains at a very low cost (Figure 9).

In this paper, we apply the FSCS-ART-with-Forgetting algorithm together with the mirroring technique to keep time complexity at O(n) while increasing the spread magnitude by up to m, where m is the number of subdomains. In such a Mirror ART (MART) with forgetting algorithm, the main test case selection process is applied to generate new (original) test cases only in the source subdomain (e.g., the upper-left quarter of Figure 9). After the generation of an original test case (e.g., P1 of Figure 9), it is immediately mapped into the mirror subdomains (e.g., P1' in mirror subdomain 1, P1'' in mirror subdomain 2, and P1''' in mirror subdomain 3 of Figure 9) with trivial computational overheads. Section 4 provides technical aspects of practical implementation of the proposed approach.

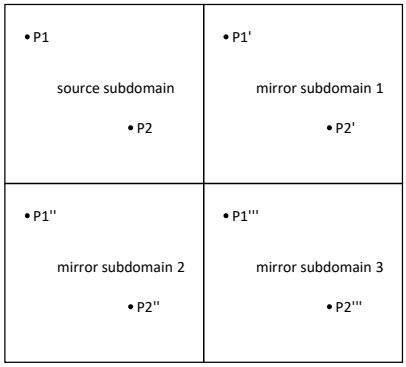

Fig. 9: Mirroring P1 and P2 in neighbouring subdomains.

3.5. *Our approach: Adaptive random sequence (ARS) for output verification*

ART is a strategy that generates or selects test cases from the program's input space. The sequence of test cases generated by ART is called an *adaptive random sequence* (ARS) [9]. A *major innovation* of the present research is the extension of the concept of ARS from the program's input space (for test case generation) into the program's output space (for the verification of resulting images), where ARS is used for the selection of pixels (reference points) when comparing two images. Our objective is to use the ART algorithms to generate ARS of pixels in order to detect differences between images more quickly than using conventional approaches.

In the case of applying MART to image comparison, the screenshot layout is divided into multiple equal-sized subdomains where only one of the subdomains acts as the source subdomain with the possible maximum test cases [a] equal to the number of total test cases in the input domain divided by the number of subdomains. A single reference point generated by the FSCS-ART-with-Forgetting algorithm in the source subdomain is mapped into mirror subdomains. The peculiarity of web layouts as described in Section 3.3 bodes well for mirroring.

When applying MART for test case generation in conventional software testing, which subdomain is designated as the source subdomain and in which order the mirror subdomains are populated might not be a major concern. However, it is a major concern in our context of image comparison, considering the fault-inclination characteristics of browser-rendered layouts discussed in Section 3.3. We postulate that the choice of the source subdomain and the order in which the mirror subdomains are populated are important decisions that would affect the effectiveness and efficiency of image comparison. We have therefore designed different MART algorithms where the upper-left or the lower-right corner of the image is used as

[a] In this paper, when discussing in the context of image comparison, the term "test case" refers to a pixel (reference point) in the image.

the source subdomain, and we hypothesize that the algorithms that first examine the lower-right corner should have improved cost effectiveness compared with those that first examine the upper-left corner (because of the domino effect pointed out in Section 3.3). We have also included ART varieties of four and nine subdomains to investigate the impact of the number of subdomains.

We configure the controller program to exclude reference points within the defined coordinates (within the boundaries of exclusion zones) to avoid detecting insignificant variations (e.g. banner areas), which would otherwise result in false positives.

We believe that combining the forgetting and mirroring strategies will remarkably reduce the computational overhead of the original FSCS-ART algorithm while still keeping an even distribution of the selected reference points. To our knowledge, this is the first work to take advantage of the web layout flow behaviour for automated testing, in both the web industry and academia.

3.6. *Metrics and measurement*

A failure detection approach in software testing is described by certain qualities and benefits such as how efficient and effective it is in terms of the cost, time and resources required. The testing process cannot be infinite and is expected to end either with or without a failure detected within a reasonable runtime. Several metrics have been developed by researchers to gauge the failure-detection capability of a testing method involved.

Among others, the three most common metrics include: P-measure: probability of finding at least one failure by the set of test cases executed; E-measure: expected number of failures detected by the set of test cases executed; and F-measure: expected number of test cases executed to detect the first failure [41, 9, 40]. P-measure and E-measure are characterized with their sampling spread drawing close in normal distribution, whereas F-measure turns out to possess a geometric probability distribution [41, 42]. Which of these metrics best characterize the effectiveness of testing depends on the constraints, conditions and other aspects of the testing. As our case study involve browser layout faults which tend to fall under the category of non-point patterns, the highest priority is given to detecting the first failure, a scenario which better simulates an industry practice where the control is passed by the tester to developers for rectification. Given the significance of discovering the first failure and ART's effectiveness in targeting failures clumped together, Chen et al. suggest that F-measure possess a preferential meaning [41, 42]. Since both P-measure and E-measure are sample size dependent, in view of the lack of justification on the sample size to be used for evaluation, it may not be meaningful to use these two evaluation metrics in our study. As a practicability indicator, execution time is also measured as it shows how quickly in practical terms the failure is detected. We note that execution time can be environment dependent and affected by the computational resources involved.

