The Journal of Systems & Software 205 (2023) 111802

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Generic and industrial scale many-criteria regression test selection ☆

Felix Dobslaw [a,*], Ruiyuan Wan [b], Yuechan Hao [b]

[a] Department of Computer and System Sciences, Mid Sweden University, Östersund, Sweden
[b] Huawei Cloud Computing Technologies Co., Ltd, Bejing, China

ARTICLE INFO

Article history:
Received 10 November 2022
Received in revised form 1 May 2023
Accepted 14 July 2023
Available online 20 July 2023

CCS Concepts:
Software and its engineering: Software testing and debugging

Keywords:
Software testing
Regression testing
Test case selection
Industrial-scale optimization

ABSTRACT

While several test case selection algorithms (heuristic and optimal) and formulations (linear and non-linear) have been proposed, no multi-criteria framework enables Pareto search — the state-of-the-art approach of doing multi-criteria optimization. Therefore, we introduce the highly parallelizable, openly available Many-Criteria Test-Optimization Algorithm (MC-TOA) framework that combines heuristic Pareto search and optimality gap knowledge per criterion. MC-TOA is largely agnostic to the criteria formulations and can incorporate many criteria where existing approaches offer limited scope (single or few objectives/constraints), lack flexibility in the expression and assurance of constraints, or run into problem complexity issues. For two large-scale systems with up to six criteria and thousands of system test cases, MC-TOA not only produces, over the board, superior Pareto fronts in terms of HVI score compared to the state-of-the-art many-objective heuristic baseline, it also does that within minutes of runtime for worst-case executions, i.e., assuming that a regression affects the entire test-suite. MC-TOA depends on convex solvers. We find that the evaluated open-source solvers are slower but suffice for smaller systems, while being less robust for larger systems. Linear formulations execute faster and obtain near-optimal results, which led to faster and better overall convergence of MC-TOA compared to integer formulations.

Editor's note: Open Science material was validated by the Journal of Systems and Software Open Science Board.

1. Introduction

The problem of selecting which test cases are most important to execute at a specific time, e.g., given a specific change to the software system, has been studied extensively. While software-developing organizations today increasingly write test cases and continuously run them to find defects and ensure quality, their associated costs tend to increase over time, often significantly. To trigger executions of entire test suites for large corporations is expensive in terms of resources used but also as an issue of time. It has long been recognized that nightly regression runs of entire test suites may not be feasible as the testing can take days, weeks, or even months to run (Rothermel et al., 2001; Vierhauser et al., 2014; Herzig et al., 2015; Garousi et al., 2018).

Thus, only tests most relevant to recent code changes should be selected. However, this is challenging to decide due to code fragments' inter-dependency. Faults may have unexpected characteristics that are masked in some test cases. Also, passing unit tests does not guarantee that system- or integration-level tests would also pass. Of course, a key goal is often to select the test cases that are most likely to fail. However, industrial practice often includes many other objectives, e.g., covering all the changed functions and constraints, a maximum total testing time of 8 h for a nightly test run, or 30 min for test runs during a work day.

Out of the three regression testing categories (Yoo and Harman, 2012), test suite reduction (TSR), test case prioritization (TCP), and test case selection (TCS), we address the latter to extend on and generalize existing work (Lin et al., 2018; Özener and Sözer, 2020). Introducing a many-objective framework in this paper, we focus on the problem of disclosing faults in large code bases affected by many code changes in short periods, which pose an acknowledged issue for software creators in practice (Bin Ali et al., 2019; Coviello et al., 2020; Hierons et al., 2020). Ideally, to be effective and practical for large industrial systems, a method to select test cases should:

- Allow *multiple/many objectives and constraints* to govern the selection process.
- *Support objectives and constraints to be flexibly selected, deselected or added* depending on the selection context.
- Support *coverage targets of multiple types*, e.g., at the function, sub-system, and feature levels.

☆ Editor:Antonia Bertolino.

* Corresponding author.
E-mail addresses: felix.dobslaw@miun.se (F. Dobslaw), wanruiyuan@huawei.com (R. Wan), haoyuechan@huawei.com (Y. Hao).

https://doi.org/10.1016/j.jss.2023.111802

- *Handle large problems* with thousands of test cases and coverage targets in even higher numbers.
- *Be fast enough* for frequently recurring and short-term selection tasks.
- *Give optimality guarantees* or indicate how far from optimal the proposed solution is.

A search criterion can come in two forms: a constraint that must be fulfilled (e.g., coverage of all changed code segments, execution time threshold) or an objective (e.g., minimized execution costs, maximized overall function coverage). Existing research focuses on single or few objectives/constraints and evaluates their algorithms on small problems or open-source scenarios with simplified assumptions (such as knowledge about faults in the problem formulation Lin et al., 2018; Özener and Sözer, 2020). However, Attempts have been made to include multiple objectives and constraints in the problem formulations and have shown improvements over simpler formulations. Another issue is that heuristic approaches proposed in the literature are limited in ensuring constraint satisfaction (e.g., Hierons et al., 2020; Coviello et al., 2020).

A limitation in many existing works is that problem formulations depend on weights for the multiple objectives, in which their assignment and the related threats are seldom discussed. Chen and Li (2022) found that apriori weighting *can be harmful*, to put it in their words. Even if clear preferences can be expressed as weights, which becomes harder with more objectives, doing so should only be done in situations where time budget is scarce and the problems very time-consuming to solve (Chen and Li, 2022). If optimality is of most importance, though, the takeaway from their broad empirical study is not to fix the weights but to go for the Pareto search instead, which produces better results in 77% of the cases. Thus, weighing the options regarding the objectives is better done on different competing solutions with their trade-off characteristics. This requires solving multiple objectives and possibly problem formulations using automated weight selection, thereby increasing the complexity of the problem.

We, therefore, contribute with a generic many-criteria algorithmic framework that can be tailored with arbitrary objectives and/or constraints. Existing work has focused on fixed and small numbers of criteria to select subsets from test suites that retain fault detection capabilities but at lower execution time costs (Mirarab et al., 2011; Mondal et al., 2015; Wang et al., 2016; Arrieta et al., 2019). However, the former approaches have yet to integrate heuristic search and convex solvers in a Pareto-optimal method for test set selection. We evaluate our method compared to a state-of-the-art many-objective evolutionary algorithm (EA) on real industry-size data sets with constraints used in production today to highlight the overall challenges in tailoring EAs to regression problems but also to show the impact on overall performance.

Our objectives with this paper are threefold. First, we introduce the MC-TOA framework, which is customizable as it (a) can incorporate existing literature formulations and (b) has built-in support for parallelization and model-based approaches to weigh objectives during search. With user-defined bounds needed for normalization, the multi (many) objective approaches allow us to translate constraints into concurrent objectives for exploration purposes. Going back and forth between objectives combinations lead to a diversified Pareto-front over time, while all solutions are valid in terms of the arbitrary domain-specific constraints.

Second, we evaluate the framework on two large real-world scenarios with realistic objectives and constraints used in industry. We reveal practical implications of combining heuristic search with optimal convex solvers for industrial scale TCS, empirically showing how different solvers (open-source and commercial) and problem formulation (linear vs. integer) affect the outcome not only theoretically or on small projects.

Third, we explain and show the problems that specialized many-objective heuristic methods struggle with regarding constraint satisfaction and performance for industrial many-criteria situations where our hybrid framework uses Pareto-optimal search to offer proven optimality gaps with a constraint satisfaction guarantee. There is no single best test set — the problem remains stochastic, and the tester must weigh the options based on the explored optimality landscape and not on a-priori selected weights in a scalarization.

The remainder of the paper is structured as follows. Section 2 provides background and related work of the optimization field for regression testing with a focus on TCS. We position this study's contributions there. Section 3 describes the multi-criteria test set optimization algorithm (MC-TOA) framework. The section even contains the mathematical problem formulation used in this paper. The experimental evaluation, including research questions and the industrial case, gets detailed in Section 4. Our findings are presented in Section 5 and discussed as responses to our research questions in Section 6. The paper concludes in Section 7 with a summary and an outlook for possible and proposed future work.

2. Test case selection

On a high level, test selection problems describe the challenge of choosing and/or prioritizing tests for execution in situations where running all tests is not feasible/desired, for instance, in the scope of regression testing. TSR, TCP, and TCS are well-understood problems covered by a growing literature body (Yoo and Harman, 2012; Khatibsyarbini et al., 2018; Khan et al., 2018). While all are related and solutions for one can often be aligned for another, we here focus on TCS problems for regressions over large test suites with frequent and large code updates per regression.

The objective of running a test suite is to disclose recently introduced faults. Under the assumption that a test case revealing a fault exists, the goal is then to maximize the likelihood that such a test case is part of the selected subset. This activity is speculative and builds on several heuristics or *metrics* that help rank individual tests and test sets against one another. Commonly applied metrics to quantify the likelihood of failure disclosure are the historical fault detection rates of test cases, diversity, as well as various coverage metrics such as code-segment coverage, branch coverage, or function coverage. Traditionally, these metrics have been used as single objectives in isolation, for instance, through greedy search algorithms (Chen and Lau, 1970), but have recently been shown to be combined for overall improved test sets (Mondal et al., 2015; Arrieta et al., 2019), which is why an integrated approach to TCS is desirable as appropriate decision support for organizations.

While much of the related literature has been focused on integrating multiple objectives into single-objective formulations (Khan et al., 2018), we here focus on the multi-criteria works that extend to general formulations and transformations as well as using multi-/many-objective approaches, which are most promising to extend to a combination of many generic criteria expressed both in objective and constraint form.

2.1. Heuristic optimization

Heuristic approaches to TCS were already introduced about 50 years ago (Chen and Lau, 1970), synthesized (Tallam and Gupta, 2005), and extended through the years (e.g., Smith and

Kapfhammer, 2009). Traditionally, Existing multi-objective attempts have primarily been based on meta-heuristics, either tailored (e.g., Mirarab et al., 2011) or from the shelf (e.g., Arrieta et al., 2019). Mondal et al. (2015) compared a Genetic Algorithm (GA) to target coverage and diversity in test sets in isolation, finding that both were less effective than a combined formulation, revealing the potential of multi-objective formulations. Arrieta et al. (2019) introduced a multi-objective black-box-only approach. With access to historical execution data and no access to coverage information, they could select better test sets than coverage-based approaches for all six included objectives. TCS approaches that integrate black-box and white-box information may have an even greater potential to find the most suitable regression test suites. At the same time, optimizing objectives independently may return many so-called non-dominating sets in the Pareto-front, i.e., solutions that are better for at least one objective than any of the found solutions but worse off for others. Pareto search enables more optimal solutions at the cost of increased complexity in decision-making. Wang et al. conducted a prioritization study applying heuristics on four objectives (Wang et al., 2016), showing that a random weight GA (Konak et al., 2006) outperforms all other six approaches, among them the popular multi-objective genetic algorithm by the name NSGA-II (Deb et al., 2002). The objectives were built on a measure of runtime cost, limits on the number of tests to be executed, resource costs, and fault detection.

Genetic Algorithms are a type of EA, a popular meta-heuristic approach that evolves a population of candidate solutions, mimicking natural selection and genetic manipulation to produce ever-fitter offspring, e.g., test sets better meeting the included testing criteria. NSGA-II has been used in many previous studies as a baseline, but it leads to substantially slower convergence of the search, thereby running into issues of scalability for more than three objectives — for instance when used for automated test-case generation (Panichella et al., 2017). The more recently introduced Borg EA *framework* (Hadka and Reed, 2013) offers better performance and scalability characteristics overall, also due to its built-in adaptive choice of operators, restart strategy, and heuristic optimization of the Pareto-front management throughout search. This makes Borg-EA a (more) suitable candidate for multi- and many-objective optimization as a baseline regarding heuristic search.

In Coviello et al. (2020), the authors investigate how the relaxation of constraints, such as coverage from a strict 100%, affects the overall performance of fault detection capabilities. Their proposed clustering-based approach showed more stable results over the test set of 19 openly available projects. Results were compared to some of the greedy algorithms mentioned above. The transformation of strict constraints into objectives led to a 30% reduction in test set size with negligible impact on fault detection. The caveat here is that the set does not guarantee the coverage criteria.

All EAs have in common that including strict constraints requires tailoring to the specific constraints. The EA community has devoted great efforts to constraint handling (Coello Coello, 2016) as no general, easy, and deterministic approaches exist. Hierons et al. investigated tens of features as objectives comparing a variety of EAs (Hierons et al., 2020) for TCS and TCP in the context of software product lines. No formal constraints were formulated. In fact, the assurance of constraints is a major deficit of heuristic solutions usually needing problem-specific tailoring (Garousi et al., 2018). At the same time, incorporating constraints is a clear strength of complex solvers that can handle linear, integer, or non-linear formulations, as explained in the next section.

In TCS, coverage criteria are strict in that they are not to be optimized but guaranteed — e.g., all branches or methods must be covered by at least one test case in a regression. These constraints, in particular, if combined, pose a severe challenge and define limitations to EA-based solutions. For better convergence and initial constraint fulfillment, it is therefore common to inject pre-processed individual solutions into the initial population to increase the chances of finding valid solutions and obtain good convergence thereafter. This initial process is also called seeding, and depending on the criteria at hand, multiple strategies can be used (Arrieta et al., 2023). It should be noted that seeding strategies can even lead to worse results (see, e.g., Arrieta et al., 2023). The more constraints, the more pre-processing is required, and may even be necessary throughout evolution to ensure a fit phenotypical population by different kinds of repair methods or penalization strategies in the objective function(s) (Coello Coello, 2016).

EAs cannot either guarantee optimal solutions. Depending on the encoded search goals, an optimal solution could be a test set of the *fewest test cases* or the *shortest expected execution* time that retains the same coverage of functions as the entire test suite or a pre-defined percentage of the coverage. By default, they cannot even express how close their solutions are to the optimum — or the optima in the case of multi-objective optimization.

