Requirement Dependencies–based Formal Approach for Test Case Prioritization in Regression Testing

Andreea Vescan, Camelia Şerban, Camelia Chisăliţă-Creţu, Laura Dioşan
Department of Computer Science,
Faculty of Mathematics and Computer Science
Babes-Bolyai University
M.Kogalniceanu 1, 400084, Cluj-Napoca, Romania
Email: {avescan, camelia, cretu, lauras}@cs.ubbcluj.ro

***Abstract*—Regression testing is the testing activity performed after changes occurred on software. Its aim is to increase confidence that achieved software adjustments have no negative impact on the already functional parts of the software.**

Test case prioritization is one technique that could be applied in regression testing with the aim to find faults early, resulting in reduced cost and shorten time of testing activities. Thus, prioritizing in the context of regression testing means to re-order test cases such that high priority ones are run first.

The current paper addresses the test case prioritization as a consistent part of a larger approach on regression testing, which combines both test case prioritization and test case selection in order to overcome the limitations of each of them.

A comprehensive formalization of test case prioritization is provided, incorporating beside the well known ingredients (test case, test requirement, fault, cost) also elements relating to the functional requirements and dependencies between requirements.

An evolutionary algorithm is used to construct the re-ordering of test cases, considering as optimization objectives fault detection and cost. A synthetic case study was used to empirically prove our perspective for test case prioritization approach.

I. Introduction

Software systems are subject to frequent changes. These may indicate the need of bug fixing, functionality improvement or adapting to new working environments. Regardless the reason for changing, the software must be tested to ensure that the new adjustment do not have an unexpected negative impact on other fully working parts of the software. This means that regression testing (RT) needs to be performed.

We have achieved a literature review on RT and then we decided to approach RT following several tracks. We have started with test case prioritization problem, and we intend to address the test selection problem as a second step. Later, we aim to merge the previous steps into a single one in order to achieve RT.

The investigation conducted indicates that both RT techniques (test case prioritization and test case selection) prove their own strengths. We plan to combine more regression techniques such that we make use of their advantages. Our approach on RT considers functional requirements along with the existing dependencies among them. The literature enquiry [5], [4] on RT showed us that testing was mainly centred on structural coverage. There are few researches that focus on functional requirements and fewer that examine the dependencies among requirements.

There is a three folded motivation for advancing this approach on RT. First, we provide a formalization to the test case prioritization problem and plea that it may be applied to any level of testing, i.e., from unit testing level to functional testing level. Second, the test case prioritization addresses the test requirements investigated by most researches on RT [5], but also the user requirements as well (that to best of our knowledge was rarely considered). Third, when achieving test case prioritization, existing dependencies among requirements are considered.

The remainder of the paper is organized as follows: Section II provides a short overview on RT. It describes the existing techniques with focus on test case prioritization problem. Section III proceeds with the test case prioritization problem formalization. Section IV provides the details on evolutionary algorithms together with the single objective approach on the test case prioritization problem, the chromosome representation, the fitness computation, and the used genetic operators. Section V presents the synthetic experiment along with the data used for the case study, the applied methodology, and the obtained results. Section VI concludes the paper and indicates future working directions.

II. Regression testing background

The regression testing problem is a key problem in the software testing domain. We have investigated various sources for establishing a proper definition of RT, from books on testing to papers publishing in the aria of RT, to ISTQB. In what follows we present the synthesis of our findings.

- In [23] the RT is viewed as *“rechecking test cases passed by previous production versions”*. And that the RT *“is specialized to the problem of efficiently checking for unintended effects of software changes”*.
- In [24] the RT is defined as *”a testing activity that is performed to gain confidence that the recent modifications to the System Under Test (SUT), e.g. bug patches or new features, did not interfere with existing functionalities”*.
- [25] defines RT as *testing a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the*

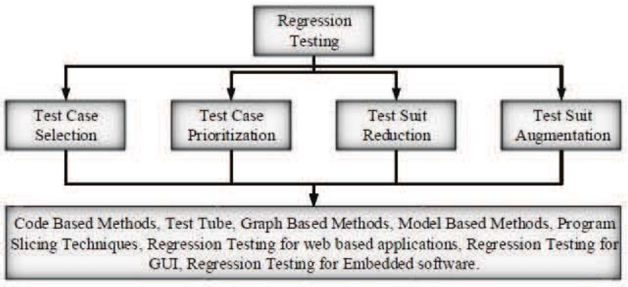
Regression Testing
Test Case Selection
Test Case Prioritization
Test Suit Reduction
Test Suit Augmentation
Code Based Methods, Test Tube, Graph Based Methods, Model Based Methods, Program Slicing Techniques, Regression Testing for web based applications, Regression Testing for GUI, Regression Testing for Embedded software.

