Automata Language Equivalence *vs.* Simulations for Model-based Mutant Equivalence: An Empirical Evaluation

Xavier Devroey*, Gilles Perrouin*, Mike Papadakis†, Axel Legay‡, Pierre-Yves Schobbens*, and Patrick Heymans*
*PReCISE Research Center, University of Namur, Belgium, Email: *firstname.lastname*@unamur.be
†SnT, SERVAL Team, University of Luxembourg, Email: michail.papadakis@uni.lu
‡INRIA Rennes, France, Email: axel.legay@inria.fr

***Abstract*—Mutation analysis is a popular test assessment method. It relies on the mutation score, which indicates how many mutants are revealed by a test suite. Yet, there are mutants whose behaviour is equivalent to the original system, wasting analysis resources and preventing the satisfaction of the full (100%) mutation score. For finite behavioural models, the Equivalent Mutant Problem (EMP) can be addressed through language equivalence of non-deterministic finite automata, which is a well-studied, yet computationally expensive, problem in automata theory. In this paper, we report on our preliminary assessment of a state-of-the-art exact language equivalence tool to handle the EMP against 3 models of size up to 15,000 states on 1170 mutants. We introduce random and mutation-biased simulation heuristics as baselines for comparison. Results show that the exact approach is often more than ten times faster in the weak mutation scenario. For strong mutation, our biased simulations are faster for models larger than 300 states. They can be up to 1,000 times faster while limiting the error of misclassifying non-equivalent mutants as equivalent to 10% on average. We therefore conclude that the approaches can be combined for improved efficiency.**

***Index Terms*—model-based mutation analysis; automata language equivalence; random simulations**

I. Introduction

Mutation analysis is a technique that assess testing robustness by introducing artificial defects, called *mutants*. Mutants are typically used for evaluating the effectiveness of test suites [1], [2] and to support test generation [3], [4]. The technique is popular as it helps simulating the behaviour of real faults [1] and reveals more faults than other test criteria [2]. Researchers also applied the method on models [5], [6] in order to test for defects related to missing functionality or misinterpreted specifications [7]. In practice, model-based mutation was proved to be powerful. As reported by Aichernig *et al.* [8], model mutants can lead to tests that reveal implementation faults that were found neither by manual tests, nor by the actual operation, of an industrial system.

Despite its potential, mutation analysis faces a number of challenges that currently prevent wider adoption [9]. One of them is the *Equivalent Mutants Problem* (EMP). It concerns the mutants whose behaviour is identical to the original artefact (code or model). Such mutants cannot be distinguished by any test, a situation that raises two issues: (i) they hamper the use of the criterion as a stopping rule by skewing the mutation score measurement (the number of detected mutants divided by the total number of mutants), and (ii) they do not bring any new value to the test generation techniques as they attempt to kill mutants that have no chance to be killed.

In this paper, we focus on the model-based formulation of the EMP, which can be expressed in terms of language equivalence. Language equivalence has been studied by the formal verification community who determined its P-SPACE complexity [10] and derived exact equivalence checking algorithms [11], [12]. While potentially helpful, such tools have, to our knowledge, never been used to tackle the EMP. This is the main contribution and novelty of this paper.

In summary, the contributions of this paper are: (i) the design of two simulation algorithms relying either on random simulations (RS) or biased simulations (BS) covering infected states [3] (*i.e.*, exploiting syntactical differences between original and mutant models) to improve the chances to distinguish non-equivalent mutants; (ii) a configurable implementation of our simulations that benefits from the fact that simulation can be easily distributed amongst processor cores; (iii) the definition of an experimental setup to apply an automata language equivalence tool (ALE) [11] to the EMP: we employed three models (from nine to 15,000 states), produced 1170 mutants using seven operators, and considered four mutation orders (one, two, five, ten), according to strong and weak mutation scenarios; (iv) the assessment of the ALE tool with respect to our algorithms. We measured the speed and accuracy of equivalence detection. For weak mutation the ALE tool is ten times faster than the simulations. However, biased simulations perform well for strong mutation on models larger than 300 states as they can be 1,000 times faster. The ratio of tagging non-equivalent mutants as equivalent is 8% for biased simulations and 15% for random ones. To ease reproducibility, our implementation, all our models, and experimental results are available at https://projects.info.unamur.be/vibes/mutants-equiv.html.

