Faster Multi-Goal Simulation-Based Testing for CyberPhysical Systems

Xiao Ling, and Tim Menzies, *Fellow, IEEE*

Abstract—For simulation-based systems, finding a set of test cases with the least cost by exploring multiple goals is a complex task. For example, Arrieta et al. reported that state-of-the-art optimizers struggle to find minimal test suites for this task. To better manage this task, we propose DoLesS (**Do**mination with **Le**as**t** **S**quares Approximation) which uses a domination predicate to sort the space of possible goals to a small number of representative examples. Multi-objective domination then divides these examples into a "best" set and the remaining "rest" set. After that, DoLesS applies an inverted least squares approximation approach to find a minimal set of tests that can distinguish best from rest in the reduced example space.
DoLesS has been tested on six simulation-based systems: a tank flow model; a model of electric car windows; a safety feature of an AC engine; and a continuous PID controller combined with a discrete state machine; a cruise controller model; and a simple case study Tiny. Comparing to the recent state-of-the-art paper attempted the same task, DoLesS performs as well or even better as state-of-the-art, while running 80-360 times faster on average (seconds instead of hours). Hence, we recommend DoLesS as a fast method to find minimal test suites for multi-goal simulation-based systems. For replication purposes, all our code is on-line: https://github.com/ai-se/DoLesS.

Index Terms—Search-based Software Engineering, Modeling and Model-Driven Engineering, Validation and Verification, Software Testing, Simulation-based Testing, Multi-goal Optimization

1 Introduction

As software engineering evolves, software engineers can take advantage of new capabilities offered by different software development processes. In this paper we explore how software testing can be improved when the development process includes the development of a high-fidelity simulator of the system being constructed, and its surrounding environment.

Simulation models play an important role in many software engineering domains. Engineers build such models to simulate complex systems [1]. In the case of cyber-physical systems, these models are sometimes shipped along with the actual device, which means that analysts can now access high-fidelity simulations of their systems. Hence, much of the work on cyber-physical testing focuses on taking full advantage of high-fidelity simulators, prior to live testing [2]. For example, analysts can use the simulators for *test suite minimization*; i.e. they can explore many tests in the simulator in order to remove tests that do not need to be explored in the real world.

Using these models for test case minimization can be a surprisingly difficult process [2]. There are two key factors in test case minimization for simulation models: *simulation runtime* and *algorithm runtime*.

Simulation runtime indicates the total time to simulate test cases for the necessary information (e.g. coverage, test execution history, and effectiveness measurements used in this study). When simulation models explore high fidelity systems (such as electronic models, physical models, AV, and drone simulation models), they can be slow to execute [3]. For example, previous literature reported that testing a high fidelity simulation model can take hours to days [2], [4], [5]. This factor mostly is *not decreaseable* because all test case minimization algorithms require test case information from simulation, and such simulation time is not changeable.

On the other hand, *algorithm runtime* indicates the time for test case minimization algorithm. This factor *can* be minimized by finding a fast approach (e.g. in our study, we find DoLesS runs significantly faster to find minimum test suite). In this study, we try to reduce the difficulty of test case minimization by finding minimum test suite in a very short feedback loop (i.e. fast algorithm runtime).

How can we simplify the complex problem of multi-goal testing for domains with complex (and possibly very slow) simulation models? Recently, Chen [6] and Agrawal et al. [7] reported successes with a variant of optimization called "DUO" (data mining using/used-by optimizers). In this approach, a data mining method firstly divides the variable space, and then an optimizer executes in each small division. Inspired by that DUO approach, in this work, we apply a sorting method on the objective space to divide the multi-objective test suite minimization problem into several smaller partitions. Our DoLesS algorithm (**Do**mination with **Le**as**t** **S**quares Approximation) applies a domination predicate to sort the objective space to a small number of representative data points (note that representative data point means the data point on the top group after the sorting predicate). More specifically, multi-objective domination divides these data points into a "best" set and the remaining "rest" set. After all that, DoLesS applies an inverted least squares approach to learn a minimal set of tests that can distinguish the best from the rest in the reduced objective

• *X. Ling, and T. Menzies are with the Department of Computer Science, North Carolina State University, Raleigh, USA. E-mail: lingxiaohzsz3ban@gmail.com, timm@ieee.org*

space.

To evaluate our proposed test case selection approach for simulation models, we compare DoLesS with the most recent state-of-the-art approach produced by Arrieta et al. [2]. In that comparison, we ask the following research questions.

RQ1: Can we verify that test case selection for multi-goal cyber-physical systems is a hard problem? Here "hard problem " means test case selection for multi-goal cyber-physical systems cannot be solved directly by applying off-the-shelf multi-goal optimizers such as NSGA-II, NSGA-III, and MOEA/D on all goals. The standard solution to this kind of problems is to apply multi-objective optimizers like NSGA-II, NSGA-III, and MOEA/D [2], [8], [9]. Arrieta et al. [2] reported that these standard approaches do not perform well in minimizing the test suite for cyber-physical systems. In fact, they only found success after a somewhat convoluted approach requiring the execution of NSGA-II multiple times (which, as we show below can be up to 360 slower than necessary). The Arrieta et al. experience (which we replicated, below) serves to motivate this work.

RQ2: Do selected test cases by DoLesS improve the results from prior state-of-the-art? Apart from the five effectiveness measurement metrics used by Arrieta et al. [2], two other evaluation scores of interest are (a) reduction in the number of test cases and (b) faults detection performance with the reduced test cases. As shown in our result section §5, DoLesS usually performs as well, if not better, than the prior state-of-the-art.

RQ3: Is DoLesS far more efficient than prior state-of-the-art in terms of running time? For all the reasons stated above, we need methods that offer faster feedback from models of cyber-physical systems. In this regard, it is significant to note that DoLesS runs 80-360 times faster than the prior state-of-the-art.

Based on the above, we say our novel contributions are:

1) We propose a novel test generation method (DoLesS).
2) We verify that DoLesS solves a hard problem (test case selection for multi-goal cyber-physical systems). This problem cannot be solved just by applying standard optimizers (NSGA-II, NSGA-III and MOEA/D) in off-the-shelf manner. Due to the many-goal nature of the problem, we need somehow to extend our optimizing technology[1].
3) We clearly document the value of doing DoLesS. When testing on six cyber-physical models, DoLesS finds test suites as good, or even better, than those found by Arrieta et al.'s approach [2]. Further, DoLesS does so while running 80-360 times faster (seconds instead of hours, mean time). Hence, we recommend DoLesS as a fast method to find minimal test cases for multi-goal cyber-physical systems.

The rest of this paper is structured as follows. Section 2 introduces the background and related work in test case selection for simulation-based testing. Section 3 introduces the problem of studying effectiveness measurement metrics in cyber-physical systems and illustrates how they are calculated by mathematical formula. Moreover, multi-objective optimizers and our proposed approach are introduced in this section as well. Section 4 introduces the case studies, performance evaluation metrics, and statistical analysis method used in this study. Section 5 shows our experimental results. Section 7 explores threats to validity and Section 8 makes the summary of our study and states the possible future work.

Based on the above, we can conclude that DoLesS is faster, yet more effective, than prior results since:

- DoLesS can handle multiple goals (in our experiment, 5 goals) simultaneously. Hence it does not need to loop the algorithm n times (where n is the number of subsets for the goals) like the prior state-of-the-art method.
- DoLesS's sorting procedure uses continuous domination to very quickly divide candidates into a very small "best" set (that we can focus on) and a much larger "rest" (that we can mostly ignore). Like much research before us, we argue that continuous domination is more informative than binary domination [11], [12], [13].
- Chen et al. [6] argues that some SE optimization problems can be solved better by *over-sampling* than via *evolutionary* methods. For example, the *evolutionary NSGA-II method* mutates 100 individuals for 250 generations (these parameters were selected to ensure comparability to the prior study). On the other hand, our *DoLesS over-sampling method* explores 10,000 individuals for one generation. This result suggests that cyber-physical system testing might be another class of problem that better to be solved via the over-sampling methods which stated by Chen et al [6].

2 Background

A repeated result is that test suites can be *minimized* (i.e. we can run fewer tests) while still being as *effective* (or better) than running the larger test suite [14], [15], [16], [17], [18]. Note that "effective" can mean different things in different domains, depending on the goals of the testing. For example, at FSE'14, Elbaum et al. [19] reported that Google could find similar number of bugs, but after far fewer tests execution. This was an important result since, at that time, the initial Google test suites were taking weeks to execute. Such long test suite runtimes is detrimental to many agile software practices.

Test case selection for traditional software testing: Research has found many test case selection techniques such as DejaVu based, firewall based, dependency based, and specification based techniques [20]. We note that different test suite minimization methods need different kinds of data. For example, in 1995, Binkley et al. proposed a semantic-based method which takes the use of differences and similarities of two consecutive versions to select test cases [21]. Rothermel et al. developed a test case selection technique for C++ software on 2020 [22]. In 2001, Chen et al. developed test case selection strategies based on the boolean specifications [23]. In 2005, a fuzzy expert system was developed in test case selection by Xu et al. [24]. In 2006, Grindal et al. presented an empirical study on evaluating five combination strategies for test case selection [25]. In 2007 to 2015, multi-objective

1. In the literature, "many-goal" usually refers to 2 or 3 goals and "multi-goal" refers to four or more. This is an important distinction since (a) our problem is a five goals task and (b) standard optimizes like NSGA-II are known to have issues with multi-goal [10].

search genetic algorithms got more attention from software testing researcher. For example, In 2014, Panichella et al. introduced MOGA which increases diversity by injecting new orthogonal individuals during test selection search process based on the mechanisms of orthogonal design and orthogonal evolution [26]. In 2015, Mondal et al. [27] firstly compared "test case diversity" and "code coverage" in several real-world case studies and then proposed new test selection technique based on NSGA-II which maximize both two criteria. Also, in 2015, Souza et al. [28] proposed two MOO algorithms and present the empirical evaluation of their performances in test selection compared to NSGA-II and MBHS. In 2017, Devroey et al. [29] formulated the test case selection problem in product line from a Feature Transition System (FTS) as an optimization problem which maximizes the coverage measure in FTS while minimizes/maximizes the number of products needed to execute all test cases. Also in 2017, an extensive study was made by Devroey et al. [30] which diverse a distance function between actions of FTS and products on which the test case may be executed.