Fig. 1. Subtypes of RT techniques [20]

software, as the result of the changes made. It is performed when the software or its environment is changed.

In conclusion, RT is an essential part of the software quality process, providing the confidence that the software has not been degraded after changes, and also that the code change has not adversely affected existing features, thus fully working modules in the software are left unharmed.

A. Regression Testing – Techniques

The existing literature [22] on test case management can be categorized into three different areas of investigation: Test Suite Minimization (or reduction, TSM), Test Case Selection (TCS) and Test Case Prioritization (TCP).

A review on various RT techniques is presented in paper [21]. A more detailed view about subtypes of RT is presented in [20]. Further research indicated Test Suite Augmentation (TSA) as another type of RT along with several subtypes and subcategories [20]. Therefore, the four main types of RT are TCS, TCP, TSM, and TSA.

Figure 1 visualizes the subtypes and also correlates the several methods that are shared among all the different RT types, such as Code Based Methods, Test Tube, Graph Based Methods, Model Based Methods, Program Slicing Techniques, RT for web based applications, RT for GUI, and RT for embedded software.

B. Test Case Prioritization

The goal of TCP is to find an optimal order in which to execute test cases. The ideal ordering of test cases would be the one that maximizes the rate of fault detection. However, since the fault information is not known to the tester in advance, prioritization techniques have to depend on surrogates. Rothermel [19] defined TCP problem as follows:

Test Case Prioritization Problem

Provided: a test suite T, the set of permutations of T denoted by PT; an objective function from PT to real numbers denoted by f.

Ensures: to find a permutation $\sigma \in PT$ such that $(\forall\sigma')(\sigma' \in PT)(\sigma' \neq \sigma)[f(\sigma) \geq f(\sigma')]$.

Various objective functions can be found in the specialised literature:

- statement coverage – a surrogate for fault detection capability in RT literature.
- δ-coverage – difference of the statement coverage between two consecutive versions.
- APFD [12], which rewards orderings with earlier fault detection abilities.
- Fault History Coverage – if we have one past fault, we can say that a test covers that fault if it is able to detect it. Fault history coverage has been used several times in test case minimization [11], [10], but has not be used in prioritization.
- Execution Cost – here we have to take into account the number of steps performed, instead of the wall-clock time, which depends on the underlying hardware and operating system.

1) Related work on TCP: We next describe some related approaches found in literature.

The existing literature approaches regarding TCP problem use metrics to evaluate their results and to compare them with other similar ones. These metrics serve as objectives function in the proposed algorithms as well. This section briefly surveys the existing TCP algorithms and related work in this area.

Greedy Algorithms have been widely employed for TCP [18], incrementally adding test cases to an initially empty sequence. The choice of which test case to add next is to achieve the maximum/minimum value for the desired metric (e.g., cost of a test case metric). Rothermel et al. [17] point out that this greedy prioritization algorithm may not always choose the optimal test case ordering for other criteria.

It is well known that Evolutionary Algorithms (EAs), of which Genetic Algorithms (GAs) are a subclass, are a form of meta–heuristic search that find solutions to combinatorial problems at a reasonable computational cost [16]. Regarding the problem of TCP in the context of RT, the application of Genetic Algorithms has been shown to be effective [15]. TCP, viewed as a search based optimization problem in RT, addresses a wide variety of objectives and several optimization techniques have been proposed to approach this problem.

Chu-Ti Lin et al. [27] describe a technique for the test case prioritization based on their history. Most of the existing test case prioritization approaches are code-based, in which software testing is considered as an independent process and most of them are memory-less and they model regression testing as a one-time activity rather than a continuous process. It addresses the issue of using the test results of the preceding software versions to schedule the test cases of the later software versions.

In [28] an algorithm is proposed to prioritize test cases based on the rate of fault detection and fault impact. It identifies the severe fault at earlier stage of the testing process and the effectiveness of prioritized test case. The effectiveness of prioritization of test cases based on faults was further investigated in [29], [30], [31].

In order to emphasize some differences and the benefits of our approach opposite to the related work, we have made an antithesis analysis in Table Ithat reveals that our proposed evolutionary algorithm considers cost and faults and also functional requirements and dependencies between functional requirements. Remark. We have only provided single objective approaches since we only investigate this aspect in the current paper. Future work will investigate also multi-objective approach.

2) TCP – comprehensive definition : Considering our investigation on RT and previous approaches, we have established an improved RT approach, in our opinion a comprehensive approach that considers: requirements and traceability regarding requirements and faults, dependencies between requirements, and model applied to any testing level. Thus when prioritizing test cases due to changes we both take into account the test cases associated with the requirement being changed and also the dependencies to other requirements.

In the next section we present the formalization of our comprehensive test case prioritization problem for the regression testing problem.