II. Background

A. Transition Systems & Finite Automata

Our research in model-based testing considers transition systems as a powerful abstract formalism to model system

behaviour. Our definition is adapted from Baier and Katoen's book [13], where atomic propositions have been omitted (we do not consider state internals): a Transition System (TS) is a tuple $(S, Act, trans, i)$ where S is a set of states, Act is a set of actions, $trans \subseteq S \times Act \times S$ is a non-deterministic transition relation (with $(s_1, \alpha, s_2) \in trans$ sometimes noted $s_1 \xrightarrow{\alpha} s_2$), and $i \in S$ is the initial state. To deal with test generation activities, where finite behaviours are sought, we first require that sets S and Act are finite. To mimic weak and strong mutation scenarios (see Section III-A), we will stop our executions in specific states. These additional requirements make our execution semantics equivalent to that of usual non-deterministic finite automata (NFA), thereby enabling the comparison of our simulations to ALE tools. In the remainder of this paper, unless otherwise stated, we will always refer to TSs with such restrictions so that the term can be used interchangeably with NFAs. Let $ts = (S, Act, trans, i)$ be a TS, let $t = (\alpha_1, \ldots, \alpha_n)$ where $\alpha_1, \ldots, \alpha_n \in Act$ be a finite sequence of actions. The trace t is valid iff:

$$ts \overset{(\alpha_1,\ldots,\alpha_n)}{\Longrightarrow} s \text{ with } s \subset S,$$

where $ts \overset{(\alpha_1,\ldots,\alpha_n)}{\Longrightarrow}$ is equivalent to $\exists s \in S : i \overset{(\alpha_1,\ldots,\alpha_n)}{\Longrightarrow} s$, meaning that there exists a non-empty sequence of transitions labelled $(\alpha_1, \ldots, \alpha_n)$ from i to a state s of the TS.

B. Equivalent Mutant Problem

In this paper, we focus on the model-based instance of the Equivalent Mutant Problem (EMP). The equivalent mutant problem is a well-known issue in mutation analysis [9]. It stems from the fact that two program variants may exhibit the same behaviour and therefore cannot be distinguished by test cases. This is particularly problematic with respect to both generation and assessment of test suites, since 100% of killed mutants is impossible to reach in case of equivalence (also the EMP leads to wasting resources spent on assessing useless mutants). Mutant equivalence can take two forms [9]: (a) equivalence between mutants and the original system; (b) equivalence between two mutants (not with the original system). Mutants of case (a) are called *equivalent* while mutants of case (b) are called *duplicate*. In the context of this paper, we focus on mutants that are behaviourally equivalent to the original system, *i.e.*, mutants of case (a).

C. Automata Language Equivalence & EMP

In our context, the EMP corresponds to a classic problem in automata theory: *Automata Language Equivalence* (ALE). The accepted language of an automaton is formed by all the sequences of actions (words) that can be accepted *i.e.*, starting in the initial state and ending in a final state. Therefore, if a mutant m accepts the same language as the original o (language-equivalent), then there is no trace t that can distinguish the mutant from the original: $\forall t, t \in \mathcal{L}(o) \Leftrightarrow t \in \mathcal{L}(m)$.

There are various forms of relations defined between two automata that we can compute to determine whether they are language-equivalent. Among them, we can cite bisimulations or trace equivalence [13]. In the last years, the verification community came up with dedicated algorithms such as bisimulations up to congruence [11] or antichains [12] to address language equivalence.

Although tackling the language equivalence and inclusion problems from different angles and heuristics, all these techniques may face exponential blow-up since both language inclusion and equivalence were demonstrated to be P-SPACE complete [10]. While worst-case complexity can seem discouraging, various heuristics have been proposed to limit the effects of this complexity in practice. One of the goals of this paper is to determine the applicability of an exact language equivalence algorithm to address the EMP [11]. The algorithm selected due to its availability, reported performance over the state of the art and ability to handle non-determinism that mutations may incur. In the next section, we also present two baseline algorithms that run generated traces to distinguish original and mutants' behaviours.