2.2. Linear and integer programming optimization

In mathematics, linear programming (LP) solves linear programs, i.e., linear objective functions over linear constraints. Classical algorithms, such as the Simplex Method, can obtain a proven optimal solution for a defined linear program much faster than random search — however, in polynomial time. It turns out that many of the test selection problems can be expressed as a subset of linear programs called integer programs (IP), where the result variables are limited to discrete integer space, which makes the problems harder to solve optimally (Wolsey, 1998) — in *non*-polynomial time. For a given coverage metric, applying IP, we can obtain a test set of the shortest theoretical length, often referred to as an optimal solution. Given these optimal solutions or *bounds*, we can now calculate how far off an existing solution from a heuristic algorithm is. This helps us compare methods, and we call this difference the *optimality gap*, as standard in the optimization literature (Wolsey, 1998). It should be noted that such a solution is only optimal in terms of length; alternative test sets of the same length with better fault detection capability may exist.

A problem can also be relaxed from IP to LP by allowing the search space of binary test activations be defined over $[0, 1]$ instead of $\{0, 1\}$, which may speed up the search. In this context, the resulting outcome of LP formulations, also called relaxations, are approximations that can be heuristically translated into test sets, for instance, by using randomized rounding. As mentioned above, solving an LP formulation is done in polynomial time and is, therefore, at least in theory, more efficient than solving the IP formulation.

Many alternative IP solver implementations, both open-source and commercial, exist. Commercial solvers often cover a broader spectrum of more complex problems, such as non-convex problems with non-linear objectives and/or constraints. To name a few, there are open-source Cbc,[1] and Clp[2] as well as commercial Gurobi[3] and CPLEX.[4] They differ in the heuristics they employ to find solutions efficiently (using variants of *cutting planes* and *branch and bound*). Still, for the same given problem, they will

[1] https://github.com/coin-or/Cbc
[2] https://github.com/coin-or/Clp
[3] https://www.gurobi.com/
[4] https://www.ibm.com/analytics/cplex-optimizer

effectively produce results of the same quality.[5] Within the scope of this paper, we refer to them as *convex solvers*, even though they may be much more capable than that.

Requirements such that a test set may not execute for longer than a certain time budget can be expressed as constraints in IP/LP. Combining multiple constraints, for instance, in fault detection capabilities, may result in long/unknown runtimes and the potential result of the formulation being infeasible.

IP-based solutions have shown promise more broadly for software testing challenges than heuristics. Dong et al. verified mixed integer programmings (MIP) superiority to the NSGA-II heuristic, even for small instances, when solving the so-called Next Release Problem with up to three objectives (cost, time, urgency) (Dong et al., 2022).

TCS can be described without constraints but not solved effectively in a naive formulation. Let $\hat{O}$ be the set of objectives formulated as maximization problems for the TCS problem at hand.[6] We could formulate the general problem of finding an optimal selection of tests expressed as binary activations in x for a test suite T in a single formula:

$$\max_{x} \qquad o_{np}(x) = \prod_{\hat{o}\in\hat{O}} \hat{o}, \qquad x \in \{0,1\}^{|T|} \tag{1}$$

Apart from not being suitable for our general problem since we have several constraints, this formulation is non-linear and does not consider balancing the objectives. What is worse, no polynomial-time algorithm is known and possibly does not exist (Cook, 2006) to solve problems of this form. A generic solution is required for software systems of ever-growing complexity in the number of test cases and functions.

Applying LP to improve on GAs for TCS was demonstrated in Williams (2013) and for TCP in Zhang et al. (2009) under the stringent consideration of time limits in single-objective formulations. Mirarab et al. expand on those ideas for two objectives and vary the objective weights in the scalarization over time to produce a Pareto-optimal set (Mirarab et al., 2011). For polynomial time calculation, they went for an approximate LP formulation, suggested a heuristic to select a single solution from the set, and evaluated it based on fault detection capabilities over five synthetic projects. Due to computational limitations, they obtain approximate solutions by transforming the IP into an LP. The solutions are, therefore, not necessarily optimal and require conversion into binary space. Even though not explained in Mirarab et al. (2011), this can be achieved by rounding as explained in the method section in this paper.

In the Nemo framework, Lin et al. (2018) compare different IP-based formulations for multi-constraint TSR, arguing the importance of considering non-linear interactions among test cases, exemplifying this by an artificially crafted fault coverage example. They outline a high-level process that involves the transformation of constrained single objective non-linear formulations into linear ones that can be solved with simpler convex solvers and, oftentimes, faster. However, how multiple objectives or industry-scale test suites are handled is not detailed or investigated. In standard MIP, these binary formulations can trivially transform from non-linear to linear. Özener and Sözer (2020) respond to Lin et al. (2018) by further contextualizing the example and showcase that the non-linear formulation (and neither its linearization) is sufficient for optimality still, and do offer a linear formulation that resolves the sub-optimality through penalties in the objective function, without the need of a transformation from non-linear to linear. Özener and Sözer further claimed that their formulation can be extended to multiple criteria without detailing how, why this is the case, its limitations, or its impact on optimality. Further, Xue and Li (2020) offer a thorough analysis of the alternative formulation types to prove that the multi-criteria test-suite minimization problem as posed in Lin et al. (2018) can be expressed linearly without introducing non-linear space complexity in the number of variables, and offer an alternative formulation to Özener and Sözer (2020) for that. They introduce multi-objective IP formulations (called *Big-M* and *Or-relation*), independent of weighing objectives, and produce sound Pareto-fronts for the given criteria as opposed to heuristic approaches including NSGA-II over the same datasets with up to three objectives. While the solution works for binary constraints, many constraints in TSC cannot be expressed that way. The criteria commonly applied in TSC have been investigated and classified into four types: coverage-based, cost-based, effect-based, and others (Khan et al., 2018). In all categories, some variants cannot be expressed as binary constraints, which depicts a limitation in work mentioned above.

While the TCS field has made much progress in the improved problem formulation (Lin et al., 2018; Özener and Sözer, 2020; Xue and Li, 2020), the literature needs further extension of the method and tools to be practically applicable. Existing formulations assume binary encoded fault information which is not available or complete at regression time as acknowledged, e.g., in Özener and Sözer (2020), Xue and Li (2020). It is not only unknown whether a fault exists, but no knowledge of the potential types of faults can be known as the code changes can introduce arbitrary and arbitrarily many such. Xue and Li (2020) express this concern as "faults ... refer to the old known bugs in regression testing, and hence revealing faults does not necessarily lead to detecting unknown bugs." Also, if given much weight, this might result in a poor selection where certain test cases over fixed code segments are included in all regressions, whereas others, for example, new ones, never.

Common in existing approaches is using scalarization to handle multiple objectives, i.e., formulating a single objective function incorporating the different objectives (e.g., Mirarab et al., 2011; Garousi et al., 2018). This simplifies the optimization process and reduces the user's complexity as the solution choice is automated (Garousi et al., 2018). However, since scalarization is done before the search, this may impact the optimality of the solutions as opposed to approaches building a Pareto-front based on various scalarizations (e.g., by varying weights) and presenting the trade-offs to the user. Chen and Li call this a-priori setting potentially *harmful* (Chen and Li, 2022). In extension, the type of constraints can vary, and usually, there are some necessary constraints (e.g., each function must be covered) and some best-effort criteria (e.g., as many functions covered as possible).

EAs require tailored constraint handling, which becomes complex for many constraints, and make it an inflexible solution for TCS. LP/IP-based solutions offer constraint satisfaction guarantees and optimality gap information but at high computational costs. A common framework that combines the strengths of the two approaches, which allows for simple inclusion and adjustment of objectives and constraints with real-world scenarios from industry, would broaden the applicability of the literature for a wider range of scenarios.

Extending on the existing findings above, we in this paper propose a novel generic multi-criteria algorithm that, in support of a convex solver, establishes a Pareto-front of non-dominating test sets and presents them for selection by a tester based on the exploration of varying weight and objective/constraint combinations in the problem formulation. The framework applies partial

[5] For instance, while two correct solvers may return different test sets of minimal length, these test sets are guaranteed to have the same proven minimal length

[6] All min objectives can be aligned to max problems by simple negation (Wolsey, 1998).

problem formulations to obtain theoretical optimal bounds, then zoom in on scalarized formulations of all objectives while varying weights to identify the Pareto-front of valid solutions. Its inherent parallelizability makes it a suitable extension and practical and scientific contribution to the literature.

3. MC-TOA

Section 3.1 defines the problem in mathematical notation. Section 3.2 gives a conceptual introduction to MC-TOA in support of the formerly defined notation to solve constraint-heavy TCS problems, followed by an algorithmic description in Section 3.3 for which we discuss implementation limitations.

3.1. Operationalizing test case selection

We here introduce a common basic formulation of the TCS problem with one objective function and a single coverage constraint to build the vocabulary. All symbols introduced through the paper are in reference Table 1. It is not claimed to be optimal here, but it serves as a starting point for an explanation of MC-TOA and can, in future work, be extended/substituted. Let T be a set of test cases that exercises a program consisting of arbitrarily interlinked functions F to reveal faults.[7] For each change of any $f \in F$ we could run all of T, which may demand long execution times and costly resources. For a change in f, we reduce the set of executed tests to $T_f \subset T$ for which we know, e.g., using static analysis or based on instrumentation and tracing, that all contain tests *cover* f. Further, since function changes over time happen frequently, we bundle recent function changes into a set $F' \subset F$ to reduce redundant test-case execution and seek $T_{F'} \subset T$, *a* set of tests that covers all functions in F'. Even $T_{F'}$ may become too *expensive* to complete, which leads us to the following multi-criteria optimization problem to decide the one test set for execution in the regression.

Now, let E be the binary *execution matrix* of size $|F| \times |T|$ with $\epsilon_{ij} = 1$ if test $t_j \in T$ covers function $f_i \in F$, otherwise $\epsilon_{ij} = 0$. Further, let E' be the matrix of size $|F'| \times |T|$, which reduces E to those rows for which $f_i \in F'$, and have E' shift index appropriately to the correct rows in E. Thus, E' is a folded version of E containing only those functions affected by the change. We seek $T_{F'}$ covering all function changes in F', encoded as a binary vector $x = x_1, \ldots, x_{|T|}$ for which $x_j = 1$ details $t_j \in T_{F'}$ and $x_j = 0$ otherwise. In the simplest formulation for test set minimization, we minimize $|T_{F'}|$, the size of the test set, to obtain a regression schedule of the smallest possible size covering all functions at least once, here expressed as an IP with one constraint group:

$$\min_{x} \quad o_{size}(x) = \sum_{j=1}^{|T|} x_j, \qquad x \in \{0,1\}^{|T|} \tag{2}$$

$$\text{subject to} \quad \sum_{j=1}^{|T|} \epsilon'_{ij} x_j \geq 1, \qquad \forall i \in \{1, \ldots, |F'|\}, \tag{3}$$

with (3) denoting that each function $f \in F'$ must be covered by at least one test $t \in T$. The formulation is a dual variant of the classical combinatorial Set Covering Problem with an equal cost of 1 for all variables.

[7] We here use function coverage, while other types, such as branch-coverage, and code-segment-coverage, can be used interchangeably as most methods, ours included, are agnostic to coverage granularity. This is further discussed in Section 6, Discussion.

Table 1
Reference for all introduced Symbols. Above the line general symbols, below the line MC-TOA formulation specific ones.

Symbol	Meaning
t	Test case
T	Set of test cases
f	Function
F	Set of functions
F'	Set of changed functions, $F' \subset F$
T_F	Subset of T covering all functions in F
E	$\|F\| \times \|T\|$ binary *execution matrix*
E'	$\|F'\| \times \|T\|$ binary *reduced execution matrix*
ϵ_{ij}	Does t_j cover function $f_i \in F$
ϵ'_{ij}	Like ϵ_{ij} with index shift for E'
x	Binary solution vector $x_1, \ldots, x_{\|T\|}$
x_i	1 **iff** $(t_i \in T_{F'})$, else 0
x_T	Vectorized presentation of test set T
$\|T\|$	Size of set T
o	Objective function
$\hat{o}$	Normalized objective function
O	Set of objectives
$\hat{O}$	Set of normalized objectives
$\cup_{\succ}$	Pareto-union operator
$\odot$	Element-wise vector multiplication
λ	Test set size limit
ϕ	Failure rate vector
ϕ_i	Failure rate for test case t_i
a	Minimum function-execution activation
K	Categorization of test cases
κ	Binary category vector
κ_i	Does test case t_i cover category κ
$\overline{\kappa}$	Category coverage lower bound
s_κ	Coverage criterion slack variable

3.2. The framework

At a glance, our proposed multi-criteria test set optimization algorithm (MC-TOA) framework accompanies an arbitrary number of user-defined test criteria as either objective or constraint into a customizable multi-phase search. MC-TOA returns a test set for a regression. Given user-defined time constraints, MC-TOA evolves a set of multi-objective near-optimal solutions and records optimality-gap readings for each individual solution to support the final selection of the test set. The optimality-gap knowledge allows the ranking of solutions to support the user in selecting a test set or to do this automatically (e.g., the one of the highest normalized rank). An algorithmic implementation is detailed in (the next) Section 3.3.

The framework consists of three phases and is illustrated in Fig. 1. The expected outcome is a test set for a regression that respects the relevant restrictions and criteria passed as input. Meta-data such as a binary matrix E' detailing which tests cover which functions and system-specific features that may cover anything from historical timing information to morphology coverage are passed along to guide the search. Finally, the user decides for each criterion whether it is an objective $o \in O$ (*best-effort*) or a constraint $c \in C$ (*a must*) and passes them along to the algorithm as such.