For the purpose of this paper, F-measure and execution time (ms) taken to detect the first failure are reported, unless indicated otherwise.

4. IMPLEMENTATION AND CONFIGURATION

The automated testing process requires a development environment set up and configured on a computer. [b] There is no specific requirement for a platform or programming language.

4.1. *Testing overview*

A basic testing flow involves a configuration file, a driver program (controller), screenshot capturing, and output verification modules. A tester enters application-specific instructions and parameters, such as application URI, exclusion zones, algorithms, keywords etc. in the configuration file as shown in Section 4.2. The driver program written in the language of choice reads the configuration file and passes parameters to the screenshot capturing module, which is an interface to webdrivers. Using an API of the web driver allows simulating user behaviour such as opening pages and invoking other actions available on the web application. There are a variety of web driver implementations by both the open-source community and the official browser vendors. We used Selenium webdriver for this experiment because of its growing popularity and acceptance rate among the web application developers [7]. For the purpose of this testing, Selenium is only used to capture web application output as screenshots for URLs passed from the driver program 4.2. The captured screenshot pairs are then used as input to the *testing strategy module* (which is the most essential component of our testing framework), which determines the (x, y) origin on the coordinate system (the next reference point to be checked). The testing strategy module includes the implementations of eight test case (reference point) selection algorithms, which will be further explained in Section 4.5. Output verification is performed based on the pairwise comparison of the reference points selected from the two screenshots. As the driver program gets screenshots from the webdriver, stores them in the local file system and calls the testing strategy module, it is where the evaluation metrics described in Section 3.6 (namely, the F-measure and execution time) are applied. These metrics are needed in this study to evaluate and compare the performance of the eight algorithms of the testing strategy module. Test execution is run for each algorithm independently. The testing process requires reliable Internet connection unless the applications are hosted within the local network. The main modules of our testing framework are depicted in Figure 10, and explained in Sections 4.2 to 4.5.

[b] We have made the source code of our testing framework available at http://www.art-research.yoomroo.com/project-output-verification/

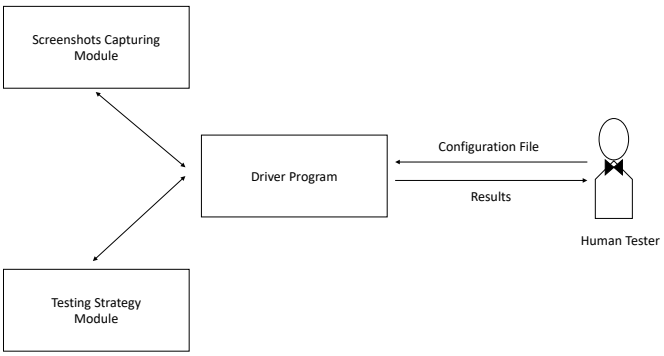

Fig. 10: Main modules of our testing framework.

4.2. *Configuration*

A configuration file written in a human and machine-readable language (YAML, XML etc.) allows using the same testing environment to test different applications. It also provides abstraction to free the tester from knowing internal implementation of the web application. The instructions provided in the configuration file is parsed in the driver program and determines the behaviour and flow of the testing process. If no specific URLs are give in the configuration, they are recursively navigated and crawled starting from the application base URL, also known as the home page.

A sample configuration file can be as simple as shown in Figure 11.

4.3. *Driver program*

The driver program acts as an entry point to start the test process which includes parsing the configuration file, executing the test process accordingly and record experiment results. During the execution of test process, the driver program interacts with the browser webdrivers to automatically capture screenshots of webpages and stores them in the local file system. Once a pair of screenshots are taken from the two versions, it invokes the testing strategy module to execute reference point selection and comparison on the pair. The test cases are reference points identified by (x, y) generated by the algorithm under evaluation in the testing strategy module. The reference points are accessed on the pair and their pixel values are compared to determine whether they look visually identical on the basis of the geometry of RGB (red, green and blue) colour space, a computationally low cost way to compare two colour pixels. Reference points falling within the exclusion areas defined in the configuration file are skipped to avoid detection of false positives due to the dynamic nature of these sections. The test process continues until a non-identical pair of reference points are encountered or stopping criterion is met.

```
---
# config.yml
#
app:
    url: http://myapp.com
    stgurl: http::// stg.myapp.com
    onlytest:
    sitemap:
    links: follow
    output_path: ~/screenshots
    resize: [1600,2000]
    algorithm: MART4
    browser:
       - firefox
       - chrome
    #exclude ad banners by coordinates
    #lt for left-top and rb for right-bottom of rec
    exclude:
    - - lt:
        - 34
        - 200
      - br:
        - 940
        - 200
    ignore-paths:
       - tools
       - admin
```

Fig. 11: A simple configuration file written in YAML.

