A Hybrid Algorithm for Multi-objective Test Case Selection

Takfarinas Saber*‡, Florian Delavernhe*†, Mike Papadakis‡, Michael O'Neill*‡, Anthony Ventresque*†
*Lero, †School of Computer Science, University College Dublin, Ireland
‡ Natural Computing Research and Applications Group, School of Business, University College Dublin, Ireland
florian.delavernhe@ucdconnect.ie, {takfarinas.saber,m.oneill,anthony.ventresque}@ucd.ie,
§ Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, michail.papadakis@uni.lu

***Abstract*—Testing is crucial to ensure the quality of software systems – but testing is an expensive process, so test managers try to *minimise* the set of tests to run to save computing resources and speed up the testing process and analysis. One problem is that there are different perspectives on what is a *good* test and it is usually not possible to compare these dimensions. This is a perfect example of a *multi-objective optimisation* problem, which is hard — especially given the scale of the search space here. In this paper, we propose a novel hybrid algorithm to address this problem. Our method is composed of three steps: a greedy algorithm to find quickly some good solutions, a genetic algorithm to increase the search space covered and a local search algorithm to refine the solutions. We demonstrate through a large scale empirical evaluation that our method is more reliable (better whatever the time budget) and more robust (better whatever the number of dimensions considered) – in the scenario with 4 objectives and a default execution time, we are 268% better in hypervolume on average than the state-of-the-art algorithms.**

***Index Terms*—Multi-objective Optimisation, Hybrid-metaheuristic, Search-based Software Engineering, Test Suite Selection,**

I. Introduction

"Test early, test often": one of the most iconic principles of modern software development methods consists in continuously testing software artefacts – in order to fix problems quickly. However, programs tend to have a large number of tests and running all of them becomes not practical (or feasible) as running and analysing tests is expensive in terms of resources (servers) and manpower [10]. Many techniques exist to address this problem, from test generation [2] to distributed testing [15], test case minimisation [25] to test case prioritisation [26]. In this paper, we address *test selection* [31], which can be defined as "find a minimal subset of all the tests that covers as much of the program as possible" – in order to speed up the testing process, and limit the impact on the material resources (servers on which to run tests) and human resources (tests results usually need to be analysed by developers/testers).

Coverage is the key here and there are multiple ways of computing it (see Section III), each of them telling something different about the program under test and there is no agreement on which one of them is the best (see Section II). Time is also a dimension of the problem, as running tests takes time: is it better to save 10 minutes or to spend these 10 minutes on stressing more the program with tests that focus on branch coverage? Do we need so many tests doing line coverage or should we save an hour of testing? Etc. This is a typical multi-objective problem: test resource managers want to make decisions based on *good* subsets of the tests, i.e., sets of tests that are better than any other possible set on a particular combination of objectives. Eventually, test resource managers look at the different possible sets and make a decision based on a local optimisation (e.g., favouring objective 1 which gives a bigger gain than objective 2 while the latter is usually more important etc.).

In such large search spaces, many optimisation techniques do not work well: exact solvers (e.g., MILP) only handle a single objective thus would require a scalarisation of some sort of the objectives, and do not scale well anyway. While greedy algorithms and neighbouring search algorithms are too poor (due to their guided and non-diverse search) or too slow (due to their local nature). Evolutionary algorithms (NSGA-II, MOEA/D), on the other hand, have recently proven to be better [32], [34], while hybrid algorithms, i.e., algorithms composed of various other algorithms, have not been studied extensively for this problem (with the notable exception of Yoo and Harman [32]). The main contribution of this paper is to perform a thorough study (based on similar data as the state-of-the-art [31], [34]) of the application of a three step method, that has recently proven to be good in large scale multi-objective problems [28], [29], [16]. In particular, we introduce our own method, GREAP (for *GReedy Evolutionary Algorithm Path-relinking*), based on a greedy algorithm to find quickly some good solutions, a genetic algorithm to increase the search space covered and a local search algorithm to refine the solutions. We show that our solution is (i) more robust than the state-of-the-art to the number of objectives; (ii) is more effective (better quality and diversity) ; (iii) is more efficient (can work in tighter time budgets); and (iv) scales better when the size of the programs and the number of tests increase. In particular, we show that GREAP loses only 2% of quality when the time budget is reduced to 25% – while state-of-the-art algorithms lose 40+% (and they cannot find any solutions for 3 out of 10 subjects). We also show that GREAP is the only algorithm which always finds solutions when the number of objectives increases from 2 to 4 (quality reduced by only 8%). In general, for 4 objectives and a "normal" time budget (see Section V-E), GREAP is 268% better than the other algorithms (if we exclude the subjects for which the other algorithms cannot find any solutions) – see Table VI.

This paper is organised as follows: Section II presents the related work, Section III defines formally the problem; Section IV introduces our algorithm; Sections V and VI present the experimental setup and the results of the evaluation; and finally Section VII concludes the paper.

II. Related Work

Testing software systems usually takes the form of creating and running test cases, which aim at checking whether the program under test is behaving as expected. Any wrong program logic is supposed to make the tests fail – while when tests pass, we assume the behaviour of the program under test is correct (as long as the tests are exhaustive and correct). The set of all test cases is called a test suite. Developers and testers have worked with all sorts of metrics to assess the quality of tests, and in particular coverage metrics are widely used. Test resource managers also consider resource consumption (e.g., tests execution time, financial cost of running the tests) as an important element. Creating tests is a skill- and labour-intensive tasks and automatic test generation [2], [14] is the focus of a lot of interest in academia and industry. Software artefacts evolve constantly though: various stakeholders and developers change their opinions about what applications should do and how to achieve their goals, parts of the code are improved or re-factored, and so on. Tests are then used to ensure that nothing bad is introduced in the programs and they are executed very often against the evolving programs. Given the size of the software artefacts and the complexity of software engineering teams and projects – this is quickly a challenge, with a lot of resources dedicated to running the tests and analysing their outputs.

