Evolving Rules for Action Selection in Automated Testing via Genetic Programming - A First Approach

Anna I. Esparcia-Alcázar(✉), Francisco Almenar,
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
{aesparcia,urueda,tvos}@pros.upv.es
http://www.testar.org

Abstract. Tools that perform automated software testing via the user interface rely on an action selection mechanism that at each step of the testing process decides what to do next. This mechanism is often based on random choice, a practice commonly referred to as *monkey testing*. In this work we evaluate a first approach to genetic programming (GP) for action selection that involves evolving IF-THEN-ELSE rules; we carry out experiments and compare the results with those obtained by random selection and also by $\mathcal{Q}$-learning, a reinforcement learning technique. Three applications are used as Software Under Test (SUT) in the experiments, two of which are proprietary desktop applications and the other one an open source web-based application. Statistical analysis is used to compare the three action selection techniques on the three SUTs; for this, a number of metrics are used that are valid even under the assumption that access to the source code is not available and testing is only possible via the GUI. Even at this preliminary stage, the analysis shows the potential of GP to evolve action selection mechanisms.

Keywords: Automated testing via the GUI · Action selection for testing · Testing metrics · Genetic Programming

1 Introduction

The relevance of testing a software application at the Graphical User Interface (GUI) level has often been stated due to several reasons, the main being that it implies 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 source code [2] in any application and are often large and complex; hence, it is difficult to test applications thoroughly through their GUI, especially because GUIs are designed to be operated by humans, not machines. Furthermore, they are usually subject to frequent changes motivated by functionality updates, usability enhancements, changing requirements or altered contexts. Automating the

G. Squillero and K. Sim (Eds.): EvoApplications 2017, Part II, LNCS 10200, pp. 82–95, 2017.
DOI: 10.1007/978-3-319-55792-2_6

process of testing via the GUI is therefore a crucial task in order to minimise time-consuming and tedious manual testing.

The existing literature in testing via the User Interface 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 [3], and *traversal-based*, which uses information from the GUI (GUI reflection) to *traverse* it [1], and can be used to check some general properties. Of the three, the latter group is considered the most resilient to changes in the SUT.

The designer of any automated tool for carrying out traversal-based testing is faced with a number of design choices. One of the most relevant is the decision of the action selection mechanism which, given the current state (or window) the system is in, involves answering the question "what do I do next?". Although most tools leave this to purely random choice (a procedure known as *monkey testing*), some authors have resorted to metaheuristics or machine learning techniques in order to decide what action to execute at each step of the testing sequence, such as $\mathcal{Q}$-learning [7] and Ant Colony Optimisation [5]. Here we present a first approach to using Genetic Programming (GP) to evolve action selection rules in traversal-based software testing. There is a large body of work that shows the power of GP to evolve programs and functions and, more specifically, rules; on the other hand, GP has also previously been used in software testing, e.g. by [11,12] but, to the best of our knowledge, not to evolve action selection rules.

In our approach GP evolves a population of rules whose quality (or *fitness*) is evaluated by using each one of them as the action selection mechanism in a traversal-based software testing tool. In order to do this suitable metrics must be defined and a number of options are available in the literature. For instance, in [6] metrics are proposed for event driven software; [10] defines a coverage criteria for GUI testing, while in [4] the number of crashes of the SUT, the average time it takes to crash and the reproducibility of these crashes are used. In this work we will follow the approach taken by [7], who propose four metrics which are suitable for testing web applications, based on the assumption that source code is not available.

In order to carry out our study we chose three applications as the SUTs: the Odoo enterprise resource planning (ERP) system, a software testing tool called Testona and the PowerPoint presentation software. These are very different types of SUT: while Odoo is an open source web application, both Testona and Powerpoint are proprietary desktop applications. Statistical analysis was carried out on the results of the three action selection methods over the three SUTs.

The rest of this paper is structured as follows. Section 2 describes the action selection mechanism using genetic programming. Section 3 introduces the metrics used for quality assessment of the testing procedure. Section 4 summarises the experimental set up, the results obtained and the statistical analysis carried out;

it also highlights the problems encountered. Finally, in Sect. 5 we present some conclusions and outline areas for future work.

