$\mathcal{Q}$-learning strategies for action selection in the TESTAR automated testing tool

Anna I. Esparcia-Alcázar, Francisco Almenar, Mirella Martínez, Urko Rueda, and Tanja E.J. Vos

Research Center on Software Production Methods (PROS)
Universitat Politècnica de València
Camino de vera s/n 46022, Valencia, Spain
{falmenar,aesparcia,mmartinez,urueda,tvos}@pros.upv.es
http://www.testar.org

Abstract. TESTAR is an open source tool for automated software testing that generates test sequences on the fly based only on information derived from the Graphical User Interface (GUI). At the core of TESTAR is the way to automatically select which actions to test; finding the right algorithm to carry out this task can make significant differences to the testing outcome.
In this work we evaluate $\mathcal{Q}$-learning as a metaheuristic for action selection and carry out experiments with a range of paramenters, using random selection as a baseline for the comparison. Two applications are used as Software Undder Test (SUT) in the experiments, namely MS Powerpoint (a proprietary desktop application) and the Odoo enterprise management system (an open source web-based application). We introduce metrics to evaluate the performance of the testing with TESTAR, which are valid even under the assumption that access to the source code is not available and testing is only possible via the GUI. These metrics are used to perform statistical analysis, showing that the superiority of action selection by $\mathcal{Q}$-learning can only be achieved through an adequate choice of parameters.

Mots-Clefs. Automated GUI Testing,Testing Metrics, Testing Web Applications, $\mathcal{Q}$-learning

1 Introduction

The Graphical User Interface (GUI) represents a central point in any application from where the user may access all the functionality. Hence, testing at the GUI level means taking the user's perspective and is thus the ultimate way of verifying a program' s correct behaviour. Current GUIs can account for 45-60% of the entire source code [**?**] in any application and are often large and complex. Consequently, it is difficult to test applications thoroughly through their GUI, especially because GUIs are designed to be operated by humans, not machines. Moreover, they are inherently non-static interfaces, subject to constant change caused by functionality updates, usability enhancements, changing requirements or altered contexts. Automating the GUI testing process is therefore a crucial task in order to minimise time-consuming and tedious manual testing.

TESTAR is an open source tool that performs automated testing via the GUI, using the operating system's Accessibility API to recognise GUI controls and their properties, and enabling programmatic interaction with them. It derives sets of possible actions for each state the GUI is in and selects and executes appropriate ones, thus creating a test sequence on the fly. TESTAR has been successfully applied to various commercial and open source applications, both deskptop and web-based ones, as shown in e.g. [**?**,**?**,**?**,**?**]; in most cases the action selection mechanism was random choice, a procedure also known as *monkey testing*

In citeBV2012,BV14 the first attempts to action selection in TESTAR based on metaheuristics, and especifically $\mathcal{Q}$-learning, are described. However, the performance metrics used for evaluation were the average time it took to crash the application under test and the reproducibility of the crashes. Although the results were promising, they revealed problems with this choice of metrics, which we try to address here. In this work we introduce four novel metrics specifically designed to testing via the GUI and without access to the source code of the applications. Using these metrics we compare various settings for $\mathcal{Q}$-learning and also use random testing as a baseline.

In order to carry out our study we chose two applications: the Odoo enterprise resource planning (ERP) system and the PowerPoint presentation software. They are two very different types of SUT: one is an open source web application and the other a proprietary desktop application.

We run experiments in three phases or iterations, refining the process after each phase, and carry out statistical analysis on the results of the third phase.

The rest of this paper is structured as follows. Section 3 describes the action selection mechanism using $\mathcal{Q}$-learning. Section 4 introduces the metrics used for quality assessment of the testing procedure. Section 5 summarises the experimental set up, the results obtained and the statistical analysis carried out; it also highlights the problems encountered. Finally, in section 6 we present some conclusions and outline areas for future work.

