Automatically Finding Hidden Industrial Criteria used in Test Selection

Cláudio Magalhães
Centro de Informática
Recife, Brazil
cjasm@cin.ufpe.br

Alexandre Mota
Centro de Informática
Recife, Brazil
acm@cin.ufpe.br

Eliot Maia
Motorola Mobility
Jaguariúna, Brazil
eliotm@motorola.com

Abstract

In this paper we propose a way to find weights of a ranking function semi-automatically. From the manual choices made by (experienced) human test architects, our idea is to propose an optimization model that tries to find the necessary weights automatically. We present some experiments by encoding our optimization model in the Z3 SMT solver and using real Motorola Mobility[1] data.

1 Introduction

Regression testing is vital for any product development. It gives a measure of how the adjustments made preserved accepted functionality. Re-run all available test cases is the easiest alternative for a regression testing campaign. In practice, however, this is usually unfeasible due to the effort needed in terms of time and human (for manual testing particularly) resources. Existing regression testing methods, such as test case selection, test case prioritization and test suite reduction [5] attempt to make regression testing more cost-effective. In this paper we focus on the (selection and) prioritization method(s).

To prioritize the best test cases, one has to set specific characteristics to create an ordering on the test cases. The most direct characteristics usually are code or requirements change coverage. However, in industry other criteria are used as well. In the work reported in [6], the authors proposed some criteria to prioritize potentially relevant test cases. In that work, real test data and criteria was used to illustrate the feasibility of that idea in an industrial context[2]. The proposed criteria were:

1. Number of executions of a test case: how much executions a test was done in previous cycles. The criterion is given by $crit_1 = executionsTC^{-1}$;
2. Test case failure status: This is the failure rate of the total number of executions of a particular test case. This criterion ($crit_2 = failuresTC/executionsTC$) values tests that were most effective in the past;
3. Number of unique failures found: A same test case may have failed countless times and revealing the same defect repeatedly. In this case, a single Change Request (CR) is opened and assigned to this test. This criterion ($crit_3 = CRs$) takes into account the defects found per test not counting repetitions
4. Regression level: each test case is classified according to a code called regression level, which can vary from 1 to 5. The last criterion

   $$crit_4 = regressionLevelPoints_n$$

   takes this into account, where $regressionLevelPoints_n$ are the points previously assigned to a test whose regression level is n.

When using a weighted ranking function, based on assumed relevant test case criteria, the challenge is to find the best criteria and weights to achieve the best ordering.

In [6] it is shown that by using certain weights on these four criteria, one can create a regression testing campaign automatically by taking the top test cases from the new ordering. Unfortunately, the work [6] left to the test architects the task to find the weights. And because these weights are not searched exhaustively, but proposed manually by guess, the ranking is not the best one in general.

Therefore, our main contributions in this paper are.

- Propose two additional test prioritization criteria (introduced in Section 4) to be considered in the ranking function;
- An optimization model to find the best weights for a ranking function;
- An embedding of the proposed optimization model in the Z3 SMT solver [2][3];
- A semi-automatic calibration strategy based on subsets of test architects choices and the optimization model.

[1] Motorola Mobility is our industrial partner

[2] The work [6] was result of a previous (in the 2004-2008 period) research partnership between Centro de Informática and Motorola Mobility.

[3] http://rise4fun.com/Z3/ (Opt library)

DOI Reference number: 10.18293/SEKE2016-198

This paper is organized as follows. Section 2 presents our proposed Mathematical Optimization Model. In Section 3 we briefly describe the Z3 tool by showing how to embed the model presented in Section 2 in its language. Section 4 describes some experiments we performed in our proposed model using real data from Motorola Mobility. Section 5 considers our conclusions and future work.

2 Mathematical model

In the work reported in [6], the authors propose a ranking function based on four criteria as presented in Section 1. The ranking function of a test is calculated using the weighted average of the standard points.

$r^{\vec{c}}_{\vec{w}} = \left(\sum_4^{i=1} crit_i \cdot weightCrit_i\right) / \left(\sum_4^{j=1} weightCrit_j\right)$

where $crit_i$ is the normalized scores on the criteria i and an element of the vector $\vec{c}$. And $weightCrit_j$ is the weight of criteria j and and element of the vector $\vec{w}$.

In this section we present a generalization of the work reported in [6] using a Mathematical Optimization Model. Without loss of generality, we do not consider the denominator ($\sum_4^{j=1} weightCrit_j$). This is important in Section 3 because the ranking function then becomes decidable and solvable by Z3.