Test case selection for model-based software: Another large group of model in software testing is the simulation-based software. In 2010, Hemmati et al. [31] explores multiple similarity measurements to support the similarity testing in State Machine models. Moreover, one year later, Cartaxo et al. implemented a similarity function for test case selection in model-based testing [32]. Pradhan et al. [33] proposed a multi-objective optimization test case selection approach which can be used with limited time constraints. Arrieta et al. also used test case execution history to select test cases [3]. Same in 2016, Arrieta et al. also conducted a study on test case selection of cyber-physical product line where their proposed method can be adopted to "X-in-the-Loop" test level [34]. In 2017, Lachmann et al. [35] did an empirical study on several black-box metrics and made comparisons on their performance in selecting test cases in system testing. On 2020, Fremont et al. conducted a test selection study which they presented a new approach to automated scenario-based testing of the safety of autonomous vehicles [36].

Due to the obtainable data in simulation models, many of the above methods are unsuitable for cyber-physical systems, for two reasons. *Firstly*, cyber-physical systems are embodied in their environment. Hence, it is not enough to explore static features of (e.g.) the code base. Rather, it is required to test how that code base reacts to its surrounding environments. Hence, using just static information such as (e.g.) code coverage metrics is not recommended for testing cyber-physical systems.

Secondly, at least for the systems studied here, cyber-physical systems make extensive use of process control theory. In that theory, the feedback controller is used to compare the value or status of process variables with the desired set-point. This controller then applies the difference as a control signal to bring the process variable output of the plant to the same value as the set-point. Hence, for test suite minimization of process control applications, the requirement is data collected from the feedback loops inside the cyber-physical systems. Accordingly, here we use input and output signals in the simulation models instead of execution history or coverage information.

In one of the IST'19 journal paper, Arrieta et al. [2] explored issues associated with test suite minimization by using the data extracted from feedback loops. They noted that feedback loops have anti-patterns; i.e. undesirable features that appear in a time series trace of the output of the system. The reason why anti-patterns are important factors to be considered in test minimization for simulation models is because a faulty behavior in simulation model may not cause the execution interruption as traditional software does. For example, if a correct $+$ operator is mistakenly coded to $-$, the whole system can still run successfully. For another example, the model can still run with no error if the **OR** logic operator is replaced to **AND**. Thus, traditional scenarios such as execution failing history has very limited impact to test case minimization in simulation models. In such situation, anti-patterns are much more important since they can directly reveal undesirable behaviors from the output signals even the model is executed successfully. Figure 1 shows three such features correspondingly from left to right in the red dash rectangle: *instability*, *discontinuity*, and *growth to infinity*. Later in this paper we will mathematically define these anti-patterns.

In all, Arrieta et al. [2] explored seven goals for cyber-physical model testing: maximizing the three anti-patterns that shown in Figure 1, maximizing three other measures of effectiveness, as well as minimizing total execution time. Arrieta et al. used *mutation testing* to check the validity of their minimized test suite. Mutation based testing is a fault-based testing technique which implements "mutation adequacy score" to assess test suite adequacy by creating

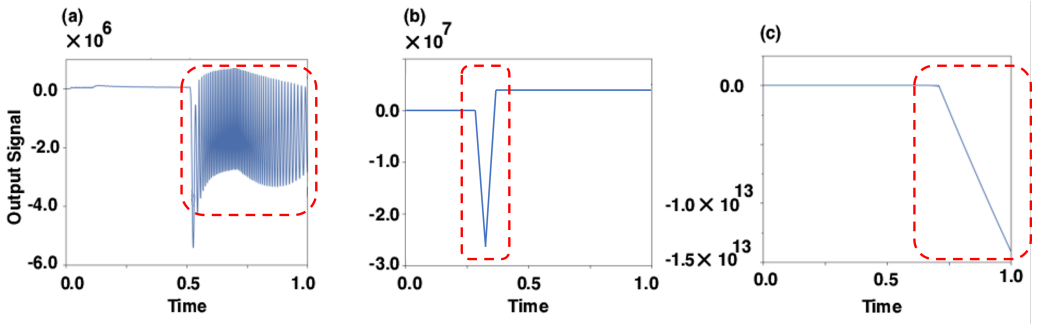


Fig. 1: Examples of anti-patterns seen for systems under feedback. The name of anti-patterns from left to right are *instability*, *discontinuity*, and *growth to negative infinity* correspondingly. The alarming patterns are shown in red marks.

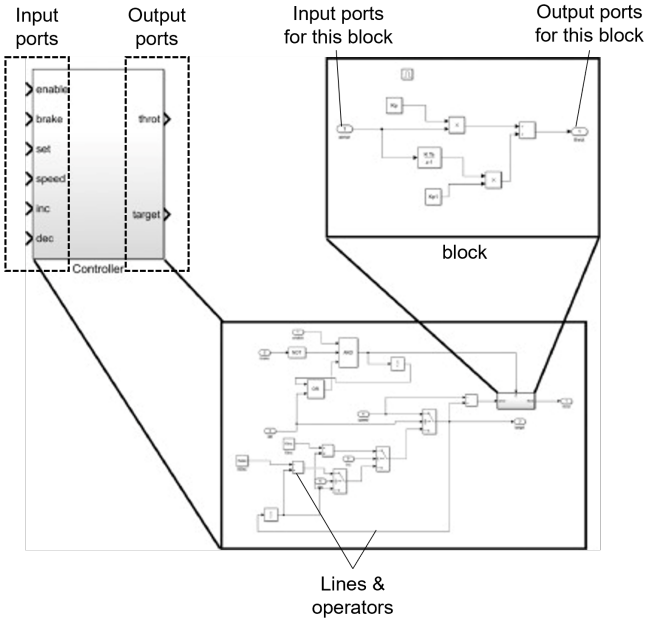


Fig. 2: Simple example of a Simulink model - Cruise Controller of a car [2].

mutants [37], and then pruning test cases which cannot distinguish the original model from the mutant. In our study, we use the mutants generated in the experiments from Arrieta et al. [2]. Those mutants were generated with Hanh et al.'s technique [38] and some of the mutants are removed if (a) they are not detected by any test case, (b) they are killed by all test cases, and (c) they are equivalent mutants [39]. Like Arrieta et al., we say a test suite is *minimal* when it retires as many mutants as a larger suite.

Mutation testing is the inner loop of Arrieta et al.'s process and, in their experiments, they found mutation testing to be an effective technique. The problem area in their work was the outer loop that optimized for seven goals. They found that standard optimizers such as NSGA-II [40] can be ineffective for more than three goals (a result that is echoed by prior work [12]). More recent optimizers like NSGA-III [41] and MOEA/D [42] also failed for this multi-goal task. Later in this paper, we replicate their experiment and strengthen that finding (see RQ1).

To address this optimization failure, they resorted to "pairwise" approach based on NSGA-II. That is, they ran NSGA-II with all 21 subsets of "choose two or three from seven" goals then returned the test suite associated with the run which has the best scores (where the "best" here is measured just on a subset of goals). While their work is definitely an extension of state-of-the-art, their study has two drawbacks. Firstly, the test cases selected in this way is only the best which measured on a subset of the optimization goals. Secondly, the "pairwise" approach increases optimization time by an order of magnitude, which is a major issue for large simulators, especially when we are running these algorithms 20 times (to check the generalizability of this stochastic process).

Hence in this work, we seek to improve the mutation based test suite minimization method from Arrieta et al.'s study [2]. Like them, we will optimize for the anti-patterns and effectiveness measures seen in process control systems. But unlike that prior work, we will offer methods that simultaneously succeed across many goals (without needing anything like the pairwise heuristic used in Arrieta et al.). Further, we show that all this can be achieved without additional runtime cost.

2.1 Testing Simulation Models

Cyber-physical system developers often use simulation tool (e.g. Simulink) to build cyber-physical models [43]. For an example Simulink models, see Figure 2. This is a model with two hierarchical levels [2]. A complex model will have far more blocks and operators.

In Simulink models, the inputs and outputs are all signals (here signal means a time series function). This means at each simulation time step ΔT, there will be a value in each input and in each output regarding to that time step. For example, if we simulate a model for 5 seconds in real time and the time step ΔT is 0.05, then there will be $5/0.05 + 1 = 101$ simulation steps, which means each input or output should be a vector of length 101.

Since the inputs of simulation models are vectors of time series, a test case t of the simulation model will also be n time series vectors $t = \{t_{i_1}, t_{i_2}, \cdots, t_{i_n}\}$ where n is the number of inputs. Assuming an initial set of n test cases $\{t_1, \cdots, t_n\}$ for a simulation, each test simulates the model from a set of unique k input signals $\{is_1, \cdots, is_k\}$ to a set of l output signals $\{os_1, \cdots, os_l\}$ [1], [2].

Our goal for this study is to select representative test cases from the initial test suite to minimize the test execution time, but not influence the testing performance. Here we can define the test case selection problem as follow:

> Given an initial test suite T which includes n test cases $T = \{t_1, t_2, t_3, \cdots, t_n\}$ and an evaluation function f which can evaluate the fault detection ability of a test suite, the goal of the test case selection is to find a new test suite $TS = \{ts_1, ts_2, \cdots, ts_a\}$ such as $TS \subseteq T$ and $f(TS) = f(T)$.

If we search in the space which contains all the subsets of the initial test suite, then the search space will be very large. For example, with only 100 test cases, there will be $2^{100} - 1$ possible subsets. Thus, cost-effectively selecting test cases is a significant problem.

2.2 Mutation Testing in Simulation Models

One persistent problem in SE testing is the "oracle problem"; i.e where can we get the expertise that can judge if a test suite is "good enough". The insight offered by mutation testing [44], [45], [46], [47], [37] is that it is possible to automate the generation of a test oracle as follows:

- Suppose we have a program P and a test suite T.
- That program P can be "mutated" to generate a syntactically valid piece of code, with some changes (e.g. swap a minus sign with a plus or any other other mutations listed in Table 1).
- The least we should expect from T is that it can recognize when P's behavior change (since otherwise, the tests are blind to operational changes in the code).