2 Related work

The existing literature in User Interface testing covers three approaches: *capture-and-replay* (C&R), which involves recording user interactions and converting them into a script that can be replayed repeatedly, *visual-based* which relies on image recognition techniques to visually interpret the images of the target UI [**?**] , and *traversal-based*, which uses information from the GUI (GUI reflection) to traverse it [**?**], and can be used to check some general properties.

Current practice of UI testing relies mainly on Capture-and-Replay (C&R, also called Record-and-Replay) tools. This is a mature technology, for which tools are widely available, be they commercial or open source. However, a major problem with this approach is maintenance, as changes in the UI usually render the created test scripts unusable. This problem becomes more severe with the new generation of Internet-based applications, as these adapt their layout dynamically according to the users' needs. Hence, in spite of some degree of automation, GUI testing still involves heavy load of manual work, which is costly and error prone [**?**].

Visual-based and traversal-based tools aim at solving the maintenance problem; the latter group, to which TESTAR belongs, is considered to be the most resilient to changes in the SUT.

In order to evaluate the quality and performance of the testing suitable metrics must be defined. For instance, in [**?**] Chaudhary et al propose metrics for event driven driven software. Memon et al [**?**] propose a coverage criteria for GUI testing. However, knowing what to measure is still an area that deserves further investigation. In this work we propose four such metrics which are suitable to measure the quality of the GUI testing on web applications, based on the assumption that source code is not available.

3 Using $\mathcal{Q}$-learning for action selection in TESTAR

The choice of an action selection mechanism is one of the two main inputs for the human tester in TESTAR (the other one being the custom protocol).

We have employed the $\mathcal{Q}$-learning algorithm to guide the action selection process. $\mathcal{Q}$-learning [**?**] is a model-free reinforcement learning technique in which an agent, at a state s, must choose one among a set of actions A_s available at that state. By performing an action $a \in A_s$, the agent can move from state to state. Executing an action in a specific state provides the agent with a reward (a numerical score which measures the utility of executing a given action in a given state). The goal of the agent is to maximise its total reward, since it allows the algorithm to look ahead when choosing actions to execute. It does this by learning which action is optimal for each state. The action that is optimal for each state is the action that has the highest long-term reward.

Our version of the $\mathcal{Q}$-learning algorithm, shown in Algorithm 3 is governed by two parameters: the maximum reward, R_{max} and the discount γ. Depending on how these are chosen the algorithm will promote exploration or exploitation of the search space. The R_{max} parameter determines the initial reward unexplored actions have; so, a high value biases the search towards executing unexplored actions. On the other hand, discount γ establishes how the reward of an action decreases after being executed. Small γ values decrease the reward faster and vice versa.

Require: $R_{max} > 0$ /* *reward for unexecuted actions* */
Require: $0 < \gamma < 1$ /* *discount factor* */
1: **begin**
2: start SUT
3: $\forall (s, a) \in S \times A : \ Q(s, a) \leftarrow R_{max}$
4: initialize s and available action A_s
5: **repeat**
6: $a^* \leftarrow max_a\{Q(s, a)|a \in A_s\}$
7: execute a^*
8: obtain state s' and available actions $A_{s'}$
9: $Q(s, a^*) \leftarrow R(s, a^*) + \gamma \cdot max_{a \in A_{s'}} Q(s', a)$
10: $ec(a^*)$ ++
11: $s \leftarrow s'$
12: **until** stopping criteria met
13: stop SUT
14: **end**

R is set as follows:

$$R(s,a,s') := \begin{cases} R_{max} & if\ x_a = 0 \\ \frac{1}{x_a} & otherwise \end{cases}$$

where x_a is the number of times action a has been executed and R_{max} is a large positive number (in order to make actions not executed before attractive for the agent)

4 Testing performance metrics

Finding appropriate metrics for assessing the quality of the testing has been a long standing issue. For instance, [**?**] defines a number of metrics for GUI testing, but these imply having access to the code of the SUT; one of the strengths of TESTAR is precisely not relying on the assumption that this is the case. However, this also implies that specific metrics must be defined.

