Faster Multi-Goal Simulation-Based Testing using DoLesS (Domination with a Least Squares Approximation)

Xiao Ling
xling4@ncsu.edu
North Carolina State University
Raleigh, USA

Tim Menzies
timm@ieee.org
North Carolina State University
Raleigh, USA

ABSTRACT

For cyber-physical 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**a**s**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 learn a minimal set of tests that can distinguish best from rest in the reduced example space.

DoLesS has been tested on four cyber-physical models: 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. 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 cyber-physical systems. For replication purposes, all our code is on-line: https://github.com/hellonull123/Test_Selection_2021.

KEYWORDS

Search-based software engineering, Machine learning with and for SE, Software testing, Modeling and Model-Driven Engineering, Validation and Verification, Embedded and cyber-physical systems

1 INTRODUCTION

Simulation models play an important role in many domains. Engineers build such models to simulate complex systems [29]. 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 [3]. 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.

However, using these models for test case minimization can be a very difficult process [3]. These simulation models are built to simulate complex systems such as electronic and physical models. Hence executing these simulation models can be very time consuming [5]. This problem gets even worse for multi-goal problems (e.g. minimizing runtime *and* maximizing the number of bugs found) since it is necessary to run the models multiple times for different subsets of the goals [5]. For example, Arrieta et al. reported that testing a high fidelity simulation model can take hours to days [3, 19, 37]. Further, they warned that state-of-the-art multi-goal optimizers (e.g. NSGA-III and MOEA/D) struggle to find minimal test suites for this task.

Recently, Chen [10] and Agrawal et al. [1] 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 problem 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**a**s**t **S**quares Approximation) applies a domination predicate to sort the example space to a small number of representative data points. 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 example 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. [3]. In that companion, we ask the following research questions.

RQ1: Can we verify that test case selection for multi-goal cyber-physical systems is a hard problem? Arrieta et al. [3] reported that standard multi-goal optimizers such as NSGA-III and MOEA/D failed in this task. This is an important observation since, if otherwise, there will be no clear motivation for this paper. Accordingly, as a first step, we replicate their results in our experiment.

RQ2: Can DoLesS find better ways to select test cases which result in test suites with higher effectiveness measure (objective) scores? Section 3.1 of this paper reviews five effectiveness measurement metrics which are used to select test cases for cyber-physical systems in the previous study [3]. Our results show that on those five metrics, general performances of **DoLesS** is better than previous state-of-the-art method.

RQ3: Do selected test cases by DoLesS beat the prior state-of-the-art? Apart from the five effectiveness measurement metrics used by Arrieta et al. [3], two other evaluation scores of interest are (a) reduction in the number of test case 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.

RQ4: 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 is a problem that defeats state-of-the-art optimizers (NSGA-III and MOEA/D).
(3) We clearly document the value of doing **DoLesS**. When testing on four cyber-physical models, **DoLesS** finds test suites as good, as even better, than those found by Arrieta et al.'s approach [3]. 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 follow. 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 6 explores threats to validity and Section 7 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 [38, 42, 52].
- Chen et al. [10] 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 [10].

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 [2, 15, 44, 45, 49]. 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. [16] 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.

Research has found many test case selection techniques such as DejaVu based, firewall based, dependency based, and specification based techniques [17]. 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 [8]. Rothermel et al. developed a test case selection technique for C++ software on 2020 [36]. In 2001, Chen et al. developed test case selection strategies based on the boolean specifications [11]. In 2005, a fuzzy expert system was developed in test case selection by Xu et al. [47]. In 2006, Grindal et al. presented an empirical study on evaluating five combination strategies for test case selection [20]. In 2011, Cartaxo et al. implemented a similarity function for test case selection in model-based testing [9]. Pradhan et al. [35] 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 [5]. Also in 2017, Lachmann et al. [23] did an empirical study on several black-box metrics and made comparisons on their performance in selecting test cases in system testing.

Due to the data requirements, 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 (for example, see the steam governor of Figure 1). 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.

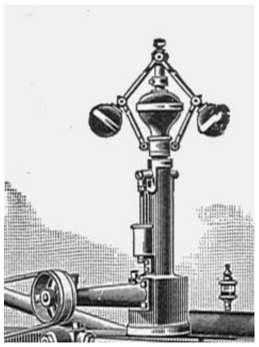
Figure 1: Feedback controller (1904). Energy is released (as steam) when the weights spin faster and pull away from the shaft. Thus, increased speed leads to energy release, which slows the system. From Wikipedia.