4.4. *Capturing screenshots*

The ability of programmatically capturing screenshots is an important part of web testing. A webdriver is used to enable communication with browsers. There are open-source and vendor webdriver implementations, providing an API to drive major browsers. As a portable browser automation framework, Selenium provides advanced API and a webdriver component to allow simulating user behaviour on the target applications and take screenshots. Selenium webdriver supports multiple programming languages, and there exist third-party bindings for languages which are not natively supported. To make the testing process simpler, Selenium webdriver does not require a server and it can directly initiate a browser instance. Browser instances running on remote machines can also be used to further reduce test execution time. Using a webdriver allows not only to capture a screenshot of the entire webpage but also a certain section of the webpage. The code snippet in Figure 12 illustrates a simple case of programmatically taking a screenshot through the Firefox browser and storing it in the local file system, using the Selenium webdriver.

4.5. *The testing strategy module*

In this research we investigate the following eight test case selection algorithms: sequential (T), sequential (B), RT, FSCS-ART-with-Forgetting, MART 4, MART 4 (B), MART 9, and MART 9 (B), where the first three serve as experimental controls and the remaining five are varieties of ART algorithms. The testing strategy module

```
WebDriver Driver = new FirefoxDriver();
Driver.get(''http://our-test-domain'');

File shot = ((TakesScreenshot)Driver).
            getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(shot, new File(ScreenshotPath));
```

Fig. 12: Taking a screenshot using the Selenium webdriver.

includes the implementations of these eight algorithms. They are explained in the following subsections. Although existing literature usually use MART 4, we included MART 9 as well to provide additional insight into the benefits of the mirroring technique.

4.5.1. *Coordinate system of a screen*

Before we describe the individual algorithms, it is necessary to introduce the screen coordinate system that is used in this study. The basic unit of measure is typically the pixel. A point on the screen is given by the x- and y-coordinate pairs. The origin (0, 0) is at the top left of the screen. The x-coordinates increase towards the right, and the y-coordinates increase towards the bottom.

4.5.2. *Algorithm 1: sequential (T)*

The first algorithm, denoted by "sequential (T)," is the most fundamental benchmark algorithm to serve as an experimental control, where "(T)" represents "top-down scan." In this algorithm, test cases (reference points) are selected sequentially by scanning the image using the following order:

(0,0), (1,0), ..., (n-1,0),
(0,1), (1,1), ..., (n-1,1),
...,
(0,m-1), (1,m-1), ..., (n-1,m-1),

where n and m are the width and height dimensions of the image. In other words, this algorithm starts by scanning the first (top) line of the image, from left to right, then the second line, from left to right, and so on, until the bottom line. The algorithm will stop when a failure (difference between the two pixels at the same reference point) is detected or when a prescribed stopping criterion is met.

For example, Figure 13 illustrates a 3×3 image containing nine pixels. The sequential (T) algorithm will generate the following sequence of reference points: Pixel_1, Pixel_2, Pixel_3, Pixel_4, Pixel_5, Pixel_6, Pixel_7, Pixel_8, Pixel_9.

$Pixel_1$	$Pixel_2$	$Pixel_3$
$Pixel_4$	$Pixel_5$	$Pixel_6$
$Pixel_7$	$Pixel_8$	$Pixel_9$

Fig. 13: A 3×3 image containing nine pixels.

4.5.3. *Algorithm 2: sequential (B)*

As pointed out in Section 3.3, the flow behaviour (domino effect) adds special interest to the *right* and *bottom* boundaries of the output space. We therefore designed the second benchmark algorithm, named sequential (B), which scans the image from bottom up, and we hypothesize that sequential (B) should be more cost-effective than the sequential (T) algorithm.

In sequential (B), test cases (reference points) are selected sequentially by scanning the vertical lines of an image from bottom up and from right to left, as follows:
(n-1,m-1), (n-1,m-2), ..., (n-1,0),
(n-2,m-1), (n-2,m-2), ..., (n-2,0),
...,
(0,m-1), (0,m-2), ..., (0,0),
For the image illustrated in Figure 13, the sequential (B) algorithm will generate the following sequence of reference points: $Pixel_9$, $Pixel_6$, $Pixel_3$, $Pixel_8$, $Pixel_5$, $Pixel_2$, $Pixel_7$, $Pixel_4$, $Pixel_1$.

It is to be noted that the above sequence is *not* a reverse order of that of sequential (T).

4.5.4. *Algorithm 3: RT*

The third benchmark is the random testing (RT) algorithm. RT selects each reference point by means of *random sampling without replacement*, according to the uniform probability distribution.

4.5.5. *Algorithm 4: FSCS-ART-with-Forgetting*

FSCS-ART To understand the FSCS-ART-with-Forgetting algorithm, it is necessary to first introduce the FSCS-ART algorithm, the pseudocode of which is shown in Figure 14. This pseudocode has been adapted from the original FSCS-ART algorithm [10] for the purpose of selecting a sequence of reference points for image comparison.