3.2.1. Phase 1: Execution time limits

Phase 1 decides the time limit/for the regression, commonly expressed as a maximum runtime or test set size. This phase can be implemented in many ways. A user may decide a time such as 3 h (top-down), or an algorithm may suggest a minimum runtime or test set size that fulfills all constraints (bottom-up, as exemplified in this paper). Objectives need to be *max-aligned* and *normalized*, with the set of all objectives denoted as $\hat{O}$ and selected from the available test metric criteria. Examples of max-aligned normalized formulations are presented later in Section 4.6.4.

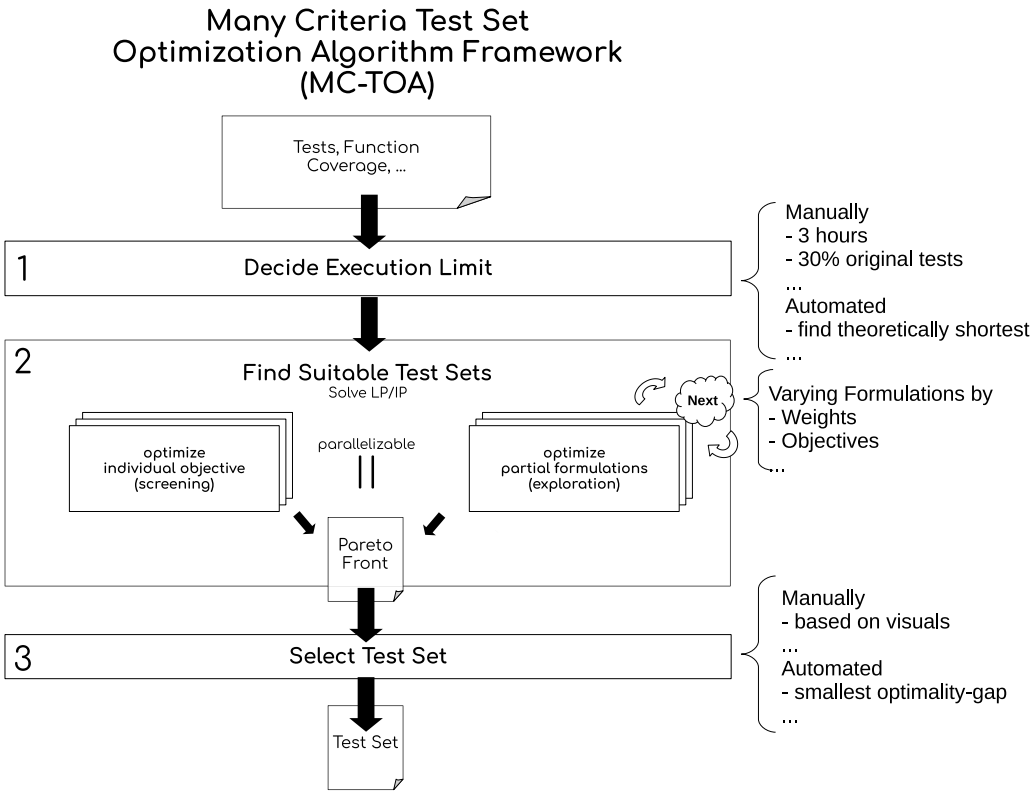

Fig. 1. A conceptual view over the Many-Criteria Test Set Optimization Algorithm Framework. The approach is highly parallelizable as all individual solver runs in both 2. screening and 2. exploration and can independently run in parallel.

3.2.2. Phase 2: Test set search

In Phase 2, the algorithm establishes a Pareto-front of non-dominating solutions by *screening* and *exploring* the search space in support of a convex solver. During *screening*, the convex solver finds partially optimal solutions concerning the individual objective formulations and thus spans the search space. The optima per objective demarcate the solution space in terms of fitness and will be used to assess the quality of any solution in absolute terms in support of the Hypervolume Indicator (HVI) metric, explained in Section 4.6. The *exploration* combines objectives in arbitrary ways to form partial formulations of the problem that produce test sets by varying the objectives. Wang et al. (2016) suggested solving LP for multiple objectives using normalization for a concrete set of objectives and random varying of weights. MC-TOA includes a generalization of that idea to arbitrary objectives and convex solver type, arbitrary constraints, and the possibility to vary weights in the exploration phase according to a learning model (see the cloud of name *Next* in Fig. 1). For the random selection of weights, all screening executions can run parallel as they solve independent problems. With a pre-defined execution plan, the exploration can run all its solvers entirely in parallel or introduce a feedback mechanism to inform the choice of formulation on the results from previous runs (feedback loop during exploration). Screening and exploration can run in parallel, updating the Pareto-front continuously until an execution threshold is reached.

3.2.3. Phase 3: Test set selection

Once a stop criterion terminates Phase 2, the Pareto-front is passed on to Phase 3, where a single test set is selected for execution. This could be done manually, for instance, in support of visualization. Alternatively, a ranking algorithm can choose the best-performing test set given a weighting of all objectives, such as HVI. An alternative voting mechanism implementation is presented in Mirarab et al. (2011).

3.3. The implementation

Algorithm 1 formalizes the implementation of MC-TOA Phase 1 and 2 for the empirical evaluation in this paper. For the purpose of performance comparison among algorithms, we implement Phase 3 using HVI. We define execution limits for the test sets in terms of test set sizes. An alternative would have been to include the expected execution time per test, but we refrained from adding that complexity for the purpose of clarity.

MC-TOA takes six inputs and returns a Pareto-front after Phase 2. The inputs are (i) a convex solver s for the solving of the independent formulations, (ii) a binary execution matrix E' based on which the test coverage can be assessed, (iii) the normalized objectives $\hat{O}$, (iv) the constraints C, and (v) + (vi) are the user-defined factors that allow the customization of the limits of the explored space relative to the scenario at hand. To have access to E', information on E must be available and collected over time. Aspects that influence the choice of the factor variables (v + vi) are explained below. Any information influencing the test set quality is assumed to be included in the objective functions.

3.3.1. Phase 1

In line 1, a minimum size test set $\hat{T}$ under all constraints C, is produced by solver s. If $\hat{O}$ is formulated in LP, this shall be treated as a special case. For LP, solutions cannot directly be translated into test sets, as their outcome is not binary but floating-point based. We use a standard procedure from the literature (see, e.g., Konak et al., 2006), applying a simple heuristic to correct for that by rounding each solution $x' \in [0, 1]^{|T|}$ into a valid solution $x \in \{0, 1\}^{|T|}$ as in:

$$x_i = \lceil x'_i \rceil \forall i \in \{1, \ldots, |T|\}. \quad (4)$$

This is a *defensive* conversion, i.e., smaller test sets might exist and be identified using heuristic search, which we do not apply

Algorithm 1 MC-TOA Implementation for this study

Input: convex solver s, Execution Matrix E', normalized max objectives $\hat{O}$, constraints C, user test set size limit λ_u, slack factor ζ
Output: Pareto-front PF

Phase 1: Decide Execution Limit
1: $\hat{T} = s.solve(E', \hat{o}_{size}, C)$ // optimal size test set
2: $PF = \{T'\}$ // Pareto-front
3: $\lambda = \max(|\hat{T}| * \zeta, \lambda_u)$ // test set size limit (bottom-up + top-down)

Phase 2: Find Suitable Test Sets
2a. Span search space (screening)
4: **for** $\hat{o} \in \hat{O} \setminus \{\hat{o}_{size}\}$ **do**
5: $T' = s.solve(E', \hat{o}, C \cup \{c_{size}(\lambda)\})$
6: $PF = PF \cup_{\succ} \{T'\}$
7: **end for**

2b. optimize partial formulations (exploration)
8: **while not** stop criterion **do**
9: $\hat{o} = \hat{o}_1, ..., \hat{o}_m = randSubSet(\hat{O})$ // $m \geq 2$, Stateless strategy
10: $o = \sum \hat{w} \odot o$ // $w \sim unif(0, 1)$
11: $\lambda' \sim unif(|\hat{T}|, \lambda)$
12: $T' = s.solve(E', o, C \cup \{c_{size}(\lambda')\})$
13: $PF = PF \cup_{\succ} \{T'\}$
14: **end while**

here for simplicity. For LP, the outcome $\hat{T}$ is near-optimal, as shorter variants may exist. This may further impact the quality of the overall MC-TOA conduct — something we also investigate as explained in 4.1.

A simple and valid approach to Phase 1 is to define a regression test suite size limit λ based on a budget and not based on the scenario at hand. There are some downsides to this. First, this will naturally lead to sets that fill the limit, with the consequence that it might lead to unnecessarily high resource utilization and cost. Second, since the size limit has a large impact on the exploration space of solutions in Phase 2, a pre-defined λ will lead to a larger space, which will require more time to execute MC-TOA to receive a diverse set of test sets to choose from in Phase 3. We, therefore, here offer a heuristic approach that takes both a time limit based on domain/user needs and scenario characteristics into consideration.

In line 2, the Pareto-front gets instantiated as $\hat{T}$. The execution time limit λ, which is the outcome of Phase 1, is decided based on $\hat{T}$ in line 3, given user-defined upper bound λ_u (input 5) and slack factor ζ (input 6) that increases the acceptable size to a tolerable maximum. The slack factor ζ is a multiplier larger or equal to 1, which defines the search range for qualifying test sets as $\{|\hat{T}|, \zeta \times |\hat{T}|\}$ in terms of size. The larger the slack factor, the wider the exploration. The upper-bound λ_u allows the user to clarify what sizes to tolerate at maximum to ensure that the regression finishes in time. Given the acceptable test set, the upper bound should be decided on a system basis, also considering the time to execute. In situations where the minimum size test set of full coverage is substantially smaller than the upper bound, the slack factor helps to reduce the upper bound λ to a smaller region of the multidimensional solution space. In practice, the slack factor can greatly impact reducing execution times for regressions and thereby save resources.

3.3.2. Phase 2

With the execution limit λ and initialized Pareto-front PF as inputs, Phase 2 sharpens and diversifies the Pareto-front to the final selection of the most appropriate test set in Phase 3.

The *screening* phase (2a., lines 4–7) optimizes all remaining objectives in $\hat{O}$ independently. Line 5 shows how the partially optimal test sets T' are a product of the solver, which now applies the test size constraint c_{size} under consideration of the limiting factor λ. In line 6, T' is invoked into the Pareto-front following the mathematical concept of Pareto union $\cup_{\succ}$. For two non-dominated sets PF_1 and PF_2, $PF = PF_1 \cup_{\succ} PF_2$ includes only those non-dominated test sets, i.e., those test sets for which no other test set has a higher fitness or objective score for any of the objective functions (according to Pareto-dominance $\succ$). This partial phase is entirely parallelizable without loss of generality.

In the *exploration* phase (2b., lines 8–14), objectives are randomly varied and weighted for the solver to produce valid heuristic solutions to expand the Pareto-front and obtain test sets of better balance among the objectives. The implemented strategy is stateless in that no model-building is applied for objective, weight, or test set size limit selection. Model-based approaches, such as Kriging (Jones et al., 1998), would be other possible approaches to structurally explore the objective landscape. Consequently, in line 9, a random selection of objectives gets drawn. The weighting mechanism applied in this paper (line 10) is inspired by random weight genetic algorithms (RWGA) as presented in Konak et al. (2006) or the work from Mirarab et al. (2011). Objectives are varied as subsets $\hat{O}' \subset \hat{O}$ and weight vectors $w \in [0, 1]^{|\hat{O}'|}$ in single objective formulations of format $\sum \hat{w} \odot o', \forall \hat{o}' \in \hat{O}'$, while respecting all of the user-defined constraints. Combining this with multiple normalized objectives, MC-TOA generalizes the single fitness-function approach to test prioritization in Wang et al. (2016), where RWGA outperformed other multi-objective Pareto-optimal algorithms such as NSGA-II. In line 11, the test set size is even here randomly drawn from within the range of exploration. Given those conditions, the solver solves the defined problem formulation (line 12), and the resulting test set qualifies for the Pareto-front (line 13). The stop criterion applies a time limit for the entire MC-TOA run. The stop criterion shall even be considered for Phases 1 and 2 if the total execution time is short (this heavily depends on the hardware on which MC-TOA is run). In our experiments, we ran the screening (2a.) and exploration (2b.) in sequence.

4. Experimental evaluation

In Section 4.1, we go through the research questions we try to answer, followed by an explanation of the scenarios we investigate in Section 4.2 and the criteria formulations in Section 4.3 that can serve as examples for how criteria can be expressed in general for MC-TOA. It follows an exposé of the experiments (Section 4.4), including algorithm details (Section 4.5) and applied measures (Section 4.6). We compare various variants of MC-TOA to the many-objective state-of-the-art EA framework, Borg (Hadka and Reed, 2013), and a greedy random approach in those cases where constraint satisfaction can be handled.

4.1. Research questions

We address three research questions that connect to *feasibility* and *flexibility* in regression TCS. Existing research has found that combining objectives in TCS leads to great improvements over single objective formulations (Mondal et al., 2015). As mentioned in Section 2 on related work, many existing algorithms in the literature are *evolutionary*, which may be only one strategy among many and not necessarily the best (Feldt and Poulding, 2015).

The choice of a suitable algorithm becomes more involved for highly constrained problems, particularly those for which the constraints vary among instances of the problem, such as in the case of test set selection. Evolutionary algorithms have limitations when it comes to constraint satisfaction (Coello Coello, 2016). While introducing the MC-TOA framework, we therefore ask.

RQ1: How does MC-TOA compare to existing state-of-the-art many-objective algorithms tailored to the problem of test set selection in terms of performance for real-world industrial data?

We compare the proposed MC-TOA framework to the Borg EA framework and a random search strategy in an experimental study on real-world data sets and focus on the overall normalized quality of regression sets using the HVI measure (Auger et al., 2012) as a means to capture the Pareto-optimality of the multi-objective solution candidates. In the second question, we try to understand how the problem formulation impacts the quality of the result. In RQ2, we investigate the impact of formulating criteria as strict constraints or best-effort objectives on the solution quality. We ask.