In one of the IST'19 journal paper, Arrieta et al. [3] explored issues associated with test suite minimization by using the data extracted from feedback loops. They noted that feedback loops

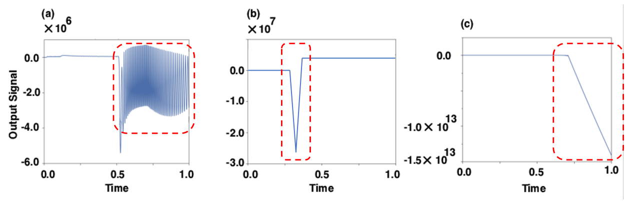

Figure 2: Examples of anti-patterns seen for systems under feedback. The name of anti-patterns from left to right are *instability, discontinuity,* and *growth to negative infinity* correspondingly. The alarming patterns are shown in red marks.

have anti-patterns; i.e. undesirable features that appear in a time series trace of the output of the system. Figure **??** 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. [3] explored seven goals for cyber-physical model testing: maximizing the three anti-patterns that shown in Figure **??**, 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 mutants [22], 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. [3]. Those mutants were generated with Hanh et al.'s technique [7] 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 [34]. 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 [14] can be ineffective for more than three goals (a result that is echoed by prior work [38]). More recent optimizers like NSGA-III [13] and MOEA/D [50] 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 definitely an extension to the state-of-the-art, Arrieta et al.'s [3] study had two drawbacks. Firstly, the test cases selected in this way was only the best which measured on a subset of the optimization goals. Secondly, the "pairwise" approach increased 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 [3]. 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 [12]. For an example Simulink models, see Figure 3. This is a model with two hierarchical levels [3]. 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.

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 $\{os_1, \cdots, os_l\}$ [3, 29].

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 selection problem as follow:

Given an initial set of n test cases $T = \{t_1, t_2, t_3, \cdots, t_n\}$, we want to find a subset of that set of test cases $TS = \{ts_1, ts_2, \cdots, ts_a\}$ which can test the model as initial test suite does $Perform(TS) = Perform(T)$ with $1 \leq |TS| \leq |T|$.

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.

3 EXPERIMENTAL METHODS

3.1 Simulation Effectiveness Metrics

In this study, we implement five out of seven effectiveness measurement metrics which Arrieta et al. [3] used in their study. The first three metrics are also widely used in previous studies [27, 28, 43], and the forth metric is proposed by Arrieta et al. [3].

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.

3.1.1 Test Execution Time. Total test execution time is the first metric we implement in our study. Wang et al. [43] 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, Arrieta et al. pointed out the problem that each test case has different execution time [3]. In our study, we use the similar

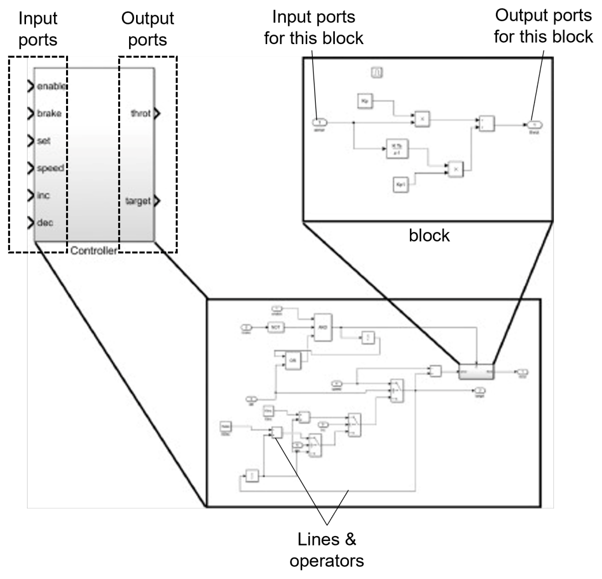

Figure 3: Simple example of a Simulink model - Cruise Controller of a car [3].

calculation that Arrieta et al. [3] did in their study to deal with the test execution time. The test execution time is calculated as follow:

Define a set of test cases $TC_{Select} = \{T_{s_1}, \cdots, T_{s_n}\}$ that are selected from the initial test suite $TC_{Init} = \{T_{i_i}, \cdots, T_{i_m}\}$. Let TET_{s_a} denotes the execution time of the test case T_{s_a} in TC_{Select}, and TET_{i_b} denotes the execution time of the test case T_{i_b} in TC_{Init}. The total test execution time of a set of selected test cases is [3]

$$Time_{Select} = \sum_{a=1}^{n} TET_{s_a} / \sum_{b=1}^{m} TET_{i_b} \tag{1}$$

In our study, we want to <u>*minimize*</u> 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. [27] 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. Assume we have N output signals $\{O_1, O_2, \cdots, O_N\}$. For discontinuity score of each output signal $discontinuity(O_j)$ where $1 \le j \le N$, Matinnejad et al. calculated it with [27, 28]