III. Mutant Equivalence Analysis

A. Strong and Weak Mutation

Our simulations detect an incorrect state if a trace that is valid with respect to the original TS is invalid on the mutant TS, and vice-versa. Indeed, when executed, a trace induces one or more *runs* (alternating sequences of states and actions), depending on the presence of non-determinism. If such a run does not contain all the actions of the trace (*i.e*, the run is *incomplete*), it is because of the presence of an incorrect state preventing the subsequent actions to be fired. If all runs are complete, the original and the mutant are assumed equivalent for this trace. Mutants affect the final states of these runs. For weak mutation, these states can map to any state of the TS. For strong mutation, we need to account for the fact that TSs have no final states. A frequent example is the modelling of user sessions in which, after a legitimate sequence of actions, the system returns to its initial state to welcome a new user. This occurs in two thirds of the systems we analyse in Section IV-1. This is why we model strong mutation by generating traces whose runs start and end in the same initial state.

The ALE approach uses automata that have explicit initial and final states. For weak mutation, we generate automata in which all states are final, and for strong mutation the initial state is the only final state.

B. Automata Language Equivalence (ALE)

The ALE approach we selected for comparison is developed by Bonchi and Pous [11]. It can be thought of an extension to non-deterministic TSs of the Hopcroft-Karp algorithm. In particular, they introduce a new bisimulation relation called *up to congruence* that requires to explore less states than the original algorithm. This approach also avoids to build the complete deterministic finite TS and performs determinisation on-the-fly. This makes such an approach particularly relevant: (i) non-determinism may be introduced locally by mutations (our original models are deterministic), thereby limiting determinisation scope, and (ii) between 0% and 15.5% of our mutants are non-deterministic (see Section IV-1).

Algorithm 1 Generic simulation

Require: $o : TS$ {the original system}
$m : TS$ {the mutant to compare to o}
N {total number of traces to generate}
k {trace length}
Ensure: returns a positive or negative trace differentiating m from o or a special value ($none$) if m is equivalent to o.

1: $traceset \leftarrow select(o, \frac{N}{2}, k)$
2: **for all** $t \in traceset$ **do**
3: **if** $\neg(m \stackrel{t}{\Longrightarrow})$ **then**
4: **return** $pos(t)$
5: **end if**
6: **end for**
7: $traceset \leftarrow select(m, \frac{N}{2}, k)$
8: **for all** $t \in traceset$ **do**
9: **if** $\neg(o \stackrel{t}{\Longrightarrow})$ **then**
10: **return** $neg(t)$
11: **end if**
12: **end for**
13: **return** $none$

C. Random and Biased Simulation

Our randomized approach to equivalence analysis is straightforward: we generate random traces from the original model and run them on the mutant model and reciprocally. If a trace fails to execute on one of the models, it serves as a counterexample and disproves equivalence. If all runs succeed, then the mutant is considered *probably equivalent* and testers have to decide if they want to perform more simulations or switch to an exact method. Algorithm 1 presents our generic simulation approach: N traces are selected (resp.) from the original model (line 1) and the mutant model (line 7), and executed (resp.) on the mutant model (line 3) and the original model (line 9). In case of non deterministic behaviour, all the possible paths are considered for the execution of the trace. If one execution fails, the algorithm stops and returns a *positive* trace such as $(o \stackrel{t}{\Longrightarrow}) \wedge \neg(m \stackrel{t}{\Longrightarrow})$ (line 4) or a *negative* trace such as $\neg(o \stackrel{t}{\Longrightarrow}) \wedge (m \stackrel{t}{\Longrightarrow})$ (line 10) . This generic simulation algorithm is instantiated through two strategies for trace generation (lines 1 and 7): *Random Simulation* (RS) and *Biased Simulation* (BS). The parameter N is computed using the Chernoff-Hoeffding bound as explained hereafter.

1) Random Simulation (RS): Random simulation (RS) assumes a uniform distribution of traces over the model, that is, such traces are selected randomly ($select$ call on lines 1 and 7 in Algorithm 1) by accumulating the actions α_i triggered by a random walk of a given length $\leq k$ in the TS. For weak mutation (WM RS), the only constraint is to start the random walk from the initial state i. Strong mutation (SM RS) requires a random walk starting from and ending in i: after few tries, this method (*i.e.*, using a random walk until the initial state i is reached) showed very poor results on our largest models (we set a timeout of one hour for one equivalence detection) and is therefore not further discussed in this paper.