RQ2: How does the formulation of the problem criteria as strict constraints compare to a best-effort formulation of criteria as objectives in terms of efficacy and constraint satisfaction?

Choosing objectives over constraints has some mathematical advantages, as discussed throughout the paper, and has been used as a go-to strategy in previous work (Hierons et al., 2020). The findings may give practitioners insights into the trade-offs for production use. Our approach is the first one to our knowledge that offers a generic solution to strict test set selection supporting multiple objectives, and we hypothesize that the constraint-based *original* problem can be solved near-optimally, i.e., producing near-optimal Pareto-fronts. The third research question investigates differences within MC-TOA that relate to its configuration. Lastly, we ask.

RQ3: What is the difference in solution quality and performance
a. when relaxing the IP to an LP formulation and
b. between a state-of-the-art commercial solver and open-source alternatives?

Whether high-quality solutions only can be obtained through the solving of the NP-hard IP formulation of the test selection problem or whether it can be sufficient to relax the problem to an LP to have it solvable in polynomial time which makes it suitable for large-scale problems, is the first aspect we cover. There might be a tipping point in the problem at which switching from IP to LP solving is preferable. We control for two variables. The proposed MC-TOA framework depends upon existing software, namely a convex solver, to do the iterative solving, a component that may differ in efficacy. We also look into the differences in runtimes for open-source vs. a high-end commercial solver, which we used under an academic license. We investigate whether the commercial solver is faster or more robust, which may lead to better Pareto-fronts in our approach compared to the open-source one.

4.2. Cases

Huawei Cloud Computing Technologies is interested in finding test sets for regressions, including several criteria for which data has continuously been collected for a longer time. Table 2 summarizes the systems under investigation, which are proprietary real-world production systems. System 1 has 11,267 test cases and 6485 functions, whereas System 2 has 3693 test cases and 5691 functions. Both systems can be considered large compared to those evaluated in the literature reviewed above — an order of magnitude for System 1. While not all details of the systems can be disclosed due to confidentiality reasons, we formalize the criteria in Section 4.3 in the following section. Due to confidentiality reasons, we can here not report on any more system metrics, such as lines of code or the type of system.

Table 2
The two systems under regression testing in direct comparison.

	System 1	System 2
Test cases	11,267	3,693
Functions	6,485	5,691
Criteria	5	6

The test coverage is summarized in histograms for both systems; see Fig. 2. Both system diagrams suggest long-tail distributions with a small number of functions covered by many test cases, whereas very few test cases cover the majority of functions. The histograms at the bottom show a similar distribution when reducing the functions to those covered by the top 40 (10) test cases, i.e., those test cases with the highest overall coverage. Without going into detail, the highly covered functions likely define core functionality that gets activated in many systems' use cases.

4.3. Criteria formulation

The problem formulation from Section 3.1 can be extended with additional objectives and constraints. In the lack of time to execute all tests, any metric that could contribute to increasing fault-finding capability could potentially be invoked into the optimization if that increases the chances of a regression revealing a fault. Information such as test case runtimes or historical fail rates collected in many companies already can serve that purpose.

We formalize the objectives and constraints of the two industrial systems in this investigation. We start with the particular case of formulating test set size as a constraint and then express fail rates, function exercising, and finally, generic coverage criteria exercising diverse configurations or different components, to name two examples. Varying criteria in the objective function are one of the built-in features of the proposed MC-TOA framework to strike a good balance between exploration and exploitation in the multi-dimensional search space.

4.3.1. Test set size

We here express the objective in (2) as a constraint that requires a maximum acceptable size to be set for the search to disregard all solutions above that. In order not to underestimate this value, and by the fact that creating an infeasible problem, a convex solver can first obtain the minimal size ms_{opt} for (2) subject to (3) to delimit the test set space to $\max\{ms_{opt}, \lambda\}$ for a tester defined limit λ. In practice, λ could be derived by setting a maximum size as the percentage ratio of tests in the test suite. For instance, if a maximum of executing 40% of test suite T with $|T| = 1.000$ is set, and given , $s_{opt} = 100$, we get $\max\{100, 400\} = 400$. The formal set size constraint definition using λ is:

$$c_{size}(\lambda, x): \quad \sum_{i=1}^{|T|} x_i \leq \max\{, s_{opt}, \lambda\}, x \in \{0, 1\}^{|T|}, \lambda \in \{1, \ldots, |T|\} \tag{5}$$

This constraint is included in all formulations solved in Phase 2 of MC-TOA, as described in Section 3.2. We exemplify this criterion and the following criteria on a minimal system in Table 3.

Example. Let the size limit be 1, i.e., $\lambda = 1$. From Table 3, we get test set $T' = \{t_1, t_3\}$ and since , $s_{opt} = 2$, this ensures that any valid result has two tests which ensure that all functions are coverable and the problem feasible, such as T'.

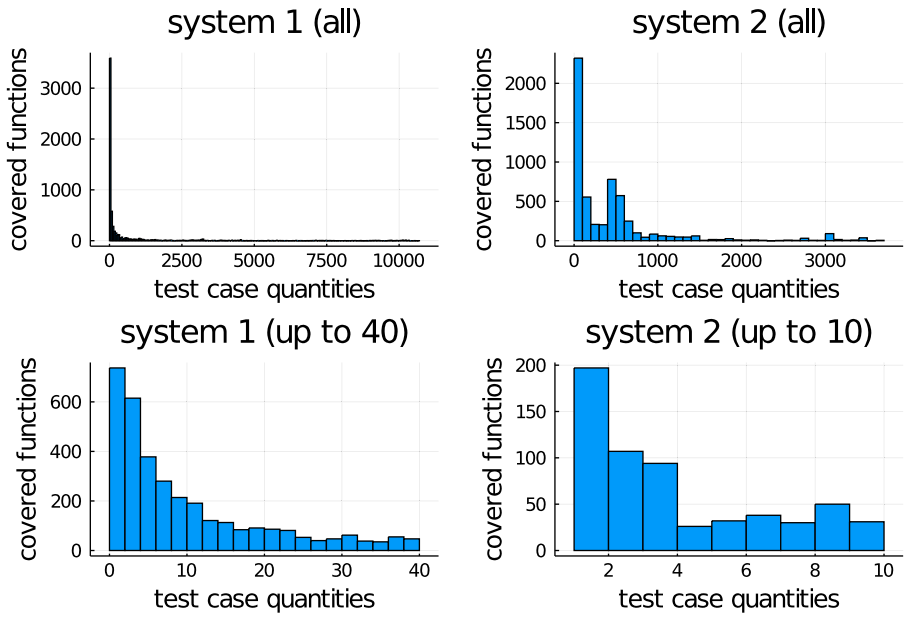

Fig. 2. Histograms over test coverage statistics for systems 1 (left) and 2 (right).

Table 3
A (folded) Execution Matrix (E') for which each test-case from $T = \{t_1, t_2, t_3\}$ is guaranteed to cover a change in at least one of $F = \{f_1, f_2, f_3\}$.

	t_1	t_2	t_3
f_1	1	1	0
f_2	0	1	1
f_3	0	0	1

4.3.2. Fail rates

Failure rates $\phi_j \in [0, 1]$ for all test cases $t_j \in T$ are, be they empirical or predicted, assessments of how likely a test is to fail. We want to trigger as many failures as possible. We thus search for the smallest set with the *highest* likelihood of failing,

$$\max_x \qquad o_{fail}(x) = \sum_{j=1}^{|T|} \phi_j x_j, \qquad x \in \{0, 1\}^{|T|} \tag{6}$$

There is no meaningful way of describing this criterion as a constraint, i.e., we cannot generally require that a test set fails a regression. Only a tiny proportion of regressions fail since code changes do, statistically, not produce guaranteed detectable defects in most cases.

Example. From Table 3, with $T'_1 = \{t_1, t_2\}$, $T'_2 = \{t_2, t_3\}$, and $\phi = (0.1, 0.01, 0.05)$, we get $o_{fail}(x_{T'_1}) = 0.11$, and $o_{fail}(x_{T'_2}) = 0.06$ which has us prefer T'_1 over T'_2 in terms of chances to provoke a failure.

4.3.3. Function exercising

We prefer tests that cover many functions over those that cover only a few. Maximizing the overall number of function exercising can be achieved by the following optimization function:

$$\max_x \qquad o_{ex}(x) = \sum_{j=1}^{|T|} \sum_{i=1}^{|F|} \epsilon_{ji} x_j, \qquad x \in \{0, 1\}^{|T|} \tag{7}$$

This basic formulation may be extended to consider, for instance, an even spread among the exercised functions, which would circumvent situations where one single function gets exercised by many test cases while others exercise only once.

Example. From Table 3, with $T'_1 = \{t_1, t_2\}$ and $T'_2 = \{t_2, t_3\}$, we get $o_{ex}(x_{T'_1}) = 3$, and $o_{ex}(x_{T'_2}) = 4$. T'_2 is to be preferred over T'_1 in terms of function exercising.

For the constraint formulation of the execution criterion, the minimum number of activations per test case $a = a_1, \ldots, a_{|T'|} \in \mathbb{N}^{+|T'|}$ should be chosen carefully, again as otherwise. the IP may become infeasible:

$$c_{ex}(a, x): \qquad \sum_{j=1}^{|T|} x_j \geq a_j, x \; in \{0, 1\}^{|T|} \tag{8}$$

Here, a natural limit for each a_j is the maximum number of functions covered by $t_j \in T'$, i.e., a_j must be chosen such that $a_j \leq \sum_{j=1}^{|T|} e'_{ij} x_j$. High activation numbers incur the risk that feasible test sets become rather large.

4.3.4. Coverage criteria

Test cases can be categorized by multiple means, such as based on what component of the program they cover or the type of hardware they run on. A category is often desired to be covered as a strict constraint or a coverage objective to be maximized. These can be criteria of structural nature that may not reveal through the code only. Thus, a problem formulation may require covering a finite number of these categorizations. Categorization K assigns test cases to categories, which may be mutually exclusive, with each test only occurring at most in one category in K, or non-exclusive, with test cases in multiple categories of K. Each category may be modeled as a function $\kappa : T \to \{0, 1\}$, which for each $t \in T$ assigns whether it belongs to the category.

Example. For t with κ : *running on hardware A*, let $\kappa(t) = 1$, else $\kappa(t) = 0$. Building on Table 3, let us assume our software consists of modules M1 and M2, respectively. Categorization K_1 then assigns each test its *belongingness* with $M1 = (1, 0, 0)$ and $M2 = (0, 1, 1)$. The categorization is mutually exclusive as a function can only belong to one module and is complete as all tests are assigned to a module.

The general logical constraint we can derive is

$$\frac{1}{|T|} \sum_{j=1}^{|T|} \kappa(t_j) x_j \geq \overline{\kappa}, \qquad \text{(criterion coverage)} \tag{9}$$

with $\overline{\kappa} \in [0, 1]$ being the minimum acceptable coverage in the test set. For required full coverage, we set $\overline{\kappa} = 1$. The more constraints we add, and the more restrictive they are, the greater the risk that the problem becomes infeasible. Each criterion can therefore be formulated as an objective to receive the best possible outcome given the Pareto property of the problem. The linear programming formulation requires the introduction of a binary slack variable per category in a categorization, s_κ, as well as a constraint that ensures the s_κ is 1 only if the category is covered. Thus, we get

$$\max_x \quad o_{cov,K}(x) = \sum_{\kappa \in K} s_\kappa, \quad s_\kappa \in \{0, 1\} \forall \kappa \in K, x \in \{0, 1\}^{|T|}, \text{ with} \tag{10}$$

$$\sum_{j=1}^{|T|} \kappa(t_j) x_j - s_\kappa \sum_{j=1}^{|T|} \kappa(t_j) \leq 0 \tag{11}$$

It should also be mentioned that a coverage criterion with n categories can be translated into n criteria with a single category. Furthermore, while many criteria for test prioritization can be expressed through coverage this way, some criteria, such as those related to time, cannot. Instead, following the general rules of linear/integer programming, they can be added to the search in their normalized/weighted forms as constraint or objective.

4.4. Experiments

We compare MC-TOA to a multi-objective EA and a greedy random search strategy as a baseline based on real-world industrial cases. Table 4 summarizes all relevant criteria, with criterion 5 being specific to System 1 and criteria 6 and 7 being specific to System 2. Table 5 gives an overview of all experiments in both series, for both systems, and with different configurations of criteria either as objectives or constraints depending on the algorithm variants in the specific comparisons (columns).

The three convex solvers we compare MC-TOA on are the open-source Clp, Cbc, and commercial Gurobi (under academic license). Clp implements the basic Simplex Method algorithm and thus only solves linear programs. Cbc can solve integer programs, making use of Clp internally for relaxation. Gurobi solves linear and mixed integer programs extending beyond that, e.g., to non-linear optimization. Cbc and Gurobi run in two versions, one solving the IP and one solving the LP. The LP results may underestimate the minimum, so we apply a heuristic approach to translate them into a test selection and still receive a solution respecting all criteria based on Formula (4).

For both systems, we conducted two experimental series (objective-based vs. constraint-based) with five MC-TOA configurations and two heuristics in comparison (see Table 5). In the objective-based (or best-effort) series, we formulated all criteria as objectives to allow for comparison to Borg and the random greedy algorithm. This results in formulations with a single objective and four constraints for System 1 and a single objective with five constraints for System 2. In the original formulation (*constraint-based*), the problem for System 1 consists of three objectives and two constraints, whereas for System 2, there are three objectives and three constraints. We call this the original formulation because, at Huawei, the situation is such that all coverage criteria (criterion 5 for System 1 and criteria 6 and 7 for System 2) must be guaranteed. Here, criterion 5, *Features*, depicts the system features a test case covers. Criterion 7, *Callchain depth 3*, signals, for each test case, whether there has been a regression change in a function in the call-chain of depth 3. Consequently, we could not include the metaheuristics in the comparison for experimental series two because of the need for a generic approach to constraint satisfaction which does not exist, as discussed in Section 2 without substantial effort in the repair algorithm or relaxing the problem which makes for an unfair comparison to the optimal solvers. Either way, the results of both series can directionally be compared since they solve similar versions of the same problem.