Suppose that we have K criteria. Thus a $L \times K$ matrix (the criteria matrix) named $C_{L\times K}$ is a collection of normalized numbers corresponding to each criterion (L corresponds to the amount of records of some entity. In our case, we have L test cases).

As for each criterion we have also an associated weight then we have a K vector of weights. This vector $\vec{w}$ can be set manually, as in [6], or semi-automatically as presented here. Thus, for us, this vector ($\vec{w}_K = (w_1, w_2, \ldots, w_K)$) belongs to our set of unknowns.

The ranking function $\vec{r}^{\vec{c}}_{\vec{w}} = (r_1, \ldots, r_K)$ we use in this paper (a K vector $r_1, \ldots, r_K$) is given by.

$\vec{r}^{\vec{c}}_{\vec{w}} = C_{L\times K} \times \vec{w}_K$

In [6], the authors assumed that certain weights were more important than others by adjusting the weights manually. In our proposal we intend to find such weights by solving a mathematical problem. To do so we use some test case selections from our test architect experts as input to define an objective function.

To show how we transform this problem into an optimization problem, suppose that *Suite* is the set of all test cases and *Chosen* is the set of test cases that a test architect has selected (*Chosen* $\subseteq$ *Suite*). Our ranking function is used in such a way that if a test case tc_i belongs to *Chosen*, then its ranking function value r_i must be greater than all others r_j ($i \neq j$), corresponding to not chosen test cases.

As the selected test cases (*Chosen*) cannot be explained using all the criteria we consider, we employ an optimization model that tries to use the majority of the elements of *Chosen* to satisfy the maximum number of criteria (A criterion is representative whether its weight is greater than zero). Thus we try to maximize the set of test cases selected by the test architects considering the set *Chosen* as our best reference. Suppose that $(\dot{tc}_1, \ldots, \dot{tc}_k)$, where $tc_i \in$ *Chosen* ($1 \leq i \leq k$), is a boolean vector corresponding to the test cases chosen by a test architect. These boolean elements are interpreted such that if $\dot{tc}_i$ is true then tc_i is *elected* to belong to the final selection. Otherwise, it is discarded. Hence, our optimization problem can be stated as.

Max $(\dot{tc}_1, \ldots, \dot{tc}_k)$, **Subject to** $r_i > r_n$, *if* $\dot{tc}_i$
where $tc_n \in (Suite \setminus Chosen) \cup \{tc_r \mid 1 \leq r \leq k \wedge \neg\, \dot{tc}_r\}$

In the above optimization problem characterization, the maximization of $(\dot{tc}_1, \ldots, \dot{tc}_k)$ means to get a fully k-vector of truth values whenever possible, but accepting a partial vector as well. The restrictive part states that if a chosen test case tc_i is elected by this model, then its ranking (r_i) has to be greater than all other rankings of the not elected ($\{tc_r \mid 1 \leq r \leq k \wedge \neg\, \dot{tc}_r\}$) as well as not chosen (*Suite* $\setminus$ *Chosen*) test cases.

It is worth noting that, as we will see in Section 4, extreme cases can happen. If none or a single test case can be elected by the above model, then the test architect's choices cannot be safely described by the criteria used in this paper.

Another interesting point is that if a weight is unnecessary, our optimization model simply sets its value to zero.

3 The Z3 SMT Solver

Z3 is an SMT solver from Microsoft Research [2]. It is targeted at solving problems that arise in software verification and software analysis. Consequently, it integrates support for a variety of theories.

Recall from Section 2 that we have weights. In our Z3 encoding, weights are described as constants (unknowns to be instantiated by Z3). For instance, our four criteria become.

```
(declare-const w1 Int) ...
(declare-const w4 Int)
```

The ranking function, corresponding to each element r_i of Section 2, can be stated in Z3 very easily as follows.

```
(define-fun f((c1 Real) (c2 Real)
              (c3 Real) (c4 Real)) Real
  (+ (* w1 c1) (* w2 c2)
     (* w3 c3) (* w4 c4)) )
```

The previous function is used to compute each ranking r_i, where $c1, \ldots, c4$ correspond to the data from our industrial partner as will be clarified as in what follows.