Three directions have been proposed to address this problem [23]: *Test Case Minimisation* [25] aims at providing subsets of the initial test suite by eliminating redundant tests [21], [4], i.e., tests that can be forgotten with no major impact *Test Case Prioritisation* [11] aims at reordering the test suite, in order to find the best order in which to apply the tests and increase the chances of finding the defects at an early stage *Test Case Selection*, the one we address in this paper and the focus of the remaining of this paragraph, aims at selecting a subset of the original test suite which balances quality and cost of the test suite. Early works have often addressed the problem as a single objective, with one property fixed (either cost or quality) and the other property optimised. Fischer et al. [13] proposed a 0-1 integer programming problem formulation while Rothermel et al. [25]) later proposed a graphical representation of the problem. In 2001, an empirical comparison study by Mansour et al. [22] compared the early algorithms applied on test case selection. In 2007, Yoo and Harman [31] presented the first work on multi-objective test case selection and introduced the concept of Pareto efficiency of selected set of tests, that they solved using an evolutionary algorithm (i.e., NSGA-II [6]). They have later extended their study with a greedy algorithm [32] and produced a hybrid algorithm. The objectives considered were execution time (cost) and statement coverage (quality). Evolutionary algorithms are the most popular algorithms for this problem: Dipesh et al. [23] use a cluster-based evolutionary algorithm and consider four objectives (execution time and three quality measures); while Zheng et al. [34] evaluate MOEAD for the problem, using a various number of coverage objectives (see Section III).

Now that the problem is widely considered multi-objective, the question of picking the right objectives becomes important. Indeed, when objectives are not independent, they have an impact on each other and optimising one may lead to optimising others – and in turn algorithms that optimise this "dominant" objective(s) have an advantage. It is not a surprise that the coverage metrics we propose in our study are not totally independent. After all, they all try to maximise *how much* of the program is tested. The relative impact of one on the others is not clear though and recent studies by Gregory Gay [17], [18] show that there is a lot of research to be done in this domain – Gay's work addresses another problem (namely test generation) but we think his conclusions can be extended to our work. In particular, Gay's finding that simple objectives (line/branch coverage) are very efficient is a sign that they cannot be dismissed. Another of Gay's conclusions is that combinations of objectives have unique results, that are difficult to achieve when using single-/mono-objectives. This also reinforces our ideas that the software testing field (and in particular test selection) needs multi-objective approaches.

III. Problem Definition

Test selection is an important test resource management problem where algorithms try to minimise the number of tests required to stress a software artefact. Each test in a test suite *covers* in some ways a part of the program and the general idea is to find the minimal set of tests that covers the whole program – or the largest part of the program. The general idea being that running less tests (or just the right number of tests) can improve the resource management.

There are various test coverage metrics: line coverage, instruction coverage, branch coverage, etc. In this paper, we select 3 of them in addition to the running time of the tests (see below the definition of the problem's objectives). Each of these coverage metrics sees the program from a different perspective: for instance as a sequence of instructions (instruction coverage) or as a tree (branch coverage). Anyway, our approach is to some extent agnostic to the different dimensions (coverage metrics) used. In fact, our goal in this paper is to compare algorithms that are designed to explore large multi-dimensional search spaces - and that is why we picked them. The choice of objectives is in this case of lesser importance.

If a subset of tests T covers, according to a coverage metric c_j, the element $p_k^j \in P^j$ of the program $\mathcal{P}$ then we have the following equation: $c_j(T, p_k^j) = 1$. We use this equation to assess how much of a program is covered by a set of tests, knowing that a program $\mathcal{P}$ is made of all its elements p_k^j. We say that a program $\mathcal{P}$ is fully covered, according to coverage metric c_j, by a set of tests $T = \{t_1, t_2, \ldots, t_n\}$ (each $t_i \in \mathcal{T}$, $\mathcal{T}$ being the original, full, set of tests), if:

$$\forall p_k^j \in P^j, \quad s.t.\ c_j(T, p_k^j) = 1 \tag{1}$$

As said in the introduction of this paper, we see this problem as a multi-objective optimisation problem, and we try 3 different combinations of objectives. Each of the different coverage metrics, as well as the time to run tests, is seen as a different objective $O_j \in O$.

The first coverage objective O_1 aims at maximising the *line coverage* c_1 of lines $p_k^1 \in P^1$ of the program $\mathcal{P}$. The second

coverage objective O_2 aims at maximising the *branch coverage* c_2 of code branches (control structures with all their branches executed at least once) $p_k^2 \in P^2$ of the program $\mathcal{P}$. The last coverage objective O_3 is inspired by the modified condition / decision coverage (MC/DC) where we keep two criteria and aim at maximising the *MC/DC* c_3 of decision coverage $p_k^3 \in P^3$ of the program $\mathcal{P}$. We consider that a statement *if* is MC/DC covered if and only if the two following criteria are met: (i) each decision takes the value true and false during testing; and (ii) each condition takes the value true and false during testing – where a condition is a boolean atomic expression and a decision is a boolean expression composed of different boolean operators and conditions. The fourth and final objective O_4 we consider in our work is the cost in execution time of running all the selected tests.

The multi-objective model of our problem can be described through the following formulation:

$$\begin{aligned} minimise: O_j &= - \sum_{p_k^j \in P^j} c_j(T, p_k^j) \quad \forall j \in \{1,2,3\} \\ O_4 &= \sum_{t_i \in T} cost(t_i) \\ subject\ to: T &= \{t_1, t_2, \ldots, t_n\} \in \wp(\mathcal{T}) \end{aligned} \tag{2}$$

The aim of multi-objective optimisation techniques is to find the set of Pareto optimal solutions [7] (a.k.a., Pareto front). A solution $T^j = \{t_1^j, t_2^j, \ldots, t_m^j\} \in \wp(\mathcal{T})$ is in the Pareto optimal set (also said as non-dominated) if and only if there is no solution $T^k = \{t_1^k, t_2^k, \ldots, t_l^k\} \in \wp(\mathcal{T})$ with all objectives (c_i's and cost in our case) better or equal than the objectives of T^j, with at least one objective strictly better. T^j is said to be dominated by T^k if and only if $\forall O_i \in O,\ O_i(T^k) \leq O_i(T^j)$ and $\exists O_k \in O \mid O_k(T^k) < O_k(T^j)$. In short, each solution on the Pareto front is a good and makes a unique trade-off between the objectives – in the sense that no other currently found solution is better than this solution on all dimensions.

IV. GREAP: GReedy Evolutionary Algorithm Path-relinking

Large and complex search spaces, such as the ones we are addressing in our work, are challenging for classical optimisation techniques. Random/greedy techniques generate a lot of poor solutions and do not work well; Local Search techniques do not progress quickly enough in such large search spaces; exact solvers are mostly mono-objective and require complex scalarisation techniques and do not scale well. The solutions that seem the most promising are evolutionary algorithms [31], [34], sometimes combined with greedy algorithms [32]. However, they are known to improve the set of solutions quite slowly, due in part to the poor initial population (hence the use of greedy algorithms to bootstrap the evolutionary algorithms).