The algorithm works as follows: Lines 1 to 4 perform initialization, where n is the number of already-selected reference points, E is the set of already-selected reference points, differenceFound is the variable that shows whether a difference (failure) has

```
1  Set n to 0;
2  Set E to ∅;
3  Set differenceFound to false;
4  Set k to 10;
5  Randomly select a reference point, t;
6  L1: Set n to n+1;
7  IF t detects a difference between the two images THEN
8    Set differenceFound to true;
9    GOTO L2;
10 ENDIF
11 IF the prescribed stopping criterion has been met THEN
12   GOTO L2;
13 ENDIF
14 Add t to E;
15 Randomly select k candidate reference points to form a candidate set C
       = {c1, c2, ..., ck}, where c1, c2, ..., ck are points that have never
       been used for image comparison;
16 FOR each ci ∈ C
17   Set di to minimum(d(ci,e1), d(ci,e2), ..., d(ci,em)), where E
       = {e1,e2,...,em} and d(x,y) denotes the Euclidean distance between x
       and y;
18 ENDFOR
19 Find cj ∈ C such that dj is the maximum among d1, d2, ..., dk. In
       situations where there is more than one candidate satisfying the
       requirement, randomly select a candidate to be cj;
20 Set t to cj;
21 GOTO L1;
22 L2: RETURN t, n, differenceFound;
```

Fig. 14: Pseudocode of the FSCS-ART algorithm adapted for image comparison.

been detected, and k is the size of the candidate set (denoted by C) and the value of k is normally set to 10 [10]. Line 5 randomly selects the first reference point. Lines 7 to 10 mean that if the two pixels at this reference point are found to be different then the algorithm will report the difference and return. Lines 11 to 13 mean that, even if no difference is detected, the algorithm can still return—this will happen when the prescribed stopping criterion is satisfied, for example, when testing resources are exhausted or when all pixels have been compared.

In line 14, the currently compared reference point is added to E, which is the set of already-selected reference points. Line 15 constructs a fixed-size-candidate-set C of size k. The elements of C are different candidate points, and C and E cannot overlap (that is, the candidate points should have never been used for image comparison in previous iterations). In situations where the number of remaining points is smaller than k, all these remaining points should be included in C. Lines 16 to 18 calculate the shortest distance from each candidate point c_i to the set E. Line 19 finds the candidate having the largest shortest distance to E, and such a candidate is selected to be the next reference point (as in line 20) and the current candidate set C is discarded. Line 21 repeats the process.

FSCS-ART-with-Forgetting The FSCS-ART algorithm described above has a time complexity of $O(n^2)$ because of the distance calculations in lines 16 to 18 of Figure 14: For each candidate c_i, its distances to all the previously selected reference points are calculated, which is time consuming. To reduce the computational

overhead, a "forgetting" strategy can be applied: Instead of looking at all the previously selected reference points, we can only look at a small and fixed number of previously selected reference points, and such an algorithm is named FSCS-ART-with-Forgetting.

The pseudocode of our implementation of the FSCS-ART-with-Forgetting algorithm is the same as that shown in Figure 14 except that lines 2 and 14 are interpreted differently: In line 2, E is initialized as an empty *queue* having a maximum size of 10 (so, our "forgetting" algorithm only remembers the last 10 reference points that have been selected) and, in line 14, "Add t to E" is an enqueue operation where, when E is full, the oldest element of E will be dequeued so that only up to 10 elements will be remembered at all times. This way, the distance calculations in lines 16 to 18 only consume a constant computational overhead because the size of C is 10 and the size of E is also 10 (hence, no more than 100 distances need to be calculated). As a result, the time complexity of our FSCS-ART-with-Forgetting algorithm is linear (O(n)).

4.5.6. *Algorithm 5: MART 4*

To further reduce the computational overhead, we further apply a mirroring strategy in addition to the forgetting strategy. The algorithm named "MART 4" refers to the FSCS-ART-with-Forgetting-and-Mirroring-with-4-subdomains algorithm, where mirroring and forgetting are both applied to the original FSCS-ART algorithm and the mirroring involves a source subdomain (the *upper-left* quarter) plus three mirror subdomains (the sequence of the three mirror subdomains is shown in Figure 9).

The MART 4 algorithm generates an adaptive random sequence (ARS) of reference points as follows:

Step 1: Apply the FSCS-ART-with-Forgetting algorithm to generate a reference point P within the source subdomain;

Step 2: Map (translate) P into the mirror subdomain 1—the mapping only involves adding the width of the subdomain to the x-coordinate of P and, hence, incurs trivial overheads;

Step 3: Map P into the mirror subdomain 2;

Step 4: Map P into the mirror subdomain 3;

The above four steps are repeated until a difference of images is detected or the stopping criterion is met. Consider the example shown in Figure 9, the generated ARS of reference points is: P1, P1', P1'', P1''', P2, P2', P2'', P2'''.

4.5.7. *Algorithm 6: MART 4 (B)*

In the MART 4 algorithm, the source subdomain is defined to be the upper-left quarter of the entire input domain, and the sequence of the three mirror subdomains

is defined as shown in Figure 9. Following the same rationale for the design of the sequential (B) algorithm, which states that a higher priority should be given to the right and bottom boundaries of the output space, we designed the MART 4 (B) algorithm.

The basic procedure of MART 4 (B) is similar to that of MART 4 except that a different source and mirror subdomain scheme is used, as shown in Figure 15. Consider the example shown in Figure 15, the generated ARS of reference points is: P1, P1', P1'', P1''', P2, P2', P2'', P2'''.