2) Biased Simulation (BS): The biased simulation (BS) approach exploits the basic characteristics of mutation testing: mutations are localised and they create (most of the time) behavioural differences. It assumes that those differences are detected by a trace t which, when executed on the original TS o or on its mutant m, goes through one of the states affected by the mutation. For instance, the transition missing (TMI in Table II) operator produces a mutant by removing a transition $a \stackrel{\alpha_i}{\longrightarrow} b$ from the original TS. The BS approach generates traces in o and m, such that their executions $m \stackrel{t}{\Longrightarrow}$ or $o \stackrel{t}{\Longrightarrow}$ cover a or b. Such states, called *infected states*, have been shown to help identifying equivalent mutants at the code level and to speed up mutation analysis [14]. This motivates us to adopt this strategy in our biased simulation.

In practice, the set of infected states S_{infect} is computed by checking syntactic differences between the original and mutant TSs. It will include: (i) connected states (i.e, states accessible from the initial state) from one model which are not present in the other, and (ii) states with differences in their input/output transitions: in number of transitions or in action names, considering any pair of states $< s_o, s_m >$ where s_o is a state in the original TS, s_m a state in the mutant TS, such that their names are identical. An alternative is to instrument the mutant generator to keep track of the list of infected states while generating the mutants. Our goal is to be able to apply this strategy without any information on how the mutants are generated (*e.g.*, generated by other frameworks than ours) and to fairly compare with an exact approach that makes no assumption on the locality of differences. Once the set of infected states S_{infect} is obtained (by any means), the second step is to generate traces that cover such infected states.

For weak mutation (WM BS), a trace t is selected ($select$ call on lines 1 and 7 in algorithm 1) by concatenating the actions of (i) the shortest walk from the initial state i to a randomly chosen state $a \in S_{infect}$ and (ii) a random walk starting from a. To proceed, the first step during trace generation is to compute the shortest distance (*i.e.*, the number of transitions) between each state of the original TS o (or its mutant m resp.) and the initial state i of o (or m resp.) using a standard breadth-first search. For strong mutation (SM BS), instead of a random walk starting from a, the algorithm will consider the actions of a path starting from a and returning to i using the computed shortest distance: the distance from a to i will (not strictly) decrease each time a transition is taken in the path.

3) Estimating the Number of Required Runs: An important parameter for simulation is the number of runs N. Herault *et al.* [15] suggested to use the Chernoff-Hoeffding bound to estimate the number N of required runs to limit the equivalence probability depending on the approximation parameter $\epsilon > 0$ and a confidence parameter $\delta < 1$, for traces that are uniformly distributed. Adapted to our two-way simulation, we have: $N = \frac{8 \log(2/\delta)}{\epsilon^2}$. In the context of biased simulations, trace distribution is not uniform as the infected states "force" traces to explore only given portions of the model, *viz.* where the mutations are. We therefore do neither interpret δ and ϵ values nor an equivalence probability. There are just a convenient means to compute N in our experiments.

TABLE I
MODELS CHARACTERISTICS

Model	States	Trans.	Act.	Avg. deg.	BFS height	Back lvl tr.
Minepump	25	41	23	4.64	15	9
AGE-RR	772	6,639	772	8.60	328	408
Random	15,000	20,488	300	1.37	11,865	4,899

TABLE II
TRANSITION SYSTEM MUTATION OPERATORS

SMI	State Missing operator removes a state (other than the initial state) and all its incoming/outgoing transitions.
WIS	Wrong Initial State operator changes the initial state.
AEX	Action Exchange operator replaces the action linked to a given transition by another action.
AMI	Action Missing operator removes an action from a transition.
TMI	Transition Missing operator removes a transition.
TAD	Transition Add operator adds a transition between two states.
TDE	Transition Destination Exchange operator modifies the destination of a transition.

IV. EMPIRICAL ASSESSMENT

This section presents our empirical assessment of the ALE, RS, and BS approaches. We define the following research questions: **RQ1** *How many non-equivalent mutants are effectively detected by the RS and BS approaches?* **RQ2** *What are the worst case execution times for the ALE and BS/RS approaches?*

To answer these RQs, we consider several models of different kinds of systems and apply the following procedure to each of them: (i) we generate a set of mutants from the model using the operators presented in Table II for orders 1, 2, 5, and 10; (ii) for each order, we sample 100 non-equivalent mutants (using the ALE algorithm to guarantee non-equivalence) to form the mutant set M; (iii) for each mutant in M, we measure the execution time and result of: 3 executions of weak mutation random and biased search (WM RS/BS), and 3 executions of strong mutation-biased search (SM BS) algorithms[1] with 4 different values of δ and ϵ; and the executions of the ALE algorithm. In the following we detail the different steps of the procedure. The assessment has been performed on a Debian 3.16.7 x86_64 GNU/Linux running on a 16 cores, 2.2 GHz, 16Gb RAM virtual machine.