Our solution uses a three stage method [28], [16] composed of three optimisation algorithms applied successively: (i) first we use a modified GRASP [12] in order to 'aggressively' produce a good initial population with good values – and also a good coverage of the non-dominated set of solutions (i.e., with good variety among the different solutions); (ii) the second phase is a classic evolutionary algorithm, NSGAII [6] in our case, but applied to the population produced by the first phase. This phase aims to exploit information from GRASP and to produce a good approximation of the Pareto set; and (iii) finally a third phase pushes locally and aggressively solutions from previous stages to the Pareto front; in our algorithm we used a path-relinking [3]. Each of the steps is executed for a maximum given time based on the overall execution. The greedy part of the first phase and the third can also be interrupted if they achieve a certain goal (e.g., when GRASP generates enough initial solutions, or when the path-relinking has paired all solutions).

A. First Step: Greedy Algorithm to Generate the Initial Population

The aim of this first step is to produce the initial population of 100 individuals – required by GREAP's second step (using NSGA-II). We use a modified GRASP algorithm [12] to create a first set of 80 solutions. The algorithm builds a solution from an empty subset of tests and at each iteration, the algorithm adds randomly one of the best tests (say, t_i) according to a particular utility function. The utility is the normalised sum of coverage values t_i can add to the solution, divided by the number of coverage objectives. This value is then multiplied by a direction λ, then we subtract the sigmoid normalised cost of the test multiplied also by a direction $1 - \lambda$. This direction is modified for each solution built to obtain totally different solutions – each solution has a different "perspective" on the coverage, and cost, objectives.

Let $T \subseteq \mathcal{T}$ be the selected tests at a given iteration of the GRASP phase, with coverage objectives $O_j^{'},\ \forall j \in \{1,2,3\}$. Let $O_j^{''},\ \forall j \in \{1,2,3\}$ be the new coverage objectives when adding a test $t_i \in \mathcal{T} \setminus T$ to T. We consider that $g^j(t_i) = O_j^{'} - O_j^{''},\ \forall j \in \{1,2,3\}$ the potential gain on each of the considered coverage objectives when adding t_i to the current set of selected tests T. We also increase the running time of the set of tests when adding test t_i, $w^j(t_i) = O_4^{''} - O_4^{'}$.

Finally, we define the utility $U(t_i)$ of adding a test t_i to the current selection of tests as follows:

$$\begin{aligned} x_1 &= \sum_{j \in \{1,2,3\}} \frac{g^j(t_i)}{max_{t_k \in \mathcal{T}}\{g^j(t_k)\}} \\ x_2 &= \frac{w^4(t_i)}{max_{t_k \in \mathcal{T}}\{w^4(t_k)\}} \\ U(t_i) &= x_1 \times \frac{\lambda}{3} - \frac{(1-\lambda)}{(1+e^{-x_2})} \end{aligned} \tag{3}$$

with $\lambda \in [0,1]$ being the value that allows the exploration of different directions in the search space.The first two λ directions picked are 0 and 1. The former builds a solution with no selected tests (so a cost objective of 0) and the latter with the maximum coverage in all objectives (not necessarily selecting all the tests). After that, the directions (values of λ) are chosen to spread the exploration as much as possible, playing with coverage and cost (splitting the space covered by 2 previous values of λ) – λ will successively be 0.5, 0.25, 0.75, 0.125, 0.375, etc.

At each iteration, the algorithm picks a (new) test from the original test suite so that the utility function is increased of at least a factor $\alpha \leq 1$. This factor allows the algorithm to pick good tests, increasing the overall values of the selected set of tests.The stopping criteria of the algorithm, i.e., the conditions that stop the algorithm from adding tests are: (i) no tests left to add, (ii) full coverage of lines, branches and MC/DCs or (iii) a negative utility for all the tests. In our implementation, we fixed the value of α at 0.70.

At the end of the algorithm, the set of solutions is made of diverse good solutions. After that, a path-relinking algorithm [19] is applied. The path-relinking algorithm takes a set of solutions and for each pair of solutions (a.k.a., "parents"), the algorithm "navigates" from one parent to the other parent by adding or removing tests one at a time. After each step (removing or adding a test) if the current solution is not dominated by its parents, the algorithm adds it to the set of solutions.

Finally, to obtain a population of 100 solutions for the second phase, we sort and keep only the non-dominated solutions. If the number of non-dominated solutions is greater than 100, we only keep the solutions having the best crowding distance [6] (to increase the variety of the solutions). If the number of non-dominated solutions is less than 100 the algorithm takes solutions of lower ranks, i.e., solutions that are dominated only by the non-dominated solutions and so on. These operations are repeated until the number of solutions is 100.

B. Second Step: Evolutionary Algorithm

GREAP's second step is an evolutionary algorithm, i.e., a stochastic optimisation method, inspired by natural evolution – in particular evolutionary algorithms implement 'their' version of the concepts of mutation, crossover and selection. In our algorithm, we use the Non-dominated Sorting Genetic Algorithm II (NSGAII) [6]. It is a Pareto-based algorithm, which aims to select solutions using Pareto dominance and in case of non domination between two individuals use the density of the neighbourhood of individuals. The initial population (a critical element for every evolutionary algorithm) is generated by the first step of our hybrid algorithm, which (as we'll see in the evaluation section) gives a good bootstrap to the evolutionary algorithm.

C. Third Step: Path-relinking

Finally, the third step is a path-relinking applied on the result of the second phase. We use the same idea seen in the first step: for every pair of solutions, the path-relinking algorithm looks for intermediary (feasible) solutions between them by adding or removing tests (one at a time). The intermediary solutions are then added to the list of good solutions if they are non-dominated. The algorithm runs for a certain time budget or until there is no more pair of solutions that have not been tried. The aim of the path-relinking step of GREAP is to "fill in" the gaps between solutions belonging to the Pareto front and to push this front as much as possible. We picked this algorithm and not any other local search algorithm (e.g., PLS as in other related work using three step methods[28], [29], [27]) as the search space in our problem is not too constrained and most intermediary solutions are feasible.

V. Experimental Setup

We present here the setup of our experiments: the metrics used to evaluate our algorithms, the different subjects, the time budget for each phase of GREAP, and the other algorithms we compare GREAP against.

