EvoMBT: Evolutionary model based testing

Article *in* Science of Computer Programming · March 2023

DOI: 10.1016/j.scico.2023.102942

5 authors, including:

Fitsum Meshesha Kifetew
Fondazione Bruno Kessler
56 PUBLICATIONS **1,233** CITATIONS
SEE PROFILE

Davide Prandi
Fondazione Bruno Kessler
108 PUBLICATIONS **11,423** CITATIONS
SEE PROFILE

`EvoMBT`: Evolutionary Model Based Testing

Raihana Ferdous, Chia-kang Hung, Fitsum Kifetew, Davide Prandi, and Angelo Susi

Fondazione Bruno Kessler, Trento, Italy.
Email: {rferdous,chung,kifetew,prandi,susi}@fbk.eu

Abstract

Writing tests for software systems is an important but expensive activity that plays a critical role in the success of the software. This is particularly true in systems where the interaction space is fine grained and in continuous change, as in the case of computer games or cyber-physical systems. In such situations model-based testing offers reasonable testing solutions as it allows to abstract away from details and focus on the relevant aspects from the point of view of testing. In this paper we present our tool `EvoMBT` that combines model-based testing with search algorithms for the generation of test cases for systems with complex and fine grained interactions. We illustrate the basic principles behind `EvoMBT` and provide examples along with empirical data from experiments on a self-driving car simulator where `EvoMBT` is used to produce road configurations that challenge the model responsible for driving the car.

Keywords: model-based testing, search-based testing, game play testing, cyber-physical systems

1. Motivation and significance

Testing software systems with complex and fine grained interaction spaces is difficult as it requires satisfying delicate constraints as well as constructing a sequence of actions that eventually stimulate desired behaviour in the system under test (SUT). Test cases written for such systems also suffer an additional drawback of being rather brittle, i.e., they are likely to break upon successive modifications of the software. In such a situation, model-based testing (MBT) offers a mechanism for abstracting away from details and focuses on the relevant aspects that make the testing effective [1]. In `EvoMBT` we combine the power of abstraction of MBT with the well established search-based software testing (SBST) in order to develop test cases

that achieve high levels of coverage of the SUT as well as exposing potential bugs [2].

EvoMBT implements the classical MBT test generation cycle where 1) abstractions of the SUT are captured in an appropriate model, 2) the model is then used to derive abstract test cases based on some criterion, 3) the abstract test cases are then concretised, and 4) the concrete test cases are executed against the SUT. EvoMBT uses extended finite state machines [3] (EFSMs) as the modelling formalism for capturing the desired SUT behaviour. Hence the abstract test generation is based on a given EFSM, the result of which are paths in the EFSM that correspond to a sequence of actions on the SUT when concretised. The core of EvoMBT, i.e., the generation of abstract test cases from a given EFSM, is completely independent of the SUT. Given an EFSM, EvoMBT applies different search heuristics to find test cases that satisfy the desired coverage criteria, e.g., transition coverage. Specifically, EvoMBT uses the well known EvoSuite[1] tool as a library for the various search algorithms. The other steps of the MBT cycle (abstraction of the model, concretisation of abstract tests, execution of concrete tests against the SUT) however are specific to the particular SUT.

EvoMBT principally supports the use of search-algorithms mainly because they are applicable in a wide range of contexts and scale fairly well on large problems, though they may not always guarantee full coverage. However, EvoMBT also provides interfaces for other approaches as well. For instance, it currently provides a model checking [4] interface where an LTL-based lightweight model checking tool is externally developed[2][3]. Overall, our goal is to let EvoMBT provide the basic infrastructure into which different approaches could be plugged in. For instance, we have also experimented with the use of planning algorithms as well, though our implementation is not currently released. EvoMBT is fully open source and publicly available through GitHub so that interested researchers could contribute different implementations so as to enrich its potential in solving problems using different approaches.

EvoMBT is developed in the context of the iv4XR[4] H2020 project for the testing of 3D games. For this reason, EvoMBT comes by default with a number

[1] www.evosuite.org

[2] https://github.com/iv4xr-project/iv4xr-mbt/blob/master/src/main/java/eu/fbk/iv4xr/mbt/efsm/modelcheckingInterface/InterfaceToIv4xrModelCheker.java

[3] https://github.com/iv4xr-project/aplib/tree/master/src/main/java/eu/iv4xr/framework/extensions/ltl

[4] www.iv4xr-project.eu

of example scenarios from a 3D game called `Lab Recruits`[5]. It also includes a random game level generator for `Lab Recruits` which is useful for experimenting with `EvoMBT` under different complexity levels. `Lab Recruits` is further discussed in Section 3.