1) Models: We carry out the assessment on 3 different models coming from different sources and with varying size detailed in Table I. The different characteristics considered are: the number of states (*St.*); the number of transitions (*Tr.*); the number of actions (*Act.*); the number of incoming plus outgoing transitions per state (*Avg. deg.*); the maximal number of states between the initial state and any other state when traversing the TS in breadth-first search (*BFS h.*); the number of transitions whose source state has a higher *BFS h.* value than its destination state (*Back lvl tr.*). The models are: the mine pump (*Minepump*) [16]; WordPress model (*AGE-RR*) that represent the navigational usages of a real WordPress instance (reverse-engineered using a 2-gram inference method [17]). The random model (*Random*) generated to have properties which are likely to represent real system [18].

2) Mutant Generation and Sampling: First-order mutants are generated using the operators presented in Table II. Each operator is applied (arbitrarily) 10 times on the *Minepump* model. Due to the small size of the model, applying the same mutation operator more than 10 times is not relevant. Operators are also applied (arbitrarily) 500 times on the other models. In the same way, N-order mutants (with N equal to 2, 5, or 10 in our case) are generated by applying the same operators 10/500 times (depending on the model) on $(N-1)$-order mutants. After the generation, we perform a random sampling of 100 mutants (when available) for orders 1, 2, 5, and 10, giving us a set M with 370 mutants for the *Minepump* models, and 400 mutants for the other models. To ease mutant generation, we use our compact representation [5].

3) Non-determinism: We checked all the 1170 mutants and found that only 4.11% of them are non-deterministic. Nevertheless, there is a great disparity amongst models as the non-determinism rate varies from 0.25% for *AGE-RR* to 11.35% for *Minepump*. Higher-order mutation greatly influenced non-determinism rates: the sole order 10 is responsible for 43% of all non-deterministic mutants.

4) Algorithm Execution: To run the language equivalence algorithms (for WM and SM), we use the HKC library [19], an OCaml implementation of the ALE algorithm [11] compiled using OCamlbuild. This tool handles non-deterministic TSs using different strategies: the automata may be processed either forward of backwards, and the exploration strategy may be breadth-first or depth-first. For each mutant, we execute the HKC library using each of the 4 possible configurations.

The random and biased simulation algorithms are implemented in Java using multi-threading to parallelize trace selection and execution as described in Algorithm 1 (lines 1, 3, 7, and 9). In our experiments, we set up the algorithm with 4 threads and run 4 instances in parallel on our virtual machine with 16 cores. We run the simulation algorithms with 4 different values of δ and ϵ determining the number of traces selected and executed (N in Algorithm 1):

- RS1/BS1: ($\delta = 1e-10$, $\epsilon = 0.01$, $N = 1,897,519$);
- RS2/BS2: ($\delta = 1e-10$, $\epsilon = 0.1$, $N = 18,975$);
- RS3/BS3: ($\delta = 1e-5$, $\epsilon = 0.1$, $N = 9,764$);
- RS4/BS4: ($\delta = 1e-1$, $\epsilon = 0.1$, $N = 2,396$).

For all the simulation configurations and all models, we fixed the trace length k to 3,000, which was our compromise between performance and non-equivalence detection: setting k to BFS height led to crashes in some cases. In order to answer RQ2, we also run each algorithm (RS1/BS1 to RS4/BS4, plus the 4 possible ALE configurations) with the model itself as the "mutant". Those (unrealistic) equivalent detection runs between the model and itself are only used to approximate the worst computation time of the different algorithms.

A. Results and Discussion

1) Non-equivalent mutant detection - Answering RQ1: To answer RQ1, we compute the non-equivalent mutant classification recall of the BS/RS algorithms (in Figure 2), *i.e.*,

[1] As explained in section III-C1, SM RS is not considered for the assessment due to the poor results during our initial attempts.