Table 4
The criteria included in this study.

Criterion	Explanation
1. Function coverage	Defined as constraint in (3)
2. Test set size	Details in Section 4.3.1
3. Fail rates	Details in Section 4.3.2
4. Function exercising	Details in Section 4.3.2
5. Features (System 1 only)	Coverage criterion (see Section 4.3.4), 159 categories
6. Morphologies (System 2 only)	Coverage criterion (see Section 4.3.4), 2 mutually exclusive categories
7. Callchain depth 3 (System 2 only)	Coverage criterion (see Section 4.3.4), 7 mutually exclusive categories

Table 5
The experimental setup for the two experimental series. Criteria link to Table 4. Borg and random search cannot generically handle constraints, so they were left out in two of the four setups where criteria were added as constraints.

Experiments	System 1		System 2	
Criteria				
Objectives	2–5	2–4	2–4 & 6,7	2–4
Constraints	1	1, 5	1	1, 6,7
Algorithms				
MC-TOA Gurobi	×	×	×	×
MC-TOA Cbc	×	×	×	×
MC-TOA Clp	×	×	×	×
MC-TOA Gurobi relaxed	×	×	×	×
MC-TOA Cbc relaxed	×	×	×	×
random	×		×	
Borg	×		×	
Criteria count	5	5	6	6

All experiments are run on cloud instances with 16 vCPUs and 16 GB RAM on a Ubuntu 18.04 operating system. The termination time for each algorithm in both studies is set to 300 s. To mitigate memory depletion issues experienced throughout prototyping, we used a swap file of size 16 GB. The experiments run on the original data sets for both systems. Because of the stochastic nature of all algorithms, each experiment is repeated 15 times.

The applied convex solvers are integrated into the *JuMP* framework.[8] The Borg implementation is based on the BlackBoxOptim optimization framework.[9] Both are written in the Julia programming language. We tailor Borg to the test set selection problem by adding support for discrete optimization in our Julia implementation fork.[10] Both code[11] and experimental data are openly available.[12]

4.5. Algorithm details

As discussed in Section 2, metaheuristics such as EAs require tailoring to address arbitrary criteria as coverage constraints (Coello Coello, 2016). For this study, the constraint handling needed tailoring for both Borg and greedy random search regarding function coverage (11), explained in the respective sections below.

8 https://github.com/jump-dev/JuMP.jl
9 https://github.com/robertfeldt/BlackBoxOptim.jl
10 https://github.com/feldob/BlackBoxOptim.jl
11 https://github.com/feldob/mc-toa-paper
12 https://zenodo.org/record/7740864#.ZBMIG4DMJH4

4.5.1. MC-TOA

The implementation follows the description in Section 3. We use convex-solver timeouts to ensure that MC-TOA terminates unfinished runs when the overall time budget of a run is exhausted. Timeouts are respected to varying degrees depending on the internal implementation of interrupts, which we bring up in Section 6, Discussion. For Phase 1, we execute the convex solver with a timeout equaling half the total search time to give sufficient time to build the Pareto-front after that. A consequence of strict timeouts is that the algorithm may be signaled to halt prematurely, returning the best solution found till that point if it exists. Therefore, even though optimum results could be obtained in finite time, in those cases, the outcome is non-optimal and transformed using Formula (4) while we ensured that the HVI comparison among algorithms is based on the actual optimal solutions.

Since size-limit λ_u and slack-factor ζ interact and impact the length of execution for regressions, they must be selected in consideration of one-another. For any use-case, the type of regression and resulting time limits will impact this decision, which is why no common default can be suggested here. For the slack-factor, we use $\zeta = 1.3$ to allow for a 30% increase in test suite size on-top of the given size-limit to give space for all objectives to get optimized. Limits close to 1 lead to very little possibility for improvements. We use a test set size limit of $\lambda_u = 0.1$, which ensures a reduction to at most 10% of the test suite. Our rationale behind that choice is heuristically based on the larger System 2, leading to regressions that correspond to nightly regressions. For simplicity, we keep the same value for System 1 in our investigations. Given basic runtime statistics for the tests, it should be rather simple to derive a suitable λ_u value, and leave some space for improvements in the remaining objectives through the slack-factor — in our case 30% was a useful bound to receive a diversified Pareto-front.

4.5.2. Greedy random search

The baseline random search strategy uses the algorithm from Li et al. (2007), which iteratively extends the test set by adding the solution maximizing function coverage using greedy 1-ply look-ahead as an efficient first valid heuristic solution. Until the time budget is exhausted, the algorithm applies the same strategy as for the first solution, introducing randomness in the choice of the 1-ply look-ahead test selection by randomly selecting a test case in the top half of the remaining test cases that improve on the test coverage. This is repeated for a solution until function coverage is 100% and repeats until timeout. Candidates are added continuously to the building Pareto-front, according to the Pareto union operator $\cup_{\succ}$. We further applied the ϵ window approach for the Pareto-front qualification, as explained below for the Borg framework, to obtain Pareto-fronts of manageable sizes.

4.5.3. Borg

Because of a more efficient Pareto-front management, Borg is suited for problems with many objectives and multi-modal landscapes and could be demonstrated to outperform other state-of-the-art EAs, such as NSGA-II (Hadka and Reed, 2013), for more than three objectives, a challenge NSGA-II has shown to struggle with (Panichella et al., 2017). This makes it the best metaheuristic choice to our knowledge for large-scale TCS problems with multiple criteria expressed as objectives. The Borg framework requires setting some configuration parameters, with defaults recommended in Hadka and Reed (2013). The ϵ value steers the granularity of the Pareto-front archive to reduce the memory and runtime burden for Pareto-fronts with many minimally dominating candidates. We apply $\epsilon = 0.1$ as the default for all dimensions. All candidates are binary encoded, i.e., $x \in \{0, 1\}^{|T|}$. The initial population bootstrapping of 10 candidates is done in support of the greedy random search heuristic above. This is to help the algorithm with an incumbent in terms of feature coverage and a diverse set of candidates that fulfill the function coverage constraint. Consequently, Borg never performs worse than random search in this study as the exploration capability of this strategy is very limited.

We used plain binary crossover and flip mutation operators with a common n-point-crossover rate of $\rho = 0.9$ for size $n = \lceil 0.01 \cdot |T| \rceil$, as well as a mutation rate of $\mu = 0.01$. The number of crossover points was thus chosen in relation to the number of test cases. Widely used and compatible Tournament selection (of size three) was applied (Shukla et al., 2015). Restarts are triggered by convergence with no improvement over ten generations and are based on the same logic for the re-initialization of the population as for the initial bootstrapping. We apply no embedding operator for the restart strategy since the produced individuals already fulfill the function-coverage constraint and ensure a specific diversity among each other.

4.6. Measures

The main criteria in this investigation are as introduced in Section 3.1 test set size, and in Table 4, failure rate, number of function executions, and several system-specific coverage criteria (e.g., features, morphologies) as also described in the dataset description in Section 4.2. We here list four internal measures used to compare the competing algorithms: *time*, *optimality gap*, *hypervolume*, and *normalized fitness*.

4.6.1. Time

We set deadlines for the conduct of each algorithm experiment and each solver execution. For MC-TOA, we track the number of successful executions of the convex-solver to assess differences among the configurations for cases of different sizes in the number of functions and tests (executions per time unit). The time assessment is limited to the actual optimization runs and does not consider the time or effort involved in obtaining, e.g., the historical coverage or failure information. To ensure fairness in this study, we ensured that all compared algorithms have access to the same information.

4.6.2. Optimality-gap

The normalization, as explained above, allows the ranking of solutions as we can extract the optimality gaps for each measure, i.e., how close a solution, or the best solution at a point in time of a search, is from the theoretical optimum for each objective singled-out. Since we, in our experiments here, obtain the optimal values due to running optimal IP solvers in the screening phase 2.a, the optimality-gap is absolute. However, since deadlines hinder the algorithms from finishing at times, sub-optimal solutions are possible but have yet to be observed in this study. Furthermore, because of the duality property of linear programs, it is often possible to retrieve optimal bounds without obtaining an actual optimal solution. However, we have not used that information here due to practical limitations in our implementation.

4.6.3. Hypervolume indicator

While there is no gold standard for quality assessment in many-objective optimization, we use the most popular measure, HVI (Auger et al., 2012). The hypervolume spans the multidimensional fitness landscape and balances the objectives equally, aggregating them into a single HVI value. It can either be based on the optimality gaps to allow globally normalized comparison, alternatively on lack of optimal performance knowledge,

Table 6
Performance statistics for System 1 in Experimental Series 1. The first (second) half of the table reports results for runs with the system-specific feature criterion as a constraint (objective).

Method	Min. set size		Max. fail.rate		Max. exerc.rate		Solver
	abs	gap (%)	abs	gap (%)	abs	gap (%)	Runs
Constraint							
gurobi	551 ± 0	0 ± 0	953.5	0 ± 0	811662 ± 0	0 ± 0	13 ± 0
Cbc	551 ± 0	0 ± 0	953.5	0 ± 0	811662 ± 0	0 ± 0	10 ± 1
gurobi_relaxed	635 ± 61	16 ± 12	953.5	0 ± 0	811662 ± 0	0 ± 0	14 ± 1
Cbc_relaxed	633 ± 65	15 ± 12	953.5	0 ± 0	811662 ± 0	0 ± 0	14 ± 0
Clp	643 ± 57	17 ± 11	953.5	0 ± 0	811662 ± 0	0 ± 0	14 ± 1
Objective							
gurobi	513 ± 0	1 ± 1	960.1	0 ± 0	821908 ± 0	0 ± 0	14 ± 0
Cbc	513 ± 0	1 ± 1	960.1	0 ± 0	821908 ± 0	0 ± 0	10 ± 1
gurobi_relaxed	541 ± 35	6 ± 7	960.1	0 ± 0	821908 ± 0	0 ± 0	15 ± 1
Cbc_relaxed	560 ± 43	10 ± 9	960.1	0 ± 0	821908 ± 0	0 ± 0	14 ± 0
Clp	538 ± 37	5 ± 8	960.1	0 ± 0	821908 ± 0	0 ± 0	14 ± 1
borg	537 ± 0	5 ± 0	545.9	44 ± 4	545430 ± 29401	34 ± 4	–
random	537 ± 0	5 ± 0	456.3	53 ± 1	474979 ± 3943	43 ± 1	–

the extreme values of all empirically observed solution performances throughout the search. The latter approach only allows for relative comparison among the competing algorithms, as the reference may change depending on the empirically observed solution fitnesses. To make objective-based and constraint-based searches comparable, we include all constraint-based criteria in the HVI calculation and set the upper bound of the volume to the normalized required value the constraint demands. For the constraint-based runs, finding solutions is harder, but any solution for those hypervolume dimensions has a hypervolume of 1 in each constraint dimension. HVI can be normalized to a percentage scale [0, 1], which is what we do here. The larger the value, the better the solution.

HVI calculations are computationally expensive for high-dimensional spaces and data sets with many data points. One way of reducing the computational load is to use the Monte-Carlo approximation (Bader et al., 2010), as applied in this study with 10.000 samples per approximation. We use HVI to show progress throughout the search and plot it over time to visualize convergence and relative performance among the algorithms. To calculate the HVI over the entire runtime, we recorded all Pareto-front changes throughout the search for all algorithms, which required the efficient use of compression for storage and reproduction. In particular, for the meta-heuristics producing many non-dominating solutions (despite using a tight ϵ value for Borg and random search).

4.6.4. Normalized fitness

For the objectives to be combined into balanced partial single objective formulations via the weighted sum approach in line 10 of Algorithm 1, they must be normalized. This is possible for all objectives because MC-TOA, through the convex solver, gives us access to valuable boundaries that we can use as denominators. For test set size, the trivially largest possible value is $|T|$; for the failure rate, it is the sum of all possible failure rates (maximizing failure rate by executing all T), and for the maximum function exercising, we can exercise at best as many as all tests combined do, based on the execution matrix E. Normalization is generally possible for any coverage criterion K over T since coverage is a normalized measure. We use the $\hat{o}$ symbol to signify a normalized objective. These are the max-normalized versions of the formulations required for MC-TOA:

$$\max_x \quad \hat{o}_{size}(x) = 1 - \frac{1}{|T|} o_{size}(x), \qquad x \in \{0,1\}^{|T|} \tag{12}$$

$$\max_x \quad \hat{o}_{ex}(x) = \frac{1}{\sum_{i=1}^{|F|}\sum_{j=1}^{|T|}\epsilon'_{ij}} o_{ex}(x), \qquad x \in \{0,1\}^{|T|} \tag{13}$$

$$\max_x \quad \hat{o}_{fail}(x) = \frac{1}{\sum_{i=1}^{|F|} f_i} o_{fail}(x), \qquad x \in \{0,1\}^{|T|} \tag{14}$$

$$\max_x \quad \hat{o}_{cov,K}(x) = \frac{1}{|K|} o_{cov,K}(x), \qquad x \in \{0,1\}^{|T|} \tag{15}$$

Formula (15) stands out in that it may appear multiple times in a problem formulation for various coverage criteria K as discussed in 4.3.4.

5. Results

Tables 6 and 7 summarize the study's main results aggregated by algorithm and system. Columns two to four contain the single objective average, the standard deviation of absolute fitness, and optimality-gap readings for the best-found solutions in the main three search criteria for System 1 and System 2, respectively. Finally, the fifth column contains the total solver executions per MC-TOA run, including Phases 1 and 2. In the strict formulations with coverage criteria being constraints (upper rows in tables), criteria coverage is guaranteed, i.e., the optimality-gap is 0 (HVI score is 1), as explained in Section 4.6.