2 Genetic Programming for Action Selection in GUI-Based Automated Testing

Tree-based Genetic Programming is the original form of GP as introduced by Koza [8]. It involves the evolution of a population of individuals, or candidate solutions, that can be represented as expression trees, given suitable nodes (functions) and leaves (terminals) are defined for the problem at hand. In this work we represent individuals as IF-THEN-ELSE rules that, given the current state of the SUT, pick the next action to execute. An example rule would be something like this:

IF *previousAction* **EQ** *typeInto*
AND *nLeftClick* **LE** *nTypeInto*
PickAny *typeInto*
ELSE
PickAnyUnexecuted

According to this rule, if the last executed action (*previousAction*) was entering text in a box (*typeInto*) and the number of clickable items (*nLeftClick*) is less than or equal to the number of text boxes (*nTypeInto*), then the next action will be typing text in any of the text boxes (**PickAny** *typeInto*); otherwise, a random action will be chosen that has not been executed before (**PickAnyUnexecuted**). Note that the text entered would be chosen at random.

The GP engine chosen was ponyGP[1] and the set up for the experiments is given in Table 1. The fitness of each individual was calculated by using it as the action selection rule for the traversal-based tool *TESTAR*[2]; metrics are collected in the process, one of which is used as the fitness value.

Figure 1 shows how the genetic programming process could be embedded within the testing tool (*TESTAR* here). The *live* version of *TESTAR* tests the SUT at hand using the best action selection rule found so far, while, in parallel, the evolutionary algorithm uses a *sandbox* version of *TESTAR* in order to evaluate the fitness of the new individuals. When a better individual is found, it is sent to the live *TESTAR*, that carries on testing using the new individual for action selection.

3 Testing Performance Metrics

As stated in Sect. 1, a number of metrics have been defined in the literature to assess the quality of the testing, e.g. those given by [10] or [4]. However, two main

[1] Developed by Erik Hemberg from the ALFA Group at MIT CSAIL http://groups.csail.mit.edu/EVO-DesignOpt/PonyGP/out/index.html.
[2] http://www.testar.org.

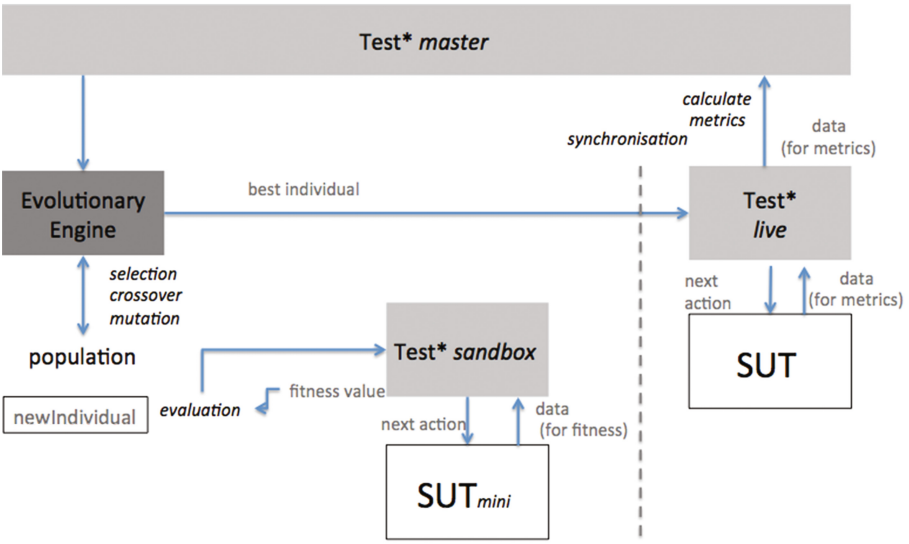

Fig. 1. The evolutionary process embedded within a traversal-based testing tool.

issues can be found with them: namely, that they either imply having access to the SUT source code (which is not always the case) or that they focus on errors encountered and reveal nothing about to what extent the SUT was explored (which is particularly relevant if no errors are detected). For these reasons, we decided on the following metrics, as defined by [7]:

- **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. An abstract state does not take into account modifications in parameters; for instance, a window containing a text box with the text "tomato" would be considered the same abstract state as the same window and text box containing the text "potato".
- **Longest path.** This is defined as the longest sequence of non-repeated consecutive states visited.
- **Minimum and maximum coverage per state.** The *state coverage* is defined 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.

It is interesting to note that longest path and maximum coverage are in a way opposed metrics, one measuring exploration and the other exploitation of the SUT.