III. TCP Formalization

TCP Problem Input

- P – the program or software under test;
- R –the requirements set, where $R = \{r_1, r_2, \dots, r_n\}$;
- $rd : R \times R \ \rightarrow \{0,\ 1\}$ – the requirements dependency mapping with the meaning:

$$rd[r_i, r_j] = \begin{cases} 1, & \text{if requirement} r_i \text{depends on } r_j,\ i,j \in \{1,\dots,n\}; \\ 0, & \text{if requirement} r_i \text{ does not depend on } r_j,\ i,j \in \{1,\dots,n\}. \end{cases}$$

  Remark that having $rd(r_i, r_j) = 1$ does not mean that $rd(r_j, r_i) = 1,\ i,\ j\ \in \{1,\dots,n\}$;
- T – the test suite, where $T = \{t_1, t_2, \dots, t_m\}$ consists of tests that cover the requirements $r_i,\ i = \overline{1,n}$;
- $\sigma_T = (t_1, t_2, \dots, t_m)$ – a permutation for the test suite T;
- $tr : T \times R \ \rightarrow \{0,\ 1\}$ – the test-requirements traceability mapping (see Table II for a small example) having the meaning that:

$$tr(t_j, r_i) = \begin{cases} 1, & \text{if requirement } r_i \text{is covered by the test } t_j, \\ & \quad \text{where } i \in \{1,\dots,n\}, j \in \{1,\dots,m\}; \\ 0, & \text{if requirement } r_i \text{ is not covered by the test } t_j\ , \\ & \quad \text{where } i \in \{1,\dots,n\}, j \in \{1,\dots,m\}; \end{cases}$$

  ,
- $tc : T \rightarrow N$ – the test execution time expressed as a test cost for each $t_j \in T, i = \overline{1,m}$; this cost is considered to be unchanged on each test run;
- F – a faults set, where $F = \{f_1, f_2, \dots, f_q\}$ are the faults seeded in the P program;
- $tf : T \times F \ \rightarrow \{0,\ 1\}$ – the fault traceability mapping (see Table III for details), indicating that:

$$tf(t_j, f_k) = \begin{cases} 1, & \text{if the fault } f_k \text{ is detected by the test } t_j \\ 0, & \text{if the fault } f_k \text{ is not detected by the test } t_j \end{cases},$$

  where $j \in \{1,\dots,m\},\ k \in \{1,\dots,q\}$.
- $P\ '$ – a changed version of P that includes the seeded faults from F and the set of changed requirements from $\boldsymbol{R}$denoted by CR, $CR \subseteq R$.

TCP Problem Output

Find a permutation σ of T such that all r_is are satisfied and the test objective Obj is met. Obj is satisfied if and only if σ is a permutation of T that optimizes $Obj(T)$.

Remark: The primary objective of the addressed RT activity may be one or several criteria that needs to be optimized. When selecting the test cases to be part of the regression suite, first we select the test cases that cover the changed requirements, then we take into consideration also the dependencies between the requirements (considering also the test cases of the dependable requirements). The first part refers tot the retest set of test cases and the second selection part refers to the actual regression suite.

IV. TCP and Evolutionary Algorithms

We have until now specified the TCP problem and we further present the algorithm for solving it. We have decided to involve EAs in our approach due to their strengths: flexibility, robustness, and ability to deal with multiobjective objective problems.

A. Evolutionary Algorithms - Overview

Evolutionary algorithms [14] represent a class of stochastic optimization methods that simulates the process of natural evolution. They are based on mechanisms inspired by biological evolution such as reproduction, mutation, recombination, natural selection and survival of the fittest. Being used for solving optimization problems, the candidate solutions to these problems play the part of individuals in a population, and the cost function determines the environment within which the solutions "live". Evolution of the population takes place during an iterative process that includes the application of the above operators.

The main idea that the desired solution could be searched among a collection of potential solutions is very common in computer science. The scientists have given also a name to it: searching in a "search space". Cf. to M. Mitchell [14] the term "search space" refers to the collection of candidate solutions to a problem and to the notion of "distance" between the potential solutions. The dimension (size) of the search space is equivalent to the number or variables and/or parameters of a particular problem.