Test cases are described in Z3 via two declarations. First we name each test case by declaring a named constant. For example, a test case 1 becomes the following constant declaration (its corresponding ranking value r_1).

```
(declare-const r1 Real)
```

As each test case is associated with four instances of the chosen selection criteria, we state an assertion for each of them as is illustrated in what follows for the test case 1.

```
(assert (= r1 (f 0.06 0.0 0.02 0.5)))
```

As Z3 does not have a min/max operator over sets, we capture this constraint by an indirect way of counting how many chosen test cases can be elected. First we create a $\{0, 1\}$ counter (from the boolean vector but this allows us to sum counters) for each chosen test case. For example, suppose test case 17 was chosen.

```
(declare-const FlTC17 Int)
(assert (or (= FlTC17 0) (= FlTC17 1)))
```

As test case 17 can be elected or not, we use a Z3 conditional construct (`ite`) as follows. In this Z3 encoding, the part starting with `and` occurs if `FlTC17` equals to 1. Otherwise, there is no constraint over test case 17 (`true`).

```
(assert (ite (= FlTC17 1)
             (and (> TC17 TC2) ...) true))
```

Finally we maximize the elected test cases (As the counters have at most the value 1, our full boolean vector of Section 2 means this sum is the maximum possible one).

```
(maximize (+ ... FlTC17 ...))
```

4 Experiments on Automatic Prioritization

During the development of this work, we identified two additional criteria of interest.

5. Creation date: From discussions with the test architect, we noted that newer tests can find a greater number of failures during testing. So, $crit_5 = creationTC^{-1}$ where *creationTC* is the number of days from the date of test case creation to the present time.
6. Date of the last execution: Like the previous criterion, the date of the last execution also increases the scores. Thus, $crit_6 = LastExecutionTC^{-1}$ where *LastExecutionTC* is the number of days from the date of the last execution of the test case to the present time.

In the following experiments we used a Core i5 1,7 GHz MacBook Air with 8 GB RAM, running El Captain 10.11.3 OS. Two test architects helped us to perform the experiments on three test suites creation. The first two experiments (see Tables 1 and 2) involving 80 test cases and the last one (see Table 3) with 427 test cases. In these tables, the test cases in bold face are common between the two test architects choices. Each table has four columns:

- **Chosen**: Number of test cases chosen manually;

Chosen	Elected	Criteria	Time	Match
12	1, 19	1, 2, 5, 6	4m 21.5s	16.7%
25	None	1, 2, 3, 5, 6	11.8s	0%

Table 1. 1st experiment (80 test cases)

Chosen	Elected	Criteria	Time	Match
20	1, 2, 3, 65	1, 2, 5, 6	1m 31.1s	20%
22	1, 2, 3, 65	1, 2, 5, 6	2m 22.8s	18.2%

Table 2. 2nd experiment (80 test cases)

- **Elected**: Test cases chosen by our optimization model;
- **Criteria**: Criteria found by the optimization model;
- **Time**: This is the effort needed to solve the Z3 model;
- **Match**: Elected divided by chosen test cases.

In Table 1 the intersection of **Chosen** test cases between the architects is of 27.6% (They both chose 29 test cases in total where 8 have been chosen by both: 8/29 = 27.6%). This low intersection in general occurs because the selection is manual and there is a short time to perform this (Just from their opinion, we get the rate as reading and analyzing 400 test cases to elect 100 test cases in 1 hour. This means electing a test case in about 2.25 seconds). This can be very error-prone. And this was captured by Z3 where one experiment elected 2 test cases and the other none at all.

Table 2 exhibited a better outcome between architects. Common choices were around 45% of all test cases involved. Four test cases were elected by Z3 and they were the same for both architects choices.

In Table 3 we have the most expensive experiment for using Z3. A total of 427 test cases with 128 selected test cases for one test architect and 195 for the other (And an intersection of 54 choices or 20%). In view of its size, the columns **Chosen** and **Elected** were only filled with the total amount of test cases.

The best percentage match obtained by [6] was 7.27% using four criteria. We get a 20% with six criteria. From [6], even with the low percentage match, the ranked tests reveals similar bug reports when compared to manual selections. Thus, this seems to indicate that: (i) the ranking is indeed relevant; (ii) further criteria can increase the percentage match; and (iii) Z3 can outperform human weight search (20% versus 7.27%). Besides, we cannot fully match architects choices because their selection is not systematic and use information beyond the criteria used here.

