Regression Test Case Prioritisation for Guava

Yi Bian[2], Serkan Kirbas[3,4], Mark Harman[1], Yue Jia[1], and Zheng Li[2]

[1] CREST, Department of Computer Science,
University College London, Malet Place, London, WC1E 6BT, UK
[2] Department of Computer Science,
Beijing University of Chemical Technology, Beijing 100029, P.R.China
[3] Department of Computer Science,
Brunel University London, Kingston Lane, Uxbridge, London, UB8 3PH, UK
[4] Computer Engineering Department,
Bogazici University, Bebek, Istanbul, 34342, Turkey

Abstract. We present a three objective formulation of regression test prioritisation. Our formulation involves the well-known, and widely-used objectives of Average Percentage of Statement Coverage (APSC) and Effective Execution Time (EET). However, we additionally include the Average Percentage of Change Coverage (APCC), which has not previously been used in search-based regression test optimisation. We apply our approach to prioritise the base and the collection package of the Guava project, which contains over 26,815 test cases. Our results demonstrate the value of search-based test case prioritisation: the sequences we find require only 0.2% of the 26,815 test cases and only 0.45% of their effective execution time. However, we find solutions that achieve more than 99.9% of both regression testing objectives; covering both changed code and existing code. We also investigate the tension between these two objectives for Guava.

Keywords: Regression Testing, Test Case Prioritisation, NSGA-II

1 Introduction

Test Case Prioritisation (TCP) reorders a sequence of test cases, based on testing objectives [2, 7, 8]. Most previous work on test case prioritisation has been single objective, though it has been argued that much more work is needed on multiple active approaches [11]. Yoo et al. [9] extended single objective test case selection to the multiple objective paradigm, but there is far less work on multi objective prioritisation [12–15].

This paper introduces a multi objective formulation of the test prioritisation problem, in which we include an additional objective which, perhaps surprisingly, has not previously been studied in any multi objective regression test optimisation work. That is, in addition to statement coverage, and execution time, which have been widely studied, we also use coverage of changed code as an objective, since this is clearly critical in regression test optimisation. Our formulation therefore balances the tension between coverage of (specifically) changed code against all code, while seeking to minimise the overall execution time.

We apply our approach to Guava [4], an open source Java project containing several google versions of core Java utility libraries. These libraries provide day-to-day functionalities including collections, data caching, concurrency support, string manipulation[5]. In this work, we selected tests suite for one of the major package of Guava, com.google.common.collect, which contains 26,815 test cases. We chose to test this package because it provides the core data collection that are used in every Java program, such as, list, set, maps, tables etc. [5].

2 Our Approach to Test Case Prioritisation

This section presents our algorithm representation and fitness functions to test case prioritisation problem. Given a test suite T with n elements, and a set of n objectives, $f_1, ..., .f_n$. We seek to find a new permutation of T, $T' = < t'_1, ..., t'_n >$ where $\exists i \in \{1, ..., n\} \wedge f_i(T') > f_i(T'')$ [11].

We use NSGA-II [10], with a permutation encoding in which the N test cases are given a sequence number from 0 to $N-1$. We used rank selection, order crossover, and order-changing mutation operators in the NSGA-II. Figure 1 shows an example of the order crossover operator. It first randomly selects two points and then swaps elements between these points and order the remainder from the beginning of the position. The crossover rate is 0.1 and mutation rate is 0.001.

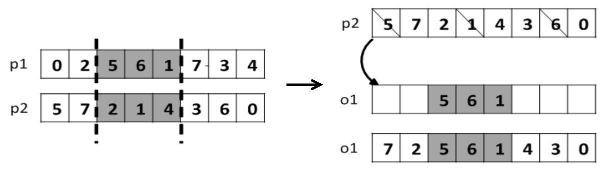

Fig. 1: An example of the order crossover

We have considered three objectives in test case prioritisation. The first one is Average Percentage of Change Coverage (APCC). APCC allows us to priories tests focused on the code that has been added or modified recently. To extract the code change information, we first extract the current version of Guava from the git repository and then use the "Blame" function to determine which lines have been changed. Given a line number as input, the "Blame" function returns the previous revision numbers in which if the line was added or a modified. We mark the line as "changed" if the date of the revision returned is after the previous release of the Guava library. APCC is defined as follow:

$$APCC = (1 - \frac{TC_1 + TC_2 + \cdots + TC_M}{NM} + \frac{1}{2N}) * 100\% \tag{1}$$