TABLE 1: Simulation fault patterns collected from literature [49], [50], [51], [52], and also listed in [48].

Pattern	Illustration
Change of constant values	Change constant value c to some other values if it is numeric. If it is boolean, negating it.
Change of arithmetic operators	Switch +/- or replace + to $\times$.
Change of relation operator	Switch $\geq$ to $<$ or $\leq$, and switch $\leq$ to $>$ or $\geq$.
Change of logical operator	Switch **AND**, **OR** and **XOR**; Remove **NOT** or add **NOT**.
Incorrect Connection	Switch the input lines of the "Switch" block.
Incorrect Signal Data Type	Switch the "double" data type to "single", or switch "fixdt(0,8,3)" data type to "fixdt(0,8,2)".
Wrong Initial conditions and delay values	Change the initial value in "Integration" and "Unit Delay" blocks.

- Given a simulation program P mutated to P', we say that a test suite is "good enough" if measurements in the output signal taken from P and P' are different[2].

In traditional software engineering projects, mutants are systematically seeded by changing a piece of code from the original project. However, mutants are quite different in the simulation models. As Figure 2 shows, simulation models are combined with multiple blocks, lines, and operators instead of pure lines of code. Therefore, mutating simulation models (e.g. Simulink projects) requires certain fault patterns [48]. Using a literature review, we found numerous typical fault patterns for the simulation models (see Table 1). As a sanity check, we manually inspect the mutants generated outputs from Arrieta et al. [2] before running our experiments to check if the generated mutants follow these fault patterns. The inspection result indicates that all observed mutants are satisfy one of the fault patterns shown in Table 1.

3 Experimental Methods

3.1 Simulation Effectiveness Metrics

In this study, we implement five out of seven effectiveness measurement metrics which Arrieta et al. [2] used in their study. The first three metrics are also widely used in previous studies [53], [54], [55], and the forth metric is proposed by Arrieta et al. [2]. Moreover, since the values in every metric differ a lot, we normalized all metrics before performing search on them to avoid the threat from different scales.

Aside: We exclude two of the metrics explored by Arrieta et al. (*Input & Output-based test similarity metrics*) since these two metrics will always result similar normalized values (0.95-0.99) with different test case selections. Such similar normalized values can affect the performance of multi-objective optimization algorithms. We will introduce the remaining five metrics in the rest of this section.

2. Our pre-experimental expectation was that statistical methods would be required to detect nuanced changes between P and P'. But, referring to Figure 1, the faults induced but our mutants tend towards very large changes in those measurements. Hence, just computing residuals sufficed for this recognizing differences.

Please note that in all following 5 metrics, T represents the initial test suite such that $T = \{t_1, t_2, \cdots, t_n\}$ and n represents the number of tests in the initial test suite. If test case i is selected, then $Select_{t_i} = 1$. Otherwise, $Select_{t_i} = 0$. Moreover, TS in the following formulation means the new test suite after the selection process.

3.1.1 Test Execution Time

Total test execution time is the first metric we implement in our study. Wang et al. [53] stated that the number of selected test cases can be treated as the measurement for selecting representative test cases from the initial test suite. However, in simulation models, different test case may has different execution time. Thus, we cannot directly count the number of selected test cases in our experiment. To mitigate that, we adopt the idea from Arrieta et al. [2] that using the overall execution time of the selected test cases as the search guidance.

To say more specifically, let tet_i be the the normalized test execution time for the test case i, the overall test execution time for a test suite can be calculated as follow

$$executionTime(TS) = \sum_{i=1}^{n} Select_{t_i} * tet_i \quad (1)$$

In our study, we want to *minimize* this metric because the goal of test case selection is to decrease the test execution time.

3.1.2 Discontinuity in Output Signal

Discontinuity is the second metric we implement in our study. As Matinnejad et al. [55] stated, the discontinuity of the output signal is a short duration pulse in the output signal, which means the output signal increases or decreases to a value in a very short time, and recovers back to normal. If executing a test case causes discontinuity in the output signal, then that test case detects the faulty behavior in the model. To be more specifically, the discontinuity score $discontinuity(O_j)$ of an output signal j can be calculated as follow:

$$discontinuity(O_j) = \max_{dt=1}^{3}(\max_{i=dt}^{k-dt}(min(lc_i, rc_i))) \quad (2)$$

where

- $lc_i = |sig(i \cdot \Delta t) - sig((i-dt) \cdot \Delta t)|/\Delta t$ is the left change rate of step i. Δt is the time stamp.
- $rc_i = |sig((i+dt) \cdot \Delta t) - sig(i \cdot \Delta t)|/\Delta t$ is the right change rate of step i.

The discontinuity score of a test case will be the sum of the discontinuity score for each output signal. Assume the normalized discontinuity score for the test case i is dc_i, the overall discontinuity score for a test suite can be calculated as follow

$$discontinuity(TS) = \sum_{i=1}^{n} Select_{t_i} * dc_i \quad (3)$$

In our study, we want to *maximize* this metric because the goal is to detect more discontinuity in the output signal.

3.1.3 Instability in Output Signal

Instability is the third metric we implement in our study. As Matinnejad et al. [55] stated, the instability of the output signal is a duration of quick and frequent oscillations in the output signal, which means the output signal increase and decrease repeatedly in a duration of time. If executing a test case causes instability in the output signal, then that test case detects the undesirable impact on physical process [55]. To be more specifically, the instability score $instability(O_j)$ of an output signal j can be calculated as follow:

$$instability(O_j) = \sum_{i=1}^{k} |sig(i \cdot \Delta t) - sig((i-1) \cdot \Delta t)| \quad (4)$$

where k is the total number of simulation steps and Δt is the time stamp in the simulation model for each step.

Similar to the discontinuity metric, the instability score of a test case will be the sum of the instability score for each output signal. The overall instability score will be calculated similar to the Equation 3. For the space reason, we are not listing it again here.

In our study, we want to *maximize* this metric because the goal is to detect more instability in the output signal.

3.1.4 Growth to Infinity in Output Signal

This is the forth metric we implemented in our study. As Matinnejad et al. [54] pointed out, the growth to infinity of the output signal is the phenomenon that the output signal increases or decreases to infinity value. If executing a test case causes growth to infinity in the output signal, then that test case detects the faulty behavior in the model. To be more specifically, the growth to infinity score $infinity(O_j)$ of an output signal j can be calculated as follow:

$$infinity(O_j) = \max_{i=1}^{k} |sig(i \cdot \Delta t)| \quad (5)$$

where k is the total number of simulation steps and Δt is the time stamp in the simulation model for each step.

Same as above, the infinity score of a test case will be the sum of the infinity score for each output signal. Moreover, the overall infinity score of a test suite will be the sum of infinity score of selected test cases (e.g. Equation 3).

In our study, we want to *maximize* this metric because the goal is to detect more growth to infinity situation in the output signal.

3.1.5 Output Minimum and Maximum Difference in Output Signal

This is the last metric we implemented in our study. Arrieta et al. proposed this metric in their work because the difference between maximum output signal and minimum output signal can indicate the level of how a model is being tested. If executing a test case results in large minimum and maximum difference in the output signal, then that test case can detect more parts in the simulation model. To be more specifically, the minmax score $minmax(O_j)$ of an output signal j can be calculated as follow:

$$minmax(O_j) = |\max_{i=1}^{k}(sig(i \cdot \Delta t)) - \min_{i=1}^{k}(sig(i \cdot \Delta t))| \quad (6)$$

where k is the total number of simulation steps and Δt is the time stamp in the simulation model for each step.

Same to the above metrics, the minmax score of a certain test case will be the sum of minmax scores for all output signals. Moreover, the overall minmax score for a test suite will be calculated like Equation 3.

In our study, we want to *maximize* this metric because the goal is to coverage more parts that can be tested.

3.2 Algorithms

3.2.1 Binary vs Continuous Domination

In the following, all the algorithms use *binary domination* except for `DoLesS` that uses *continuous domination*.

Binary domination say one individual is better than another if it is better on at least one goal and worse on none. Many studies [11], [12], [13] warn that binary domination struggles to distinguish candidates for three or more goals.

For many-goal problems, Zilter's *continuous domination* predicate [11] is useful [11], [12], [13]. Continuous domination judges the domination status of pair of individuals by running a "what-if" query which checks the situation when we jump from one individual to another, and back again. Specifically:

- For the forward jump, we compute $s_1 = -\sum_i e^{w_i * (a_i - b_i)/n}$.
- For the reverse jump we compute $s_2 = -\sum_i e^{w_i * (b_i - a_i)/n}$.

where a_i and b_i are the values on the same index from two individuals, n is the number of goals (in our case $n = 5$), and w_i is the weight {-1,1} if we are minimizing or maximizing the goal i correspondingly. According to Zitler [11], one example is preferred to another if we lost the least jumping to it; i.e. $s_1 < s_2$.

Specifically, in this work, we use this predicate to select better goal sets that (a) **minimize** test execution time, (b) **maximize** discontinuity score, (c) **maximize** instability score, (d) **maximize** growth to infinity score, and (e) **maximize** output minimum & maximum difference.

3.2.2 NSGA-II

NSGA-II is a common evolutionary genetic algorithm [40]. Firstly, it generates an initial set of population as the starter of the entire algorithm. Secondly, these candidates will evolve to offsprings in a series of generations by implementing the crossover and mutation operators with their individual probability. In our reproduction experiment, we use single point crossover with 0.8 probability and bit-flip mutation with $1/N$ probability (N is the number of test cases). Thirdly, parents for next generation will be selected by selection operator, which utilizes a non-dominated sorting algorithm to select top non-dominated solutions [10]. In the situation where a front needs to be divided because it exceeds the total number of population, NSGA-II uses the crowding distance to split candidates in that group.