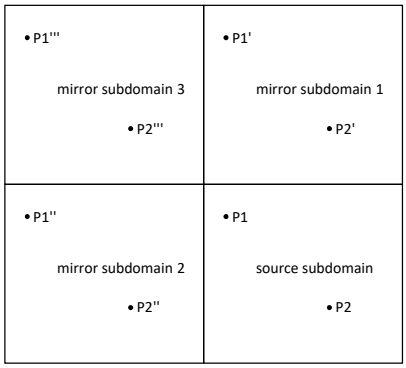

Fig. 15: Source and mirror subdomains of MART 4 (B).

4.5.8. *Algorithm 7: MART 9*

The MART 9 algorithm is similar to MART 4 except that nine rather than four subdomains are used, as illustrated in Figure 16.

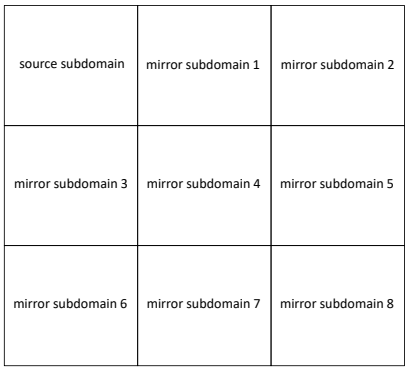

Fig. 16: Source and mirror subdomains of MART 9.

4.5.9. *Algorithm 8: MART 9 (B)*

The MART 9 (B) algorithm is similar to MART 4 (B) except that nine rather than four subdomains are used, as illustrated in Figure 17.

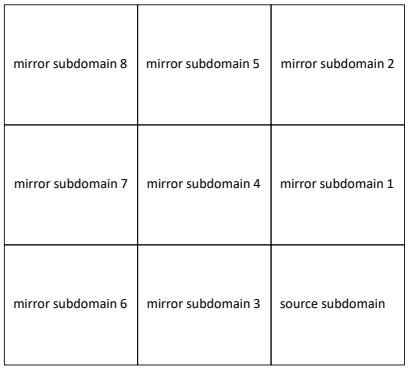

Fig. 17: Source and mirror subdomains of MART 9 (B).

4.6. *Characteristics of subject applications*

As a black box testing technique, our approach does not require any information about the internal implementation of the applications under test, enabling the tester to compile a configuration file based on observation from outside of the system. However, the tester is expected to be familiar with the general external behaviour and appearance of applications. This is especially useful in reducing false positive results when the application contains embedded third-party content, such as advertisements.

As part of the controlled experiments to be discussed in Section 5, the testing was conducted on seven custom e-commerce web applications developed with the Linux, Apache, MySQL, and PHP software bundle, a popular open-source model in the web industry. The front-end implementations include custom, Bootstrap and Foundation frameworks. These applications were developed and maintained by Tickets.com Pty Ltd. (www.tickets.com) for their business customers and were specialized in live entertainment events and ticket sales, providing a large set of frequently updated webpages to conduct testing on.

Due to confidentiality requirements of our industry partner, the specific names of the seven subject applications are not listed in this paper.

5. EMPIRICAL STUDY AND DISCUSSIONS

To put into practice and evaluate the proposed approach, a set of seven real world web applications from the industry were tested. Web applications in staging and

production versions were used for pairwise comparisons in the controlled experiments. In practice, web development agencies tend to keep two versions of web applications where the staging version with restricted public access provides a testing environment to test new features before putting them into production. Our controlled experiments involved the use of faulty webpages collected from multiple version cycles.

5.1. *Scale and categorization*

The testing was carried out on Windows 7 Enterprise Edition with four most popular web browsers (Google Chrome Version 35.0.1916.114m, Mozilla Firefox Version 29.0.1, Opera Version 17.0.1241.45 and Microsoft Internet Explorer 10.0.15) with screen size of 1600×900 involving 3115 pairs of webpages from the seven web applications under controlled experiment. [c] Screenshots were categorized according to their failure rates [d] where Category A (1347 pairs of screenshots) had a calculated failure rate between 0.0020 and 0.0070 (an average of 0.0053), and Category B (1768 pairs of screenshots) had a calculated failure rate between 0.0008 and 0.0020 (an average of 0.0014).

5.2. *Results of experiments*

Each pair of screenshots (production page, staging page) was tested for 1000 times on each algorithm in order to yield statistically meaningful results (except sequential (T) and sequential (B) as these two algorithms are deterministic and hence only require one run). Therefore, the scale of the experiments is nontrivial. All F-measures were collected and the average F-measure was calculated for each algorithm. Similarly, the mean execution time to the first failure was also calculated for each algorithm.

Results of experiments are shown in Table 1 and Table 2. A number of interesting observations can be made, as discussed below:

First, the F-measure and execution time results shown in both Table 1 and Table 2 indicate that sequential comparison (sequential (T) and sequential (B)) is the worst option. However, it is clear that, within the two sequential comparison algorithms, sequential (B) always outperformed sequential (T). This means that our *failure-based testing* strategy is effective, where a higher priority is given to the right and bottom boundaries of the output space for the "domino effect" failure pattern in web layouts.