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

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

The discontinuity rate of a set of selected test cases is calculated as follow:

With same definition of test case above, let DC_{s_a} denotes the discontinuity score of the test case T_{s_a} in TC_{Select}, and DC_{i_b} denotes the discontinuity score of the test case T_{i_b} in TC_{Init}. The total discontinuity score of a set of selected test cases is [3]

$$DC_{Select} = \sum_{a=1}^{n} DC_{s_a} / \sum_{b=1}^{m} DC_{i_b} \tag{3}$$

$DC_k = \frac{\sum_{i=1}^{N} discontinuity(O_{k_i})}{max_{l=1}^{N}(discontinuity(O_l))*N}$ is the normalized discontinuity score of the test case k among N output signals $\{O_1, O_2, \cdots, O_N\}$.

In our study, we want to <u>*maximize*</u> 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. [27] 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 [27]. Assume we have N output signals $\{O_1, O_2, \cdots, O_N\}$. For instability score of each output signal $instability(O_j)$ where $1 \le j \le N$, Matinnejad et al. calculated it with [27]

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

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

The instability score of a set of selected test cases is calculated as follow:

With same definition of test cases above, let IS_{s_a} denotes the instability score of the test case T_{s_a} in TC_{Select}, and IS_{i_b} denotes the instability score of the test case T_{i_b} in TC_{Init}. The total instability score of a set of selected test cases is [3]

$$IS_{Select} = \sum_{a=1}^{n} IS_{s_a} / \sum_{b=1}^{m} IS_{i_b} \tag{5}$$

$IS_k = \frac{\sum_{i=1}^{N} instability(O_{k_i})}{max_{l=1}^{N}(instability(O_l))*N}$ is the normalized instability score of the test case k among N output signals $\{O_1, O_2, \cdots, O_N\}$.

In our study, we want to <u>*maximize*</u> 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. [28] 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. Assume we have N output signals $\{O_1, O_2, \cdots, O_N\}$. For growth to infinity score of each output signal $infinity(O_j)$ where $1 \le j \le N$, Matinnejad et al. calculated it with [28]

$$infinity(O_j) = \max_{i=1}^{k} |sig(i \cdot \Delta t)| \tag{6}$$

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

The growth to infinity score of a set of selected test cases is calculated as follow:

With same definition above, let IF_{s_a} denotes the infinity score of the test case T_{s_a} in TC_{Select}, and IF_{i_b} denotes the infinity score of the test case T_{i_b} in TC_{Init}. The total growth to infinity score of a set of selected test cases is [3]

$$IF_{Select} = \sum_{a=1}^{n} IF_{s_a} / \sum_{b=1}^{m} IF_{i_b} \tag{7}$$

$IF_k = \frac{\sum_{i=1}^{N} infinity(O_{k_i})}{max_{l=1}^{N}(infinity(O_l))*N}$ is the normalized infinity score of the test case k among N output signals $\{O_1, O_2, \cdots, O_N\}$.

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. Assume we have N output signals $\{O_1, O_2, \cdots, O_N\}$. For output minimum and maximum difference score of each output signal $minmax(O_j)$ where $1 \leq j \leq N$, Arrieta et al. calculated it with [3]

$$minmax(O_j) = |\max_{i=1}^{k}(sig(i \cdot \Delta t)) - \min_{i=1}^{k}(sig(i \cdot \Delta t))| \tag{8}$$

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

The difference of output minimum and maximum of a set of selected test cases is calculated as follow:

With same definition of test case above, let MMD_{s_a} denotes the output minimum and maximum difference of the test case T_{s_a} in TC_{Select}, and MMD_{i_b} denotes the output minimum and maximum difference of the test case T_{i_b} in TC_{Init}. The total output minimum and maximum difference score of a set of selected test cases is [3]

$$MMD_{Select} = \sum_{a=1}^{n} MMD_{s_a} / \sum_{b=1}^{m} MMD_{i_b} \tag{9}$$

$MMD_k = \frac{\sum_{i=1}^{N} minmax(O_{k_i})}{max_{l=1}^{N}(minmax(O_l))*N}$ is the normalized output minimum and maximum difference score of the test case k among N output signals $\{O_1, O_2, \cdots, O_N\}$.

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 decides one individual is better than another if it is better on at least one goal and worse on none. Numerous studies [38, 42, 52] warn that binary domination is hard to distinguish candidates once the number of goals grows to three or more.