3.2.3 NSGA-III

NSGA-III is an improved NSGA-II algorithm [41]. In NSGA-III, all procedures such as initial population generation, crossover, and mutation are similar to NSGA-II, except selection procedure. In NSGA-III, the selection procedure is applied based on a set of reference points. The reference

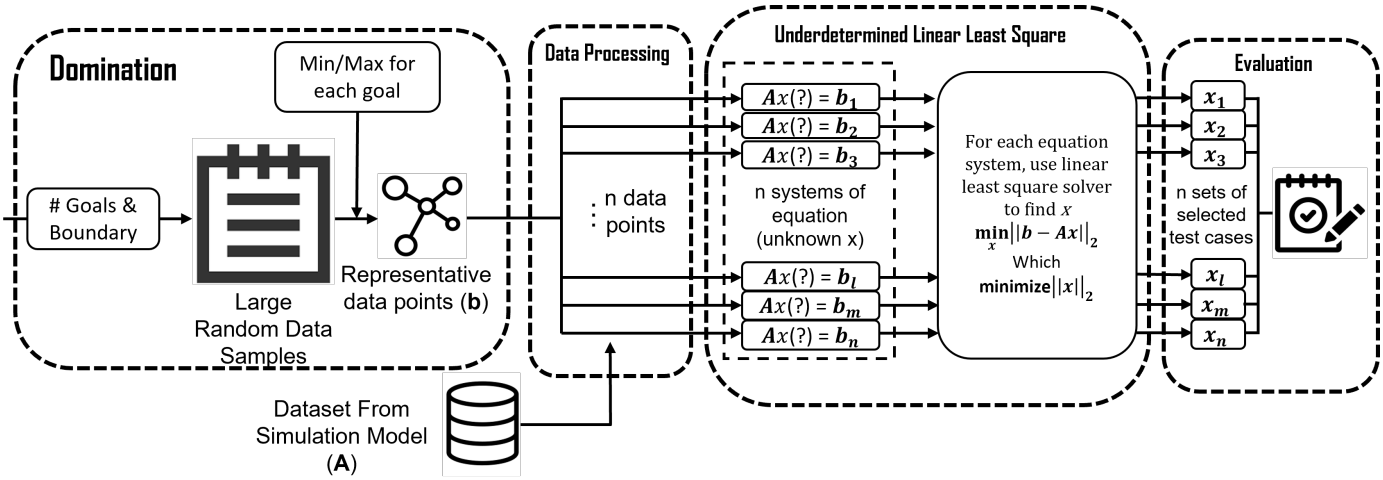


Fig. 3: DoLesS Framework - Continuous domination selects representative goals from the large initial random population; Data processing reads the data set in and forms the linear equation system; Linear least square solver solves the least square approximation; and Evaluation evaluates the selected test cases.

points are uniformly distributed on the normalized hyperplane with some division number p [41]. After that, each objective point is normalized adaptively and associated with a reference point by calculating its distance to the corresponding reference line. Niche-Preservation operation is then applied to select candidates which will be used in the next generation [41].

3.2.4 MOEA/D

MOEA/D [42] decomposes the problem into multiple subproblems with less objectives in each subgroup and solves these sub-problems simultaneously [42]. To do so, prior to inference, all examples get random weights assigned to their goals. Examples are then clustered by those weights such that all examples know the space of other examples that weighted in a similar direction. Next, during the execution, if one example X_0 finds a way to improve itself, its local neighborhood will move in the same direction as X_0.

3.2.5 DoLesS

Figure 3 shows the entire framework of our approach. Unlike above evolutionary algorithms, our proposed approach DoLesS (**Do**mination with **Le**as**t** **S**quares Approximation):

- Uses *continuous domination* (defined below, see the first block in Figure 3) to reduce the size of initial large random sets of goals and find a "best" group of representative samples.
- For each data entry in the "best" group, DoLesS then uses a *least square approximation technique* (the third block in Figure 3) to inversely predict the test selection outcomes which can fit the representative sets of goals best. This least square approximation is discussed below.

After sorting via domination, DoLesS divides data into:

- The $\sqrt{n}$ "best" items. We randomly generate 10000 initial candidates so the first 100 candidates with highest domination score are grouped the "best" group.
- And the remaining "rest" items.

Here we set final population to 100 since: (a) 10000 random initial population is large enough to cover a wide range of possible outcomes and (b) to make comparison fair, we select same number of final candidates as previous work [2].

In the *data processing* stage of Figure 3, we take data from each model (which Arrieta et al. also used in their study [2]), and then process it into the form of least square approximation structure by combining with the representative goals generated from continuous domination. Table 2(i) shows a simple example of effectiveness measurement data collected from the models. We can find that each test case will have a single score for all effectiveness measurement data (a_{ij} means the score of test case i in effectiveness measure j). The corresponding matrix equation system for the above example is shown in Table 2(ii). This equation shows the linear relationship of test selection outcomes and the final effectiveness measure scores (e.g. the final score of effectiveness measure 1 can be obtained by $em_1 = a_{11} \cdot t_1 + a_{12} \cdot t_2 + a_{13} \cdot t_3 + a_{14} \cdot t_4$ where t_i is the outcome of test selection). In this example, our goal is to find the best outcomes of t_1 to t_4 which can result em_1 to em_5. To summarize the above example, in our approach, we collect effectiveness measurement data for n test cases (like Table 2(i)) and want to find the best set of outcomes for t_1 to t_n which can get the closest scores to representative goals which are selected by continuous domination.

Linear Least Squares Approximation is a mathematical method for estimating the true value of variables in the variable space based on a consideration of errors in measurements. In *"Interactive Linear Algebra"* by Dan Margalit and Joseph Rabinoff [56], the best approximation solution to an inconsistent matrix equation $Ax = b$ is defined as follow:

Definition: Let A be an $m \times n$ coefficient matrix and let b be a vector of objective values in $\mathbb{R}^m$. A least square solution of the matrix equation $Ax = b$ is a vector $\hat{x}$ in $\mathbb{R}^n$ such that

$$dist(b, A\hat{x}) \leq dist(b, Ax) \tag{7}$$

for all other vector x in $\mathbb{R}^n$.

TABLE 2: An example of (i) collected effectiveness measurement data (EM means effectiveness measure) and (ii) its corresponding matrix equation form

	t_1	t_2	t_3	t_4
EM 1	a_{11}	a_{12}	a_{13}	a_{14}
EM 2	a_{21}	a_{22}	a_{23}	a_{24}
EM 3	a_{31}	a_{32}	a_{33}	a_{34}
EM 4	a_{41}	a_{42}	a_{43}	a_{44}
EM 5	a_{51}	a_{52}	a_{53}	a_{54}

(i)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{51} & a_{52} & a_{53} & a_{54} \end{bmatrix} \cdot \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = \begin{bmatrix} em_1 \\ em_2 \\ em_3 \\ em_4 \\ em_5 \end{bmatrix}$$

(ii)

Here the distance function is the 2-norm of the vector. Then we can rephrase the linear least square approximation problem to the mathematical formula by following the above definition:

$$\min ||Ax - b||_2^2 + ||x||_2^2 \tag{8}$$

As we can see, $||Ax - b||_2$ is the distance from the vector Ax to the vector b. To avoid the negative value, we take the power two of the distance value and the ultimate goal is to find the minimum distance between the vector Ax and the vector b. In some cases, there will have multiple possible optimal solutions for the problem. Thus, we add a 2-norm of variable vector x so that the variables in x are as small as possible.

Next we will explain how test case minimization problem with effectiveness measurements can be formulated to the above linear least square problem. First of all, as we presented in Section §3.1, the overall score of each effectiveness measurement is the sum of each individual effectiveness measurement score for selected test cases. As shown in Table 2(i), each test case T_j has its own effectiveness measurement score a_{ij} where i represents the ith effectiveness measurement. The overall score of ith effectiveness measurement will be calculated by $\sum_{j=1}^{4} a_{ij} \times t_j$ where t_j will be the binary variable which represents whether test case j is selected or not. Overall there will have i equations since the number of effectiveness measurements is i (In our study, i = 5). The overall equation systems can be rewritten to the inconsistent matrix equation $Ax = b$ where A has elements a_{ij}, x is the vector which contains tests minimization outcome, and b represents the final scores of i effectiveness measurements (The matrix representation is shown in Table 2). At this point, the test case minimization problem is reshaped to the linear least squares approximation problem and the optimal test minimization outcome will be the x which minimize $min||Ax - b||_2^2$.

However, there is an issue that since there are more tests than the number of effectiveness measurements, solving the linear least square problem can result multiple possible solutions. To find the best one among these possible solutions, we add a term $||x||_2^2$ after the least square formulation which minimizes the overall elements in the vector x. Please note that the test case minimization problem is trying to find *minimal* tests, so the extra term can make sure the final optimal solution has minimal tests in it.

TABLE 3: Summary of number of I/O signals, number of test cases, and number of mutants in six case studies, due to space situation, in the below table, TT means "Two Tanks" project and ACE means "AC Engine" project.

Project	TT	CW	ACE	EMB	CC	Tiny
# Inputs	11	15	4	1	5	3
# Outputs	7	4	1	1	2	1
# Test Case	150	133	120	150	150	150
# Mutants	6	96	12	18	20	12

In the formulation $Ax = b$, where x is an outcome vector of test minimization, we want to predict the value (0/1) for each entry of x. In our linear least square formulation, the final outcome of x is a vector of float numbers (ranged from 0 to 1 by controller) which indicates lower effect to the final score with coefficient $\rightarrow$ higher effect to the final score with coefficient from $0 \rightarrow 1$. Since test selection outcomes can only have 0 (discard that test) and 1 (select that test), we use the threshold of 0.5 to indicate higher probability or lower probability. A value < 0.5 means higher chance to be 0 and a value > 0.5 means higher chance to be 1. For each representative candidate found by continuous domination, `DoLesS` finds the test selection which can get the closest score to that candidate. Although there exists delta between original ideal scores and truth scores generated by optimized results because of the approximation procedure, our results show that least square approximation can find adequate test cases which perform as well or better as the previous state-of-the-art. Our implementation of above step uses **scipy.optimize**, a python library, and uses the function called **lsq_linear**, which solves the above problem by using either dense QR decomposition technique or Singular Value Decomposition technique.

Finally, in the *Evaluation* stage of Figure 3, `DoLesS` selects the Pareto front set[3] from the final population as Arrieta et al. did in their study [2]. All evaluations are made through 20 repeats for each of algorithm.