As expected, it can be observed that both Gurobi and Cbc solve the IP formulations for the strict versions to optimality in all dimensions due to the screening in MC-TOA Phase 2.a for both systems. Over the relaxed formulations, the convex solvers show minor performance deterioration for the smaller System 2 (3%–4% in the execution rate dimension). A more considerable negative impact can be measured for System 1 (around 16% for all linear solvers in the test set size dimension). Random search and Borg perform significantly worse than MC-TOA in any configuration for the objective-based search. For the smaller System 2, the optimality gap for Borg is 3%–11% for the main dimensions, though not as great as for System 1 with 5%–44%. This may suggest that MC-TOA scales better in the size of the problem formulation but would need a more dedicated scalability investigation. Borg found the best solutions in the execution rate dimension for System 2. Since min size acted as an objective in Borg, we could not meaningfully bind the search upwards without affecting convergence, which is why some very large, otherwise of low quality, solutions found throughout the search obtained values that were better in this one dimension compared to MC-TOA. This was true on very few occasions, even for the random search, which explains the overall single-dimensional worse performance of exercise rates for the MC-TOA configurations while performing substantially better in all other dimensions for System 2. This phenomenon could not be observed in the larger System 1, likely due to the early greedy solutions used for bootstrapping, which nudged the search directly into more optimal areas of the search space regarding the test

Table 7
Performance statistics for System 2 in Experimental Series 1. The first (second) half of the table reports results for runs with the system-specific morphology criterion as objective (constraint).

Method	Min. set size		Max. fail.rate		Max. exerc.rate		Solver
	abs	gap (%)	abs	gap (%)	abs	gap (%)	Runs
Constraint							
gurobi	141 ± 0	0 ± 0	206.6	0 ± 0	564239 ± 0	0 ± 0	37 ± 1
Cbc	141 ± 0	0 ± 0	206.6	0 ± 0	564239 ± 0	0 ± 0	31 ± 1
gurobi_relaxed	141 ± 0	0 ± 0	206.6	0 ± 0	548722 ± 12938	3 ± 3	41 ± 1
Cbc_relaxed	141 ± 0	0 ± 0	206.6	0 ± 0	545150 ± 15211	4 ± 3	40 ± 0
Clp	149 ± 8	6 ± 6	206.6	0 ± 0	550936 ± 18549	3 ± 4	40 ± 1
Objective							
gurobi	141 ± 0	0 ± 0	206.6	0 ± 0	564239 ± 0	0 ± 0	37 ± 1
Cbc	141 ± 0	0 ± 0	206.6	0 ± 0	564239 ± 0	0 ± 0	31 ± 1
gurobi_relaxed	141 ± 0	0 ± 0	206.6	0 ± 0	548722 ± 12938	3 ± 3	41 ± 1
Cbc_relaxed	141 ± 0	0 ± 0	206.6	0 ± 0	545150 ± 15211	4 ± 3	40 ± 0
Clp	149 ± 8	6 ± 6	206.6	0 ± 0	550936 ± 18549	3 ± 4	40 ± 1

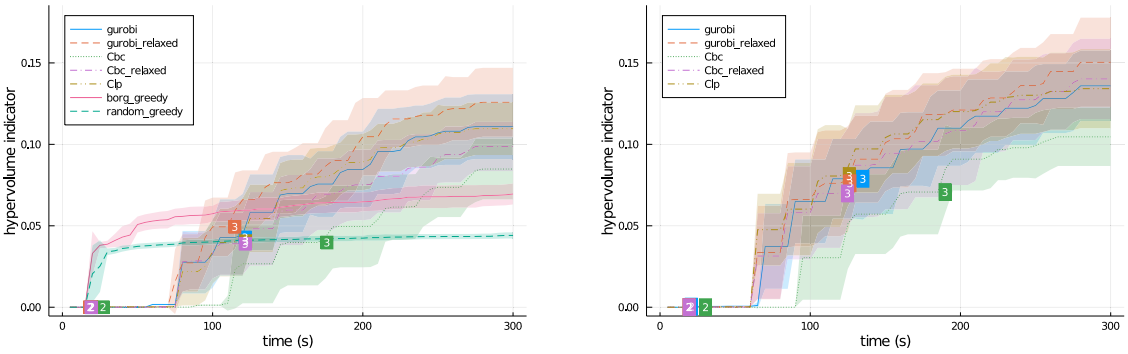

Fig. 3. Progress plot showing the HVI trajectory for all algorithms on System 1 with feature coverage as objective (left) and constraints (right). For each variant, the MC-TOA *screening* step starts at colored marker 2 and the *exploration* step at colored marker 3.

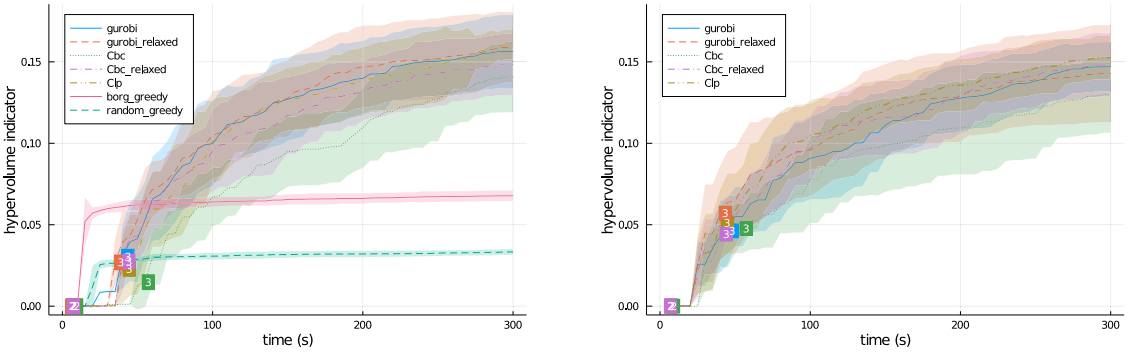

Fig. 4. Progress plot showing the HVI trajectory for all algorithms on System 2 with feature coverage as objective (left) and constraints (right). For each variant, the MC-TOA *screening* step starts at colored marker 2 and the *exploration* step at colored marker 3.

set size straight out. Expressed in test-suite reduction, a common measure for expressing the success of a selection, we find that for the constraint-based formulations, i.e., those respecting all non-negotiable constraints, the best solutions produce a reduction of 551/11,267 (4.89%) for System 1 and 141/6485 (2.17%) for System 2.

While the single objective readings in Tables 6 and 7 only indicate how explorative the algorithms are, does the HVI summarize the efficiency in obtaining balanced solutions in the Pareto-front. How the algorithms compare over the entire criteria, the spectrum can be seen in HVI graphs in Figs. 3 and 4 for both the objective and constraint-based problem formulations of both investigated systems.

All MC-TOA variants follow a typical convergence pattern. While the search for System 2, for all MC-TOA variants, is close to convergence, convergence was not achieved after the total runtime of 300 s for System 1. This suggests that for longer runtimes, better results could be expected. All MC-TOA variants surpass Borg after less than 120 s for System 1 and less than 100 s for System 2 while producing better overall Pareto-fronts according to HVI after that. Borg converges for System 1 after about 100 s, while for System 2 after about 30 s. The random search converges prematurely to a consistently inferior Pareto-front performance as of HVI.

The HVI scales for the objective and constrained variants (Figs. 3 and 4) look similar but do not assemble the same HVI spaces and cannot directly be compared. The partial volumes are equal for all coverage criteria and normalized to [0, 1]. The relaxation of constraints in the objective-based search can obtain results better than the optimum in the constrained version in

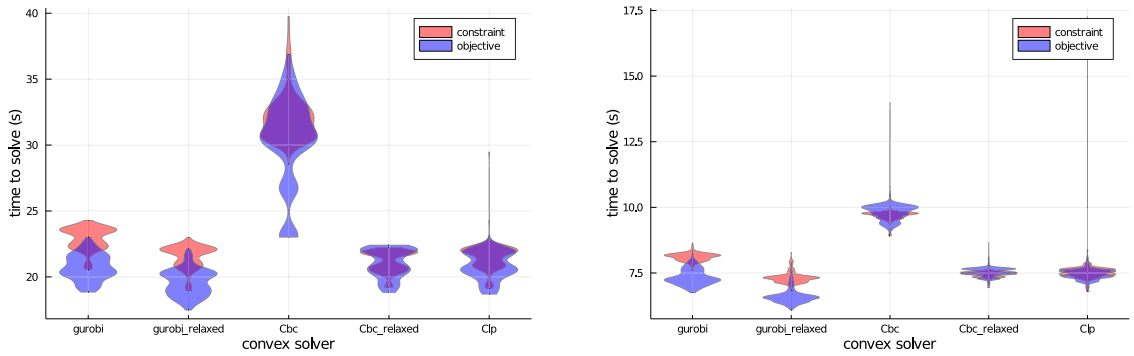

Fig. 5. Comparing execution time performance for the different convex solvers on System 1 (left) and System 2 (right) when applied in our MC-TOA framework. Notice the difference in the *y*-axis range on execution time.

other dimensions (e.g., absolute *min. Set Size* average for System 1 in Table 6 of 551 for one and 513 for the other). As a consequence, the constraint-based results are conservative if compared directly to the objective ones, i.e., for equal HVI, the constraint result can be considered at least as good as the objective-based result, suggesting that solving the original harder constrained problem can lead to better solutions using MC-TOA than solving the simplified objective-based formulation.

The numbered boxes along the trajectories in the HVI plots are positioned at the average point where the respective phases 2.a and 2.b were reached. The objective-based experiments get through the initial phases 1. and 2. seemingly faster. Even though the convex solvers over IP formulations reliably find solutions with optimal single-objective readings, the HVI readings for the solutions found by LP formulations are better for both Gurobi and Cbc, in particular for the bigger System 1, where convergence is not as advanced after the 300 s. Overall, the relaxed Gurobi formulation performed best, possibly because of the extra runs the solver could execute till success within the limited time window. Cbc consistently performs the worst among all MC-TOA configurations.

The convex solvers differ in runtime. Consequently, the number of successful solver iterations making up for the Pareto-front per run differ in size and quality. The solver run numbers suggest that relaxed formulations are solved fast, particularly for Cbc. Fig. 5 compares the MC-TOA configurations for both systems regarding execution time aggregates over all executions using violin plots. A significant within-difference between objective and constraint formulations can only be observed for Gurobi, with strict formulations requiring more time. Cbc is slower in IP solving than Gurobi, almost by a factor of two for the larger System 1. The within-difference per solver for IP vs. LP is visible for both Gurobi and even more so for Cbc. Both solve the LP formulations faster — in particular, Cbc.

The difference in Pareto-front quality between the heuristic search through Borg and MC-TOA can be visualized by two-dimensional reductions of the many-dimensional solution space, exemplified by individual arbitrarily chosen optimization runs. Fig. 6 presents six 2D plots for System 1 Pareto-fronts, and Fig. 7 three for System 2. Each point represents a non-dominated solution. Not all dimensions are shown here for brevity. The primary purpose of the plots is to show how Borg and even more random searches get stuck in sub-optimal areas of the search space. In contrast, irrespective of the solver choice, MC-TOA results fall along each dimension's optimal perimeter. Selecting a test set for execution among the Pareto-front would require handling many more and less differentiated points for Borg than for MC-TOA or some beforehand screening or filtering of the Pareto-front.

6. Discussion

We address the three research questions, followed by a more open discussion, including future avenues and threats to validity.

RQ1: How does MC-TOA compare to existing state-of-the-art many-objective algorithms tailored to the problem of test set selection in terms of performance for real-world industrial data?

In all tested settings, MC-TOA outperforms Borg and random search in both scenarios over the board, except for the very early stages of the search. For the smaller System 2, the overall average HVI of the best solution so far over all MC-TOA configurations was about 133% better (0.14 vs. 0.06), and for the larger System 1, with incomplete convergence, about 43% better (0.1 vs. 0.07). Borg requires much tailoring and cannot handle criteria as constraints out of the box, which required us to split the experiments into a best-effort category to allow for comparison and a strict problem formulation with fixed constraints. Despite all efforts to customize Borg with designated greedy starting points and varying operators for mutation and crossover, it fails to grasp the sizeable many-dimensional landscape. The results of running MC-TOA for the strict scenarios again are superior to the best-effort ones for which MC-TOA outperformed Borg. The numbers above can therefore serve as a conservative lower bound. This highlights that MC-TOA is more flexible as well as better performing.

A timeframe for regression can be subdivided into the time to decide the test set to run and the actual execution of the test set. For timeframes in the hour ranges, for instance, highly regressions, some minutes may be reserved for the actual decision process. In this context, the results of this study suggest using MC-TOA instead of Borg or random search. For very short timeframes, such as for regular codebase commits, the number of affected functions in the activation matrix E' are likely much smaller than the presented ones here, and runtimes would be substantially faster there too. We have, though not tested this scenario further. Evolutionary algorithms may be an alternative for short decision times in the second range. However, since decision time can be made proportional to execution time, we see the situation as many tests that execute very fast as rare — it begs the question of whether the optimization is even required/meaningful in that case overall.

RQ2: How does the formulation of the problem criteria as strict constraints compare to a best-effort formulation of criteria as objectives in terms of efficacy and constraint satisfaction?