This search process actually appears when a problem must be solved. Usually, a global search and (single-objective) optimization problem P is represented in the following way: let S denote the solution set containing all the candidate solutions to the given problem P. The elements of this set are formal solutions, not relying on any specific underlying representation. Note the difference between solution set and solution space: the first refers to a collection of elements, while the second implies a structure over its elements. The purpose of a search problem P is to find specific solutions in the search space that optimize (maximize or minimize) an objective (cost) function f [14]:

- **Given:** a function $f : S \rightarrow \Re$ from a search space S to the set of real numbers
- **Required:** a solution $s_0 \in S$ such that $f(s_0) \leq f(s)$ for all $s \in S$ ("minimization") or such that $f(s_0) \geq f(s)$ for all $s \in S$ ("maximization"). Furthermore, some constraints could be present: inequality constraints

Reference	Testing Level	Functional requirements and dependencies	Performance Criteria - Metrics	Method
Our Approach	system level -coverage and requirements based information	✓	cost, coverage, faults	EA
[3]	system level - coverage based information	–	the percentage of code coverage of the test case in previous version	Sorting algorithm (Greedy)
[2]	system level - coverage based information	–	number of affected nodes covered by a test case	Sorting algorithm (Greedy)
[1]	system level - coverage based information	–	average percentage of coverage achieved, average percentage of coverage of changed code, and average percentage of past fault coverage.	EA

TABLE I
AN ANTITHESIS ANALYSIS OF TCP APPROACHES

	r_1	r_2	r_3	$\dots$	r_{n-1}	r_n
t_1	1	1	0	$\dots$	1	0
t_2	0	1	1	$\dots$	0	1
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
t_m	1	0	0	$\dots$	0	1

TABLE II
TEST-REQUIREMENTS (TR) TRACEABILITY MATRIX

	f_1	f_2	f_3	$\dots$	f_{q-1}	f_q
t_1	1	0	0	$\dots$	0	0
t_2	0	1	1	$\dots$	0	0
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
t_m	0	0	0	$\dots$	0	1

TABLE III
TEST-FAULTS (TF) TRACEABILITY MATRIX

$g_j(s_0) \leq 0$, $j = 1, 2, \dots, n_1$ and/or equality constraints $h_j(s_0) = 0$, $j = 1, 2, \dots, n_2$.

Such formulation is called an optimization problem or a mathematical programming problem [13]. Let note also that the global optima are well defined and are independent of any structure defined on the search space S [14].

B. TCP as Optimization Problem

The prioritization problems could be considered an optimisation problem; one or more criteria (number of tests, execution time, various coverages, different efficiency metrics) could be taken into account.

The algorithm applied in order to solve this optimization problem can be:

- a single-objective optimization algorithm – when two potential solutions are compared by using a single quality function (that incorporates more criteria — e.g., each objective is multiplied by a strictly positive scalar, obtaining a weighted linear combination of criteria). There are several problems of such approach: the optimal scale factors must be identified w_i and the weighted sum finds interesting, solutions but misses many solutions of interest (miss the concave region of the search space).
- a multi-objective optimization algorithm – when two possible solutions are compared by using simultaneously more quality functions. This will be further investigated in our future work.

C. Evolutionary Algorithm Ingredients for the TCP Problem

For solving TCP problem, a classical EA was used. In the next sections we will present the basic ingredients of it: the representation, the fitness and the search operators.

1) Chromosome Representation: A permutation-based representation is used for the chromosomes in order to encode a potential solution to TCP problem. The available tests must be ordered, obtaining a prioritization of them. We will encode this order by using chromosomes with a number of genes equal to the number of tests, each gene being an element from a permutation of size equal to the number of tests.

Preliminary steps for solving TCP by a EA:

1) identify the tests for re-testing: set $ReTests$
2) identify the tests for RT: set $RegrTests$
3) unify the previously identified sets of tests: $ActualTests = ReTest \cup RegrTests$

For instance, we consider a simple example of TCP problem with a test suite T composed of $m = 10$ tests ($T = \{t_1, t_2, \dots, t_{10}\}$), 8 requirements ($R = \{r_1, r_2, \dots, r_8\}$) and 3 faults ($F = \{f_1, f_2, f_3\}$). The requirement matrix is given in Table IV (the first 8 columns). The last column of Table IV lists the costs associated to all the considered tests. The fault matrix is given in Table V. The modified requirements are: r_2 and r_5. Dependent requirements are as follows: r_1 and r_8 depend on r_2; r_4, r_7, r_8 depend on r_5. In this context, $ReTests = \{t_3, t_5, t_8\}$ and $RegrTests = \{t_1, t_2, t_6, t_7$ (for r_2), t_4, t_6, t_7, t_8 (for r_5)$\}$. Therefore, $ActualTests = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$

A possible execution order of tests could be $t_7, t_2, t_6, t_8, t_1, t_5, t_4, t_3$ and it is encoded into a chromosome as the permutation described in Figure 2.

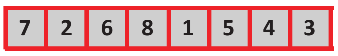

Fig. 2. Example of chromosome encoding for TCP

We can observe that the first 6 tests from this permutation ($t_7, t_2, t_6, t_8, t_1, t_5$) covered all changed (modified and dependent) requirements. The cost associated to this part is 27, while the total number of faults covered by them is 3.