Table 1. Genetic programming parameters.

Feature	Value
Population size	20
Max tree size	20
Functions	Pick, PickAny, PickAnyUnexecuted, AND, OR, LE, EQ, NOT
Terminals	nActions, nTypeInto, nLeftClick, previousAction, RND, typeLeftClick, typeTypeInto, Any
Evolutionary operators	Mutation and crossover
Evolutionary method	Steady state
Selection method	Tournament of size 5
Termination criterion	Generating more than 30 different states

4 Experiments and Results

4.1 Procedure

We have taken a simplified approach which involves evolving action selection rules by genetic programming using PowerPoint as the *sandbox* SUT and then validating the best evolved rule by using it to test the three different SUTs described below. For the latter phase we carried out 30 runs of 1000 actions each. In this way we can ascertain how well the GP-evolved rule generalises to SUTs not encountered during evolution.

The best evolved rule was as follows:

IF *nLeftClick* **LT** *nTypeInto*
PickAny *leftClick*
ELSE
PickAnyUnexecuted

In order to carry out statistical comparisons, the validation process was repeated using random and $\mathcal{Q}$-learning-based action selection. $\mathcal{Q}$-learning [13] 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. The choice of the algorithm's two parameters, maximum reward, R_{max} and discount γ, will promote exploration or exploitation of the search space. In our case we chose those that had provided best results in [7].

A summary of the experimental settings is given in Table 2.

Table 2. Experimental set up.

Set	Action selection algorithm	Parameters	Max. actions per run	Runs
Ev	GP-evolved rule	See Table 1	1000	30
Qlearning	$\mathcal{Q}$-learning	$R_{max} = 9999999$; $\gamma = 0.95$	1000	30
RND	Random	N/A	1000	30

4.2 The Software Under Test (SUT)

We used three different applications in order to evaluate our action selection approach, namely Odoo, PowerPoint and Testona. **Odoo** is an open source Enterprise Resource Planning software consisting of several enterprise management applications; of these, we installed the mail, calendar, contacts, sales, inventory and project applications in order to test a wide number of options. **PowerPoint** is a slide show presentation program part of the productivity software Microsoft Office. It is currently one of the most commonly used presentation programs available. **Testona** (formerly known as *Classification Tree Editor*) is a software testing tool that runs on Windows. It implements tree classification, which involves classifying the domain of the application under test and assigning tests to each of its leaves.

4.3 Statistical Analysis

We run the Kruskal-Wallis non parametric test, with $\alpha = 0.05$, on the results for the three action selection mechanisms. The test shows that all the metrics have significant differences among the sets. Running pair-wise comparisons by means of the Mann-Whitney-Wilcoxon test, provides the results shown in the boxplots contained in Figs. 2, 3, 4, 5 and 6; these results are ordered in Table 3, where the shaded column is the best option. It can be seen that the GP approach wins in the abstract states and longest path metrics for both Powerpoint and Odoo and comes second in Testona, where, surprisingly, random testing performs best (Fig. 7).

One metric we have not considered in the statistical analysis is the number of failures encountered, shown in Table 4. Here we can see that in general, the evolutionary approach finds the most real failures[3].

[3] Note that ascertaining whether these failures are associated to any defects is beyond the scope of the *TESTAR* tool.