[c] Each of the 3115 pairs included two non-identical screenshots. In our controlled experiments, we excluded identical screenshots.

[d] In the context of comparing two screenshots, we define the concept of "failure rate" as the number of different reference points divided by the total number of reference points. For example, if two screenshots (each having $1600 \times 900 = 1440000$ pixels) differ only in a 50×50 area then the failure rate is calculated as $(50 \times 50)/(1600 \times 900) = 0.0017$. We were able to compute the precise failure rate of each screenshot pair in the controlled experiments.

Method	Mean time to first failure (ms)	Mean F-measure
sequential (T)	26940	128903
sequential (B)	20445	97826
RT	59	191
FSCS-ART-with-Forgetting	213	131
MART 4	54	129
MART 4 (B)	51	119
MART 9	29	130
MART 9 (B)	28	127

Table 1: Mean execution time and F-measure (**Category A**).

Method	Mean time to first failure (ms)	Mean F-measure
sequential (T)	86336	413092
sequential (B)	75112	359390
RT	293	972
FSCS-ART-with-Forgetting	1049	643
MART 4	268	640
MART 4 (B)	267	637
MART 9	136	645
MART 9 (B)	135	643

Table 2: Mean execution time and F-measure (**Category B**).

Secondly, Table 1 and Table 2 show that all five ART algorithms are superior to RT in terms of F-measure—they required a fewer number of pixel comparisons to detect the first failure (that is, the first different pixel). This means that using an even distribution of checkpoints across the *output* domain is a more *effective* strategy than RT.

Thirdly, in terms of *efficiency*, the first ART algorithm (FSCS-ART-with-Forgetting) is worse than RT in execution time due to the distance computation required in ART. This cost, however, is significantly reduced with the mirroring technique, even to the extent that it is cheaper than RT, as randomly generating a new reference point is more expensive than mapping an existing one (which, on average, involves no more than one addition or subtraction operation on the x- or y-coordinate). Consequently, as shown in both Table 1 and Table 2, all four MART algorithms have outperformed RT in execution time. This means that all four MART algorithms are not only more *effective* but also more *efficient* than RT.

Fourthly, when we compare the results of the four MART algorithms with the results of FSCS-ART-with-Forgetting, it is found that the former not only outperformed the latter in execution time but also achieved comparable F-measures. It

is also shown that increasing the number of subdomains from four to nine largely reduced the execution time but not the F-measure.

Fifthly, an important observation is that MART 4 (B) and MART 9 (B) always outperformed MART 4 and MART 9, respectively. This observation is consistent with the observation that sequential (B) outperformed sequential (T). In other words, all of the bottom-up approaches outperformed their top-down counterparts, demonstrating the effectiveness of our *failure-based testing* strategy, where testing priorities are given to regions that are more likely to incur a failure. This finding also indicates that the effectiveness and efficiency of the mirroring technique is related to the order of mirror subdomains, and hence provides additional flexibility to control proliferation direction in ART when there is knowledge of the possible failure inclination (certain sections of the input/output domain are more likely to have failures).

Finally, it is to be noted that Table 1 and Table 2 correspond to screenshots of larger and smaller failure rates, respectively. A smaller failure rate in web application screenshots indicates that the failure type is unlikely to be structural layout fault, that is, the failure is unlikely to flow as described in Section 3.3. So, searching for it from the bottom has smaller benefit. However, lower areas of the application screen are still slightly failure-inclined because header areas are usually static (less frequently changing) in web applications.

6. CONCLUSION

A failure-based testing strategy is defined as a concept that envisions selecting test cases based on the knowledge available of various aspects of failure patterns [9]. In this paper we designed failure-based testing techniques for the detection of web application layout faults. Our techniques are based on both the observation of web application failure patterns and the concept of adaptive random sequence.

ART is a technique that selects test cases from the *input space*. This study has successfully extended the concept and application of ART to the *output space*. We showed that output verification in terms of oracle automation can be facilitated by the use of application domain knowledge. In this regard, ART is guided with failure-pattern-based heuristics and outperforms random testing (RT) in both failure detection *effectiveness* (the F-measure) and failure detection *efficiency* (the execution time).

At the same time, it is found that how many mirror subdomains are used and in which order they are selected can have noticeable effect on the failure detection capacity.

The empirical results demonstrate that our approach is promising in the light of attempts at automating output verification in an cost-effective way. If used properly, this method can minimize the manual work and efforts required from developers and testers during the conduct of regression testing of web applications. Most importantly, this method is not dependent on particular testing of the web application,

and can also be used in other domains for output verification. It is to be noted, however, that we did not consider the problem of image scaling and translation variations, as these topics are beyond the scope of this study. Interested readers are referred to Kıraç et al. [8] for treatments that eliminate differences caused by scaling and translation.

In summary, our proposed approach has a number of main points of novelty in terms of both practical application and academic contribution: use of the characteristics and failure patterns of browser layout rendering to improve screenshot comparison effectiveness and efficiency; even distribution of reference points across the *output* space as an extension and novel application of the concepts of ART and ARS; and division of the output space into multiple subdomains to reduce the fault-detection execution time even to the extent that it is cheaper than RT. The empirical results presented in this paper add further justifications for the emergence of the area of failure-based testing, which covers ART as a special instance [9].