4 Experimental Setup

4.1 Case Studies

One challenge with research in this area is that public domain case studies are hard to find (which this has implications for reproducibility of results). After scouring the literature on cyber-physical testing we find that most of the studies utilized 2-4 models for their evaluation. After collecting all the public models we could find, we find six public Cyber-physical models to evaluate our proposed approach.

For each of those models, we implemented the test cases and mutants generated by Arrieta et al.'s study[4]. The summary of number of initial test cases and number of mutants are shown in Table 3. In that table: Two Tanks project is a model that simulate the incoming and outgoing flows of the tanks [57]; CW project is a model that simulate the electrics and mechanics of four car windows [2]; AC Engine project is

3. In multi-objective optimization, the Pareto front of a set of individuals are all examples not dominated by anything else.

4. https://github.com/aitorarrietamarcos/IST2019Paper

a model that simulate some safety functionalities in the AC engine [58], [59]; EMB project simulates the software model controller which includes a continuous PID controller and a discrete state machine [55]; CC is a cruise controller system and Tiny is a simple physical model [2].

At first glance, the case studies in Table 3 may appear to contain very small test cases. But appearances can be deceiving; e.g. the number of input signals is a poor measure of the internal complexity of a cyber-physical system. As shown in our **RQ1** results, the systems of Table 3 are so complex that, for the purposes of test suite minimization, they defeated state-of-the-art optimizers (NSGA-III and MOEA/D).

4.2 Performance Criteria

To evaluate the selected test cases, we use two evaluation metrics from prior work [2]. These two evaluation metrics are (a) normalized test execution time and (b) mutant detection score. Previous study used these two metrics to calculate the hypervolumne indicator and average weighted sum of mutation score and normalized test execution time [2], while in our study, we directly compare the performance of algorithms in these two metrics.

Normalized test execution time (TET-): Our goal for selecting test cases from the initial test suite is to speed up the testing process. Thus, test execution time is a very important indicator which can indicate whether selected test cases can significantly reduce the cost of testing. In this study we want to *minimize* this value since, as discussed in our introduction, the whole point of this paper is to reduce the time required for testing cyber-physical systems.

Mutant detection score (MS+): If a set of selected test cases can significantly reduce the test execution time, but cannot detect most of the mutants, then such a selection is a bad choice. Our goal for selecting test cases is detecting as much as mutants when minimizing the test execution time. Therefore, mutant detection score becomes another important evaluation metric. In this study we want to *maximize* this value since higher value means the test suite is better since it can detect more mutants.

That is, a good test case selection approach can both (a) minimize the test execution time and (b) maximize the mutant detection score.

4.3 Statistical Analysis

In our study, we record the value of above two evaluation metrics in 20 repeats. To compare the total performance of different algorithms, we implement A Scott-Knott analysis [60]. The Scott-Knott analysis can sort the candidates by their values, and assign candidates to different ranks if the values of candidate at position i is significantly different (by more than a small effect size) to the values of candidate at position $i-1$ [61].

More precisely, Scott-Knott sorts the candidates by their median scores (and in our study, the candidates are the test case selection approaches). Scott-Knott method will split the sorted candidates into two sub-lists which maximize the expected value of differences in the observed performances before and after division [62]. After that, Scott-Knott will declare the one of the split as the best split. The best split should maximize the difference $E(\Delta)$ in the expected mean value before and after the split [63], [64]:

$$E(\Delta) = \frac{|l_1|}{|l|} abs(\overline{l_1} - \bar{l})^2 + \frac{|l_2|}{|l|} abs(\overline{l_2} - \bar{l})^2 \tag{9}$$

where $|l|$, $|l_1|$, and $|l_2|$ are size of list l, l_1, and l_2. $\bar{l}$, $\overline{l_1}$, and $\overline{l_2}$ are mean value of list l, l_1, and l_2.

After the best split, Scott-Knott then implements some statistical hypothesis tests to check the division. If two items d_1 and d_2 after division differ significantly by applying hypothesis test H, then such division is defined as a "useful" division. Scott-Knott will run recursively on each half of the best division until no division can be made. In our study, we use cliff's delta non-parametric effect size measure as the hypothesis test. Cliff's delta quantifies the number of difference between two lists of observations beyond p-values interpolation [63]. The division passes the hypothesis test if it is not a "small" effect ($Delta \geq 0.147$). The cliff's delta non-parametric effect size test explores two list A and B with size $|A|$ and $|B|$:

$$Delta = \frac{\sum_{x \in A} \sum_{y \in B} \begin{cases} +1, \text{ if } x > y \\ -1, \text{ if } x < y \\ 0, \text{ if } x = y \end{cases}}{|A||B|} \tag{10}$$

Cliff's delta estimates the probability that a value in the list A is greater than a value in the list B, minus the reverse probability [65] in the above formula. This hypothesis test and its effect size is supported by Hess and Kromery [66].

5 Results

Returning now to the research questions offered in the introduction, we offer the following results.

TABLE 4: Replication results of state-of-the-art approach, as well as the off-the-shelf optimizers. The metric with "-" means less is better while "+" means more is better. The light gray cell in each project means that approach wins others significantly (as computed by the statistical method in §4.3). Moreover, the dark gray on column **Runtime** marks the significant longer runtime for state-of-the-art NSGA-II method.

Project	Approach	TET-	MS+	Runtime (s)
Twotanks	NSGA-II	0.30	1.00	11964.6
	NSGA-III	0.49	1.00	2484.3
	MOEA/D	0.54	1.00	1296.2
CW	NSGA-II	0.39	0.99	15409.9
	NSGA-III	0.61	0.98	2185.0
	MOEA/D	0.68	0.99	1205.2
ACEngine	NSGA-II	0.38	0.73	14042.1
	NSGA-III	0.61	0.72	1895.7
	MOEA/D	0.65	0.73	1221.7
EMB	NSGA-II	0.37	1.00	12585.6
	NSGA-III	0.54	1.00	2019.6
	MOEA/D	0.63	1.00	1300.9
CC	NSGA-II	0.29	0.99	12522.8
	NSGA-III	0.48	1.00	1922.7
	MOEA/D	0.55	1.00	1306.8
Tiny	NSGA-II	0.36	0.98	15696.0
	NSGA-III	0.51	1.00	2572.3
	MOEA/D	0.60	1.00	1408.5

RQ1: Can We Verify that Test Case Selection for Multi-goal Cyber-physical Systems is a Hard Problem? To answer RQ1, we replicate Arrieta et al.'s experiment [2] in Table 4 (using 20 repeats, with 20 different random number seeds). Two algorithms differ significantly if they separate in different ranks in the Scott-Knott analysis.

As seen in Table 4, we can found the approach with NSGA-II that Arrieta et al. proposed [2] performs better than other multi-goal optimizers (MOEA/D and NSGA-III). Specifically, in all six case studies, NSGA-II can find the most minimal test suite among 3 off-the-shelf search algorithms while the minimized test suite from these 3 algorithms can achieve similar mutation scores. This means the feedback loop of executing test suite minimized by NSGA-II is much shorter than those minimized by NSGA-III and MOEA/D (We will discuss the indicator of why NSGA-III and MOEA/D may failed in this task in discussion section later). However, NSGA-II has limitations on the number of optimized goals as we discussed in §2. Specifically, NSGA-II can only handle two or three goals which is verified by Panichella et al. [10]. Hence, the state-of-the-art approach has to be re-run multiple times to explore all pairs of five goals, which requires much longer feedback loop during the test case minimization as shown in the last column in Table 4. As shown below, we can achieve better results, orders of magnitude faster. Hence, for **RQ1**, we say:

> Test case selection for multi-goal cyber-physical models is a *hard problem* that cannot be solved by merely applying, off-the-shelf, the recent optimizer technology (e.g. NSGA-III and MOEA/D). Moreover, the state-of-the-art approach can address this problem by utilizing NSGA-II in their algorithm, but needs extensive CPU execution before finding the best choice. To summarize that, developing a multi-goals optimizer with fast feedback loop is a hard problem in test case selection for cyber-physical systems.

These **RQ1** results motivate the rest of this paper where we develop a fast approach, which can handle multiple goals simultaneously for the cyber-physical systems.

TABLE 5: RQ2 results: Scores of two evaluation metrics calculated by sets of selected test cases. All entries are reported the median score of 20 repeats. In the title row, the metric with "-" means less is better while "+" means more is better. The light gray cells mark the winning approach (as computed by the statistical method in §4.3) in that metric. Last column counts the number of wins for each approach.

Project	Approach	TET-	MS+	Wins
Twotanks	NSGA-II	0.30	1	**2**
	DoLesS	0.30	1.00	**2**
CW	NSGA-II	0.39	0.98	**1**
	DoLesS	0.36	0.95	**1**
ACEngine	NSGA-II	0.39	0.72	1
	DoLesS	0.30	0.72	**2**
EMB	NSGA-II	0.37	1	1
	DoLesS	0.35	1	**2**
CC	NSGA-II	0.29	0.99	**2**
	DoLesS	0.29	1.00	**2**
Tiny	NSGA-II	0.36	0.98	0
	DoLesS	0.31	1.00	**2**

RQ2: Is DoLesS far more effective than prior state-of-the-art in terms of two evaluation metrics? To answer RQ2, we compare scores of TET- (*normalized test execution time*) and MS+ (*mutant detection score*) between the prior state-of-the-art and DoLesS. It is important to have higher performance on these two evaluation metrics since they directly indicate whether a test case selection is good or not. Table 6 shows our simulation results (note that TET - test execution time & MS - Mutation Score). For each method in each project, we repeat experiment 20 times and calculate the value of two evaluation metrics for each repeat. To obtain the final conclusion, we implement Scott-Knott statistical method to check if our approach significantly differ to state-of-the-art approach in each metric. The light gray cells mark the winning method (in the first rank) resulted from the Scott-Knott test. Moreover, we count the number of higher performance in these two evaluation metrics for each algorithm and record the number of wins in the last column.