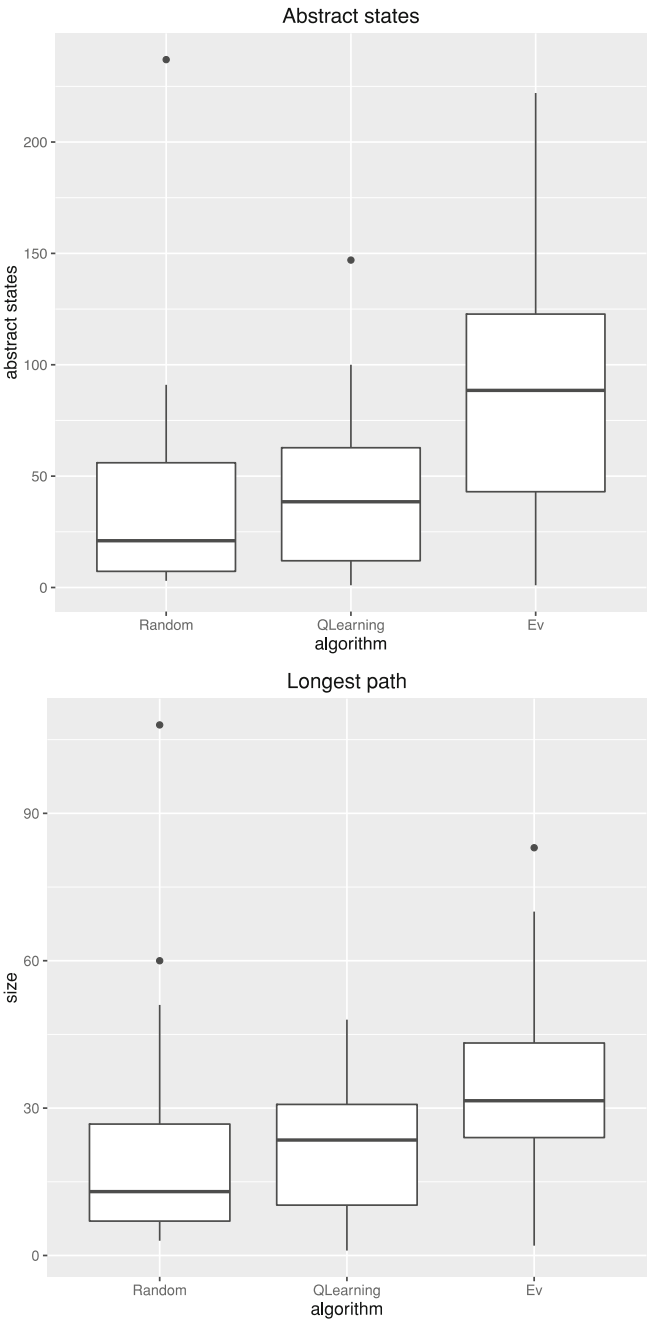

Fig. 2. Boxplots for the abstract states and longest path metrics with the results obtained for Odoo

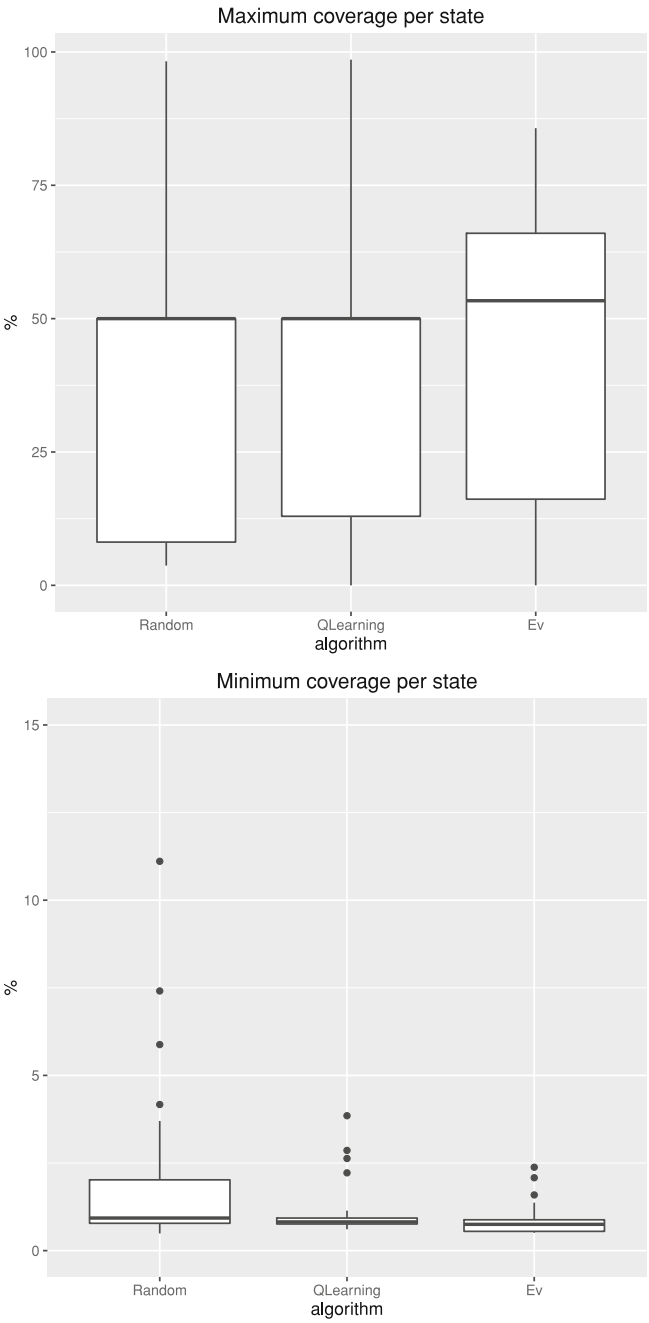

Fig. 3. Boxplots for the maximum and minimum coverage metrics with the results obtained for Odoo