A. Metrics

Comparing different sets of non-dominated solutions in a multi-objective context is a well-known problem [24]. First of all, because of the complexity and the size of the problem, it is usually not possible to obtain the exact Pareto frontier, i.e., the exact solutions. Second of all, because of the difficulties to compare (and visualise) solutions in a multi-dimensional space. Different metrics have been proposed in the literature, often to measure the sets of non-dominated solutions from various perspectives – namely the quality of the sets of solutions and the spread of the sets of solutions. Some of the metrics require to have the exact set of non-dominated solutions (the ideal Pareto front). As we do not have this (it is in practice often impossible to obtain) we use a classical estimation: the best set solutions given by all the algorithms. All of the metrics we use are known to be good ones for the comparison of sets of non-dominated solutions [24].

- We use the Hypervolume (HV) [35] (one of the most popular metrics) to assess the quality and the diversity of the sets of solutions. The Hypervolume computes the space between all solutions from the non-dominated set of solutions and a reference point. The exact Pareto front produces the best Hypervolume, as its solutions get the best values in every objective. Because of its well-known utility, popularity [24] and good performance when it comes to comparing solutions, the Hypervolume is our favourite metric (and many of our experiments will only consider this one).
- The second metric, which focuses on quality, is the generational distance (GD) [30] which computes the average Euclidean distance between solutions from the result of an algorithm and the nearest solution from our approximation of the exact Pareto front. The smaller the GD the better the solution is.
- The third metric is the inverted generational distance (IGD) [5], similar to the generational distance but computes the minimum distance between the result set of an algorithm and the approximated Pareto set. IGD can be used for both diversity and quality, and it is also to be minimised.
- The generalised spread metric (GS) [24] computes the spread of a population using the lower and upper bounds of objective values found and can be used for the diversity of solution sets. This metric is useful if the sets of solutions have similar qualities. For the generalised spread metric, the lower the better.

	Data-set									
	gzip-v3	gzip-v4	space-v38	schedule-v2	totinfo-v1	tcas-v1	space13k-v38	grep	sed	make-v1
LOC	7259	7359	6199	413	407	174	6199	10068	14427	35545
# tests	214	214	150	2650	1052	1608	13585	809	370	1044

Table I
CHARACTERISTICS OF OUR DATA SET: 9 SUBJECTS AND THEIR NUMBER OF LINES OF CODE (LOC) AND NUMBER OF TESTS.

- We also use the Pareto front size metric (PFS), which returns the number of solutions in the result set of an algorithm that also belongs to the Pareto front. PFS is diversity-oriented, and the higher the better for PFS.
- Finally, our last metric is the epsilon metric (ϵ) [36], which estimates the minimal distance to transform every solution of a result set into a solution on the Pareto front. ϵ addresses both diversity and quality and the lower ϵ the better.

All these metrics are computed with the metric tool of JMetal [9].

B. Data Set

We have picked 6 out of 10 subjects based on recent and/or widely considered state-of-the-art studies [31], [34]. We also selected larger subjects, doing what we think is a thorough analysis[1] We summarise the most interesting characteristics of the subjects in Table I. In particular, we report the number of lines of code (LOC) and number of tests. All the subjects (see Table I) are C programs, obtained from the Software-artifact Infrastructure Repository (SIR [8], [1]), a large repository of software artefacts popular in academia.

We use the tool Gcov to measure the coverage metrics. Gcov is a source code coverage analysis and profiling tool that gives which line/instruction/branch is executed by which test. We also collected the execution time of all tests – which gives us the cost value for each test.

C. Algorithms

We have compared our own solution (GREAP) against two well-known evolutionary algorithms: NSGAII [6] and MOEAD [33]. They are both executed in the exact same conditions as our solution. NSGA-II is one of the most popular genetic algorithm – note that we use NSGA-II in the second phase of GREAP (see Section IV-B). MOEAD has recently been used in a multi-objective test selection work [34] and has proven to have very good results.

Note that we did not have access to the implementation provided by the authors of [34] and [6]. We decided to stay as generic and impartial as possible, and we used the MOEA framework [20]. MOEA is a free and open source Java library with a panel of multi-objective evolutionary algorithms implemented and ready to use. Our algorithm is also implemented in Java, and also use the MOEA framework for its second phase (NSGA-II). The parameters used for the two algorithms and for our second phase are the default values from MOEA framework (population sizes equal to 100).

[1]Running much larger subjects would have been tricky anyway for our study, as they would require a lot of pre-processing (identifying individual tests and running them is not always an easy task) and the evaluation would also require a lot of resources (computing resources and time). We believe the subjects we have picked are relevant for our study as they are recognised by the community and of decent size.

D. Setting the Time Budget for Each Phase

GREAP is a three step hybrid algorithm, and we need to allocate an execution time for each of the steps. We have selected 12 different triplets of parameters, each parameter representing the percentage of time allowed to the corresponding phase. We ran GREAP 10 times with each triplet on one of the hardest problem (i.e., Schedule with 4 objectives) and evaluated the average results on all our metrics.

Table II reports the triplets alongside their average result metrics with the following format: a triplet "a, b, c" corresponds to $a\%$ of the time allowed on the first phase, $b\%$ on the second phase and $c\%$ on the last. In bold are the best values. The results show the importance of the first and third steps: the worst results are coming from parameters with no time allowed to the first or the third steps. However, cases with a long time allowed for these two steps (first and third) do not produce good solutions either. Secondly, two different parameters seem better and distinct from the others: "10,80,10" and "10,70,20". The former favours the quality of the solution, whereas the latter produces more solutions and a better diversity. Therefore, we decide to pick the last triplet for all following experiments in this work: 10% for the first phase, 75% for the second and 15% for the last, which offers the best trade-off between diversity and quality, with a good set of quality metrics, close to the best "10,80,10" and a good diversity, second best after "10,70,20".

E. Experimental Process

Our experiments[2] were run on the ten subjects presented earlier, for the three combinations of objectives (two, three, four) and the three algorithms (NSGAII, MOEA/D and our own GREAP). Because evolutionary algorithms and our greedy step are stochastic methods, we performed 10 runs for each algorithm in each experimental set up to minimise the impact of randomness.

We varied the number of objectives for the problem, using 2, 3 or 4 objectives by taking (in this order): (i) the cost, (ii) the line coverage, (ii) the branch coverage and (iv) the MC/DC coverage. The more objectives there are in the problem the more difficult the problem is. We also varied the time budget (see Table III) for the algorithms, starting with a value T that corresponds to what was allowed in [34] for the six first subjects – we then decreased that value to $T/2$ and $T/4$. We also evaluated the algorithms on a larger budget, $2T$ to

[2]Note that we report in the following Section VI only the most relevant results, but that all our results will be uploaded online upon acceptance of this paper.