As seen in Table 6, DoLesS gets better performance in both two evaluation metrics in three out of six projects (ACEngine, EMB, & Tiny). Moreover, in Twotanks and CC projects, DoLesS and state-of-the-art method achieve the same performance (both two evaluation metrics are tied in the first rank). In the CW project, DoLesS has higher performance in minimizing test execution time when state-of-the-art method gets a little bit higher mutation score than DoLesS. Taking above comparisons together, we can

TABLE 6: RQ3 results: Runtime comparison for our proposed DoLesS and the state-of-the-art approach on totally 20 repeats. The light gray cell marked the fastest approach in each project. The last column marks how many **times faster** our proposed approach compare to other optimizers.

Project	Approach	RunTime (s)	Speed Up
Twotanks	MOEA/D	1296.2	9
	NSGA-III	2484.3	18
	NSGA-II on 5 goals	2913.4	21
	NSGA-II	11964.6	83
	DoLesS	144.7	-
CW	MOEA/D	1205.2	29
	NSGA-III	2185.0	52
	NSGA-II on 5 goals	3001.0	71
	NSGA-II	15409.9	362
	DoLesS	42.6	-
ACEngine	MOEA/D	1221.7	16
	NSGA-III	1895.7	25
	NSGA-II on 5 goals	2613.7	34
	NSGA-II	14042.1	179
	DoLesS	78.6	
EMB	MOEA/D	1300.9	33
	NSGA-III	2019.6	52
	NSGA-II on 5 goals	3109.8	79
	NSGA-II	12585.6	319
	DoLesS	39.5	-
CC	MOEA/D	1306.8	10
	NSGA-III	1922.7	14
	NSGA-II on 5 goals	3175.8	23
	NSGA-II	12522.8	89
	DoLesS	140.9	-
Tiny	MOEA/D	1408.5	19
	NSGA-III	2572.3	35
	NSGA-II on 5 goals	2741.3	37
	NSGA-II	15696.0	210
	DoLesS	74.6	-

conclude that test cases selected by DoLesS can achieve similar or better performance in minimizing test execution time and detecting more mutants in all projects.

By summarizing above findings, we answer **RQ2** as follow:

> In **all** projects, comparing to the state-of-the-art, DoLesS can get *similar or better* performance on minimizing the execution time of selected test cases while *keeping* to detect most of the mutants.

RQ3: Is DoLesS far more efficient than prior state-of-the-art in terms of running time? To answer RQ3, we count the execution time for both algorithms during the experiment. To make comparison fair enough, we run both two algorithms on the same 64-bit Windows 10 machine with a 4.2 GHz 8-core Intel Core i7 processor and 16 GB of RAM. Moreover, when running experiments, we make sure no huge process is starting or ending in our machine.

Table 6 shows the recorded runtime for each project. For each method, we repeat experiments 20 times and record the total runtime. The light gray cells mark the fastest approach.

As seen in Table 6, in all six projects, DoLesS runs significantly faster (80-360 times faster) than the previous state-of-the-art method. By analyzing our proposed algorithm and state-of-the-art approach, we find previous approach implemented NSGA-II as their multi-objective optimization, which designed for 2 or 3 objectives [10]. To handle this issue, Arrieta et al. group objectives into 21 different combinations with two or three objectives in each combination, and select one of the best combinations by repeating their approach in those 21 groups [2]. However, in our approach, we just need continuous domination to find "ideal goals" and approximate corresponding test cases inversely.

Moreover, comparing DoLesS to other off-the-shelf optimizers (e.g. NSGA-III, MOEA/D, and NSGA-II on 5 goals), we can find DoLesS can still run significantly faster than these optimizers. Moreover, in the next section, we will discuss that DoLesS can solve the issues which cannot be solved by these optimizers.

By summarizing above findings, we answer **RQ3** as follow:

> In **both six** projects, DoLesS run significantly faster (80-360 times) than the previous method. In other words, DoLesS is far more efficient than state-of-the-art approach.

Even though our current empirical results can only boost a speed of (up to) 300 times faster, we can make a theoretical case that, if the number of goals increases, our technique would be even more comparatively faster (please note that in the following counts, test execution time is the metric that must be in every combination):

- **5** goals will result **10** different combinations of 2 or 3 objectives with text execution time must be included.
- **7** goals will result **21** different combinations of 2 or 3 objectives with text execution time must be included.
- **9** goals will result **36** different combinations of 2 or 3 objectives with text execution time must be included.
- The above pattern shows that with more goals being utilized, the number of repeats for state-of-the-art NSGA-II approach increases exponentially. However, our approach can handle multiple goals simultaneously in one time. This can show the efficiency of our approach.

6 Discussion

6.1 Why Other Optimizers May Fail

Recall from our introduction that Arrieta et al. [2], [4], [5] found state-of-the-art multi-goal optimizers (e.g. NSGA-III and MOEA/D) struggle to find minimum test suites for this domain. In this section, we take a deeper dive into our results to suggest why that might be so.

Table 7 expands on all our results and shows specific numbers for our five goals. The header on the each of the right-hand-side columns marks if those columns are to be minimize or maximized. Recall that:

- e.g. "Discontinuity+" needs to be maximized since our tests are trying to create progressively worse cases of deviation from normal behavior.
- e.g. "Time-" needs to be minimized since this is the overall execution time for selected test cases running in the simulation models (and less test execution time is desirable since this means faster feedback when running these tests).

(Aside: The "Time-" values of Table 7 show less variation than the "Speed up" column since the latter is *total* time for all our processing while the former is the time to run a test suite *after it is generated* by the methods of this paper.)

At first glance, Table 7 seems to show that MOEA/D and NSGA-III are out-performing DoLesS. However, when we consider *all* the goals, we can see that MOEA/D and NSGA-III are failing to deal with our five-goal problem (evidence: NSGA-III and MOEA/D performance worse than other methods at minimizing the overall test execution time).

Figure 4 shows why this might be so. That figure shows what happens to the mutant detection score (on the y-axis) as we increase the effectiveness measurements along the x-axis. To place those figures in context, we note that most of the DoLesS results in Table 7 for Discontinuity+, Infinity+, Instability+, and MinMax+ are around $x > .45$ This is important since, in terms of for detecting mutants, $x = .45$ can be as good as anything else (evidence: in Figure 4, the y-axis values at $x > 0.45$ are not necessarily better than $x < .45$)

Based on the above discussion from this section, now we can offer the reason why NSGA-III and MOEA/D are not suitable for this task:

They are optimizing too much for superfluous details.

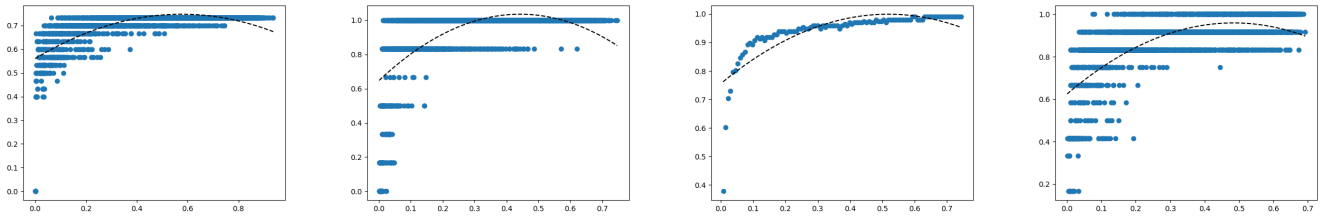
More specifically, NSGA-III and MOEA/D extensively optimize the 4 maximization goals while concentrate less on optimizing the only 1 minimization goal. But the minimization goal (e.g. "Time-") is important because it directly reflects the overall simulation execution time with the selected test cases (if the feedback loop is too long for testing a simulation model, then the test case selection process is not useful).

6.2 The Generality of DoLesS

In this section we will discuss the generality of our proposed method DoLesS.

TABLE 7: The sanity check results of 5 different approaches for each case study. In the right-hand-side of the header row, the entry with "-" at the end means minimization goal and the entry with "+" at the end means maximization goal.

Project	Approach	Best Combination	Speed up	Time-	Discontinuity+	Infinity+	Instability+	MinMax+
Twotanks	MOEA/D	-	9	0.54	0.81	0.92	0.83	0.92
	NSGA-III	-	18	0.49	0.70	0.81	0.75	0.81
	NSGA-II	-	21	0.35	0.58	0.67	0.64	0.67
	NSGA-II	Time, Infinite, Minmax	83	0.30	0.54	0.65	0.55	0.65
	DoLesS	-	-	0.30	0.56	0.67	0.56	0.67
CW	MOEA/D	-	29	0.68	0.82	0.79	0.84	0.79
	NSGA-III	-	52	0.61	0.74	0.67	0.76	0.67
	NSGA-II	-	71	0.46	0.60	0.57	0.60	0.57
	NSGA-II	Time, Instability	362	0.39	0.55	0.34	0.64	0.34
	DoLesS	-	-	0.36	0.50	0.58	0.44	0.58
ACEngine	MOEA/D	-	16	0.65	0.84	0.80	0.78	0.80
	NSGA-III	-	25	0.61	0.77	0.74	0.72	0.74
	NSGA-II	-	34	0.40	0.56	0.53	0.51	0.53
	NSGA-II	Time, Discontinuity, Instability	179	0.39	0.53	0.49	0.49	0.49
	DoLesS	-	-	0.30	0.47	0.44	0.41	0.44
EMB	MOEA/D	-	33	0.63	0.83	0.83	0.84	0.83
	NSGA-III	-	52	0.54	0.70	0.70	0.70	0.70
	NSGA-II	-	79	0.42	0.61	0.61	0.59	0.61
	NSGA-II	Time, Instability	319	0.37	0.50	0.49	0.57	0.50
	DoLesS	-	-	0.35	0.61	0.61	0.48	0.61
CC	MOEA/D	-	10	0.55	0.87	0.86	0.88	0.86
	NSGA-III	-	14	0.48	0.77	0.74	0.76	0.74
	NSGA-II	-	23	0.35	0.64	0.58	0.61	0.58
	NSGA-II	Time, Discontinuity	89	0.29	0.62	0.43	0.50	0.43
	DoLesS	-	-	0.29	0.56	0.56	0.57	0.56
Tiny	MOEA/D	-	19	0.60	0.83	0.83	0.84	0.83
	NSGA-III	-	35	0.52	0.72	0.71	0.73	0.71
	NSGA-II	-	37	0.40	0.63	0.62	0.66	0.62
	NSGA-II	Time, Instability	210	0.36	0.57	0.56	0.64	0.56
	DoLesS	-	-	0.31	0.58	0.57	0.57	0.57



(a) Discontinuity (b) Instability (c) Growth to infinity (d) Minmax difference

Fig. 4: Diminishing returns in the y-axis as the effectiveness measurement scores increase along the x-axis (y-axis shows the mutant detection score). In this figure we note that there is limited improvement in the y value above an threshold of $x = 0.5$. This figure is generated as follows: (a) to selected different combinations of tests, 50 times is iterated from one to size of the test suite for each model/effectiveness score; (b) these combinations were summarized as 5000 (x, y) pairs where x is the effectiveness measurements and y is the mutant detection score of that test; (c) if many pairs have the same effectiveness value x_i, we replace those with one pair of $x = x_i$ and $y = median(y)$; (d) the black dash line is a polynomial curve fit (aside: we tried different parametric forms, and the polynomial of degree two has the largest correlation). For space reason, this figure shows a small sample of these charts (for all figures, please see https://github.com/ai-se/DoLesS).