In future research, we intend to extend our failure-based testing approach by applying more domain-specific knowledge leverages to further enhance the effectiveness and efficiency of output verification.

ACKNOWLEDGMENTS

This research was supported in part by a linkage grant of the Australian Research Council (project ID: LP160101691).

References

[1] E. Selay, Z. Q. Zhou, and J. Zou, "Adaptive random testing for image comparison in regression web testing," in *Proceedings of the International Conference on Digital Image Computing: Techniques and Applications (DICTA 2014)*. IEEE Computer Society Press, 2014, pp. 1–7.

[2] UN International Telecommunications Union, "ITU releases 2015 ICT figures," 2015, available at https://www.itu.int/net/pressoffice/press_releases/2015/17.aspx, [accessed 19-July-2015].

[3] S. R. Choudhary, H. Versee, and A. Orso, "WEBDIFF: Automated identification of cross-browser issues in web applications," in *Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM'10)*, Sept 2010, pp. 1–10.

[4] D. Robins and J. Holmes, "Aesthetics and credibility in web site design," *Information Processing and Management*, vol. 44, no. 1, pp. 386 – 399, 2008.

[5] G. Rothermel, R. H. Untch, C. Chengyun, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 10 2001.

[6] S. Raina and A. P. Agarwal, "An automated tool for regression testing in web applications," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 4, pp. 1–4, jul 2013.

[7] OpenQA, "Selenium web application testing system," available at http://seleniumhq.org, [accessed 19-July-2015].

[8] M. F. Kıraç, B. Aktemur, and H. Sözer, "VISOR: A fast image processing pipeline

with scaling and translation invariance for test oracle automation of visual output systems," *Journal of Systems and Software*, vol. 136, pp. 266–277, 2018.

[9] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: The ART of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[10] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Proceedings of the 9th Asian Computing Science Conference (ASIAN'04), Lecture Notes in Computer Science 3321*. Springer-Verlag, 2004, pp. 320–329.

[11] A. Grosskurth and M. W. Godfrey, "A reference architecture for web browsers," in *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, 2005, pp. 661–664.

[12] E. Marcotte, "Responsive web design, A List Apart," May 2010, available at http://www.alistapart.com/articles/responsive-web-design/. [Online]. Available: http://www.alistapart.com/articles/responsive-web-design/

[13] A. Hori, S. Takada, H. Tanno, and M. Oinuma, "An oracle based on image comparison for regression testing of web applications," in *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE'15)*, 2015, pp. 639–645.

[14] D. J. Richardson, S. L. Aha, and T. O'Malley, "Specification-based test oracles for reactive systems," in *Proceedings of the 14th International Conference on Software Engineering (ICSE'92)*, 1992, p. 105.

[15] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[16] I. Banerjee, B. Nguyen, V. Garousi, and A. Memon, "Graphical user interface (GUI) testing: Systematic mapping and repository." *Information and Software Technology*, vol. 55, pp. 1679 – 1694, 2013.

[17] Microsoft Corporation, "Expression web," available at http://www.microsoft.com/expression/, [accessed 03-April-2015].

[18] Adobe Systems Incorporated, "Browser lab," available at http://blogs.adobe.com/browserlab/, [accessed 05-April-2015].

[19] M. Benedikt, J. Freire, and P. Godefroid, "VeriWeb: Automatically testing dynamic web sites," in *Proceedings of the 11th International World Wide Web Conference (WWW'02*, 2002, pp. 654–668.

[20] P. Tonella and F. Ricca, "Dynamic model extraction and statistical analysis of web applications," in *Proceedings of the 4th International Workshop on Web Site Evolution*, 2002, pp. 43–52.

[21] F. Ricca and P. Tonella, "Web testing: a roadmap for the empirical research," in *Proceedings of the 7th IEEE International Symposium on Web Site Evolution (WSE'05)*, Sept 2005, pp. 63–70.

[22] S. Bedi and P. J. Schroeder, "Observations on the implementation and testing of scripted web applications," in *Proceedings of the 6th IEEE International Workshop on Web Site Evolution (WSE'04)*, Sept 2004, pp. 20–27.

[23] G. A. D. Lucca and A. R. Fasolino, "Testing web-based applications: The state of the art and future trends," *Information and Software Technology*, vol. 48, no. 12, pp. 1172 – 1186, 2006, quality Assurance and Testing of Web-Based Applications.

[24] A. Marchetto, F. Ricca, and P. Tonella, "Empirical validation of a web fault taxonomy and its usage for fault seeding," in *Proceedings of the 9th IEEE International Workshop on Web Site Evolution (WSE'07)*, Oct 2007, pp. 31–38.

[25] C. Eaton and A. M. Memon, "An empirical approach to evaluating web application

compliance across diverse client platform configurations," *International Journal of Web Engineering and Technology*, vol. 3, no. 3, pp. 227–253, 2007.