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	cost
t_1	0	0	1	0	0	1	0	0	3
t_2	1	0	0	0	0	0	0	0	5
t_3	1	0	0	0	0	0	0	0	2
t_4	0	1	0	0	0	0	0	0	8
t_5	0	0	0	1	0	0	0	0	1
t_6	0	0	0	0	1	0	0	0	4
t_7	0	0	0	1	0	0	0	1	7
t_8	1	0	0	0	0	0	1	0	3
t_9	0	1	0	0	0	0	0	1	6
t_{10}	0	0	0	0	0	1	0	0	4

TABLE IV
REQUIREMENT MATRIX (EXAMPLE)

	f_1	f_2	f_3
t_1	1	0	1
t_2	0	0	1
t_3	1	0	0
t_4	1	0	0
t_5	0	1	0
t_6	0	1	0
t_7	1	0	1
t_8	0	1	1
t_9	1	1	0
t_{10}	0	1	0

TABLE V
FAULT MATRIX (EXAMPLE)

2) Fitness: The quality of a potential order of tests is evaluated taking into account: the correctness of the selection — all requirements are satisfied (modified requirements and dependent requirements) — and the efficiency of the selection — as few as possible tests, tests of minimal cost, tests of maximal fault.

Therefore, the fitness function could take into account different metrics. Some of these metrics are inspired by the concept of epistatic test case segment, introduced in [8] and defined as "a test cases segment which starts from the first test case in the execution sequence and ends with the test case that first reaches the maximum value of the test object".

- the total cost of selected tests (see Equation 1); because the (modified and dependent) requirements can be covered by few tests than those from $ActualTests$, the total cost will be computed over these few tests (denoted as required tests or rt) only; in this case we deal by a minimization problem.

$$\text{Total Cost} = \sum_{t \in ActualTests} tc[t] \qquad (1)$$

- the number of tests that cover all the faults (see Equation 2); in this case we deal by a maximization problem.

$$\text{Number Tests All Faults} = |\{TCAF\}| + \text{penalty}, \quad (2)$$

where:
  - $TCAF$ is the set of tests (from $ActualTests$) that cover all faults;
  - penalty — each requirement (modified or dependent requirements) that is not satisfied increase the penalty by a constant value (a large value establish according to the number of tests and requirements of the current problem)

As we already said, a possible solution for TCP described in Section IV-C1 could be encoded as a permutation $(t_7, t_2, t_6, t_8, t_1, t_5, t_4, t_3)$. By taking into account the cost objective we obtain: quality (fitness) = 27, total cost = 27, no of covered faults = 3, no of required tests = 6, that means an actual solution composed only by the first 6 tests $(t_7, t_2, t_6, t_8, t_1, t_5)$ because these six tests cover all the modified requirements (r_2 and r_5) and the dependent requirements (r_1, r_8, r_4, r_7, r_8); in addition, these tests cover all the faults.

3) Genetic Operations: In order to automatically discover, over several generations and by using biological-inspired operations (selection, crossover and mutation), efficient sets of tests, the following steps are repeated until a given number of iterations are reached: two parents are selected using a binary selection procedure; the parents are recombined in order to obtain an offspring *Off*; the offspring is than considered for the mutation; the new individual *Off** (obtained after mutation) replaces the worst individual W in the current population if *Off** is better than W.

a) Selection: Binary tournament selection is actually used in our approach. Two parents are randomly considered and the best of them, through the fitness function, is selected to be the first parent. The same procedure is repeated for selecting the second parent.

b) Crossover: After two chromosomes are selected from the current population, they are recombined. The order crossover is actually involved in our approach. It is one of the first crossover operators defined for permutations. In Figure 3 we present how two parents (two test permutations (t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8) and (t_7,t_2,t_6,t_8,t_1,t_5,t_4,t_3)) are recombined and a new order of tests ((t_2,t_8,t_1,t_4,t_5,t_6,t_7,t_3)) is obtained.

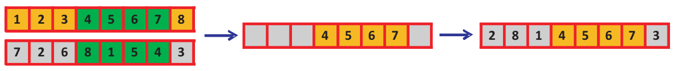

Fig. 3. Order crossover

c) Mutation: After crossover, the mutation operator is applied. In the case of a permutation encoding, we can have a swap mutation (or a bit-flip mutation) [7], [6] when two elements of the permutation are interchanged. In Figure 4 we present how a possible new permutation (t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8) is obtained from an old one (t_1,t_6,t_3,t_4,t_5,t_2,t_7,t_8) by mutation.

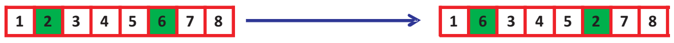

Fig. 4. Bit-flip mutation

Both perturbation operations (crossover and mutation) are applied in a probabilistic manner.

For solving TCP problem, a steady-state Genetic Algorithm (see Algorithm 1) is involved in the experimental validation.