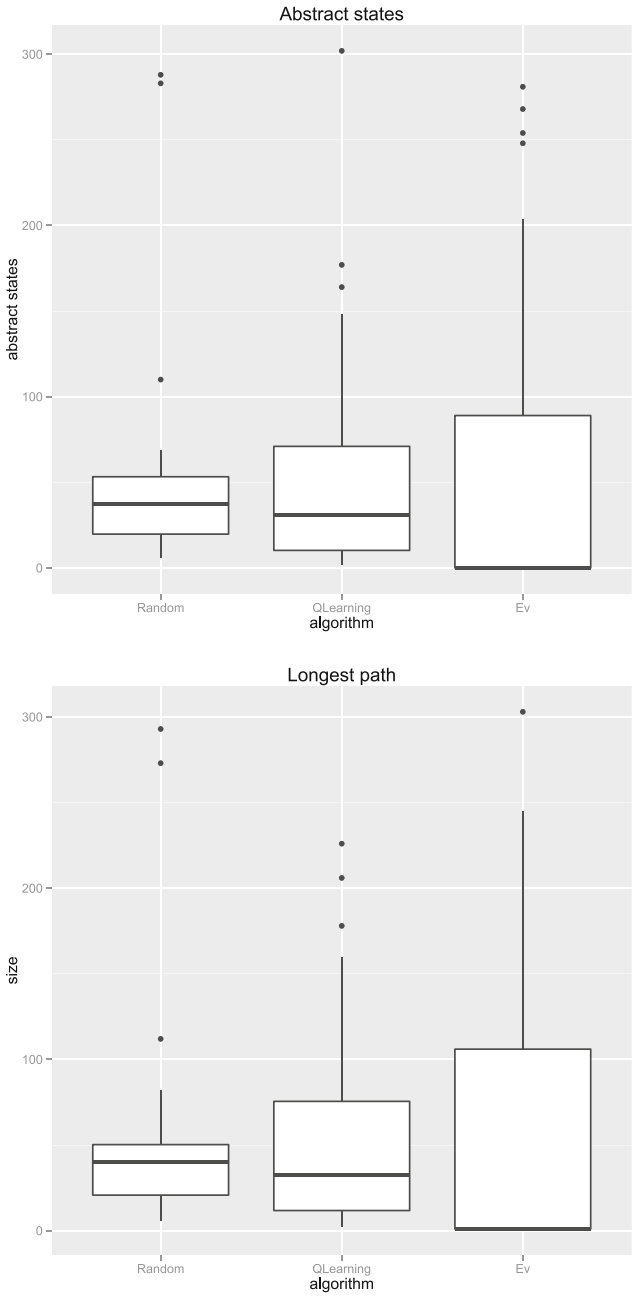

Fig. 4. Boxplots for the abstract states and longest path metrics with the results obtained for PowerPoint.