For industrial optimization problems, many relevance criteria require strict guarantees, often because of time or space limitations. MC-TOA allows the inclusion of any criterion for test

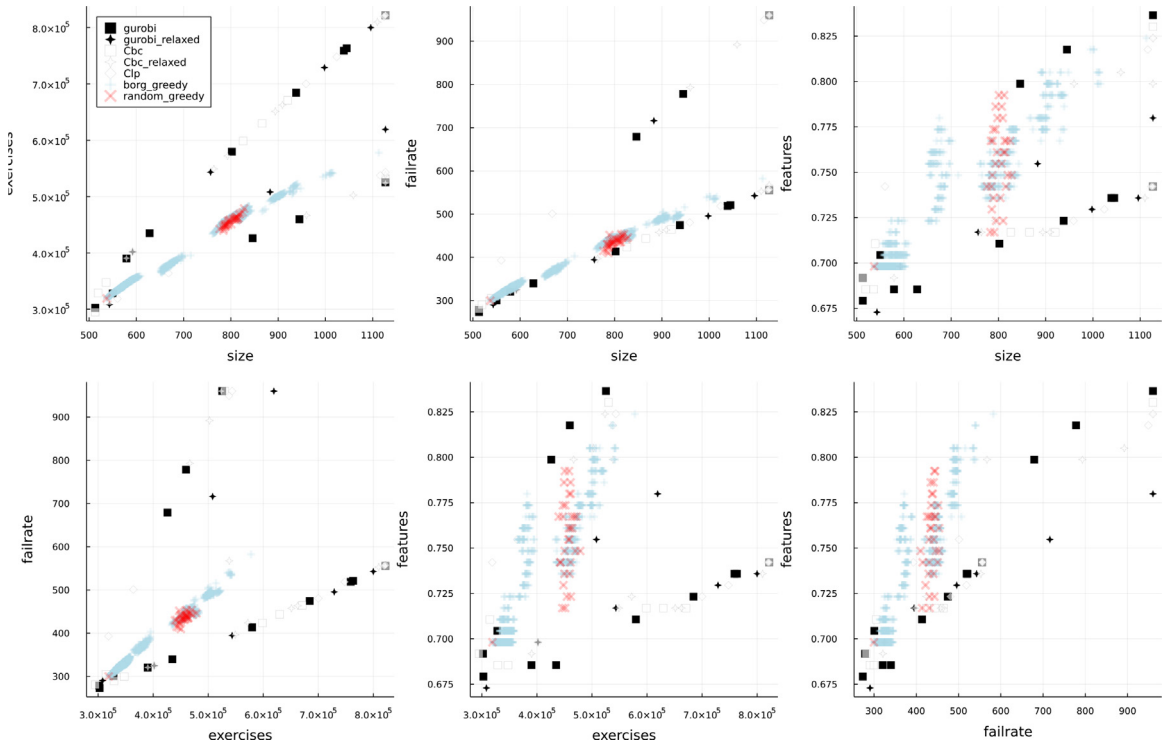

Fig. 6. Two-dimensional collapses of Pareto-fronts for System 1 (objective-based series).

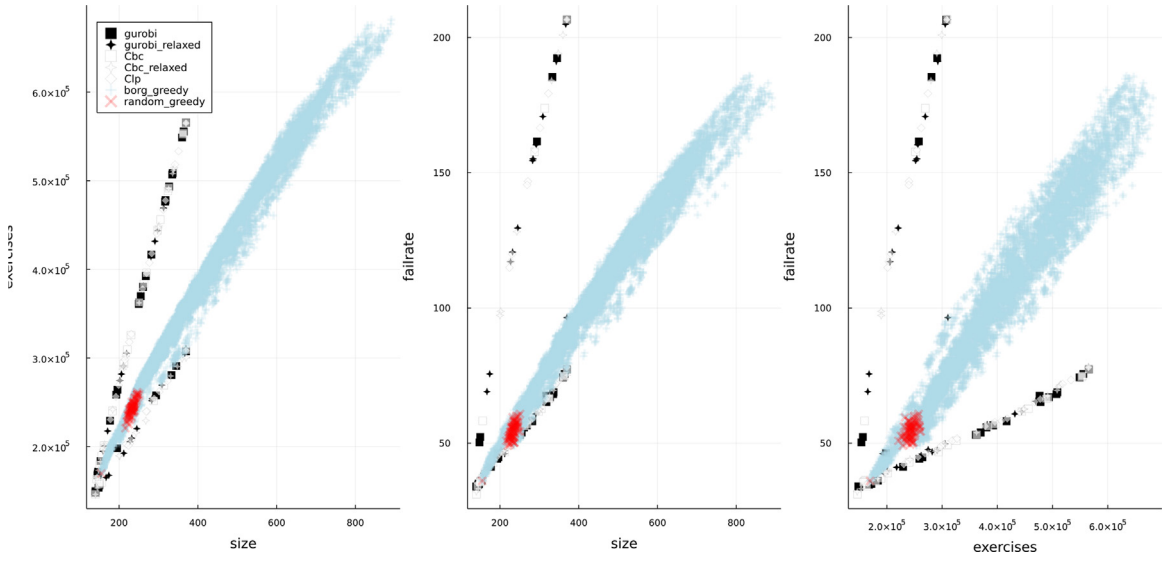

Fig. 7. Two-dimensional collapses of Pareto-fronts for System 2 (objective-based series).

set selection either as a constraint or objective. Regarding efficacy, we see that constraint-encoded formulations lead to better HVI in total numbers than objective-encoded ones for System 1 (see Fig. 3) and comparable ones for System 2 (see Fig. 4). As mentioned above, the numbers are incomparable, and the constraint-based HVI for the constraint variants for the same problem can be considered superior. Noteworthy is that for the larger System 1, the results over the constraint-based formulations are clearly better in convergence and performance after 300 s. It should also be noted that objective-based executions of solvers are faster in execution, notably for Gurobi (see Fig. 5). Constraints add extra complexity, and we explain below why the number of criteria considered in MC-TOA is always higher in the constraint-based formulations, which may explain the difference in individual execution times.

Regarding constraint satisfaction, our experimental results suggest that MC-TOA over objective-based formulations seldom finds solutions that fulfill the coverage constraints, i.e., lead to optimal solutions in the sub-dimension. This can be explained by how MC-TOA Phase 2.b explores the search space by varying objectives into normalized partial single objective formulations with random weights (see Algorithm 1, lines 9–10). While *all* criteria formulated as constraints are considered for any partial formulations (line 12), this is not the case for all objective-encoded

criteria. Additionally, the single-objective formulation will assign low weight to the formulation, leading to optimal solutions with non-optimal coverage. For future work, this design may be tested in different settings, for instance, by always considering specific prioritized objective-encoded criteria with high weights. However, the overall performance benefits achieved with constraint-encoded criteria will remain since a strict threshold is already known in most situations.

RQ3.a: What is the difference in solution quality and performance when relaxing the IP to an LP formulation?

The relaxed formulations in this study show some exciting properties that suggest they are better candidates for MC-TOA than their IP counterparts. While the single objective results of Tables 6 and 7 show an advantage for the IP solutions due to the guarantee of optimality at the extremes, do the HVI summaries reveal that the MC-TOA configurations with relaxed formulations performed better over the board by conceivable margins for both convex solvers Gurobi and Cbc. The difference in performance is likely explained by the fact that the number of relaxed versions solved within the same timeframe was consistently higher (see Tables 6 and 7, last column), leading to a better search space exploration. Future work shall verify this by controlling for the number of solvers' runs per MC-TOA execution instead of time. The faster solving of relaxed problems is due to their lower computational complexity. Therefore, the divide in performance can be expected to widen further the larger the test suites. The smaller differences in HVI for the smaller System 2 (Fig. 4) as compared to the larger System 1 (Fig. 3) also indicate that.

RQ3.b: What is the difference in solution quality and performance between a state-of-the-art commercial solver and open-source alternatives?

The commercial solver Gurobi shows better performance over open-source Cbc throughout. The Simplex Method implementation in Clp did not notably differ in execution times compared to the relaxed executions of Cbc, which internally uses Clp (see Fig. 5). Most notably, the difference in solving the relaxed and IP formulations was for Gurobi (average of 15% for System 1 and 20% for System 2) much smaller and less affected by test suite size than for Cbc (average 41% for System 1 and 26% for System 2). Gurobi got faster at solving the IP formulation than the relaxed one for the larger system. Whether or to what extent this may be affected by the number and difficulty of the constraints has yet to be investigated.

Further, Cbc (and Clp) behaved less stable because a few outlier problem formulations had to be manually killed as the interrupt handling did not work — the algorithm seems stuck. In practice, this had the consequence that some, but few, of the results were likely outside the set-out timeframe, which renders the results optimistic or best-case. We did not follow up on this, and it could very well be a problem with the *JuMP* bridge, which otherwise acted very stable.

Even here, the likely explanation for Gurobi's superiority in performance is the number of the total convex solver executions per run for Gurobi compared to Cbc as of the last columns in Tables 6 and 7. Either way, given that Gurobi and Cbc return correct solutions of the same quality for the same formulations, apart from the stability issues mentioned above, it is mostly a matter of longer execution time to obtain the same result quality with the open-source solver.

It can be observed that HVI achievements are small in absolute numbers, with convergence around 0.15 for both systems in the constraint case over a normalized space $[0, 1]$. Overall low HVI can be explained by the fact that formulations with many criteria cannot reach a very high value because the $[0, 1]$ normalized fitness values multiply. The extremes are derived per dimension while the solutions compare on a multidimensional landscape. For instance, for a solution with eight objectives, each having a fitness of 0.8 (1.0 being the greatest), the HVI is $0.8^8 = .168$. Real problems that allow for solutions with such high ratings for all objectives are rare because the competing objectives usually lead to compromises that can be visualized through Pareto-front diagrams.

Here follow some further observations of a general character. MC-TOA is highly parallelizable as solving the various formulations can be delegated to external computation units with minimal communication overhead. To speed up the optimization, Phase 1 can be decided manually to shorten the start phase of what we observed to be between 6 and 30 s in our experiments. MC-TOA is entirely parallelizable in Phase 2. In practice, the total runtime of Phase 2 is delimited by the slowest convex solver execution, which we observed to be of execution length up to 40 s in our experiments. EAs are also parallelizable in the size of the population per generation but have issues with handling constraints, as seen in the results and other studies, such as in Xue and Li (2020). For each additional constraint, time must be spent on resolving whether and how it is possible and effective to include it in the encoding of the problem, which is not required for convex solver-based approaches, which MC-TOA attempts to generalize for TCS. At the same time, MC-TOA is parallelizable on two levels, (i) when solving the individual formulations on independent units, and (ii) each convex solver can further distribute calculations to whatever number of units suitable/available. Units here may be CPUs or even external machines.[13] Finally, an efficient solution with negligible computational overhead in the seconds range for Phase 3 is the ranking according to HVI. Given that regressions often run nights and for hours, a runtime in the minutes is likely acceptable. While we have not investigated the trade-off between the budget allocated for optimization vs. the time allocated for running the tests, this is a relevant investigation avenue.

Beyond using optimality-gap and HVI for presentation and visual comparison, further applications can be considered, such as for ranking purposes in Phase 3 of MC-TOA to decide *the best* test set for the regression. Alternatively, more sophisticated ways of including the user in the decision of selecting the most appropriate test set shall be investigated in future work.

The included criteria in a study and the number of investigated categories in the search for multiple coverage criteria can be misleading. As already mentioned, coverage criteria with many categories can be translated into multiple coverage criteria with few categories, which is why we highlight the importance of future work to include clear descriptions of the categories of a criterion and their type (e.g., binary). Differentiating the formulations into objective vs. constraint was based on using standard nomenclature from the traditional optimization literature as much as possible while other terms have been proposed (see, e.g., Coviello et al., 2020). We separated criteria into objectives and constraints. Regarding objectives, the only limitation of MC-TOA is the possibility of converting each objective into a normalized version to allow for meaningful balancing in the scalarized objective function. Concerning constraints C, there is no specific limitation as long as they can be written as linear or integer constraints. Depending on the solver applied, even non-linear constraints may be possible to be handled. We have not evaluated that aspect, but Gurobi allows this. In summary, the limitations are due to the normalization requirement for objectives and, otherwise, the limitations of the solver being applied within MC-TOA.

[13] Primarily, commercial solvers are designed to allow for master–slave parallelization.

We investigate the test coverage of *functions* — the unit our industrial partner considers valuable. However, the proposed algorithm is agnostic to whether the unit of test-code coverage is measured as presented based on functions, classes, methods, or statements. Coviello et al. (2018) propose what may be intuitive, namely that a more fine-grained resolution in the unit can achieve better coverage in the tests. This comes, however, with a hit in performance as there are, for instance, more statements than methods and more methods than classes. It is, therefore, up to the product owner to strike a good balance between abstraction level and practical feasibility. Another downside on fine granularity, such as code segments in Coviello et al. (2018), is that changes on lower-level (e.g., method over class) happen more frequently, which will incur a more considerable overhead in recalculating the coverage matrix and limit the reusability of historical statistics more severely.

6.1. Threats to validity

The four main validity threat categories are addressed in the sections below.

6.1.1. Conclusion validity

Since we are working with heuristic optimization, random variation in the results is to be expected. We addressed this by repeating experiments 15 times and reporting the mean and standard deviation for all findings in numbers and graphically. Further, because of experienced reliability issues with the time limit function for some solvers in MC-TOA, there is a risk that time limits got surpassed for some runs leading to slightly better results than would be obtained otherwise. We highlight those few occurrences in the text where applicable.

6.1.2. Internal validity

When implementing all three algorithms, we depended on external libraries to the largest degree possible. Our MC-TOA framework is based on two highly used open-source optimization frameworks, BlackBoxOptim.jl (399 GitHub stars) and *JuMP* (2k GitHub stars). The Borg implementation is taken from BlackBoxOptim.jl and adjusted for discrete optimization. All three convex solvers applied are heavily used, and the two open-source ones are highly rated, which was a selection criterion. The algorithm's parameter choices were mainly based on suggested defaults from the literature.