For many-goal problems, Zilter's *continuous domination* predicate [52] is useful [38, 42, 52]. 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 minimization or maximizing the goal i correspondingly. According to Zitler [52], 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 [14]. 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 [32]. 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 [13]. 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 points are uniformly distributed on the normalized hyper-plane with some division number p [13]. 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 [13].

3.2.4 MOEA/D. MOEA/D is the first multi-objective optimization algorithm which utilizes decomposition technique [50]. More specifically, MOEA/D explicitly decomposes the problem into multiple sub-problems with less objectives in each subgroup and solves these sub-problems simultaneously [50]. To 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

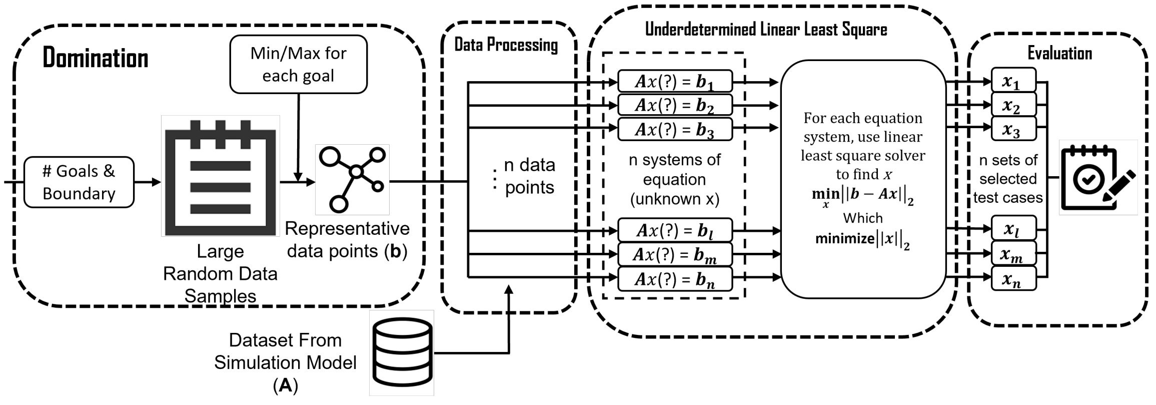
Domination
Min/Max for each goal
# Goals & Boundary
Large Random Data Samples
Representative data points (b)
Data Processing
n data points
Dataset From Simulation Model (A)
Underdetermined Linear Least Square
$Ax(?) = b_1$
$Ax(?) = b_2$
$Ax(?) = b_3$
n systems of equation (unknown x)
$Ax(?) = b_l$
$Ax(?) = b_m$
$Ax(?) = b_n$
For each equation system, use linear least square solver to find x $\min_x \|b - Ax\|_2$ Which minimize $\|x\|_2$
Evaluation
x_1
x_2
x_3
n sets of selected test cases
x_l
x_m
x_n

Figure 4: 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.

itself, its local neighborhood will move in the same direction as X_0 does.

3.2.5 **DoLesS**. Figure 4 shows the entire framework of our approach. Unlike above evolutionary algorithms, our proposed approach **DoLesS** (**Do**mination with **Le**ast **S**quares Approximation):

- Uses *continuous domination* (defined below, see the first block in Figure 4) 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 4) to inversely predict the test selection outcomes which can fit the representative sets of goals best. This least square approximation technique is discussed below.

After sorting on the domination score, **DoLesS** divides data into:

- The $\sqrt{n}$ "best" items. In our case study, we randomly generate 10000 initial candidates, hence the first 100 candidates with highest domination score are grouped into the "best" group.
- And the remaining "rest" items.

Here we select the number of final population as 100 with two reasons: (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 [3].

In the *data processing* stage of Figure 4, we take data from each model (which Arrieta et al. also used in their study [3]), and then processes it into the form of least square approximation structure by combining with the representative goals generated from continuous domination. Table 1(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 1(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

Table 1: 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)

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 1(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 method which predicts the best value of a set of unknown variables that fits the relationship between expected and observed sets of data. In general, solving a system of linear equations ($Ax = b$) will result no solution or infinite solutions. This always happens when (a) the number of constraints (equations) greater than the number of variables (overdetermined) or (b) the number of variables greater than the number of constraints (underdetermined). The way to find the best approximate solution is called the linear least square approximation. As mentioned in Section §3.2.5, in our study, we have 5 goals and n number of test cases (where $100 \leq n \leq 150$). Thus, our equation system contains 5 equations (5 goals) and n variables (where n must greater than 5). In this case, finding possible selections of test cases becomes a **underdetermined least square approximation**.