To make use of `EvoMBT` for testing a given SUT, the user needs to provide suitable EFSM models corresponding to the scenarios in SUT they would like to test. Subsequently, they would also need to provide a concretisation mechanism for generating executable test cases from the abstract test cases produced by `EvoMBT`. At this point the executable test cases could be run against the SUT and analysed. If desired objectives are not achieved at the SUT level, the process could be repeated as needed by using different models corresponding to different scenarios of the SUT. More details about `EvoMBT` and experimental data on `Lab Recruits` could be found in an earlier publication [2].

2. Software description

`EvoMBT` is developed to be generic and extensible while at the same time being immediately usable. It comes with a number of models so that users can directly try it out, out of the box, and play with the various options for test generation. It also comes ready for trying out the full MBT cycle by using the `Lab Recruits` 3D game as a use case. The GitHub[6] repository contains all the necessary resources to help users get started quickly. To get a taste of the tool, it suffices to simply clone or download the repository and run a single maven command. This launches the tool on an example model included in the tool using some default test generation options. Hence new users can start familiarising themselves using a hands-on approach. A comprehensive list of parameters is described in the online manual so that eventually users get to experience advanced features.

2.1. Software architecture

`EvoMBT` has 5 main components that are glued together via clearly defined interfaces and with a Main class that serves as a point of entry by exposing the various parameters of `EvoMBT` for command line as well as API access.

Model representation This is a core component that provides the representation formalism of EFSMs and all the corresponding operations on them.

[5] `https://github.com/iv4xr-project/labrecruits`
[6] `https://github.com/iv4xr-project/iv4xr-mbt`

Test case representation This component handles the representation of abstract test cases and test suites for use by the search algorithm. Test cases are paths in the EFSM and test suites are sets of test cases. This component also provides different implementations of test case factories that are used for initialising the search algorithms with initial candidate test cases.

Coverage goals This component handles the representation of the various coverage goals. `EvoMBT` provides implementations for state, transition, and k-transition. Further coverage goals could easily be implemented by extending existing ones or by providing new implementations to the generic interfaces defined in this component.

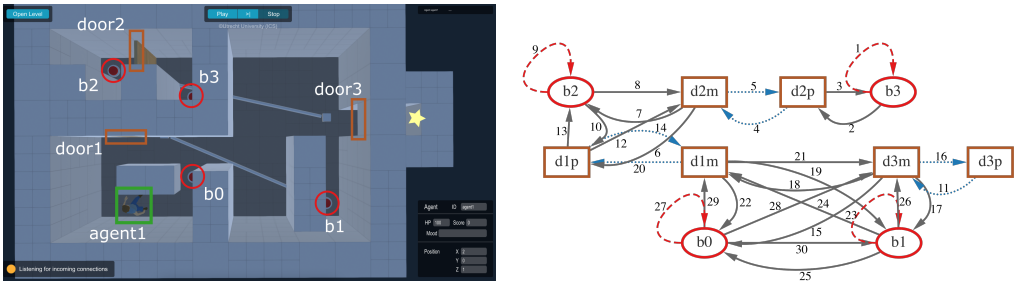
Test case execution This component handles execution of abstract test cases on the model for the purpose of computing the corresponding fitness value, as well as the concretisation and execution of test cases on the SUT, whenever available. An execution of an abstract test case corresponds to replaying the path represented in the test case on the model, starting from its initial state. During execution, different observers are notified of different events, e.g., a transition traversed. One such observer is the coverage goal manager that deals with keeping track of the execution trace and calculation of fitness values using the well established approach level and branch distance heuristic [2].

Algorithm This component is responsible for the various algorithms used for test generation. `EvoMBT` uses the algorithms implemented in EvoSuite. Since the interfaces used in `EvoMBT` are compatible with those in EvoSuite, most of the algorithms implemented in EvoSuite are readily usable in `EvoMBT`. Whenever there are strong deviations in some algorithms, they can be adapted for `EvoMBT` accordingly.

2.2. Main software functionalities

`EvoMBT` offers the following main functionalities:

1. generate tests using various search algorithms and coverage criteria. It is also extensible and new algorithms and criteria can easily be added by implementing clearly defined interfaces.
2. execution of test cases on the SUT, assuming that the concretisation functions are properly defined and the SUT execution environment is properly configured. This is available out of the box for `Lab Recruits`.
3. exporting various artefacts in different formats. Test cases are exported in different formats (e.g., textual, as graphs using dot syntax, serialised



(a) An example of a level in `Lab Recruits`. In this level there are four buttons (`b0, b1, b2, b3`) and three doors (`door1, door2, door3`)

(b) EFSM model of the `Lab Recruits` level. States represent buttons or sides of doors, transitions represent movement or action (pressing a button)

Figure 1: A level of the `Lab Recruits` game and the corresponding EFSM model [2]

binary format, as well as other custom format such as CSVs). Models are also exported using binary formats for later use. In the case of `Lab Recruits`, the levels corresponding to the randomly generated models are also exported.