In previous work [**?**] we used the number of crashes and the time to crash as a measure of the testing performance, but these pose problems too, because they reveal nothing about to what extent the SUT was explored, a fact particularly relevant if no crashes are detected. Aiming to circumvent that issue, in this work the metrics were chosen as follows:

- **Abstract states** This metric refers to the number of different states, or windows in the GUI, that are visited in the course of an execution.
- **Longest path** Any automated testing tool must ensure the deepest parts of the GUI are tested. To measure whether the tool has just stayed on the surface or it has reached deeper, we define the longest path as the longest sequence of non-repeated (i.e. excluding loops) consecutive states visited.
- **Minimum and maximum coverage per state** We define the *state coverage* as the rate of executed over total available actions in a given state/window; the metrics are the highest and lowest such values across all windows. This allows us to know to what extent actions pertaining to states were explored.

A consequence of not having access to the source code is that the metrics given above can be used to compare the efficiency of different testing methods, but not to assess the overall goodness of a method in isolation, because we do not know the global optima for each metric; for instance, we cannot know exactly how many different states there are.

5 Experiments and results

5.1 The software under test (SUT)

We used two different applications in order to evaluate our $\mathcal{Q}$-learning approach in TESTAR, Odoo and PowerPoint.

Odoo [1] is an open source Enterprise Resource Planning software consisting of several enterprise management applications that can be installed or not depending on the user needs. It can be used to create websites, manage human resource (HR), finance, sales, projects and others. Odoo has a client-server architecture and uses a PostgreSQL database as a management system. Once deployed, we installed the mail, calendar, contacts, sales, inventory and project applications in order to test a wide number of options.

PowerPoint Microsoft PowerPoint is a slide show presentation program currently developed by Microsoft and part of its productivity software Microsoft Office. It is currently one of the most commonly used presentation programs available.

[1] See https://github.com/odoo/odoo for Odoo's git repository and issue tracker, including a manual with instructions on how to deploy the server and its requirements.

5.2 Procedure

In order to test Odoo with TESTAR a server version of Odoo must first be deployed[2]. Then TESTAR must be configured by supplying the URL that accesses the Odoo client and the browser that will be used to launch it.

On the other hand, to test PowerPoint with TESTAR we must first install it and then TESTAR must be configured by providing the specific command that would be used to run PowerPoint using the cmd (Windows command prompt).

Next, for both tools we run TESTAR in *spy mode*; this uncovers possible problems with items that may not be detected well, such as emergent windows. In addition, it helps detecting undesired actions that might be performed by TESTAR that may bring problems such as involuntary file deletion. A number of parameters must also be set up, which are given in Table 1. With these settings and a first version of the TESTAR *protocol*[3] we carried out three iterations of the testing process, improving the protocol each time so as to remove the problems encountered.

Table 1. Experimental set up. We carried out three iterations involving the five sets. After each iteration the results obtained were used to refine the TESTAR protocol so as to better adapt it to the application.

Set	Max. actions per run	Runs	Action Selection Algorithm	Parameters	
				R_{max}	γ
Q1	1000	30	$\mathcal{Q}$-learning	1	0.20
Q20	1000	30	$\mathcal{Q}$-learning	20	0.20
Q99	1000	30	$\mathcal{Q}$-learning	99	0.50
Q10M	1000	30	$\mathcal{Q}$-learning	9999999	0.95
RND	1000	30	random	N/A	N/A

5.3 Statistical analysis

We run the Kruskal-Wallis non parametric test, with $\alpha = 0.05$, on the results for the five sets. In iteration 3 the test shows that all the metrics have significant differences among the sets. Running pair-wise comparisons provides the results shown in the boxplots contained in Figures 1 and 2; these results are ordered in Table 2, where the shaded column is the best option. It can be seen that for each SUT and metric the best choices are different; also, random selection turns out not to be such a bad choice in most cases. This highlights the importance of an adequate choice of parameters when using $\mathcal{Q}$-learning for action selection.