Problem: Find x that minimize

$$\min \|Ax - b\|_2^2 + \|x\|_2^2 \tag{10}$$

In the formulation $Ax = b$, where x is an outcome vector of test cases, we want to predict the value (0/1) for each entry of x. 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

Table 2: Summary of number of I/O signals, number of test cases, and number of mutants in four case studies

Project Name	Two Tanks	CW	AC Engine	EMB
# Input Signals	11	15	4	1
# Output Signals	7	4	1	1
# Test Cases	150	133	120	150
# Mutants	6	96	12	18

coefficient → higher effect to the final score with coefficient from 0 → 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 predicted 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 4, **DoLesS** selects the Pareto front set[1] from the final population as Arrieta et al. did in their study [3]. All evaluations are made through 20 repeats for each of algorithm.

4 EXPERIMENTAL SETUP

4.1 Case Studies

We use four Cyber-physical system (CPS) models to evaluate our proposed approach. These four models come from the previous state-of-the-art study [3]. We implement the test cases and mutants that Arrieta et al. generated from these four models[2]. The summary of number of initial test cases and number of mutants are shown in Table 2. In that table:

- Two Tanks project is a model that simulate the incoming and outgoing flows of the tanks [30];
- CW project is a model that simulate the electrics and mechanics of four car windows [3];
- AC Engine project is a model that simulate some safety functionalities in the AC engine [4, 6];
- and EMB project simulates the software model controller which includes a continuous PID controller and a discrete state machine [27].

At first glance, the case studies in Table 2 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 2 are so complex that, for the purposes of test suite minimization, they defeated state-of-the-art optimizers (NSGA-III and MOEA/D).

[1]In multi-objective optimization, the Pareto front of a set of individuals are all examples not dominated by anything else.
[2]https://github.com/aitorarrietamarcos/IST2019Paper

4.2 Performance Criteria

To evaluate the selected test cases, we use two evaluation metrics from prior work [3]. 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 [3], 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 [31]. 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

Table 3: RQ1 results: Reproduction results of Arrieta et al.'s study [3]. 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).

Project	Approach	TET-	MS+
Twotanks	NSGA-II	0.30	1
	NSGA-III	0.49	1
	MOEA/D	0.54	1
CW	NSGA-II	0.39	0.99
	NSGA-III	0.61	0.98
	MOEA/D	0.68	0.99
ACEngine	NSGA-II	0.38	0.73
	NSGA-III	0.61	0.72
	MOEA/D	0.65	0.73
EMB	NSGA-II	0.37	1
	NSGA-III	0.54	1
	MOEA/D	0.63	1

Table 4: RQ2 results: Scores of five effectiveness measurement metrics calculated by sets of selected test cases. All entries report the median score of 20 repeats. In the title row, the metric with "-" means we want to minimize that metric and the metric with "+" means we want to maximize that metric. The light gray cell in each project means that approach wins over another approach significantly (as computed by the statistical method of §4.3) in that metric. Last column counts the number of wins for each approach.

Project	Approach	Best Combination	Time-	Discontinuity+	Infinity+	Instability+	MinMax+	Wins
Twotanks	NSGA-II	Time, Infinite, Minmax	0.30	0.54	0.65	0.55	0.65	**1**
	DoLesS	-	0.30	0.56	0.67	0.56	0.67	**5**
CW	NSGA-II	Time, Instability	0.39	0.55	0.34	0.64	0.34	**2**
	DoLesS	-	0.36	0.50	0.58	0.44	0.58	**3**
ACEngine	NSGA-II	Time, Discontinuity, Instability	0.38	0.53	0.49	0.49	0.49	**4**
	DoLesS	-	0.30	0.47	0.44	0.41	0.44	1
EMB	NSGA-II	Time, Instability	0.37	0.50	0.49	0.57	0.50	1
	DoLesS	-	0.36	0.61	0.61	0.48	0.61	**4**

different (by more than a small effect size) to the values of candidate at position $i-1$ [24].

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 [41]. 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 [40, 46]:

$$E(\Delta) = \frac{|l_1|}{|l|} abs(\overline{l_1} - \bar{l})^2 + \frac{|l_2|}{|l|} abs(\overline{l_2} - \bar{l})^2 \tag{11}$$

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 [46]. 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 $|B|$:

$$Delta = \frac{\sum_{x \in A} \sum_{y \in B} \begin{cases} +1, & \text{if } x > y \\ -1, & \text{if } x < y \\ 0, & \text{if } x = y \end{cases}}{|A||B|} \tag{12}$$

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 [25] in the above formula. This hypothesis test and its effect size is supported by Hess and Kromery [21].