4. `EvoMBT` is also capable of performing mutation analysis on the SUT, assuming the corresponding artefacts are properly configured. This functionality is available out of the box for `Lab Recruits` where a mutation represents modification of the game level in which the connection between a button and a door is removed.

All functionalities of `EvoMBT` are available as online help when running the tool itself.

3. Illustrative examples

`EvoMBT` has been extensively used to test `Lab Recruits` [2], a 3D game developed for experimenting with intelligent agents. `Lab Recruits` supports the definition of mazes, i.e., a collection of rooms linked by doors that can be opened by pressing buttons. An example of a level in `Lab Recruits` is shown in Figure 1a. The objective is to find the path to a specific room by opening doors in the proper sequence. Both humans and artificial agents can play the game [5]. To fully exploit the `EvoMBT` features, we implemented the whole MBT cycle for `Lab Recruits` [2]. In particular, `EvoMBT` includes 1) a `Lab Recruits` random generator to create a maze model and the associated EFSM; 2) the generation of abstract test cases, i.e., EFSM paths, using different search algorithms; 3) an abstract test case executor able to run EFSM paths directly on `Lab Recruits` by controlling an in-game agent. The EFSM model for a `Lab Recruits` level is composed of states that correspond to the location of entities in the game (e.g., buttons, both sides of doors,

goal flags, fires, etc) and transitions that correspond to player actions (e.g., pressing a button) and movements from one entity to another. If such a movement involves traversing a door, the corresponding transition is guarded by a predicate the checks for the door being open. The EFSM model and the corresponding level are shown in Figure 1.

We also applied `EvoMBT` in the context of Cyber-Physical Systems (CPS) testing by participating in the tool competition at the 15^{th} International Workshop on Search-Based Software Testing [6]. The challenge is to produce a set of virtual roads to test the lane-keeping capability of a self-driving system. `BeamNG.tech` driving simulator[7] is used to evaluate generated roads. The purpose is to find roads that result in an out of bound (oob), i.e., a failure in the self-driving system. A road consists of a list of points in the 2D space that is transformed into an input for the `BeamNG.tech` tool through interpolation [7]. Figure 2a shows an example of a test case corresponding to a road. The 2D space is $80m \times 80m$ and the road comprises of six points, starting from coordinates $(10, 20)$. The arrows indicate the order of points, the solid green line represents the resulting interpolated street, and the light green curve shows the street border that the car must not cross to avoid an *oob*.

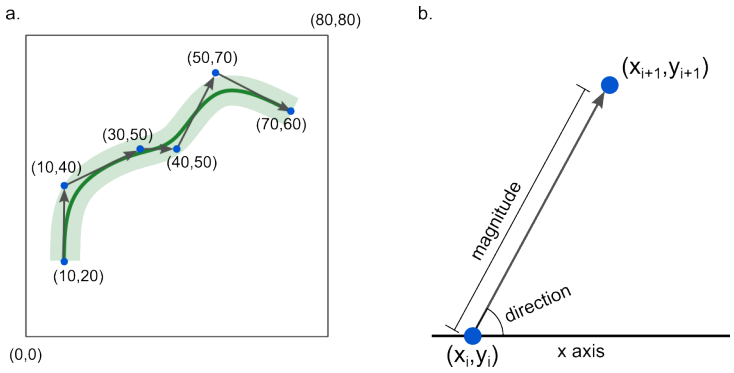


Figure 2: `BeamNG.tech` input structure. **a.** Example of test case. **b.** EFSM representation of a road.

To apply `EvoMBT`, we modelled a road as a list of direction vectors from a point to the next one, as sketched in Figure 2b. Starting from (x_i, y_i), the next point (x_{i+1}, y_{i+1}) is defined by its direction θ and its magnitude m. The direction θ represents the degree from the x axis, while magnitude m specifies the length of the vector. `EvoMBT` has a built-in EFSM generator [6] that relies

[7] `www.beamng.tech`

on the vector representation of a road to generate test cases for `BeamNG.tech` simulator. The generator takes as input the possible directions and magnitude of the direction vectors and creates an associated EFSM. The model generator allows to represent more directions and magnitudes, making the model representative of a variety of road configurations. `EvoMBT` implements test case concretisation, i.e., an EFSM path is transformed in a sequence of 2D points. Not all generated EFSM paths correspond to a valid test and hence they are filtered out before running the `BeamNG.tech` simulator. Note here that the EFSM models used by `EvoMBT` model the environment of the car, rather than the car itself. From these models, `EvoMBT` generates roads, which are input to the simulator. Since the models do not represent the car behaviour, `EvoMBT` does not generate the corresponding oracles. This later aspect is handled by the runtime infrastructure which assesses the behaviour of the car when driven on the input road and adjudicates it as correct/incorrect.