Data-set	Metric	Parameters											
		00,90,10	0,80,20	10,90,00	10,80,10	10,70,20	20,80,00	20,70,10	20,60,20	20,40,40	33,33,33	40,40,20	10,75,15
Schedule	HV	0.9638	0.9635	0.6018	**0.9649**	0.9635	0.6110	0.9624	0.9639	0.9640	0.9631	0.9631	0.9644
	GD	0.2248	0.0105	0.0703	0.0102	0.0097	0.0709	0.0093	0.0114	0.0107	**0.0078**	0.0104	0.0094
	IGD	0.1622	0.0497	0.0498	0.0501	0.0495	0.0503	**0.0494**	0.0500	0.0499	0.0500	0.0497	0.0498
	ϵ	0.0014	0.0015	0.0295	**0.0012**	0.0015	0.0296	0.0017	0.0014	0.0016	0.0015	0.0015	0.0014
	PFS	12.4	27.0	11.9	28.7	**31.7**	10.9	27.2	28.4	29.0	30.0	28.0	30.4
	GS	0.9041	0.9196	1.0787	0.9064	**0.8943**	1.1036	0.9077	0.9305	0.9338	0.9011	0.9384	0.9008

Table II

EVALUATION OF THE TIME ALLOWED FOR EACH OF THE THREE STEPS OF OUR HYBRID ALGORITHM GREAP. EACH COLUMN CORRESPONDS TO A TRIPLET OF PERCENTAGE VALUES (1 FOR EACH OF THE THREE STEPS OF GREAP). WE SHOW THE RESULTS FOR EVERY METRIC, WHEN RUNNING THE SCHEDULE SUBJECT, WITH 4 OBJECTIVES AND A GLOBAL EXECUTION TIME OF 100 SECONDS. BEST RESULTS ARE IN BOLD.

Data-set	Two objectives	Three objectives	Four objectives
gzip-v3	176.12	224.01	335.38
gzip-v4	179.38	254.06	319.70
schedule-v2	716.48	764.61	844.12
tcas-v1	204.75	217.88	243.72
tot_info-v1	294.38	297.70	335.00
Space-v38	170.00	240.00	310.00
Space13k-v38	1,000.00	1,100.00	1,200.00
grep	290.00	290.00	330.00
sed	200.00	240.00	350.00
make	295.00	300.00	340.00

Table III

SUMMARY OF THE EXECUTION TIMES T FOR EACH INSTANCE ACCORDING TO THE NUMBER OF OBJECTIVES – SEE [34] FOR DETAILS.

see how well the algorithms performed in a less constrained environment. Note that in the work of [34] they do not run their experiments on Space with three or four objectives, therefore we chose arbitrary times, similar to the ones used for the two versions of gzip. For Space13k, also not used during previous papers, we allow a big arbitrary time for 2 objectives, but afterwards for 3 and 4 objectives, times are computed using a similar scale as the increase of times alongside objectives for the three big data-sets (Tcas, Totinfo and Schedule). Finally for Make, Grep and Sed, also arbitrary times are used, picked to have similar behaviour than other data-sets.

VI. EXPERIMENTS

We aim at answering the following four questions in our study:

- **RQ 1**: Is GREAP robust against the increase of the number of objectives, which is known to be challenging?
- **RQ 2**: Is GREAP more effective (better quality and diversity) than the others evolutionary algorithms?
- **RQ 3**: How does GREAP compare to other algorithms when the time budget shrinks?
- **RQ 4**: Does GREAP react well to an increase of the test suite size (when we vary the test subjects)?

All the results presented in the current section are average values of ten runs. In order to be more readable, the values are rounded; yet, in all cases we kept enough decimals to still be able to compare the algorithms.

Values in bold are the best values from a particular perspective (described in the tables) and by default they are statistically significant (we used a Mann-Whitney U test) with a p-value below 0.05. If these (best) values (in bold) are followed by an asterisk (*) this means they are not statistically significant.

A. RQ1: Robustness against a Varying Number of Objectives

In Table IV, we report the evolution of the Hypervolume when the number of objectives increases (from 2 to 4) for each subject. The time used in the experiments is T – as this is the typical time budget in related studies. In short, we fix the metric (Hypervolume) and the time (T) and we only vary the number of objectives.

In general, the results show better performance for GREAP than for the other algorithms. Furthermore, the values are always statistically significant. However, these values are close for small subjects (the two versions of Gzip and Space), but this gap tends to increase with the size of the subjects (for medium subjects such as Grep and Sed but also for larger data-sets: Totinfo, Tcas, Schedule, Make and Space13k). Moreover, the performance of GREAP becomes clearer with the increase of the number of objectives: the efficiency of the other evolutionary algorithms tends to drop faster than for GREAP. Indeed, our algorithm has good Hypervolume values for the simpler case (2 objectives), but keeps good Hypervolume values when the problem becomes more complex (3 then 4 objectives): GREAP is less impacted by the increase in the number of objectives than the other algorithms. For Space13k, in the 3 combinations of objectives, evolutionary algorithms can't produce good solutions. However, GREAP manages to always find a good set of solutions, even for the hardest problem with 4 objectives.

This first set of experiments proves that GREAP does not struggle against more complex problems (more objectives) as other evolutionary algorithms do.

B. RQ2: Efficiency of our Algorithm

Table V is a thorough evaluation of the algorithms against various metrics, for all the subjects and with a fixed number of objectives (we chose four objectives to be in the most difficult context, for all algorithms). We also chose to compare with same times: T. In short, we fix the number of objectives and the time (T) and we only vary the metrics.

First, we notice that GREAP gets significantly better quality results (given by the four metrics: HV, IGD, GD and ϵ) than the other algorithms. The only quality result for which GREAP is not the best algorithm is GD for Tcas and Grep, and IGD for Sed. This is anyway a good demonstration that GREAP is the best algorithm for all quality metrics (37 best values out of 40 quality metrics), with at least 3 out of 4 metrics showing better results for GREAP for all subjects.