5 RESULTS

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

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 [3]. Table 3 shows our replication results with 20 repeats (with different random number seeds). Two algorithms differ significantly if they separate in different ranks in the Scott-Knott analysis.

As seen in Table 3, we can found the approach with NSGA-II that Arrieta et al. proposed [3] performs better than other multi-goal optimizers (MOEA/D and NSGA-III). Specifically, NSGA-II has higher performance in both *test execution time* and *mutation score* in four case studies. However, even though NSGA-II beats the other methods, we cannot sanction its use. NSGA-II has all the problems discussed in §2. Specifically, NSGA-II can only handle pairs of goals. Hence, it has to be re-run multiple times to explore all pairs of five goals. As shown below, we can achieve better results, orders of magnitude faster. Therefore, we can answer **RQ1** as follow:

Test case selection for multi-goal cyber-physical models is a *hard problem* that cannot be solved by merely applying, off-the-shelf, the latest optimizer technology.

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.

RQ2: Can DoLesS Find Better Ways to Select Test Cases which Result in Test Suites with Higher Effectiveness Measure (Objective) Scores? To answer RQ2, we calculate the scores of effectiveness measures *(test execution time, discontinuity, growth to infinity, instability, minimum and maximum difference)* after we generate the subsets of selected test cases.

Table 4 shows our simulation results. For each project, we use Scott-Knott statistical analysis to compare the performance across 20 repeats. Table 4 reports the median scores for each effectiveness measurement metric. To visualize the final results, we mark the winning approach in each metric by light gray , and count the number of wins in the last column.

As seen in Table 4, **DoLesS** wins in three out of four projects (in Twotanks, CW, and EMB). Moreover, in Twotanks and EMB, our proposed approach results higher scores on most of the effectiveness measurement metrics while previous approach only wins in one (**DoLesS** wins in all effectiveness measurement metrics in Twotanks and wins in 4 out of 5 effectiveness measurement metrics in EMB). These outcomes can indicate that in most of the cases, our proposed approach gets significant improvement than previous approach in these five metrics. Moreover, in half projects, the state-of-the-art approach concentrates on optimizing the *Time* metric and *Instability* metric (because of the algorithm design). However, by using **DoLesS**, we can find that most of the "goals" are equally optimized. Hence, **DoLesS** can handle multiple goals all the time while state-of-the-art method can only handle one or two goals in some cases.

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

DoLesS can find *better* test selections than state-of-the-art approach in terms of five effectiveness measurement metrics.

RQ3: Do selected test cases by DoLesS beat the prior state-of-the-art? To answer RQ3, 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 5 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 5, **DoLesS** gets better performance in both two evaluation metrics in two out of four projects (ACEngine & EMB). Moreover, in Twotanks, **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 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 **RQ3** as follow:

Table 5: RQ3 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	**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**

Table 6: RQ4 results: Runtime comparison for our proposed DoLesS and the state-of-the-art approach. The light gray cell marked the fastest approach in each project. The last column marks how many times DoLesS faster than state-of-the-art.

Project	Approach	RunTime (s)	Speed Up (times faster)
Twotanks	NSGA-II	11964.6	83
	DoLesS	144.7	
CW	NSGA-II	15409.9	362
	DoLesS	42.6	
ACEngine	NSGA-II	14042.1	179
	DoLesS	78.6	
EMB	NSGA-II	12585.6	319
	DoLesS	39.5	

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.

RQ4: Is DoLesS far more efficient than prior state-of-the-art in terms of running time? To answer RQ4, 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 both four projects, **DoLesS** runs significantly faster (80-360 times faster) than the previous 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 [32]. 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 [3]. However, in our approach, we just need continuous domination to find "ideal goals" and approximate corresponding test cases inversely.

By summarizing above findings, we answer **RQ4** as follow:

In **both four** 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.
- **7** goals will result **21** different combinations of 2 or 3 objectives.
- **9** goals will result **36** different combinations of 2 or 3 objectives.

- 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 THREATS TO VALIDITY

This section discusses issues raised by Feldt et al. [18].

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 [39, 40] 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. 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.

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. [3] 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 four real-world Simulink cyber-physical systems. When applying our method into other case studies, these concerns may raised: (a) **DoLesS** may not be applicable in projects which cannot obtain the effectiveness measurement data; (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; (c) Other algorithms may perform better if other information is utilized.

7 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 word, 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. [3] 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. [3]. 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 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) requires far more fast 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 [26, 33, 48, 51]. 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 blinded for review.