To demonstrate `EvoMBT`, we designed a small experiment to compare the performance of two test generation strategies, `MOSA` and `RANDOM`, for coverage criteria k-transition, with k equal to 1 (i.e., transition coverage) and 2. For each configuration, we performed ten independent runs with different seeds to account for inherent randomness in the algorithms, each run with a time budget of $180s$. For each run we collect the coverage achieved on the model (Figure 3a), the number of generated test cases suitable for `BeamNG.tech` (Figure 3b), and the number of *oob*. We hypothesise that `MOSA` performs better than `RANDOM` and that $k = 2$ would generate more interesting tests than $k = 1$. Wilcoxon test with 5% significance level is used to compare the results. Both strategies achieve a median coverage 95.8% (184 covered goals) for $k = 1$. With $k = 2$, the number of goals increases to 2304 and `MOSA` (median 40.7%) significantly outperforms `RANDOM` (median 29.4%). When comparing the number of valid tests generated, `MOSA` is significantly better than `RANDOM` for both values of k. Finally, mean *oob* is 0.56 ($k = 1$) and 2.25 ($k = 2$) for `MOSA`, while 0.88 ($k = 1$) and 1.8 ($k = 2$) for `RANDOM`. Interestingly, the difference is statistically significant when comparing *oob* for $k = 1$ and $k = 2$.

4. Impact

As mentioned earlier, `EvoMBT` is developed in the context of the EU funded H2020 project iv4XR that works towards the use of artificial intelligence for the verification and validation of extended reality based systems. `EvoMBT` has been developed because there was a gap that needed to be filled regarding test generation for complex software systems such as 3D games. In particular,

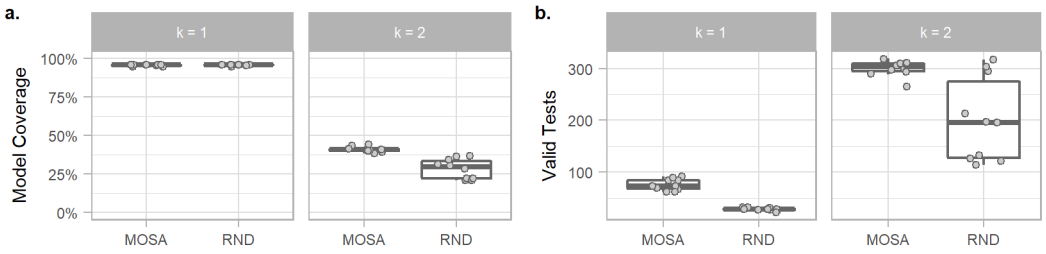


Figure 3: Summary plots for the 10 runs of the Cyber-physical system (CPS) experiment. **a**. Boxplot comparing `MOSA` and `RANDOM` coverage on CPS model for k-transition coverage criteria, with k equal to 1 and 2. Dots represent experimental runs of the same configuration. **b**. Total number of valid tests for `BeamNG.tech` engine considering the same configurations of **a**.

games such as Space Engineers[8], which is one of the use cases of the project, have testing needs that are currently not fulfilled with existing solutions. Hence `EvoMBT` strives to fill this gap.

Beyond the iv4XR project, `EvoMBT` is an openly available tool for researchers to be able to build their solutions for different problem domains by customizing it as needed. `EvoMBT` could be used as a generic MBT tool provided a suitable EFSM model. Furthermore, researchers working on MBT could further experiment on novel algorithms for handling difficult coverage criteria such as path coverage.

As showed in Section 3, we have participated in the 2022 edition of the tool competition for cyber-physical systems, organized as part of the SBST workshop at ICSE 2022. We have developed a custom EFSM model for the road configurations and used `EvoMBT` to generate test cases. The results [8, 6] are promising and show the versatility of `EvoMBT` and its potential to be applied in different domains. In Section 3, We have performed further experiments with the self-driving car using k-transition coverage criterion and we observe that `EvoMBT` is able to generate different road configurations that challenge the self-driving car.