Chosen	Elected	Criteria	Time	Match
195	12	1, 2, 5, 6	2322m 4s	6.1%
128	1	1, 2, 5, 6	1631m 4s	0.8%

Table 3. 3rd experiment (427 test cases)

Threats to validity Our technique depends on three main resources (test selection criteria, test criteria data and test architect choices). The most sensible input is the test architect choices. We saw in the previous section that when there is a representative common choice between test architects, the criteria is found more easily. However, as reported in Table 1, for diverging choices, the six criteria used in this paper sometimes are not enough to describe them.

Internal threats can be test selection criteria and test criteria data. As test criteria data change over time, it is possible that test selection criteria cannot be always the same. But as our Mathematical model is general enough we can simply propose new test selection criteria that the Z3 solver reports us with the right criteria that describe the data.

5 Conclusion

This work has shown that it is feasible, although costly, to automatically balance a ranking function by formulating the work reported in [6] as an optimization problem. As such an automatic balancing is supposed to be used in a well-spaced period of time, the effort taking by Z3 can be affordable. Besides, after knowing the more adequate criteria, their use in practice is almost inexpensive because it is just an arithmetic calculation.

In our experiments, we have used the criteria proposed in the work reported in [6] and added two other criteria. This revealed an increase in the amount of test cases the ranking function can match with the test cases chosen by the test architects. We can also observe that even with the additional two criteria we considered here, the ranking does not fit the human selection with a high percentage match. This suggests investigating additional criteria that can be used in practice. For instance, information retrieval [7] also uses ranking functions based on keyword indexing and frequency. We intend to consider this new criterion as one of our future work.

The work reported in [1] addresses the problem of determining the next set of releases in the course of software evolution in an industrial setting using an optimization problem formulation. It employs simulated annealing and a greedy algorithm to solve the optimization problem and compare both approaches. In [8], the authors use a similar approach but solving the optimization problem using binary constrained particle swarm optimization (BC-PSO). The work reported in [3] follows a non-orthodox approach to test suite reduction. That work proposes FLOWER, which leverages the Ford-Fulkerson method to compute maximum flows and Constraint Programming techniques to search among optimal flows. In our work we also have an optimization problem formulation, but we employ an SMT solver instead of search-based specific algorithms and our goal is to find the weights and not the ranking per se. Maybe the work that is closest to ours in the sense of using off-the-shelf solvers is [4]. But this work focuses on test suite minimization, removing redundant test cases based on fault coverage, which is a bit different of our goal. As in general all works implement their own search algorithms whereas we have reused an off-the-shelf SMT solver, we intend to investigate the advantages/disadvantages of doing this in future work.

As our experiments exhibited a considerable variation in execution time from one test architect choice to the other, we intend to investigate this as future work.

Another future work we intend to pursue based on the feedback of our test architect is to automatically identify when a test case is a prefix of another test case. One hidden criteria used in practice is to employ just the greatest test case. This task needs a natural language processing as well as a notion of phrase dependency.

Acknowledgments We thank Alice Arashiro, Viviana Toledo, and Lucas Heredia from Motorola Mobility, and Juliano Iyoda for suggestions on the paper. This research is supported by Motorola Mobility.

References

[1] P. Baker, M. Harman, K. Steinhofel, and A. Skaliotis. Search based approaches to component selection and prioritization for the next release problem. In *ICSM*, pages 176–185, Sept 2006.

[2] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *14th TACAS*, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] A. Gotlieb and D. Marijan. Flower: Optimal test suite reduction as a network maximum flow. In *ISSTA*, pages 171–180. ACM, 2014.

[4] H.-Y. Hsu and A. Orso. Mints: A general framework and tool for supporting test-suite minimization. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 419–429. IEEE, 2009.

[5] J.-H. Kwon, I.-Y. Ko, G. Rothermel, and M. Staats. Test case prioritization based on information retrieval concepts. 1:19–26, Dec 2014.

[6] J. Mafra, B. Miranda, J. Iyoda, and A. Sampaio. Test case selector: Uma ferramenta para seleção de testes (in portuguese). In *SAST*, pages 1–10, 2009.

[7] R. K. Saha, L. Zhang, S. Khurshid, and D. E. Perry. An information retrieval approach for regression test prioritization based on program changes. In *37th ICSE (vol. 1)*, pages 268–279. IEEE Press, 2015.

[8] L. Souza, R. Prudêncio, F. Barros, and E. Aranha. Search based constrained test case selection using execution effort. *Expert Systems with Applications*, 40(12):4887 – 4896, 2013.