REFERENCES

[1] Amritanshu Agrawal, Tim Menzies, Leandro L Minku, Markus Wagner, and Zhe Yu. 2020. Better software analytics via "DUO": Data mining algorithms using/used-by optimizers. *Empirical Software Engineering* 25, 3 (2020), 2099–2136.

[2] Bestoun S Ahmed. 2016. Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing. *Engineering Science and Technology, an International Journal* 19, 2 (2016), 737–753.

[3] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Ainhoa Arruabarrena, Leire Etxeberria, and Goiuria Sagardui. 2019. Pareto efficient multi-objective black-box test case selection for simulation-based testing. *Information and Software Technology* 114 (2019), 137–154.

[4] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. 2017. Employing multi-objective search to enhance reactive test case generation and prioritization for testing industrial cyber-physical systems. *IEEE Transactions on Industrial Informatics* 14, 3 (2017), 1055–1066.

[5] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. 2016. 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*. 297–306.

[6] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. 2019. Search-Based test case prioritization for simulation-Based testing of cyber-Physical system product lines. *Journal of Systems and Software* 149 (2019), 1–34.

[7] Nguyen Thanh Binh, Khuat Thanh Tung, et al. 2016. A novel fitness function of metaheuristic algorithms for test data generation for simulink models based on mutation analysis. *Journal of Systems and Software* 120 (2016), 17–30.

[8] David Binkley. 1995. Reducing the cost of regression testing by semantics guided test case selection. In *Proceedings of International Conference on Software Maintenance*. IEEE, 251–260.

[9] Emanuela G Cartaxo, Patrícia DL Machado, and Francisco G Oliveira Neto. 2011. On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability* 21, 2 (2011), 75–100.

[10] Jianfeng Chen, Vivek Nair, Rahul Krishna, and Tim Menzies. 2018. "Sampling" as a baseline optimizer for search-based software engineering. *IEEE Transactions on Software Engineering* 45, 6 (2018), 597–614.

[11] Tsong Yueh Chen and Man Fai Lau. 2001. Test case selection strategies based on boolean specifications. *Software Testing, Verification and Reliability* 11, 3 (2001), 165–180.

[12] Shafiul Azam Chowdhury, Soumik Mohian, Sidharth Mehra, Siddhant Gawsane, Taylor T Johnson, and Christoph Csallner. 2018. Automatically finding bugs in a commercial cyber-physical system development tool chain with SLforge. In *Proceedings of the 40th International Conference on Software Engineering*. 981–992.

[13] Kalyanmoy Deb and Himanshu Jain. 2013. 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* 18, 4 (2013), 577–601.

[14] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[15] Daniel Di Nardo, Nadia Alshahwan, Lionel Briand, and Yvan Labiche. 2015. Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system. *Software Testing, Verification and Reliability* 25, 4 (2015), 371–396.

[16] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 235–245.

[17] Emelie Engström, Per Runeson, and Mats Skoglund. 2010. A systematic review on regression test selection techniques. *Information and Software Technology* 52, 1 (2010), 14–30.

[18] Robert Feldt and Ana Magazinius. 2010. Validity threats in empirical software engineering research-an initial survey.. In *Seke*. 374–379.

[19] Carlos A González, Mojtaba Varmazyar, Shiva Nejati, Lionel C Briand, and Yago Isasi. 2018. Enabling model testing of cyber-physical systems. In *Proceedings of the 21th ACM/IEEE international conference on model driven engineering languages and systems*. 176–186.

[20] Mats Grindal, Birgitta Lindström, Jeff Offutt, and Sten F Andler. 2006. An evaluation of combination strategies for test case selection. *Empirical Software Engineering* 11, 4 (2006), 583–611.

[21] Melinda R Hess and Jeffrey D Kromrey. 2004. 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, 1–30.

[22] Yue Jia and Mark Harman. 2010. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering* 37, 5 (2010), 649–678.

[23] Remo Lachmann, Michael Felderer, Manuel Nieke, Sandro Schulze, Christoph Seidl, and Ina Schaefer. 2017. Multi-objective black-box test case selection for system testing. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1311–1318.

[24] Xiao Ling, Rishabh Agrawal, and Tim Menzies. 2021. How Different is Test Case Prioritization for Open and Closed Source Projects. *IEEE Transactions on Software Engineering* (2021).

[25] Guillermo Macbeth, Eugenia Razumiejczyk, and Rubén Daniel Ledesma. 2011. Cliff's Delta Calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica* 10, 2 (2011), 545–555.