Algorithm 1 Steady-state GA

```
Randomly_initialize_the_population $Pop$;
Evaluate_the_population $Pop$;
for g=1 to $NoOfGenerations \times PopSize$ do
  $p_1$ = Selection($Pop$);
  $p_2$ = Selection($Pop$);
  off = Crossover($p_1$, $p_2$, $p_c$);
  off* = Mutation(off, $p_m$);
  Fitness(off*);
  if off* is better then the worst individual $worst$ from $Pop$ then
    Replace($worst$, off*);
  end if
end for
solution = Best($Pop$)
```

In order to validate our approach we have considered a synthetic theoretic case study is considered. In the next section the used data is presented, along with the obtained results.

V. Experimental Analysis

This section presents and analyses the results of the empirical evaluation.

In order to evaluate our approach, we have analyzed, in detail a case study that uses theoretical data. Thereafter, it is assumed that test cases are defined and run. A single requirement is considered to be changed within the next version of the program along with 15 faults that were seeded in the program. Full experimental data of this case study is available at [9].

Input

- the program is denoted by P;
- the requirements set is $R = \{r_1, r_2, \ldots, r_{10}\}$;
- $rd : R \times R \rightarrow \{0, 1\}$ – a part of the requirements dependency mapping rd is presented shortly by Table VI; the full data of the the requirements dependency matrix is provided in [9];

	r_1	r_2	r_3	...	r_9	r_{10}
r_1	1	0	0	...	0	0
r_2	0	1	0	...	0	0
...	...	...	...	...	...	...
r_9	0	0	0	...	1	0
r_{10}	0	0	0	...	0	1

TABLE VI
Requirements dependency matrix for R

- the test suite is $T = \{t_1, t_2, \ldots, t_{50}\}$;
- $tr : T \times R \rightarrow \{0, 1\}$ – a part of the test traceability mapping is depicted by Table VII; the full traceability matrix is provided in [9];
- $tc : T \rightarrow N^*$ – the test cost associated with each $t_j \in T, i = \overline{1, 50}$ is presented as last column of the Table VII; the full data related to test costs is presented at [9]
- the faults seeded in the the program P is the set $F = \{f_1, f_2, \ldots, f_{15}\}$;

	r_1	r_2	r_3	...	r_9	r_{10}	cost
t_1	1	0	1	...	0	0	20
t_2	0	1	0	...	0	0	35
...	...	...	...	...	...	...	...
t_{49}	1	0	1	...	0	0	15
t_{50}	0	0	0	...	1	1	37

TABLE VII
Test traceability matrix for requirements set R. Costs associated to tests (last column)

- $tf : T \times F \rightarrow \{0, 1\}$ – a part of the fault traceability mapping is presented in Table VIII; the complete data related to faults seeded and tests that expose these faults are fully presented in [9].

	f_1	f_2	f_3	...	f_{14}	f_{15}
t_1	1	0	0	...	0	0
t_2	0	1	1	...	0	0
...	...	...	...	...	...	...
t_{49}	1	0	0	...	0	1
t_{50}	0	0	0	...	0	1

TABLE VIII
Seeded faults traceability matrix

- the changed version of P is P' and contains:
  - the faults from set F;
  - the set of changed requirements compared to P is $CR = \{r_2\}$, where $CR \subseteq R$.

A. Methodology of TCP Optimization

Two approaches were investigated for solving the TCP problem:

- in the first one the fitness function associated to a prioritization is represented by the total cost of the selected tests; in this case we deal by a minimization problem; we denote this approach by GA_{cost};
- in the second one the fitness function is represented by the number of tests that cover all the faults; in this case we deal by a maximization problem; we denote this approach by GA_{fault}.

A GA with chromosomes that encode permutations is involved in our numerical experiments. The fitness of a chromosome evaluates the quality of tests encoded into such a permutation. Since various objective functions can be considered for this evaluation, we investigate a single-objective optimization approach — when one metric discussed in Section IV-C2 is used for fitness computation,

The GA parameters are set as follows: population size: 100; number of generations: 100; code length (number of possible tests): size of $ActualTests$; crossover probability: 0.8; mutation probability: 0.1.

B. Numerical Results

By applying the preliminary steps on the provided data of theoretical case study, 21 test have been retained in $ActualTest$ set and 7 faults correspond to the 21 tests.