Regarding the diversity metrics (PFS and GS), results show that many more solutions are provided by GREAP than by the two evolutionary algorithms. The generalised spread of the solutions is not always in favour of GREAP though, but when another algorithm gets better results than GREAP, this

# Obj.	Algorithm	Subjects									
		gzip-v3	gzip-v4	Space	Schedule	Totinfo	Tcas	Space13k	Grep	Sed	Make
2 objectives	GREAP	**0.952425**	**0.9491085**	**0.96230**	**0.9748**	**0.7924**	**0.8495**	**0.96870**	**0.9448**	**0.93541**	**0.9954**
	NSGAII	0.952423	0.9491078	0.96225	0.0022	0.7893	0.1477	0.0	0.8610	0.93528	0.8948
	MOEAD	0.806236	0.8038152	0.82380	0.0	0.3774	0.0049	0.0	0.2117	0.32200	0.7821
3 objectives	GREAP	**0.92674**	**0.918496**	**0.95732**	**0.9746**	**0.7883**	**0.8476**	**0.96385**	**0.9232**	**0.9103**	**0.9939**
	NSGAII	0.92671	0.918474	0.95724	0.0195	0.7870	0.01966	0.0	0.7704	0.9092	0.8650
	MOEAD	0.79580	0.786363	0.85937	0.0025	0.4147	0.0191	0.0	0.2388	0.3073	0.7635
4 objectives	GREAP	**0.8792**	**0.8693**	**0.8174**	**0.9639**	**0.7331**	**0.6843**	**0.90710**	**0.8660**	**0.8655**	**0.9773**
	NSGAII	0.8789	0.8687	0.8153	0.0447	0.7101	0.1668	0.0	0.7255	0.8539	0.8054
	MOEAD	0.7329	0.7229	0.6942	0.0	0.3071	0.0150	0.0	0.3347	0.3748	0.6711

Table IV

HYPERVOLUME FOR ALL THE ALGORITHMS RUNNING WITH A TIME BUDGET T AGAINST ALL THE SUBJECTS AND FOR A VARIOUS NUMBER OF OBJECTIVES (2, 3 AND 4). BEST AND STATISTICALLY SIGNIFICANT RESULTS IN BOLD. NOTE THAT THE MOST COMPLEX (YET REALISTIC) SCENARIOS HAVE MORE OBJECTIVES.

Data-set	Algorithm	Metric					
		HV	GD	IGD	ϵ	PFS	GS
gzip-v3	GREAP	**0.8792**	**0.00026**	**0.00023**	**0.0002**	**2114.8**	**0.997**
	NSGAII	0.8789	0.00032	0.00025	0.0014	1403.6	0.965
	MOEAD	0.7329	0.00131	0.00496	0.1123	40.3	0.905
gzip-v4	GREAP	**0.8693**	**0.00027**	**0.000197**	**0.000197**	**2616.2**	0.9456
	NSGAII	0.8687	0.00037	0.000204	0.00189	1562.3	**0.9497***
	MOEAD	0.7229	0.00178	0.004236	0.140752	42.1	0.8870
Space	GREAP	**0.8174**	**0.0007**	**0.00027**	**0.001**	**2389.4**	0.812
	NSGAII	0.8153	0.0010	0.00031	0.005	1450.7	0.771
	MOEAD	0.6942	0.0030	0.00763	0.214	57.1	**0.841***
Schedule	GREAP	**0.9639**	**0.0113**	**0.0495**	**0.0013**	**27.7**	0.9054
	NSGAII	0.0477	0.0314	0.1368	0.0592	4.2	0.9164
	MOEAD	0.0	0.0901	0.1599	0.0678	1.1	**0.9979**
Totinfo	GREAP	**0.7331**	**0.0114**	**0.0120**	**5e-0.5**	**39.1**	0.6892
	NSGAII	0.7101	0.0130	0.0151	0.001	27.6	0.7992
	MOEAD	0.3071	0.0280	0.0835	0.005	2.9	**0.8920**
Tcas	GREAP	**0.6843**	0.1008	**0.1160**	**0.0**	**10.5**	0.6609
	NSGAII	0.1688	**0.0516**	0.1728	0.009	4.6	0.7887
	MOEAD	0.0150	0.1050	0.2416	0.013	1.9	**0.9356**
Space13k	GREAP	**0.90710**	**0.00008**	**0.00008**	**0.30526**	**4136.1**	**0.6229**
	NSGAII	0.0	11.24289	0.13241	15.94575	1.1	0.9997
	MOEAD	0.0	12.63097	0.143561	17.28161	1.0	1.0
Grep	GREAP	**0.8660**	0.00050	**0.00071**	**0.0276**	**5098.1**	0.6956
	NSGAII	0.7255	**0.00039**	0.00259	0.0595	551.8	**0.6095**
	MOEAD	0.3347	0.00439	0.00916	0.1787	10.0	0.9784
Sed	GREAP	**0.8655**	**0.00015**	0.00019	**0.0020**	**4600.1**	0.75
	NSGAII	0.8539	0.00020	**0.00017**	0.0069	838.3	**0.6448**
	MOEAD	0.3748	0.00230	0.00848	0.4573	16.3	0.9174
Make	GREAP	**0.9773**	**0.00007**	**0.00003**	**0.8270**	**3355.5**	**0.6834**
	NSGAII	0.8054	0.00157	0.00276	2.8760	54.8	1.1120
	MOEAD	0.6711	0.01420	0.00388	4.9919	13.9	1.0690

Table V

RESULTS FOR ALL THE METRICS FOR ALL THE ALGORITHMS RUNNING WITH A TIME BUDGET T AGAINST ALL THE SUBJECTS AND 4 OBJECTIVES. BEST AND STATISTICALLY SIGNIFICANT RESULTS IN BOLD. RESULTS WITH AN ASTERISK (*) ARE NOT STATISTICALLY SIGNIFICANTLY BETTER.

algorithm has very poor quality and size values – which to some extent beats the purpose of multi-objective optimisation.

We can conclude from this set of experiments that GREAP provides better solutions in terms of quality and produces solution sets with better diversity. Consequently, we conclude a better efficiency for GREAP than for evolutionary algorithms.

C. Impact of the Time Budget on the Algorithms

Now (see Table VI), we answer **RQ 3**, and show the 'aggressive' behaviour of GREAP. In this set of experiments we decrease the time budget (from T to $T/2$ and then $T/4$) assigned to the algorithms, making the problem more difficult. We also increase the time budget ($2T$) to evaluate whether this has a different impact on the other algorithms than on GREAP. In short, we fix the number of objectives and the metrics, and we only vary the time (T).