One metric we have not considered in the statistical analysis is the number of failures encountered, shown in Table 3

In the case of Odoo we can see that although Q20 did not perform so well in the other metrics, it does on the other hand find the higher number of failures (which involve stopping the execution and hence having a lesser chance of increasing the value of other metrics); this must also be taken into account when evaluating the different algorithms. However, for PowerPoint no failures were encountered, so this metric reveals no information.

6 Conclusions

We have shown here the successful application of a $\mathcal{Q}$-learning action selection strategy within the TESTAR tool to the automated testing of the Odoo management software and the commercial application PowerPoint. $\mathcal{Q}$-learning was also compared to *monkey testing*, i.e. using random choice for action selection. Four metrics were defined in order to evaluate the performance. Statistical

[2] See the source install tutorial available from https://www.odoo.com/documentation/8.0/setup/install.html

[3] For more details the reader is referred to the tutorial available from www.testar.org

Table 2. Results of the statistical comparison for the sets obtained in the third iteration. The shaded column represents the best choice, the remaining ones are in order of preference.

Metric (Odoo)	Set				
Abstract states	Q10M	RND	Q1	Q99	Q20
Longest path	Q10M	Q99	RND	Q1	Q20
Maximum coverage per state	Q10M	Q20	RND	Q1	Q99
Minimum coverage per state	Q99	Q20	Q10M	Q1	RND
Metric (PowerPoint)	**Set**				
Abstract states	Q99	Q20	RND	Q10M	Q1
Longest path	Q20	Q99	RND	Q10M	Q1
Maximum coverage per state	Q10M	Q20	Q99	RND	Q1
Minimum coverage per state	Q1	Q20	Q10M	Q99	RND

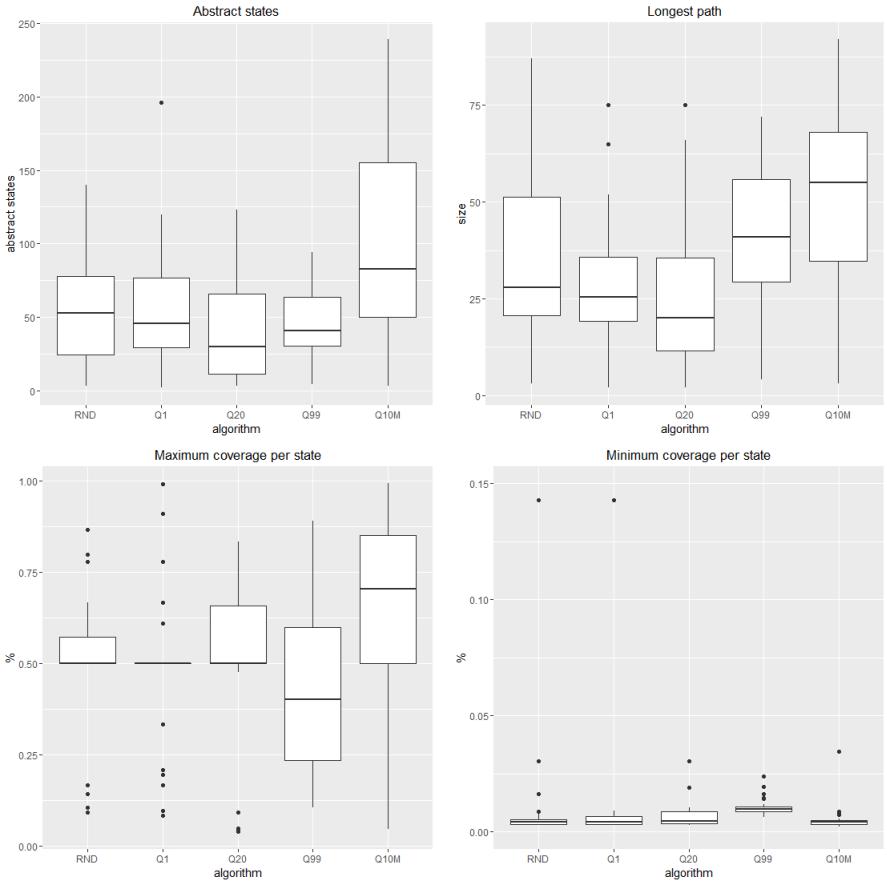

Fig. 1. Boxplots for the four metrics with the results obtained for Odoo in Iteration 3.