No	Optimisation method	Solution (selected tests)	Total cost	NoFaults	NoRequiredTests
1	GA_{cost}	t_{27},t_{32}	36	4	2
2	$Greedy_{cost}$	t_1, t_5, t_8, t_{10}, t_{14}, t_{16}, t_{18}, t_{27}, t_{28}, t_{30}	171	7	10
3	GA_{fault}	t_{15}, t_{45}, t_6, t_{16}	114	7	4
4	$Greedy_{fault}$	t_1, t_2, t_6, t_8, t_9, t_{13}, t_{14}, t_{16}, t_{20}, t_{22}, t_{24}, t_{27}, t_{30}, t_{35}, t_{37}, t_{38}	471	7	16

TABLE IX
PRIORITIZED TESTS OBTAINED BY USING DIFFERENT APPROACHES

The ordered tests, obtained by using GA_{cost} and GA_{fault} are presented in Table IX. Table IX contains also the results obtained by using two versions of a Greedy algorithm:

- The first one sorts all the tests by their cost (in an increasing order) and considers the tests, one by one, until all requirements (regression and re-testing requirements) are covered; ; we denote this approach by $Greedy_{cost}$;
- The second one sorts all the test by their number of faults (in a decreasing order) and considers the tests, one by one, until all faults are covered; ; we denote this approach by $Greedy_{fault}$.

We can observe that in the case of GA_{cost} method, even if the permutation is composed by 21 tests (with 7 corresponding faults), only the first two tests are necessary in order to cover all the requirements. The cost associated to them is 36 (= 6 + 30) and the number of discovered faults by these two tests is 4. Note that when a permutation of tests is analyzed, the tests are considered oen by one, computing (by aggregation) various metrics (e.g. the total cost, the number of faults, etc.). When all the requirements are covered, the analyzing process is stopped. Therefore, it is possible that a set of two tests to cover all the requirements, but not to cover all the faults.

The first two rows in Table IX indicate results obtained by using the *total cost criteria* in the GA and the Greedy algorithm. The selected GA solution is acceptable, considering the cost (36 milliseconds) and number of faults covered (4 faults). In addition, GA_{cost} in better than the $Greedy_{cost}$ by taking into account the cost and the number of tests, but it is weaker from the fault point of view (only 4 faults are cover by GA_{cost}).

Going further, the last two rows in Table IX consists of the second criteria used for our case study, which is the *total number of tests that cover all faults*. The solution obtained by GA_{fault} indicates that the all the faults (7) are covered by only 4 tests, compared to 16 tests indicated by the Greedy algorithm solution. The GA-based solution is better in terms of costs also (114 *vs.* 471) because is constructed by the first 4 tests from the permutation (related to 16 tests required by the Greedy-based solution).

Both approaches, using the cost and the total number of tests that cover all faults, obtain reasonable results by them selves. The data suggests to build a compound criteria consisting of the *cost* (to be minimized) and the *total number of tests that cover all faults* (to be maximized). Therefore, further experiments should be deployed to investigate the way they work together in a single-objective or multi-objective approach.

We plan to apply our algorithm to more complex case studies, the regression problem involving more changed requirements and more requirement dependencies; also benchmark case studies will be used.

VI. Conclusion

Regression testing is one of the means to provide confidence that the software's quality has not been degraded after changes.

In this paper we have used TCP as a first step in our broader approach (that is based on three steps: applying TCP and TCS individually and then applying one after the other for the same set of test cases). TCP re-order the test cases in such a way that high priority test cases can be executed first. The priority of this test case can be determined on the basis of coverage information from structural code, fault detection ability of test cases or the cost of execution of the test cases. We have applied single objective evolutionary approach, considering various criteria (from structural code, fault detection to execution cost) for a theoretical case study.

Future work will investigate: test case prioritization using multi-objective evolutionary approach and also the test case selection problem and the combination of the results with the TCP approach.

Aforestated work will be followed by investigating other degrees of dependencies existing among requirements, i.e., if r_1 depends on r_2 and r_2 depends on r_3 then r_1 depends on r_3. Consistent numerical results corresponding to various real life working scenarios need to be addressed in the future.

Acknowledgments

This research was supported in part by the Accenture Industrial Software Solutions S. A. grant to the University of Babes-Bolyai (Number 907 on April 2016).

References

[1] Michael G. Epitropakis and Shin Yoo and Mark Harman and Edmund K. Burke, *Empirical evaluation of pareto efficient multi-objective regression test case prioritisation*, Proceedings of the International Symposium on Software Testing and Analysis, pp. 234–245, 2015

[2] Panigrahi, Chhabi Rani and Mall, Rajib, *An approach to prioritize the regression test cases of object-oriented programs*, CSI Transactions on ICT, 1(2), pp. 159–173, 2013

[3] Khalilian, Alireza and Abdollahi Azgomi, Mohammad and Fazlalizadeh, Yaldak *An Improved Method for Test Case Prioritization by Incorporating Historical Test Case Data*, Sci. Comput. Program., 78 (1), 93–116, 2012

[4] Salehie, Mazeiar and Li, Sen and Tahvildari, Ladan and Dara, Rozita and Li, Shimin and Moore, Mark, *Prioritizing requirements-based regression test cases: A goal-driven practice*, Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on, pp. 329–332, 2011