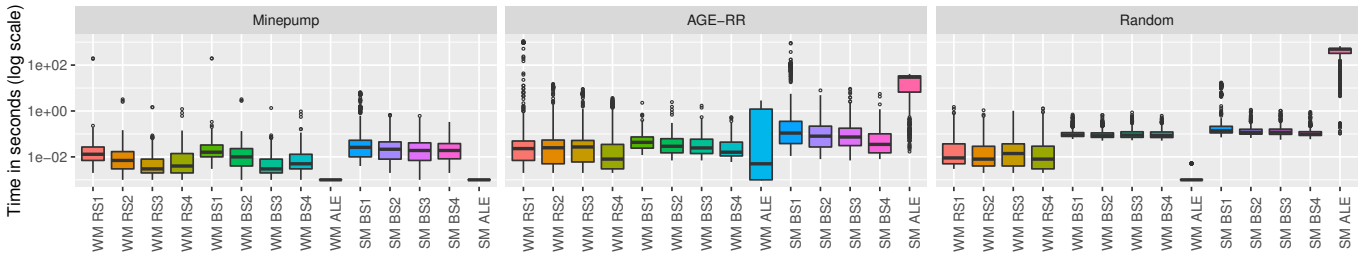

Fig. 1. Execution time of the equivalent mutant detection

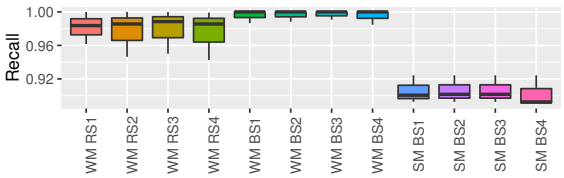

Fig. 2. Recall

the percentage of non-equivalent mutants detected by the BS/RS amongst the selected mutants. By construction, the ALE algorithm has a recall of 100%, it is therefore not shown here. It is also noted that the precision is 100% since all the non-equivalent mutants detected are indeed killable, by construction of our mutant set.

All our simulations obtain a recall higher than 90%, with a clear advantage for biased simulations which never achieve worse than 98% for the weak mutation scenario. As for time, deviation in the recall is smaller for biased simulations thus making the approach more predictable in addition of being more reliable. We also observe that the random simulations are more sensitive to the number of runs: we need more of them to discover discrepancies by luck. This effect cannot be observed for biased simulations. A possible explanation is that the number of runs required to cover infected states with traces is lower than the number we provided.

For strong mutation, the BS approach's recall decreases to around 90% ($\overline{recall} = 90\%$, $\sigma = 1\%$): amongst the 1615 non-equivalent mutant non-detections (over a total of 21688 non-equivalent mutant evaluations), 570 (35%) were TAD mutants, 570 (35%) were WIS mutants, 167 (10%) were TDE mutants, and 189 (11%) were 2nd-order TAD mutants (*i.e.*, TAD-TAD mutants); the rest of non-equivalent mutants not detected is distributed amongst different operators with less than 2% for each. This decrease may be due to the difficulty to find a path to the initial state: for strong mutation, the BS trace selection algorithm will consider traces starting from, and ending in, the initial state. This means that mutations creating (TAD) or modifying (TDE) a back-level transition will not be detected using SM BS. Concerning WIS mutants, we believe that, as the WIS operator only changes the initial state of the TS, the set of infected states (S_{infect}) is empty, which is equivalent in our implementation to considering all the states infected.

2) Worst case scenario (execution time) - Answering RQ2: Figure 3 presents a compact view of the worst execution time of the different algorithms (RQ2). As expected, the RS/BS execution time is directly correlated to the δ and ϵ values: a lower number of traces selected and executed (N) takes less time. Overall, the time of the ALE executions grows with the size of the model, reaching 5660 seconds (more than one and a half hour) for the worst WM ALE execution time on the Random model.

B. Lessons Learned

From our experiment we draw the following lessons. (i) Regarding weak mutation and independently of the size or nature of the models, the ALE approach provides faster and exact answers. This indicates that state-of-the-art language equivalence algorithms can be used successfully for such a task. (ii) Regarding strong mutation, biased random simulations are of interest for the web and the random models, and gains increase with the size (from one to three orders of magnitude). Recalls of 90% and above allow to use such simulations as reasonably reliable fast filters to discard non-equivalent mutants, leaving to ALE algorithms difficult cases so as to accelerate the analysis of large mutants bases. (iii) Biased simulations are more predictable in terms of execution time and recall. Additionally, drastically increasing the number of runs does not affect their performance as opposed to random simulations. (iv) The configuration of the ALE algorithm (forward/backward processing, or breadth-first or depth-first exploration) has very little influence on the total execution time (regarding equivalent mutant detection). This may be explained by the fact that mutations occur randomly and therefore do not privilege any graph traversal strategy.