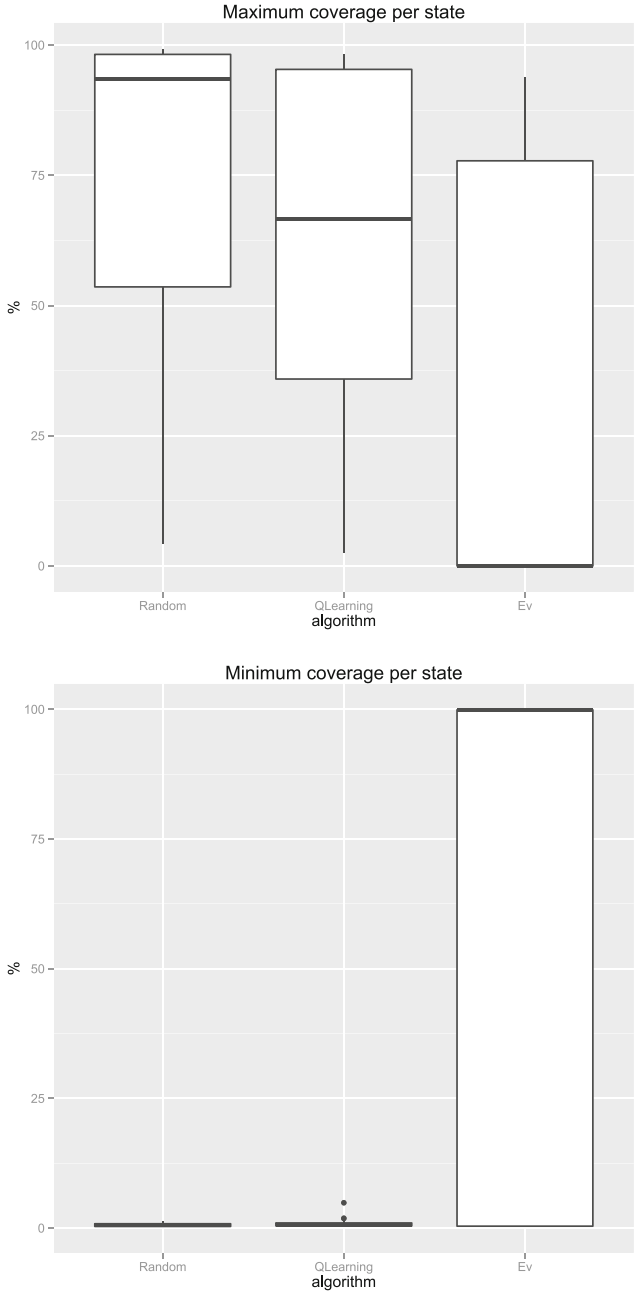

Fig. 5. Boxplots for the maximum and minimum coverage metrics with the results obtained for PowerPoint.

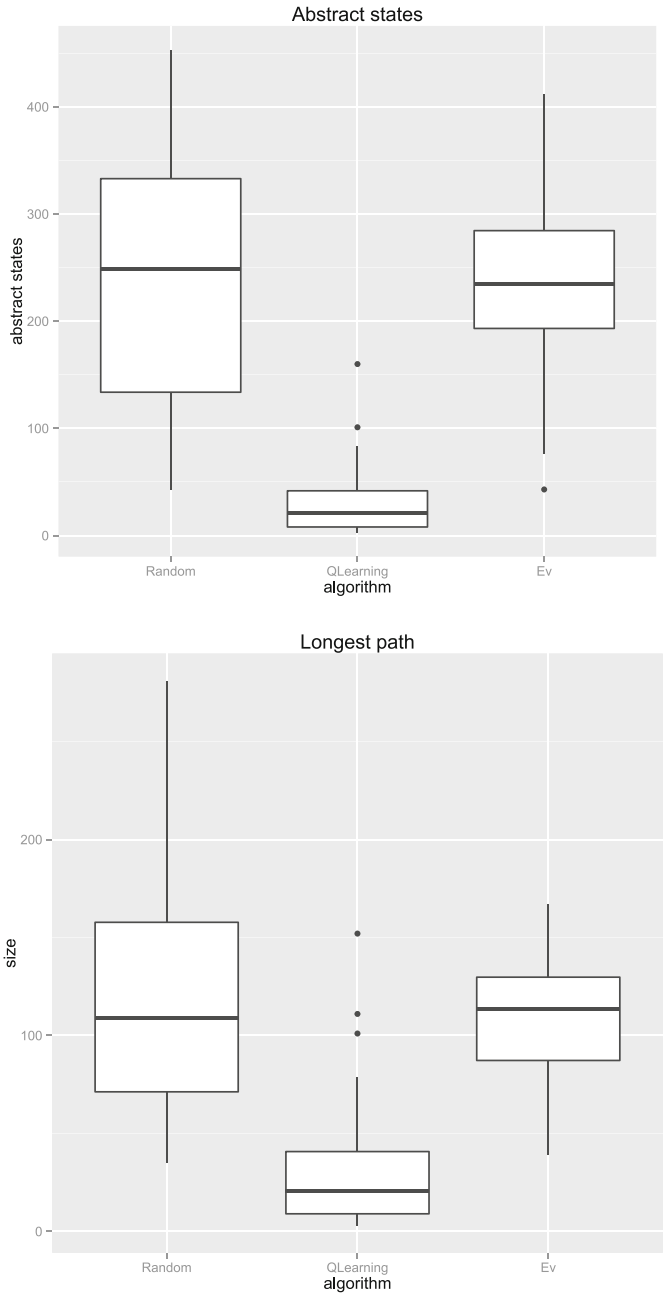

Fig. 6. Boxplots for the abstract states and longest path metrics with the results obtained for Testona.