First, all the results show statistically better performance for GREAP. Furthermore, we observe an increase of this improvement when the time budget decreases. This says that while the other evolutionary algorithms are impacted substantially by the decrease in time budget, GREAP keeps good Hypervolume values - especially for the six hardest subjects (Grep, Make, Space13k, Schedule, Totinfo and Tcas). For these six subjects, the other algorithms perform really poorly (Hypervolume values can even tend to 0 in the hardest cases) while GREAP does not vary much. Note that for the small subjects the impact on the other algorithms' performance is limited, due to the simpler nature of these subjects.

Anyway, GREAP always keeps good Hypervolume values even with smaller time budgets. The decrease in performance is not significant for small subjects and stays acceptable for larger ones. Furthermore, because of the small evolution of the Hypervolume in some cases, GREAP seems to have already converged to good solutions in four out of the ten subjects (The two Gzip versions, Space and Schedule), for the smallest time ($T/4$). For the other subjects, good solutions are already found at $T/4$, but GREAP keeps improving a bit the solutions.It's only for Space13k, the hardest subject, that GREAP improves

Time	Algorithm	Data-set									
		gzip-v3	gzip-v4	Space	Schedule	Totinfo	Tcas	Space13k	Grep	Sed	Make
$2T$	GREAP	**0.87920**	**0.86933**	**0.81758**	**0.9698**	**0.7347**	**0.7190**	**0.91952**	**0.8698**	**0.8666**	**0.9791**
	NSGAII	0.87908	0.86896	0.81612	0.2242	0.7252	0.3485	0.0	0.7889	0.8553	0.8462
	MOEAD	0.73826	0.72709	0.70408	0.0957	0.3609	0.1257	0.0	0.3647	0.4072	0.7568
T	GREAP	**0.8792**	**0.8693**	**0.8174**	**0.9639**	**0.7331**	**0.6843**	**0.90710**	**0.8660**	**0.8655**	**0.9773**
	NSGAII	0.8789	0.8687	0.8153	0.0447	0.7101	0.1668	0.0	0.7255	0.8539	0.8054
	MOEAD	0.7329	0.7229	0.6942	0.0	0.3071	0.0150	0.0	0.3347	0.3748	0.6711
$T/2$	GREAP	**0.8791**	**0.8693**	**0.8169**	**0.9623**	**0.7296**	**0.6911**	**0.90273**	**0.8554**	**0.8642**	**0.9734**
	NSGAII	0.8788	0.8685	0.8147	0.0	0.6766	0.0282	0.0	0.6192	0.8525	0.7142
	MOEAD	0.7284	0.7173	0.6498	0.0	0.2147	0.0	0.0	0.2930	0.3378	0.5012
$T/4$	GREAP	**0.8790**	**0.8691**	**0.8167**	**0.9635**	**0.7121**	**0.6799**	**0.82848**	**0.8387**	**0.8624**	**0.9436**
	NSGAII	0.8786	0.8681	0.8137	0.0	0.6061	0.0	0.0	0.4977	0.8500	0.3420
	MOEAD	0.7229	0.7066	0.5914	0.0	0.0804	0.0	0.0	0.2204	0.2760	0.0

Table VI

HYPERVOLUME VALUES FOR ALL THE ALGORITHMS RUNNING WITH DIFFERENT TIME BUDGETS (T, $T/2$, $T/4$ AND $2T$) AGAINST ALL THE SUBJECTS AND 4 OBJECTIVES. BEST AND STATISTICALLY SIGNIFICANT RESULTS IN BOLD.

the Hypervolume significantly after $T/4$.

In terms of improvement figures, if we look at the impact on the performance of the algorithms when the time budget is reduced from T to $T/4$, we see that GREAP loses only 2% of quality, while NSGA-II (34%) and MOEAD (45%) are impacted much more and cannot get any solutions in some cases (Space13k, Schedule and Tcas). If we focus on a time budget of T (reminder: we have 4 objectives here) then GREAP is 268% better than the other algorithms (when they find any solutions at all).

D. Impact of Subjects Size

Finally, looking at the previous three tables, we can evaluate the quality and the diversity of the solutions produced by the three algorithms when the size of the subjects vary. To simplify the analysis, we classify the subjects into three categories: small-scale (Gzip v3, Gzip v4 and Space), medium-scale (Totinfo, Grep and Sed) and large-scale (Space13k, Make, Tcas and Schedule).

The first thing we notice is that the quality and diversity of the evolutionary algorithms is inversely impacted by the size of the subjects: the larger the subject the poorer the results. For bigger subjects, evolutionary algorithms have a bad performance, and the number of solutions found is decreasing with the size of the test suites. We see that NSGAII and MOEAD struggle to find a good set of solutions to push toward the Pareto frontier. We can conclude that those algorithms do not scale well when the size of the problem/subjects increases.

On the contrary, GREAP has good performance when dealing with all scales, and while the results are a little less impressive for medium- and large-scale subjects they are still good and better than for the other algorithms.

As an example, take Space13k, the hardest problem/subject. In the different experiments, the two evolutionary algorithms find only one solution (except for one run of NSGAII in the 4 objectives case, where two solutions are found when allowing time T or more). This can be explained by the structure of the problem: the subject has a large test suite and a small number of LOC so the tests tend to have a large overlap between them (they cover similar elements of the subjects). Only a small number of tests from the original test suite are already enough to obtain an optimal quality. Therefore, the random initialisation of the evolutionary algorithms is full of mostly redundant tests, making it hard for the evolutionary algorithms to improve on what's already a (near) optimal set of solutions. As a consequence, we see in our experiments that the two evolutionary algorithms optimise only the cost (and with a lot of difficulty). Anyway, GREAP manages to offer more solutions, with different qualities and with smaller costs than the initial set of tests. The solutions are of a good diversity over the different objectives, which helps the second phase to explore a larger search space and more trade-offs. This difference is due to the first and third phases, which provide a good initial population with small test suites and then a good shape at the end. We can conclude that the search space is mostly composed of solutions with the same quality as the initial test suites and a random initialisation poorly performs. GREAP is the algorithm that addressed the problems the best with a big test suite. This observation can be generalised to explain the difficulties of NSGAII and MOEAD to optimise the other difficult subjects.

Furthermore, GREAP successfully provides a larger set of solutions with both a good quality and a good diversity for the large-scale problems, in all different combinations of objectives. Even for the hardest problems, our algorithm has good solutions for small times. The larger the test suite the more GREAP outperforms the other evolutionary algorithms.