[5] Srikanth, Hema and Hettiarachchi, Charitha and Do, Hyunsook, *Requirements based test prioritization using risk factors: An industrial study*, Information and Software Technology, 69, pp. 71–83, 2016

[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989

[7] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975

[8] Yuan, Fang and Bian, Yi and Li, Zheng and Zhao, Ruilian, *Epistatic genetic algorithm for test case prioritization*, International Symposium on Search Based Software Engineering, pp. 109–124, 2015

[9] Andreea Vescan and Laura Diosan and Camelia Cretu and Camelia Serban, *SBSE techniques for regression testing in IoT*, Accenture Research grant, http://www.cs.ubbcluj.ro/∼avescan/?q=node/237, 2016

[10] Shin Yoo and Robert Nilsson and Mark Harman, *Faster Fault Finding at Google using Multi Objective Regression Test Optimisation*, Proceedings of the 8th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '11), pp. 1–12, 2011

[11] Shin Yoo and Mark Harman, *Pareto Efficient Multi-Objective Test Case Selection*,Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA '07), pp. 140–150, 2007

[12] Rothermel, Gregg and Untch, Roland H and Chu, Chengyun and Harrold, Mary Jean, *Test case prioritization: an empirical study*, International Conference on Software Maintenance, pp. 179-188, 1999

[13] S. Boyd and L. Vandenberghe, *Convex optimization*, Cambridge University Press, 2004

[14] M. Mitchell, *An Introduction to Genetic Algorithms*,The MIT Press, 1996

[15] Husbands, Philip, *Genetic algorithms for scheduling*, AISB Quarterly, 89, pp. 38–45, 1994

[16] Banzhaf, Wolfgang, *Combinatorial optimization using genetic algorithms.*, Proceedings of the Spring Meeting of the Physics Society of Japan, 430–431, 1990

[17] Rothermel, Gregg and Untch, Roland J. and Chu, Chengyun, *Prioritizing Test Cases For Regression Testing*,IEEE Trans. Software Eng, 4 (3), pp. 225–237, 2007

[18] Zheng Li and Mark Harman and Robert M. Hierons, *Search Algorithms for Regression Test Case Prioritization*,IEEE Trans. Software Eng, 27 (10), pp. 929–948, 2001

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

[20] Rava, Mohammad and Wan-Kadir, Wan M.N., *A Review on Prioritization Techniques in Regression Testing*, International Journal of Software Engineering and Its Applications, 1 (1), 221–232, 2016

[21] Choudhary, Anjali and Dalal, Tarun, *A Review on Regression Testing Techniques*, IJ of Emerging Trends and Technology in Computer Science, 4 (3), 56–59, 2015

[22] Yoo, Shin and Harman, Mark, *Using Hybrid Algorithm for Pareto Efficient Multi-objective Test Suite Minimisation*, J. Syst. Softw., 83 (4), pp.689–701, 2010.

[23] Young, Michal and Pezze, Mauro, *Software Testing and Analysis: Process, Principles and Techniques*, John Wiley and Sons, 2005

[24] Harman, Mark and McMinn, Phil and De Souza, Jerffeson Teixeira and Yoo, Shin, *Search based software engineering: Techniques, taxonomy, tutorial*, Springer, pp. 1–59, 2012

[25] Graham, Dorothy and Van Veenendaal, Erik and Evans, Isabel, *Foundations of software testing: ISTQB certification*, Cengage Learning EMEA, 2008.

[26] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[27] Chu-Ti Lin, Cheng-Ding Chen, Chang-Shi Tsai, Gregory M. Kapfhammer *History-based Test Case Prioritization with Software Version Awareness*, IEEE 18th International Conference on Engineering of Complex Computer Systems, 17-19 July, 2013.

[28] Kavitha, A and Sureshkumar, N, *Test Case Prioritization for Regression Testing based on Severity of Fault*, International Journal on Computer Science and Engineering, 2 (5), pp.1462–1466, 2010.

[29] Nayak, Soumen, Chiranjeev Kumar, Sachin Tripathi *Effectiveness of prioritization of test cases based on Faults*, IEEE 3rd International Conference on Recent Advances in Information Technology, 3-5 March, 2016.

[30] Muthusamy, Thillaikarasi, Seetharaman, K, *A Test Case Prioritization Method with Weight Factors in Regression Testing Based on Measurement Metrics*, International Journal on Computer Science and Information Technology, vol. 4, pp.41–51, 2014.

[31] Muthusamy, Thillaikarasi, Seetharaman, K, *Efficiency of Test Case Prioritization Technique Based on Practical Priority Factors*, International Journal on Software Cmputing, vol. 10, pp.183–188, 2015.