We selected a baseline algorithm as a greedy algorithm (Chvatal, 1979) for test set minimization and Borg as a representative of multi-objective EA. It could be argued that for this specific problem, the older NSGA-II has been evaluated as a suitable candidate, but since that algorithm has severe scalability problems in the number of objectives and has proven to be superseded by Borg (Hadka and Reed, 2013), the results should be representative of what multi-objective EAs have to offer for this problem as of today.

We applied the same time budget to all algorithms in the study. We have no affiliation with any convex solver creator. There is, therefore, not either any conflict of interest.

6.1.3. Construct validity

The evaluation in this study is entirely based on the combined quality of the optimization criteria, not the actual fault-finding capabilities. Many of the criteria are based on code coverage, and it has been discussed in the literature whether it is the best way to assess test quality as it does not cover bug severity. There is no consensus, but what speaks for code coverage is that it strikes a good balance between offering helpful information and being easy to obtain, assuming access to the source code. Without deepening the discussion, we look into ways to incorporate fault-finding capabilities in future work.

Even optimal performance does not guarantee that faults are found. An investigation of the actual test sets was practically not feasible, which can be seen as a common challenge for complex software dependent on specific hardware.

6.1.4. External validity

Since we are basing the study on two specific use cases with explicit objectives and constraints, this affects the generalizability of the results to other scenarios. We try to address this threat by investigating data from actual systems highlighting the practical relevance.

We cannot draw general conclusions about open-source and commercial solvers as we only tested three. The choice was motivated by selecting among the most used[14] and best-performing solvers.[15] Further, only two systems have been investigated in worst-case scenarios, i.e., assuming that the regression affects all functions, negatively affecting generalizability. In any case, the systems represent production software of great importance to the organization.

Those constraints that consider fault-finding capabilities or coverage require the availability of that extracted information for the system at hand. This information must be collected over time, which might not be the case for all existing systems. Our industrial partner has been proactively collecting metrics for a long time because of the practical need to understand and improve efficiency, which is why this data was available to us.

7. Conclusions and future work

We proposed and introduced the MC-TOA framework for efficient, large-scale test set selection in support of arbitrary search criteria. MC-TOA can be tailored to the characteristics of any system, be they strictly required or best-effort objectives. We investigated two real-world systems and compared MC-TOA to the state-of-the-art many-objective-optimization search framework Borg and random search as a baseline. A study over two systems can only say so much, but the collected statistics suggest that MC-TOA not only outperforms regular metaheuristic search but also allows the addressing of a much broader scope of problems with its simple way of including search dimensions.

The results imply that fast multi-objective optimization can be done in minutes for large-scale systems with thousands of test cases while knowing how far from optimal the test sets are in all search dimensions. It can be achieved in support of open-source solvers, which may though be slower and less robust than commercial ones. This is important as the study reveals industry-relevant collected metrics and tightens the gap between the literature and industry needs. With the given examples, MC-TOA further hopefully lowers the entry bar for organizations to apply multi-criteria selection.

Many avenues open up for future work. Dynamicity, i.e., adding or removing functions throughout the search, is a field to be investigated, as this may allow adaptive human-in-the-loop solutions. This may even extend to running tests incrementally while the optimizer is working. Even though the findings here suggest that constraint-based search on relaxed LP formulations seems most promising, more empirical evidence is required to better understand how formulation complexity compares among the linear vs. integer program and strict vs. best-effort search dimensions. Parallelizability influences the scalability of MC-TOA

[14] Clp has 324 and Cbc 631 GitHub stars.

[15] https://www.gurobi.com/solutions/gurobi-optimizer/

positively, as the runtime is bound by the slowest individual execution of a single solver, even for larger systems with more complex constraints and assuming access to sufficiently many resources (trading time for resources). Of further relevance is the closer investigation of scalability, i.e., for what systems may it *not* scale? Also, while open-source solutions performed worse than commercial ones here, it should be further evaluated whether this generalizes and what the expected drop in performance may be. The MC-TOA framework further allows for reinforcement learning in Phase 2.b, where approaches such as surrogate modeling or traditional machine learning techniques may be tried for faster convergence in the choice of weights and/or objectives for the partial formulations. Including criteria considering test execution time, something very relevant in practice, is another way to extend the framework's empirical support and applicability. Another important area is the selection process from the Pareto-front, where user experience should be investigated as well as the choice of good metrics for automated selection.

CRediT authorship contribution statement

Felix Dobslaw: Conceptualization, Methodology, Software, Data curation, Writing – original draft, Visualization, Investigation, Validation, Writing – review & editing. **Ruiyuan Wan:** Data Collection, Company contact. **Yuechan Hao:** Data collection, Company contact.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

The authors would like to express sincere gratitude to Robert Feldt for his support, in particular during the ideation phase of this research.

References

Arrieta, Aitor, Valle, Pablo, Agirre, Joseba A., Sagardui, Goiuria, 2023. Some seeds are strong: Seeding strategies for search-based test case selection. ACM Trans. Softw. Eng. Methodol. 32 (1), 1–47.

Arrieta, Aitor, Wang, Shuai, Markiegi, Urtzi, Arruabarrena, Ainhoa, Etxeberria, Leire, Sagardui, Goiuria, 2019. Pareto efficient multi-objective black-box test case selection for simulation-based testing. Inf. Softw. Technol. 114, 137–154.

Auger, Anne, Bader, Johannes, Brockhoff, Dimo, Zitzler, Eckart, 2012. Hypervolume-based multiobjective optimization: Theoretical foundations and practical implications. Theoret. Comput. Sci. 425, 75–103.

Bader, Johannes, Deb, Kalyanmoy, Zitzler, Eckart, 2010. Faster hypervolume-based search using Monte Carlo sampling. In: Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems. Springer, pp. 313–326.

Bin Ali, Nauman, Engström, Emelie, Taromirad, Masoumeh, Mousavi, Mohammad Reza, Minhas, Nasir Mehmood, Helgesson, Daniel, Kunze, Sebastian, Varshosaz, Mahsa, 2019. On the search for industry-relevant regression testing research. Empir. Softw. Eng. 24 (4), 2020–2055.

Chen, T.Y., Lau, M.F., 1970. Heuristics towards the optimization of the size of a test suite. WIT Trans. Inf. Commun. Technol. 14.

Chen, Tao, Li, Miqing, 2022. The weights can be harmful: Pareto search versus weighted search in multi-objective search-based software engineering. ACM Trans. Softw. Eng. Methodol..

Chvatal, Vasek, 1979. A greedy heuristic for the set-covering problem. Math. Oper. Res. 4 (3), 233–235.

Coello Coello, Carlos A., 2016. Constraint-handling techniques used with evolutionary algorithms. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion. pp. 563–587.

Cook, Stephen, 2006. The P versus NP problem. In: The Millennium Prize Problems. pp. 87–104.

Coviello, Carmen, Romano, Simone, Scanniello, Giuseppe, 2018. An empirical study of inadequate and adequate test suite reduction approaches. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–10.

Coviello, Carmen, Romano, Simone, Scanniello, Giuseppe, Marchetto, Alessandro, Corazza, Anna, Antoniol, Giuliano, 2020. Adequate vs. inadequate test suite reduction approaches. Inf. Softw. Technol. 119, 106224.

Deb, Kalyanmoy, Pratap, Amrit, Agarwal, Sameer, Meyarivan, TAMT, 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6 (2), 182–197.

Dong, Shi, Xue, Yinxing, Brinkkemper, Sjaak, Li, Yan-Fu, 2022. Multi-objective integer programming approaches to next release problem—Enhancing exact methods for finding whole pareto front. Inf. Softw. Technol. 147, 106825.

Feldt, Robert, Poulding, Simon, 2015. Broadening the search in search-based software testing: It need not be evolutionary. In: Search-Based Software Testing (SBST), 2015 IEEE Eigth Int. Workshop on. IEEE.

Garousi, Vahid, Özkan, Ramazan, Betin-Can, Aysu, 2018. Multi-objective regression test selection in practice: An empirical study in the defense software industry. Inf. Softw. Technol. 103, 40–54.

Hadka, David, Reed, Patrick, 2013. Borg: An auto-adaptive many-objective evolutionary computing framework. Evol. Comput. 21 (2), 231–259.

Herzig, Kim, Greiler, Michaela, Czerwonka, Jacek, Murphy, Brendan, 2015. The art of testing less without sacrificing quality. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Vol. 1, IEEE, pp. 483–493.

Hierons, Robert M., Li, Miqing, Liu, Xiaohui, Parejo, Jose Antonio, Segura, Sergio, Yao, Xin, 2020. Many-objective test suite generation for software product lines. ACM Trans. Softw. Eng. Methodol. (TOSEM) 29 (1), 1–46.

Jones, Donald R., Schonlau, Matthias, Welch, William J., 1998. Efficient global optimization of expensive black-box functions. J. Global Optim. 13 (4), 455.

Khan, Saif Ur Rehman, Lee, Sai Peck, Javaid, Nadeem, Abdul, Wadood, 2018. A systematic review on test suite reduction: Approaches, experiment's quality evaluation, and guidelines. IEEE Access 6, 11816–11841.

Khatibsyarbini, Muhammad, Isa, Mohd Adham, Jawawi, Dayang N.A., Tumeng, Rooster, 2018. Test case prioritization approaches in regression testing: A systematic literature review. Inf. Softw. Technol. 93, 74–93.

Konak, Abdullah, Coit, David W., Smith, Alice E., 2006. Multi-objective optimization using genetic algorithms: A tutorial. Reliab. Eng. Syst. Saf. 91 (9), 992–1007.

Li, Zheng, Harman, Mark, Hierons, Robert M., 2007. Search algorithms for regression test case prioritization. IEEE Trans. Softw. Eng. 33 (4), 225–237.

Lin, Jun-Wei, Jabbarvand, Reyhaneh, Garcia, Joshua, Malek, Sam, 2018. Nemo: multi-criteria test-suite minimization with integer nonlinear programming. In: 2018 IEEE/ACM 40th International Conference on Software Engineering. ICSE, IEEE, pp. 1039–1049.

Mirarab, Siavash, Akhlaghi, Soroush, Tahvildari, Ladan, 2011. Size-constrained regression test case selection using multicriteria optimization. IEEE Trans. Softw. Eng. 38 (4), 936–956.

Mondal, D., Hemmati, H., Durocher, S., 2015. 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, pp. 1–10.

Özener, O. Örsan, Sözer, Hasan, 2020. An effective formulation of the multi-criteria test suite minimization problem. J. Syst. Softw. 168, 110632.

Panichella, Annibale, Kifetew, Fitsum Meshesha, Tonella, Paolo, 2017. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. IEEE Trans. Softw. Eng. 44 (2), 122–158.

Rothermel, Gregg, Untch, Roland H., Chu, Chengyun, Harrold, Mary Jean, 2001. Prioritizing test cases for regression testing. IEEE Trans. Softw. Eng. 27 (10), 929–948.

Shukla, Anupriya, Pandey, Hari Mohan, Mehrotra, Deepti, 2015. Comparative review of selection techniques in genetic algorithm. In: 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management. ABLAZE, IEEE, pp. 515–519.

Smith, Adam M., Kapfhammer, Gregory M., 2009. An empirical study of incorporating cost into test suite reduction and prioritization. In: Proceedings of the 2009 ACM Symposium on Applied Computing. pp. 461–467.

Tallam, Sriraman, Gupta, Neelam, 2005. A concept analysis inspired greedy algorithm for test suite minimization. ACM SIGSOFT Softw. Eng. Notes 31 (1), 35–42.

Vierhauser, Michael, Rabiser, Rick, Grünbacher, Paul, 2014. A case study on testing, commissioning, and operation of very-large-scale software systems. In: Companion Proceedings of the 36th International Conference on Software Engineering. pp. 125–134.

Wang, Shuai, Ali, Shaukat, Yue, Tao, Bakkeli, Øyvind, Liaaen, Marius, 2016. Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search. In: Proceedings of the 38th International Conference on Software Engineering Companion. ICSE '16, Association for Computing Machinery, pp. 182–191.

Williams, H. Paul, 2013. Model Building in Mathematical Programming. John Wiley & Sons.

Wolsey, Laurence A., 1998. Integer Programming. Vol. 52, John Wiley & Sons.

Xue, Yinxing, Li, Yan-Fu, 2020. Multi-objective integer programming approaches for solving the multi-criteria test-suite minimization problem: Towards sound and complete solutions of a particular search-based software-engineering problem. ACM Trans. Softw. Eng. Methodol. (TOSEM) 29 (3), 1–50.

Yoo, Shin, Harman, Mark, 2012. Regression testing minimization, selection and prioritization: a survey. Softw. Test. Verif. Reliab. 22 (2), 67–120.

Zhang, Lu, Hou, Shan-Shan, Guo, Chao, Xie, Tao, Mei, Hong, 2009. Time-aware test-case prioritization using integer linear programming. In: Proceedings of the Eighteenth International Symposium on Software Testing and Analysis. pp. 213–224.

Felix Dobslaw is an assistant professor with the Mid Sweden University, where he initiated and heads the Software Engineering & Education research group since 2023. He holds a Ph.D. in Computer Engineering (2015), and did a Postdoc in Software Engineering at the Chalmers University of Technology (2019–2021). His research interests include augmented and automated software testing, the gap between academia and industry, as well as explainable AI.

Ruiyuan Wan obtained a Ph.D. degree from Tsinghua University EE in 2012. Her expertise is in software and system testing, and she currently works with cloud computing at Huawei Technologies Co. She has more than ten years of experience in the field of Cyber Physical System level testing methodology and toolchain. The last six years she has focused on AI for software testing. She has published several papers in top conferences and journals, and holds several patents.

Yuechan Hao received her Master of Engineering and Ph.D. degrees in Computer and Information Sciences from the Tokyo University of Agriculture and Technology (TUAT), Japan, in 2014 and 2017, respectively. From 2017 to present, Yuechan has worked at Huawei Technologies Co., Ltd. as a senior engineer conducting in-depth research on intelligent R&D tools.