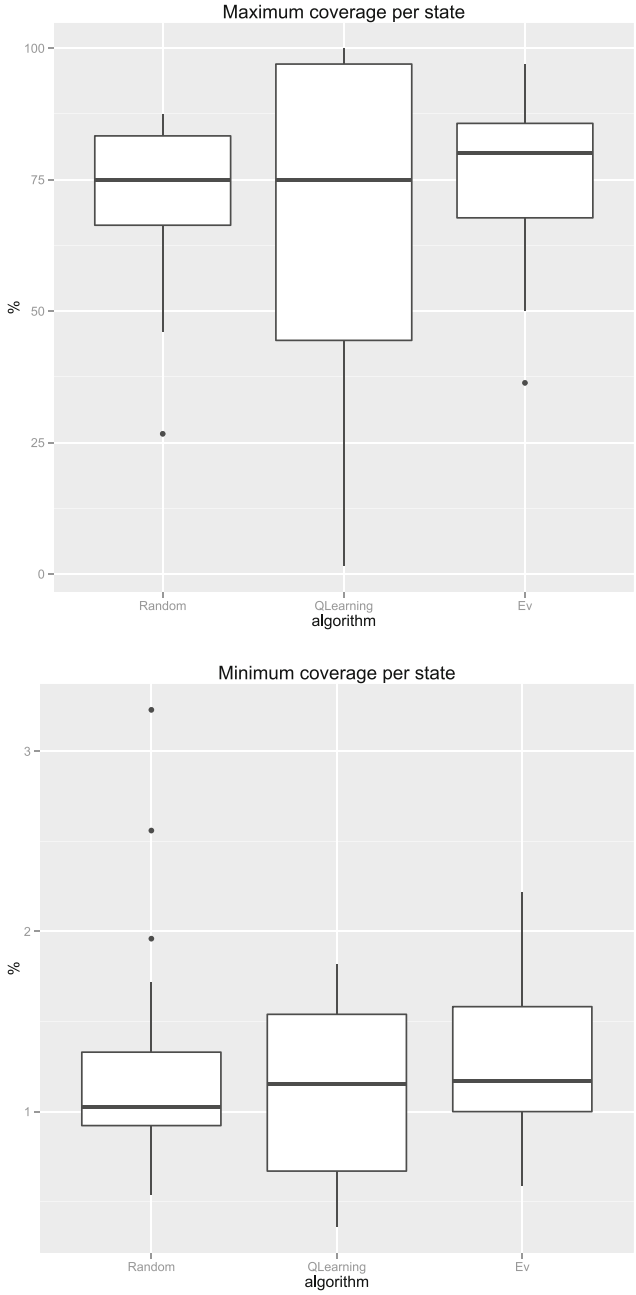

Fig. 7. Boxplots for the maximum and minimum coverage metrics with the results obtained for Testona.

Table 3. Results of the statistical comparison for all algorithms and metrics, in the three different SUTs. The shaded column represents the best choice, the remaining ones are in order of preference.

PowerPoint	Set		
Abstract states	Ev	$\mathcal{Q}$-learning	RND
Longest path	Ev	$\mathcal{Q}$-learning	RND
Maximum coverage per state	RND	$\mathcal{Q}$-learning	Ev
Minimum coverage per state	$\mathcal{Q}$-learning	Ev	RND
Odoo	**Set**		
Abstract states	Ev	$\mathcal{Q}$-learning	RND
Longest path	Ev	$\mathcal{Q}$-learning	RND
Maximum coverage per state	Ev	$\mathcal{Q}$-learning	RND
Minimum coverage per state	RND	$\mathcal{Q}$-learning	Ev
Testona	**Set**		
Abstract states	RND	Ev	$\mathcal{Q}$-learning
Longest path	RND	Ev	$\mathcal{Q}$-learning
Maximum coverage per state	Ev	$\mathcal{Q}$-learning	RND
Minimum coverage per state	Ev	$\mathcal{Q}$-learning	RND

Table 4. Number of failures encountered per SUT and algorithm.