Finally, we can conclude that with the increase of the initial test suite size, the gap between GREAP and evolutionary algorithms is increasing, and while GREAP is slightly better for small subjects, it outperforms by far other algorithms for harder problems.

VII. CONCLUSION

Test selection is a very challenging yet critical problem in software testing. The number of tests that are associated with a program can be huge and running all of them is often impossible or at least prohibitively expensive. In this paper, we propose a novel hybrid algorithm (GREAP) to address this problem. Our method is composed of three steps: a greedy algorithm to find quickly some good solutions, a genetic algorithm to increase the search space covered and a local search algorithm to refine the solutions. We answer 4 research questions through a thorough empirical evaluation (based on what the previous state-of-the-art technique [34] was evaluated against):

- The answer to **RQ 1**: Is GREAP robust against the increase in the number of objectives, which is known to be challenging? is **Yes**.
- The answer to **RQ 2**: Is GREAP more effective (better quality and diversity) than the other evolutionary algorithms? is **Yes**.
- The answer to **RQ 3**: How does GREAP compare to other algorithms when the time budget shrinks? is **It is a really reliable algorithm that is not impacted (as much as others) by the time budget**.
- The answer to **RQ 4**: Does GREAP react well to an increase of the test suite size (when we vary the test subjects)? is **Yes**.

Now that we have proven that a three step method, such as GREAP, is faster at finding good solutions, and is more reliable and more robust than the state-of-the-art techniques, we would like to explore further the combination of the three steps. We know from observation that each step has a different impact on the solutions: the third phase impacts the diversity, the first one impacts the quality. To understand this better we would need to run large numbers of experiments combining each of these phases and so on. We would also like to use our 3 step method on other Software Engineering problems to evaluate how general this technique is, as on test suite minimisation which is really close to our problem, and will probably have the same difficulties to scale on large instances.

Acknowledgement

This work was supported by Science Foundation Ireland grant 13/RC/2094 to Lero.

References

[1] http://sir.unl.edu.

[2] Shaukat Ali, Lionel C Briand, Hadi Hemmati, and Rajwinder Kaur Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on Software Engineering*, 36(6):742–762, 2010.

[3] Matthieu Basseur, Franck Seynhaeve, and El-Ghazali Talbi. Path relinking in pareto multi-objective genetic algorithms. In *EMO*, pages 120–134, 2005.

[4] Tsong Yueh Chen and Man Fai Lau. Dividing strategies for the optimization of a test suite. *Information Processing Letters*, 60(3):135–141, 1996.

[5] Carlos A Coello Coello and Nareli Cruz Cortés. Solving multiobjective optimization problems using an artificial immune system. *Genetic Programming and Evolvable Machines*, 6(2):163–190, 2005.

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

[7] Kalyanmoy Deb, Karthik Sindhya, and Jussi Hakanen. Multi-objective optimization. In *Decision Sciences: Theory and Practice*, pages 145–184. CRC Press, 2016.

[8] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10(4):405–435, 2005.

[9] Juan J Durillo and Antonio J Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[10] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. *Prioritizing test cases for regression testing*, volume 25. ACM, 2000.

[11] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering*, 28(2):159–182, 2002.

[12] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[13] Kurt F Fischer. A test case selection method for the validation of software maintenance modifications. In *Proceedings of COMPSAC*, volume 77, pages 421–426, 1977.

[14] Gordon Fraser and Andrea Arcuri. Evosuite: automatic test suite generation for object-oriented software. In *ESEC/FSE*, pages 416–419, 2011.

[15] Alessio Gambi, Alessandra Gorla, and Andreas Zeller. O!snap: Cost-efficient testing in the cloud. In *International Conference on Software Testing (ICST)*, pages 454–459, 2017.

[16] Xavier Gandibleux. Peek–shape–grab: A methodology in three stages for approximating the non-dominated points of multiobjective discrete/combinatorial optimization problems with a multiobjective metaheuristic. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 221–235. Springer, 2017.

[17] Gregory Gay. The fitness function for the job: Search-based generation of test suites that detect real faults. In *ICST*, pages 345–355, 2017.

[18] Gregory Gay. Generating effective test suites by combining coverage criteria. In *SSBSE*, 2017.

[19] Fred Glover, Manuel Laguna, and Rafael Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 29(3):653–684, 2000.

[20] David Hadka. Moea framework: a free and open source java framework for multiobjective optimization, 2012.

[21] M Jean Harrold, Rajiv Gupta, and Mary Lou Soffa. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2(3):270–285, 1993.

[22] Nashat Mansour, Rami Bahsoon, and Ghinwa Baradhi. Empirical comparison of regression test selection algorithms. *Journal of Systems and Software*, 57(1):79–90, 2001.

[23] Dipesh Pradhan, Shuai Wang, Shaukat Ali, Tao Yue, and Marius Liaaen. Cbga-es: A cluster-based genetic algorithm with elitist selection for supporting multi-objective test optimization. IEEE, 2017.

[24] Nery Riquelme, Christian Von Lücken, and Benjamin Baran. Performance metrics in multi-objective optimization. In *Computing Conference (CLEI), 2015 Latin American*, pages 1–11. IEEE, 2015.

[25] Gregg Rothermel, Mary Jean Harrold, Jeffery Ostrin, and Christie Hong. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 34–43. IEEE, 1998.

[26] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, 27(10):929–948, 2001.

[27] Takfarinas Saber, James Thorburn, Liam Murphy, and Anthony Ventresque. Vm reassignment in hybrid clouds for large decentralised companies: A multi-objective challenge. *Future Generation Computer Systems*, 2017.

[28] Takfarinas Saber, Anthony Ventresque, Xavier Gandibleux, and Liam Murphy. Genepi: A multi-objective machine reassignment algorithm for data centres. In *International Workshop on Hybrid Metaheuristics*, pages 115–129. Springer, 2014.

[29] Takfarinas Saber, Anthony Ventresque, Joao Marques-Silva, James Thorburn, and Liam Murphy. Milp for the multi-objective vm reassignment problem. In *ICTAI*, pages 41–48, 2015.

[30] David A Van Veldhuizen. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical report, DTIC Document, 1999.

[31] Shin Yoo and Mark Harman. Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 international symposium on Software testing and analysis*, pages 140–150. ACM, 2007.

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

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

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

[35] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *PPSN*, pages 292–301, 1998.

[36] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation*, 7(2):117–132, 2003.