V. Related Work

Monte Carlo simulations have been used to devise statistical model-checking techniques [15], [20] that alleviate state explosion. Poulding and Feldt [21] used Nested Monte-Carlo Search, to generate random data structures to be used for testing.All these methods are related to ours since they use Monte-Carlo but, none of them aims at modelling mutants or tackling the equivalent mutant problem.

Walkinshaw and Bogdanov [22] advocate that using random selection to compare automata languages may be biased due to the impossibility to obtain a representative sample of the language. In their work, they use a model-based testing approach to compare two automata from the accepted language perspective, and a *diff* algorithm to compare them with respect to their transition structures (which is a more elaborate version

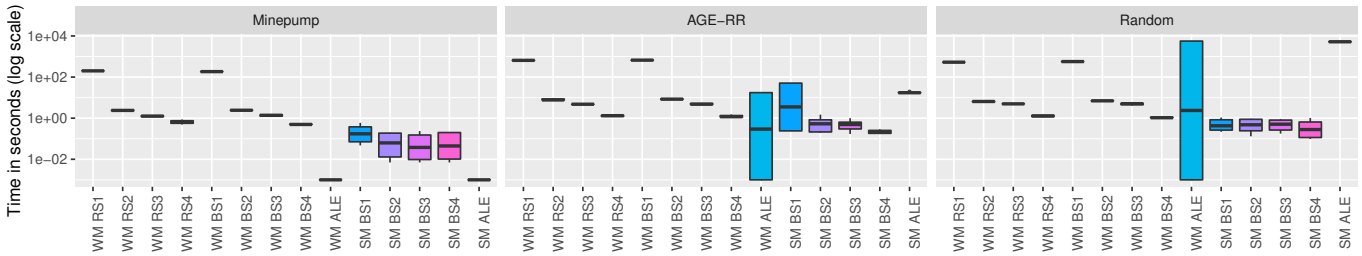

Fig. 3. Worst execution time of the equivalent mutant detection using the model itself as mutant

of our heuristic used to compute the set of infected states S_{infect}). In contrast, we look for difference instead of similarity, which motivates the choice of easier-to-compute random heuristics as baselines to compare with an ALE approach.

Code-based equivalent mutants can be identified using compiler optimizations [9], program slicing [23] and formal verification [14]. Although promising, these techniques are not applicable to model mutants.

Non-determinism complicates equivalence detection both at the code [24] and model levels [25]. Patel and Hierons [24] associate predictions from pairs of inputs and outputs of the mutant program and check whether these predictions can be discarded by the original program, hence showing non-equivalence. This is not applicable to our case since our models do not have outputs. Aichernig and Jöbstl [25] also encode the semantics of the action models in terms of constraints and use refinement to check conformance in the context of non-determinism. In our case, RS/BS manage non determinism in the TSs by considering all the possible runs.

Regarding behavioural models, Aichernig *et al.* [8] developed a mutation-based test generation technique for state machines. Belli *et al.* [26] compare mutation-testing strategies when applied on event-based and state-based models, and found that both had similar effectiveness.

VI. Conclusion

We investigated the relevance of an exact language equivalence approach to tackle the equivalent mutant problem at the model level. We offered two baseline algorithms using random simulation, and compared them to language equivalence under weak and strong mutation scenarios. Our experiments demonstrated the efficiency of the exact approach for the weak mutation scenario. However, for strong mutation, our biased simulations are efficient (up to 1,000 times faster) on models that contain more than 300 states, with detection errors of 8%. Thus, our results suggest that equivalence analysis can be significantly accelerated by our simulations.

References

[1] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria," *IEEE TSE*, vol. 32, no. 8, pp. 608–624, 2006.

[2] T. T. Chekam, M. Papadakis, Y. Le Traon, and M. Harman, "An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption," in *ICSE*, 2017.