[26] J. Takahashi, "An automated oracle for verifying GUI objects," *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 4, pp. 83–88, Jul. 2001.

[27] A. Mesbah and M. R. Prasad, "Automated cross-browser compatibility testing," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, 2011, pp. 561 – 570.

[28] V. Dallmeier, M. Burger, T. Orth, and A. Zeller, "Webmate: A tool for testing web 2.0 applications," in *Proceedings of the Workshop on JavaScript Tools (JSTools'12)*. New York, NY, USA: ACM Press, Jun. 2012, pp. 11–15.

[29] PhantomCSS, available at https://github.com/Huddle/PhantomCSS, [accessed 20-July-2015].

[30] Applitools, available at https://applitools.com/, [accessed 20-July-2015].

[31] Screenster, available at http://www.creamtec.com/products/screenster/, [accessed 20-July-2015].

[32] Y. Rubner, C. Tomasi, and L. Guibas, "The earth mover's distance as a metric for image retrieval," *International Journal of Computer Vision*, vol. 40, no. 2, pp. 99–121, 2000.

[33] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by image and video content: The QBIC system," *Computer*, vol. 28, no. 9, pp. 23–32, 1995.

[34] H. Ling and K. Okada, "An efficient earth mover's distance algorithm for robust histogram comparison," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 5, pp. 840 – 853, 2007.

[35] R. S. Boyer, B. Elspas, and K. N. Levitt, "Select- a formal system for testing and debugging programs by symbolic execution," in *Proceedings of the International Conference on Reliable Software*, 1975, pp. 234–245.

[36] L. J. White and E. I. Cohen, "A domain strategy for computer program testing," *IEEE Transactions on Software Engineering*, vol. 6, no. 3, pp. 247 – 257, 1980.

[37] P. E. Ammann and J. C. Knight, "Data diversity: an approach to software fault tolerance," *IEEE Transactions on Computers*, vol. 37, no. 4, pp. 418–425, 1988.

[38] G. B. Finelli, "NASA software failure characterization experiments," *Reliability Engineering and System Safety*, vol. 32, pp. 155 – 169, 1991.

[39] P. G. Bishop, "The variation of software survival time for different operational input profiles (or why you can wait a long time for a big bug to fail)," in *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23)*, 1993, pp. 98–107.

[40] F. T. Chan, T. Y. Chen, I. K. Mak, and Y. T. Yu, "Proportional sampling strategy: guidelines for software testing practitioners," *Information and Software Technology*, vol. 38, no. 12, pp. 775 – 782, 1996.

[41] T. Y. Chen, F.-C. Kuo, and R. Merkel, "On the statistical properties of the F-measure," in *Proceedings of the 4th International Conference on Quality Software (QSIC'04)*, 2004, pp. 146–153.

[42] ——, "On the statistical properties of testing effectiveness measures," *Journal of Systems and Software*, vol. 79, no. 5, pp. 591 – 601, 2006.

[43] "W3C Specifications," available at http://www.w3.org/Style/CSS/specs.en.html, [accessed 09-April-2015].

[44] Microsoft Corporation, "Microsoft Developer Network," available at http://msdn.microsoft.com/en-us/library/ms533005(v=vs.85).aspx, [accessed 21-June-2015].

[45] E. N. Narciso, M. E. Delamaro, and F. D. L. D. S. Nunes, "Test case selection:

A systematic literature review," *International Journal of Software Engineering and Knowledge Engineering*, vol. 24, no. 4, pp. 653 – 676, 2014.

[46] T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Q. Zhou, "A revisit of three studies related to random testing," *SCIENCE CHINA Information Sciences*, vol. 58, pp. 052 104:1–052 104:9, 2015, Springer-Verlag.

[47] S. Morasca and S. Serra-Capizzano, "On the analytical comparison of testing techniques," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'04)*, 2004, pp. 154–164.

[48] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and S. P. Ng, "Mirror adaptive random testing," *Information and Software Technology*, vol. 46, no. 15, pp. 1001 – 1010, 2004.

[49] K. P. Chan, T. Y. Chen, and D. Towey, "Forgetting test cases," in *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*. IEEE Computer Society Press, 2006, pp. 485–494.

[50] A. F. Tappenden and J. Miller, "A novel evolutionary approach for adaptive random testing," *IEEE Transactions on Reliability*, vol. 58, no. 4, pp. 619 – 633, 2009.

[51] E. Ibrahimov, J. Wang, and Z. Q. Zhou, "Similarity-based search for model checking: a pilot study with Java Pathfinder," in *Proceedings of the 13th International Conference on Quality Software (QSIC'13), The Symposium on Engineering Test Harness (TSETH'13)*. IEEE Computer Society Press, 2013, pp. 238–244.

[52] A. Shahbazi, A. F. Tappenden, and J. Miller, "Centroidal voronoi tessellations – a new approach to random testing," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 163–183, 2013.

[53] A. C. Barus, T. Y. Chen, F.-C. Kuo, H. Liu, R. Merkel, and G. Rothermel, "A cost-effective random testing method for programs with non-numeric inputs," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3509–3523, 2016.