5. Ongoing and future work

`EvoMBT` is a tool for model-based testing of potentially complex systems where directly testing the SUT is difficult. However it is not limited to just such systems, rather it can be applied to any system modelled as EFSM.

[8] `https://www.spaceengineersgame.com/`

It comes with a number of algorithms and heuristics for out of the box use but it can also be used as a library, defining new EFSM models or extending both coverage criteria and search algorithms.

`EvoMBT` is still actively in development in a number of directions. It is being applied to new problem domains both in the context of the iv4XR project (e.g., for testing the Space Engineers game[9]) as well as outside the project. While the core of `EvoMBT` is quite generic, technology/platform specific application domains could also be explored, as for instance integrating with tools such as XState[10] for JavaScript and TypeScript.

We are also working towards integrating a user friendly editor where users can easily develop EFSM models for their SUT. Currently the EFSM models need to be written in Java by implementing the interfaces provided. However, this could be simplified by providing a graphical editor that users could use to build the model. This could be achieved by integrating into `EvoMBT` functionalities implemented in other open source tools such as GraphWalker[11]. Furthermore, `EvoMBT` can also support importing models from different formats that are serialised by other model designing tools.

`EvoMBT` is open source with the code as well as all necessary artefacts such as examples, models, documentation, as well as releases are available from GitHub: `https://github.com/iv4xr-project/iv4xr-mbt`

Acknowledgements

This work is funded by the EU H2020 iv4XR project, grant number 856716.

References

[1] M. Utting, A. Pretschner, B. Legeard, A taxonomy of model-based testing approaches, Softw. Test. Verification Reliab. 22 (5) (2012) 297–312. `doi:10.1002/stvr.456`.

[2] R. Ferdous, F. M. Kifetew, D. Prandi, I. S. W. B. Prasetya, S. Shirzadehhajimahmood, A. Susi, Search-based automated play testing of computer games: A model-based approach, in: U. O'Reilly, X. Devroey (Eds.), Search-Based Software Engineering - 13th International Symposium, SSBSE 2021, Bari, Italy, October 11-12, 2021, Proceedings, Vol.

[9] `https://iv4xr-project.eu/use-cases/`

[10] `https://github.com/statelyai/xstate`

[11] `(https://graphwalker.github.io`

12914 of Lecture Notes in Computer Science, Springer, 2021, pp. 56–71. doi:10.1007/978-3-030-88106-1_5.

[3] K.-T. Cheng, A. S. Krishnakumar, Automatic functional test generation using the extended finite state machine model, in: 30th ACM/IEEE Design Automation Conference, IEEE, 1993, pp. 86–91.

[4] G. Fraser, F. Wotawa, P. Ammann, Testing with model checkers: a survey, Softw. Test. Verification Reliab. 19 (3) (2009) 215–261. doi:10.1002/stvr.402.
URL https://doi.org/10.1002/stvr.402

[5] I. Prasetya, M. Dastani, Aplib: An agent programming library for testing games, in: Proceedings of the 19th international conference on autonomous agents and multiagent systems, 2020, pp. 1972–1974.

[6] A. Gambi, G. Jahangirova, V. Riccio, F. Zampetti, SBST tool competition 2022, in: 15th IEEE/ACM International Workshop on Search-Based Software Testing, SBST@ICSE 2022, Pittsburgh, PA, USA, May 9, 2022, IEEE, 2022, pp. 25–32. doi:10.1145/3526072.3527538.

[7] V. Riccio, P. Tonella, Model-based exploration of the frontier of behaviours for deep learning system testing, in: Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE '20, Association for Computing Machinery, 2020, p. 13 pages. doi:10.1145/3368089.3409730.

[8] R. Ferdous, C. Hung, F. M. Kifetew, D. Prandi, A. Susi, EvoMBT at the SBST 2022 tool competition, in: 15th IEEE/ACM International Workshop on Search-Based Software Testing, SBST@ICSE 2022, Pittsburgh, PA, USA, May 9, 2022, IEEE, 2022, pp. 51–52. doi:10.1145/3526072.3527534.

Nr.	Code metadata description	
C1	Current code version	1.2.2
C2	Permanent link to code/repository used for this code version	`https://github.com/iv4xr-project/iv4xr-mbt`
C3	Permanent link to Reproducible Capsule	doi.org/10.5281/zenodo.7494688
C4	Legal Code License	Apache License 2.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Java
C7	Compilation requirements, operating environments and dependencies	Maven 3, Java 11 or later, Linux/MacOS/Windows
C8	If available, link to developer documentation/manual	`https://github.com/iv4xr-project/iv4xr-mbt/wiki`
C9	Support email for questions	kifetew@fbk.eu

Table 1: Code metadata