Table 3. Number of failures encountered per algorithm in the 3rd iteration when testing Odoo. No failures were encountered in PowerPoint.

Set (Odoo)	Total Failures	Unique failures
Q10M	3	1
Q99	0	0
Q20	6	2
Q1	2	1
RND	1	1

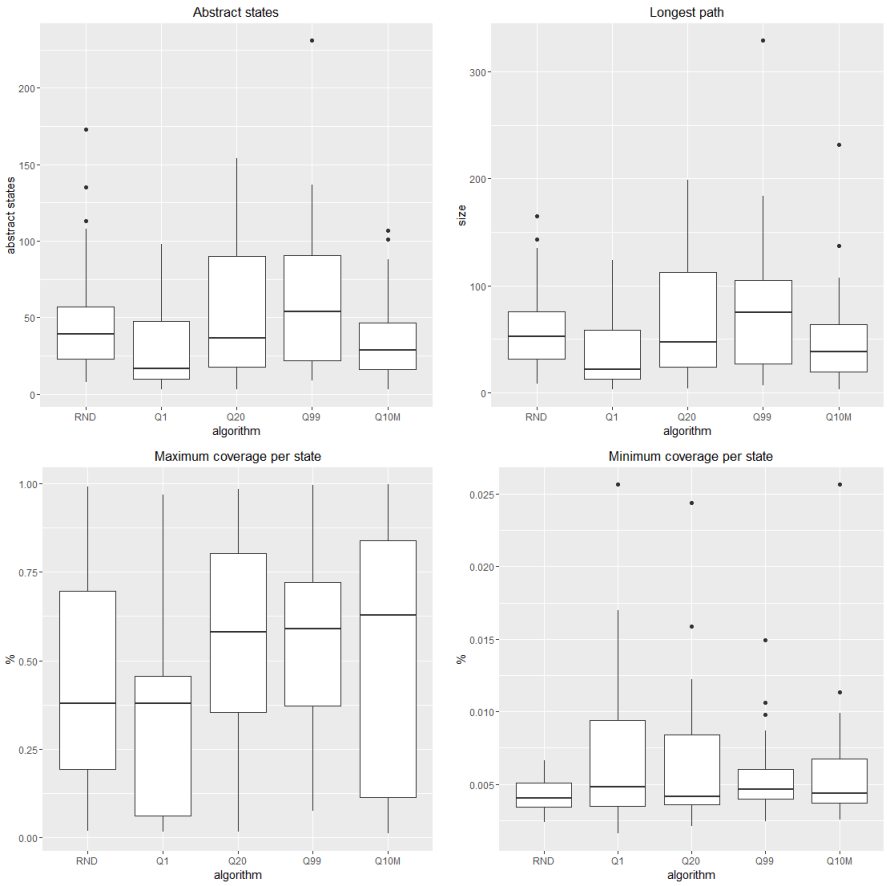

Fig. 2. Boxplots for the four metrics with the results obtained for PowerPoint in Iteration 3.

analysis reveals the superiority of the $\mathcal{Q}$-learning-based method, provided the parameters of the algorithm have been properly selected.

Further work will involve the improvement of the metrics. We will also explore more complex metaheuristics for action selection, especially population based ones (such as ant colony optimisation and genetic programming) in order to improve over the relatively simple $\mathcal{Q}$-learning algorithm.

Acknowledgments.

This work was partially funded by projects **SHIP** (*SMEs and HEIs in Innovation Partnerships*, ref: EACEA/A2/UHB/CL 554187) and **PERTEST** (TIN2013-46928-C3-1-R).