SUT	Algorithm	Errors	Freezes	False positives
Odoo	Ev	**4**	0	2
	RND	0	0	4
	$\mathcal{Q}$-learning	1	1	6
PowerPoint	Ev	**1**	0	5
	RND	0	1	2
	$\mathcal{Q}$-learning	0	1	5
Testona	Ev	**2**	2	3
	RND	0	3	6
	$\mathcal{Q}$-learning	1	1	3

5 Conclusions

We have shown here the successful application of a genetic programming-evolved action selection rule within an automated testing tool. The GP-evolved rule was also compared to $\mathcal{Q}$-learning and random, or *monkey testing*. The performance was evaluated on three SUTs (PowerPoint, Odoo and Testona) and according to four metrics. Statistical analysis reveals the superiority of the GP approach in PowerPoint and Odoo, although not in Testona.

Further work will involve developing more complex rules by introducing new functions and terminals. A further step ahead will also involve eliminating the fitness function and guiding the evolution based on novelty only [9].

Acknowledgments. This work was partially funded by project **SHIP** (*SMEs and HEIs in Innovation Partnerships*, ref: EACEA/A2/UHB/CL 554187).

References

1. Aho, P., Menz, N., Rty, T.: Dynamic reverse engineering of GUI models for testing. In: Proceedings of the 2013 International Conference on Control, Decision and Information Technologies (CoDIT 2013), May 2013
2. Aho, P., Oliveira, R., Algroth, E., Vos, T.: Evolution of automated testing of software systems through graphical user interface. In: International Conference on Advances in Computation, Communications and Services, Valencia (2016)
3. Alegroth, E., Feldt, R., Ryrholm, L.: Visual GUI testing in practice: challenges, problems and limitations. Empirical Softw. Eng. **20**, 694–744 (2014)
4. Bauersfeld, S., Vos, T.E.J.: User interface level testing with TESTAR: what about more sophisticated action specification and selection? In: Post-proceedings of the Seventh Seminar on Advanced Techniques and Tools for Software Evolution, SATToSE 2014, L'Aquila, Italy, 9–11 July 2014. pp. 60–78 (2014). http://ceur-ws.org/Vol-1354/paper-06.pdf
5. Bauersfeld, S., Wappler, S., Wegener, J.: A metaheuristic approach to test sequence generation for applications with a GUI. In: Cohen, M.B., Ó Cinnéide, M. (eds.) SSBSE 2011. LNCS, vol. 6956, pp. 173–187. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23716-4_17
6. Chaudhary, N., Sangwan, O.: Metrics for event driven software. Int. J. Adv. Comput. Sci. Appl. (IJACSA) **7**(1), 85–89 (2016)
7. Esparcia-Alcázar, A.I., Almenar, F., Martínez, M., Rueda, U., Vos, T.E.: Q-learning strategies for action selection in the TESTAR automated testing tool. In: Proceedings of META 2016 6th International Conference on Metaheuristics and Nature Inspired Computing, pp. 174–180 (2016)
8. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992). http://mitpress.mit.edu/books/genetic-programming
9. Lehman, J., Stanley, K.O.: Novelty search and the problem with objectives. In: Riolo, R., Vladislavleva, E., Moore, J.H. (eds.) Genetic Programming Theory and Practice IX. Genetic and Evolutionary Computation, pp. 37–56. Springer, New York (2011)
10. Memon, A.M., Soffa, M.L., Pollack, M.E.: Coverage criteria for GUI testing. In: Proceedings of ESEC/FSE 2001, pp. 256–267 (2001)
11. Seesing, A., Gross, H.G.: A genetic programming approach to automated test generation for object-oriented software. Int. Trans. Syst. Sci. Appl. **1**(2), 127–134 (2006)
12. Wappler, S., Wegener, J.: Evolutionary unit testing of object-oriented software using strongly-typed genetic programming. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO 2006, pp. 1925–1932. ACM, New York (2006). http://doi.acm.org/10.1145/1143997.1144317
13. Watkins, C.: Learning from Delayed Rewards. Ph.D. thesis, Cambridge University (1989)