DoLesS implements the idea from Chen et al.'s study that uses a domination predicate to sort the space of possible goals to a small number of representative examples. Many-objective continuous domination then splits these candidates into two regions where one region contains samples that dominate the samples in another region. After that, it applies an inverted least square approximation approach to find a minimal set of tests. In cyber-physical system test case selection problem that utilizes the effectiveness measurements, least square approximation is considered as an adequate approach to inversely find the minimal test suite from objective space to variable space because the formulation of this problem from the variable space to the objective space can be treated as a linear equation system (See section §3.2.5). Thus, the first possible adequate application of our method in other software engineering domain is the multi-objective optimization problems which the relation of the variable space to the objective space can be re-formulated as a linear equation system. In such problems, DoLesS can be directly applied with only small modifications on the formulation.

However, a large proportion of problems in software engineering domain cannot be re-formulated as a linear equation system since the relationship between variable space and objective space is non-linear. In such scenarios, our proposed DoLesS approach needs to be modified by changing the linear least square approximation to other approaches (e.g. non-linear least square approximation). However, we hypothesis that the overall framework of DoLesS can still be applied as a fast method in these multi-

objective optimization problems.

7 Threats to Validity

This section discusses issues raised by Feldt et al. [67].

Construct validity: The construct validity threat mainly exists in the parameter settings of algorithms. For example, in our replication experiment, we use one point crossover with 0.8 crossover probability and bitflip mutation with 1/(number of variables) mutate probability as prior studies did in order to keep consistent. For another example, in least square approximation, we use 0.5 threshold to indicate whether a test case has large probability to be chose or not. Moreover, we use the default setting of Python least square solver in our algorithm. Changing these parameters can result differences in selecting test cases. Therefore, our observation may differ when different parameters are used. We would consider hyper-parameter tuning [64], [68] in future work to mitigate this threat.

Conclusion validity: The conclusion validity threat in this study is related to the random variations of our algorithm and the access to the real faults. To reduce the effect caused by this threat, we repeat all experiments 20 times in the same machine. Moreover, we apply Scott-Knott statistical test to compare if the outcomes of our proposed approach and the previous methods differ significantly. Moreover, the systems we test are all public systems thus no faulty versions can be utilized in our analysis. To mitigate this threat, we use the same mutants which created by Arrieta et al. [2] based on common system faulty behaviors, which can make mutants more similar to the real faults. We use these mutants to better address the threat that caused by lacking access to the real faults.

Internal validity: Internal validity focuses on the correctness of the treatment caused the outcome. In this study, we constraint our simulations to the same data set. Moreover, we evaluate our approach and the previous approach in the same workflow. Another internal validity threat can refer to the mutants generated from the projects. To mitigate this threat, we use the same mutants that Arrieta et al. [2] used in their study, which they have removed duplicated mutants.

External validity: External validity concerns the application of our algorithm in other problems. In this study, we generate our conclusion from six real-world Simulink cyber-physical systems. When applying our method into other case studies, these concerns may raised: (a) Six real-world open-source systems may not represent the society of cyber-physical system. However, to the best of our knowledge, these six models are the most commonly used models in previous studies. Moreover, cyber-physical models are hard to obtain thus almost all previous studies only utilize 3 to 4 models (either private or public). Hence, we used all current resources to better address this threat in this study. In the future, more models can be used to test `DoLesS` if they become available. (b) `DoLesS` may needs modification for those projects which effectiveness measurement data and test cases are not in the linear relationship. For those projects, we would consider non-linear least square approximation as future work to mitigate this threat.

8 Conclusion & Future Work

Finding representative test cases from the initial test suite is an important task in simulation based model. Better test case selection methods can not only reduce the test execution time in the future testing, but also maintain the same testing performance as usual. In other words, a good test case selection approach can (a) minimize the test execution time and (b) maximize the mutation score.

Previous literature by Arrieta et al. [2] has shown the great success on using NSGA-II as the multi-objective optimization method to select representative test cases. However, their design has a deficiency where they need to evaluate 21 combinations first to select the best subset. Moreover, since NSGA-II is a randomized algorithm, repeats are necessary during the experiment. Therefore, their approach will execute NSGA-II 420 times with 20 repeats.

In this study, we address this deficiency by selecting test cases from all effectiveness measurement metrics. To do that, we use a very fast approach - continuous domination to select representative goals. Moreover, we make a better use of the linear relationship between test cases and goals to find the best test selections correspond to the representative goals (by linear least square approximation). Our experimental results show that our proposed approach `DoLesS` can (a) achieve better scores in terms of five effectiveness measurement metrics, (b) achieve better scores in terms of two evaluation metrics, and (c) solve test selection problem in a very fast speed (80-360 times faster than the state-of-the-art method) for cyber-physical models.

We conjecture that our method would be a better candidate for scaling to large systems than the method proposed by Arrieta et al. [2]. To see that, consider the following scenario: To successfully perform test case selection on selected cyber-physical case studies, Arrieta et al.'s approach required 3-5 hours algorithm execution time. Now imagine in some higher complexity simulation models (e.g. drone simulation models) with dozens more of test cases in the initial test suite, and these models have more signal processing criteria in I/O signals, both evaluation time and objectives are increased. In such scenario, the execution time of running NSGA-II in all subsets of objectives will exponentially increase as we mentioned at the end of **RQ4**. Moreover, such models (e.g. drone simulation models) require much faster feedback than usual cyber-physical models. Due to the above reasons, the ideal test case selection approach for complex simulation models needs to handle multiple goals (more than 4 goals) in the same time and perform selection in a very short execution time for the fast feedback. For that task, we recommend `DoLesS`.

As to further work, apart from extending this exploration of feedback loop anti-patterns, we conjecture that our methods could be useful for other multi-objective reasoning tasks. Standard practice in this area is to mutate large populations across a Pareto frontier. This has certainly been a fruitful research agenda [69], [70], [26], [71]. But perhaps the testing community could reason about more goals, faster, if used our domination methods and least squares methods to "reason backwards" from goal space to decision space. Hence, future works can be conducted with

- Finding more simulation projects which can strength our approach.
- Developing more effectiveness measurement metrics which can better indicate representative test cases.
- Adjusting our approach based on the testing scenarios of different projects. Moreover, in some cases, a new test case selection approach is needed for those projects.

ACKNOWLEDGMENTS

This research was partially funded by the National Science Foundation under Grant No. 1908762. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, "Automated test suite generation for time-continuous simulink models," in *proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 595–606.

[2] A. Arrieta, S. Wang, U. Markiegi, A. Arruabarrena, L. Etxeberria, and G. Sagardui, "Pareto efficient multi-objective black-box test case selection for simulation-based testing," *Information and Software Technology*, vol. 114, pp. 137–154, 2019.

[3] A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria, "Search-based test case selection of cyber-physical system product lines for simulation-based validation," in *Proceedings of the 20th International Systems and Software Product Line Conference*, 2016, pp. 297–306.

[4] G. Sagardui, J. Agirre, U. Markiegi, A. Arrieta, C. F. Nicolás, and J. M. Martín, "Multiplex: A co-simulation architecture for elevators validation," in *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*. IEEE, 2017, pp. 1–6.

[5] C. A. González, M. Varmazyar, S. Nejati, L. C. Briand, and Y. Isasi, "Enabling model testing of cyber-physical systems," in *Proceedings of the 21th ACM/IEEE international conference on model driven engineering languages and systems*, 2018, pp. 176–186.

[6] J. Chen, V. Nair, R. Krishna, and T. Menzies, ""sampling" as a baseline optimizer for search-based software engineering," *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 597–614, 2018.

[7] A. Agrawal, T. Menzies, L. L. Minku, M. Wagner, and Z. Yu, "Better software analytics via "duo": Data mining algorithms using/used-by optimizers," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2099–2136, 2020.

[8] A. Arrieta, S. Wang, A. Arruabarrena, U. Markiegi, G. Sagardui, and L. Etxeberria, "Multi-objective black-box test case selection for cost-effectively testing simulation models," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 1411–1418.

[9] A. Arrieta, J. A. Agirre, and G. Sagardui, "Seeding strategies for multi-objective test case selection: an application on simulation-based testing," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 1222–1231.

[10] A. Panichella, F. M. Kifetew, and P. Tonella, "Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets," *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 122–158, 2017.

[11] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, pp. 832–842. [Online]. Available: https://doi.org/10.1007/978-3-540-30217-9_84

[12] A. S. Sayyad, T. Menzies, and H. Ammar, "On the value of user preferences in search-based software engineering: A case study in software product lines," in *2013 35Th international conference on software engineering (ICSE)*. IEEE, 2013, pp. 492–501.

[13] T. Wagner, N. Beume, and B. Naujoks, "Pareto-, aggregation-, and indicator-based methods in many-objective optimization," in *Evolutionary Multi-Criterion Optimization*, S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 742–756.

[14] B. S. Ahmed, "Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing," *Engineering Science and Technology, an International Journal*, vol. 19, no. 2, pp. 737–753, 2016.

[15] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche, "Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system," *Software Testing, Verification and Reliability*, vol. 25, no. 4, pp. 371–396, 2015.