[26] Camila Loiola Brito Maia, Rafael Augusto Ferreira Do Carmo, Fabrıcio Gomes de Freitas, Gustavo Augusto Lima De Campos, and Jerffeson Teixeira De Souza. 2009. A multi-objective approach for the regression test case selection problem. In *Proceedings of Anais do XLI Simposio Brasileiro de Pesquisa Operacional (SBPO 2009)*. 1824–1835.

[27] Reza Matinnejad, Shiva Nejati, and Lionel C Briand. 2017. Automated testing of hybrid Simulink/Stateflow controllers: industrial case studies. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 938–943.

[28] Reza Matinnejad, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. 2015. Effective test suites for mixed discrete-continuous stateflow controllers. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 84–95.

[29] Reza Matinnejad, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. 2016. Automated test suite generation for time-continuous simulink models. In *proceedings of the 38th International Conference on Software Engineering*. 595–606.

[30] Claudio Menghi, Shiva Nejati, Khouloud Gaaloul, and Lionel C Briand. 2019. 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*. 27–38.

[31] Nikolaos Mittas and Lefteris Angelis. 2012. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. *IEEE Transactions on software engineering* 39, 4 (2012), 537–551.

[32] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2017. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Transactions on Software Engineering* 44, 2 (2017), 122–158.

[33] Annibale Panichella, Rocco Oliveto, Massimiliano Di Penta, and Andrea De Lucia. 2014. Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Transactions on Software Engineering* 41, 4 (2014), 358–383.

[34] Mike Papadakis, Yue Jia, Mark Harman, and Yves Le Traon. 2015. 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, 936–946.

[35] Dipesh Pradhan, Shuai Wang, Shaukat Ali, and Tao Yue. 2016. Search-based cost-effective test case selection within a time budget: An empirical study. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 1085–1092.

[36] Gregg Rothermel, Mary Jean Harrold, and Jeinay Dedhia. 2000. Regression test selection for C++ software. *Software Testing, Verification and Reliability* 10, 2 (2000), 77–109.

[37] Goiuria Sagardui, Joseba Agirre, Urtzi Markiegi, Aitor Arrieta, Carlos Fernando Nicolás, and Jose María Martín. 2017. 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, 1–6.

[38] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. 2013. 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, 492–501.

[39] Huy Tu and Vivek Nair. 2018. While Tuning is Good, No Tuner is Best. In *FSE SWAN*.

[40] Huy Tu, George Papadimitriou, Mariam Kiran, Cong Wang, Anirban Mandal, Ewa Deelman, and Tim Menzies. 2021. Mining Workflows for Anomalous Data Transfers. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 1–12. https://doi.org/10.1109/MSR52588.2021.00013

[41] H. Tu, Z. Yu, and T. Menzies. 2020. Better Data Labelling with EMBLEM (and how that Impacts Defect Prediction). *TSE* (2020). https://doi.org/10.1109/TSE.2020.2986415

[42] Tobias Wagner, Nicola Beume, and Boris Naujoks. 2007. Pareto-, Aggregation-, and Indicator-Based Methods in Many-Objective Optimization. In *Evolutionary Multi-Criterion Optimization*, Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu, and Tadahiko Murata (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 742–756.

[43] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. 2013. Minimizing test suites in software product lines using weight-based genetic algorithms. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 1493–1500.

[44] W Eric Wong, Joseph R Horgan, Saul London, and Aditya P Mathur. 1998. Effect of test set minimization on fault detection effectiveness. *Software: Practice and Experience* 28, 4 (1998), 347–369.

[45] W Eric Wong, Joseph Robert Horgan, Aditya P Mathur, and Alberto Pasquini. 1997. 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, 522–528.

[46] Tianpei Xia, Rahul Krishna, Jianfeng Chen, George Mathew, Xipeng Shen, and Tim Menzies. 2018. Hyperparameter optimization for effort estimation. *arXiv preprint arXiv:1805.00336* (2018).

[47] Zhiwei Xu, Kehan Gao, and Taghi M Khoshgoftaar. 2005. 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, 120–125.

[48] Shin Yoo and Mark Harman. 2010. Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *Journal of Systems and Software* 83, 4 (2010), 689–701.

[49] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability* 22, 2 (2012), 67–120.

[50] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation* 11, 6 (2007), 712–731.

[51] Wei Zheng, Robert M Hierons, Miqing Li, XiaoHui Liu, and Veronica Vinciotti. 2016. Multi-objective optimisation for regression testing. *Information Sciences* 334 (2016), 1–16.

[52] Eckart Zitzler and Simon Künzli. 2004. Indicator-Based Selection in Multiobjective Search. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 832–842. https://doi.org/10.1007/978-3-540-30217-9_84