[3] M. Papadakis and N. Malevris, "Automatic mutation test case generation via dynamic symbolic execution," in *ISSRE*, 2010, pp. 121–130.

[4] G. Fraser and A. Arcuri, "Achieving scalable mutation-based generation of whole test suites," *Empirical Software Engineering*, pp. 1–30, 2014.

[5] X. Devroey, G. Perrouin, M. Papadakis, P.-Y. Schobbens, and P. Heymans, "Featured Model-based Mutation Analysis," in *ICSE*, 2016.

[6] S. Fabbri, J. C. Maldonado, T. Sugeta, and P. C. Masiero, "Mutation testing applied to validate specifications based on statecharts," in *ISSRE*, 1999, pp. 210–219.

[7] T. A. Budd and A. S. Gopal, "Program testing by specification mutation," *Computer Languages*, vol. 10, no. 1, pp. 63–73, Jan. 1985.

[8] B. K. Aichernig, J. Auer, E. Jöbstl, R. Korosec, W. Krenn, R. Schlick, and B. V. Schmidt, "Model-based mutation testing of an industrial measurement device," in *TAP*, 2014, pp. 1–19.

[9] M. Papadakis, Y. Jia, M. Harman, and Y. Le Traon, "Trivial compiler equivalence: A large scale empirical study of a simple fast and effective equivalent mutant detection technique," in *ICSE*, 2015, pp. 936–946.

[10] O. Kupferman and M. Y. Vardi, "Verification of fair transition systems," in *Computer Aided Verification*. Springer, 1996, pp. 372–382.

[11] F. Bonchi and D. Pous, "Checking NFA equivalence with bisimulations up to congruence," in *POPL*, 2013, pp. 457–468.

[12] L. Doyen and J. Raskin, "Antichain algorithms for finite automata," in *TACAS*, 2010, pp. 2–22.

[13] C. Baier and J. Katoen, *Principles of model checking*. MIT Press, 2008.

[14] S. Bardin, M. Delahaye, R. David, N. Kosmatov, M. Papadakis, Y. Le Traon, and J. Marion, "Sound and quasi-complete detection of infeasible test requirements," in *ICST*, 2015, pp. 1–10.

[15] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet, "Approximate probabilistic model checking," in *VMCAI*, 2004, pp. 73–84.

[16] A. Classen, "Modelling with FTS: a Collection of Illustrative Examples," PReCISE, University of Namur, Namur, Belgium, Tech. Rep. P-CS-TR SPLMC-00000001, 2010.

[17] S. E. Sprenkle, L. L. Pollock, and L. M. Simko, "Configuring effective navigation models and abstract test cases for web applications by analysing user behaviour," *STVR*, vol. 23, no. 6, pp. 439–464, 2013.

[18] R. Pelánek, "Properties of state spaces and their applications," *STTT*, vol. 10, no. 5, pp. 443–454, 2008.

[19] F. Bonchi and D. Pous, "HKC Library v. 1.0," https://perso.ens-lyon.fr/damien.pous/hknt/, 2013.

[20] H. L. S. Younes and R. G. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling," in *CAV*. Springer, 2002, pp. 223–235.

[21] S. M. Poulding and R. Feldt, "Generating structured test data with specific properties using nested monte-carlo search," in *GECCO*, 2014, pp. 1279–1286.

[22] N. Walkinshaw and K. Bogdanov, "Automated Comparison of State-Based Software Models in Terms of Their Language and Structure," *TOSEM*, vol. 22, no. 2, pp. 1–37, 2013.

[23] R. M. Hierons, M. Harman, and S. Danicic, "Using program slicing to assist in the detection of equivalent mutants," *STVR*, vol. 9, no. 4, pp. 233–262, 1999.

[24] K. Patel and R. M. Hierons, "Resolving the equivalent mutant problem in the presence of non-determinism and conicidental correctness," in *ICTSS*, 2016, pp. 123–138.

[25] B. K. Aichernig and E. Jobstl, "Towards symbolic model-based mutation testing: Pitfalls in expressing semantics as constraints," in *ICST*, 2012, pp. 752–757.

[26] F. Belli, C. J. Budnik, A. Hollmann, T. Tuglular, and W. E. Wong, "Model-based mutation testing - approach and case studies," *Science of Computer Programming*, vol. 120, pp. 25–48, 2016.