[16] W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini, "Test set size minimization and fault detection effectiveness: A case study in a space application," in *Proceedings Twenty-First Annual International Computer Software and Applications Conference (COMPSAC'97)*. IEEE, 1997, pp. 522–528.

[17] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," *Software: Practice and Experience*, vol. 28, no. 4, pp. 347–369, 1998.

[18] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software testing, verification and reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[19] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 235–245.

[20] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, vol. 52, no. 1, pp. 14–30, 2010.

[21] D. Binkley, "Reducing the cost of regression testing by semantics guided test case selection," in *Proceedings of International Conference on Software Maintenance*. IEEE, 1995, pp. 251–260.

[22] G. Rothermel, M. J. Harrold, and J. Dedhia, "Regression test selection for c++ software," *Software Testing, Verification and Reliability*, vol. 10, no. 2, pp. 77–109, 2000.

[23] T. Y. Chen and M. F. Lau, "Test case selection strategies based on boolean specifications," *Software Testing, Verification and Reliability*, vol. 11, no. 3, pp. 165–180, 2001.

[24] Z. Xu, K. Gao, and T. M. Khoshgoftaar, "Application of fuzzy expert system in test case selection for system regression test," in *IRI-2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005.* IEEE, 2005, pp. 120–125.

[25] M. Grindal, B. Lindström, J. Offutt, and S. F. Andler, "An evaluation of combination strategies for test case selection," *Empirical Software Engineering*, vol. 11, no. 4, pp. 583–611, 2006.

[26] A. Panichella, R. Oliveto, M. Di Penta, and A. De Lucia, "Improving multi-objective test case selection by injecting diversity in genetic algorithms," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 358–383, 2014.

[27] D. Mondal, H. Hemmati, and S. Durocher, "Exploring test suite diversification and code coverage in multi-objective test case selection," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2015, pp. 1–10.

[28] L. S. de Souza, R. B. Cavalcante Prudêncio, and F. A. de Barros, "A hybrid particle swarm optimization and harmony search algorithm approach for multi-objective test case selection," *Journal of the Brazilian Computer Society*, vol. 21, no. 1, pp. 1–20, 2015.

[29] X. Devroey, G. Perrouin, A. Legay, P.-Y. Schobbens, and P. Heymans, "Search-based similarity-driven behavioural spl testing," in *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, 2016, pp. 89–96.

[30] X. Devroey, G. Perrouin, A. Legay, P. Heymans, and P. Heymans, "Dissimilar test case selection for behavioural software product line testing," 2017.

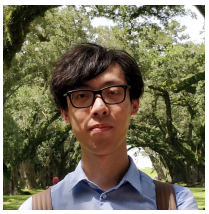
[31] H. Hemmati and L. Briand, "An industrial investigation of similarity measures for model-based test case selection," in *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, 2010, pp. 141–150.

[32] E. G. Cartaxo, P. D. Machado, and F. G. O. Neto, "On the use of a similarity function for test case selection in the context of model-based testing," *Software Testing, Verification and Reliability*, vol. 21, no. 2, pp. 75–100, 2011.

[33] D. Pradhan, S. Wang, S. Ali, and T. Yue, "Search-based cost-effective test case selection within a time budget: An empirical study," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 1085–1092.

[34] A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria, "Search-based test case selection of cyber-physical system product lines for simulation-based validation," in *Proceedings of the 20th International Systems and Software Product Line Conference*, 2016, pp. 297–306.

[35] R. Lachmann, M. Felderer, M. Nieke, S. Schulze, C. Seidl, and I. Schaefer, "Multi-objective black-box test case selection for system testing," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 1311–1318.
[36] D. J. Fremont, E. Kim, Y. V. Pant, S. A. Seshia, A. Acharya, X. Bruso, P. Wells, S. Lemke, Q. Lu, and S. Mehta, "Formal scenario-based testing of autonomous vehicles: From simulation to the real world," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–8.
[37] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2010.
[38] N. T. Binh, K. T. Tung *et al.*, "A novel fitness function of metaheuristic algorithms for test data generation for simulink models based on mutation analysis," *Journal of Systems and Software*, vol. 120, pp. 17–30, 2016.
[39] 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 *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 936–946.
[40] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.
[41] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints," *IEEE transactions on evolutionary computation*, vol. 18, no. 4, pp. 577–601, 2013.
[42] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on evolutionary computation*, vol. 11, no. 6, pp. 712–731, 2007.
[43] S. A. Chowdhury, S. Mohian, S. Mehra, S. Gawsane, T. T. Johnson, and C. Csallner, "Automatically finding bugs in a commercial cyber-physical system development tool chain with slforge," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 981–992.
[44] A. J. Offutt, J. Pan, K. Tewary, and T. Zhang, "An experimental evaluation of data flow and mutation testing," *Software: Practice and Experience*, vol. 26, no. 2, pp. 165–176, 1996.
[45] D. Schuler and A. Zeller, "Javalanche: Efficient mutation testing for java," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 297–298.
[46] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, "Mutation testing advances: an analysis and survey," in *Advances in Computers*. Elsevier, 2019, vol. 112, pp. 275–378.
[47] A. V. Pizzoleto, F. C. Ferrari, J. Offutt, L. Fernandes, and M. Ribeiro, "A systematic literature review of techniques and metrics to reduce the cost of mutation testing," *Journal of Systems and Software*, vol. 157, p. 110388, 2019.
[48] R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, "Test generation and test prioritization for simulink models with dynamic behavior," *IEEE Transactions on Software Engineering*, vol. 45, no. 9, pp. 919–944, 2018.
[49] N. T. Binh *et al.*, "Mutation operators for simulink models," in *2012 Fourth International Conference on Knowledge and Systems Engineering*. IEEE, 2012, pp. 54–59.
[50] A. Brillout, N. He, M. Mazzucchi, D. Kroening, M. Purandare, P. Rümmer, and G. Weissenbacher, "Mutation-based test case generation for simulink models," in *International Symposium on Formal Methods for Components and Objects*. Springer, 2009, pp. 208–227.
[51] Y. F. Yin, Y. B. Zhou, and Y. R. Wang, "Research and improvements on mutation operators for simulink models," in *Applied mechanics and materials*, vol. 687. Trans Tech Publ, 2014, pp. 1389–1393.
[52] Y. Zhan and J. A. Clark, "Search-based mutation testing for simulink models," in *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, 2005, pp. 1061–1068.
[53] S. Wang, S. Ali, and A. Gotlieb, "Minimizing test suites in software product lines using weight-based genetic algorithms," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013, pp. 1493–1500.
[54] R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, "Effective test suites for mixed discrete-continuous stateflow controllers," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 84–95.
[55] R. Matinnejad, S. Nejati, and L. C. Briand, "Automated testing of hybrid simulink/stateflow controllers: industrial case studies," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 938–943.
[56] D. Margalit, J. Rabinoff, and L. Rolen, "Interactive linear algebra," *Georgia Institute of Technology*, 2017.
[57] C. Menghi, S. Nejati, K. Gaaloul, and L. C. Briand, "Generating automated and online test oracles for simulink models with continuous and uncertain behaviors," in *Proceedings of the 2019 27th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 27–38.
[58] A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria, "Search-based test case prioritization for simulation-based testing of cyber-physical system product lines," *Journal of Systems and Software*, vol. 149, pp. 1–34, 2019.
[59] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, and L. Etxeberria, "Employing multi-objective search to enhance reactive test case generation and prioritization for testing industrial cyber-physical systems," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 3, pp. 1055–1066, 2017.
[60] N. Mittas and L. Angelis, "Ranking and clustering software cost estimation models through a multiple comparisons algorithm," *IEEE Transactions on software engineering*, vol. 39, no. 4, pp. 537–551, 2012.
[61] X. Ling, R. Agrawal, and T. Menzies, "How different is test case prioritization for open and closed source projects," *IEEE Transactions on Software Engineering*, 2021.
[62] H. Tu, Z. Yu, and T. Menzies, "Better data labelling with emblem (and how that impacts defect prediction)," *TSE*, 2020.
[63] T. Xia, R. Krishna, J. Chen, G. Mathew, X. Shen, and T. Menzies, "Hyperparameter optimization for effort estimation," *arXiv preprint arXiv:1805.00336*, 2018.
[64] H. Tu, G. Papadimitriou, M. Kiran, C. Wang, A. Mandal, E. Deelman, and T. Menzies, "Mining workflows for anomalous data transfers," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 1–12.
[65] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.
[66] M. R. Hess and J. D. Kromrey, "Robust confidence intervals for effect sizes: A comparative study of cohen'sd and cliff's delta under non-normality and heterogeneous variances," in *annual meeting of the American Educational Research Association*. Citeseer, 2004, pp. 1–30.
[67] R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research-an initial survey." in *Seke*, 2010, pp. 374–379.
[68] H. Tu and V. Nair, "While tuning is good, no tuner is best," in *FSE SWAN*, 2018.
[69] C. L. B. Maia, R. A. F. Do Carmo, F. G. de Freitas, G. A. L. De Campos, and J. T. De Souza, "A multi-objective approach for the regression test case selection problem," in *Proceedings of Anais do XLI Simposio Brasileiro de Pesquisa Operacional (SBPO 2009)*, 2009, pp. 1824–1835.
[70] S. Yoo and M. Harman, "Using hybrid algorithm for pareto efficient multi-objective test suite minimisation," *Journal of Systems and Software*, vol. 83, no. 4, pp. 689–701, 2010.
[71] W. Zheng, R. M. Hierons, M. Li, X. Liu, and V. Vinciotti, "Multi-objective optimisation for regression testing," *Information Sciences*, vol. 334, pp. 1–16, 2016.



Xiao Ling is a third-year PhD student in Computer Science at NC State University. His research interests include automated software testing and machine learning for software engineering.

Tim Menzies (IEEE Fellow, Ph.D. UNSW, 1995) is a Professor in computer science at NC State University, USA, where he teaches software engineering, automated software engineering, and programming languages. His research interests include software engineering (SE), data mining, artificial intelligence, and search-based SE, open access science. For more information, please visit http://menzies.us.