[5] https://code.google.com/p/guava-libraries/

In this formula, N is the number of test cases, M is total number of changed statements, and TC_i denotes the identifier of the test case that first covers the changed statement i in the execution sequence. A higher APCC value means the given test sequence cover more source code changed faster.

In addition to prioritising for the objective of covering changed code, we also include two more standard (and previously studied) objectives: Average Percentage of Statements Covered (APSC) and Effective Execution Time (EET) [14]. APSC has been widely used in TCP work [8], which measures the rate of average number of lines of code covered by given execution of test sequence. Effective Execution Time (EET) calculates execution time required for the test sequence to achieve 100% of the test objectives (so, 100% of either APCC, APSC or both depending on the problem formulation). Let ET_i be the execution time of test case i, and N_{length} is the number of test cases that achieve the test objectives, EET is defined as follow:

$$EET = \sum_{i=0}^{N_{length}} ET_i \tag{2}$$

3 Experiments and Results

In order to understand the impact of the newly introduced APCC metric, we have carried out three different multiple objective experiments, as set out in Table 1.

Table 1: Three groups of experiments are conducted

Group	Optimisation Objectives
G1	APSC and EET
G2	APCC and EET
G3	APSC, APCC and EET

All experiments were run on a CentOS 5.11 with 8 Intel E5426 CPU cores and 16G memory. In each experiment, there are 26,815 test cases in the optimisation sequences for 62 classes in the collection package. We manually extracted the code change information as explained in the approach section. Based on this Git analysis, we found that 140 lines have changed. As the Guava is quite mature and stable now, so there wasn't many major changes to the collection package.

For our search, we use the popular and widely-used NSGA-II algorithm. We experimented with three different population sizes: 100, 200, and 500, Each with a generation upper limit (termination condition) of 1000 generations. We also terminate the search if the sum value of average change in different optimisation objectives is smaller than 0.0001 in 10 consecutive generations. We compared the original sequence and average results from a set of random sequences. We generated 100,000 random sequences, from which we construct a pareto front using elite sorting based APSC, APCC and EET, repeating this process 100 times, so that 10,000,000 random test sequences are constructed in total.

The results for each of the three experiments are presented in Table 2. In Table 2, the APSC APCC and EET columns show the average best fitness value respectively over 100 runs. The Length and Time(s) show the number of selected

Table 2: Average value of objectives in different group of experiments

Strategy		APSC	APCC	EET(s)	Length	Time(s)	Generation	Front set
original		24.8085%	9.1243%	210.47	26808.00	-	-	-
random		86.1261%	89.7498%	188.26	24031.36	442.02	-	19.30
100	G1	99.9584%	-	1.13	80.65	71.82	800.09	2.39
	G2	-	99.9834%	0.39	22.94	4.06	82.40	1.24
	G3	99.9563%	99.9900%	1.40	111.32	71.48	721.30	4.84
200	G1	99.9674%	-	1.03	59.02	99.65	477.70	2.78
	G2	-	99.9925%	0.28	7.13	5.47	34.57	1.66
	G3	99.9657%	99.9921%	1.08	65.76	107.39	432.58	5.61
500	G1	99.9709%	-	0.96	49.93	180.33	214.70	2.73
	G2	-	99.9927%	0.28	7.00	12.91	31.50	1.77
	G3	99.9692%	99.9923%	0.94	48.15	210.27	223.69	7.43

tests to achieve maximum coverage (all statement coverage for G1 and G3 and statement coverage for G2) and execution time on average. The generation and front set columns show the average number of generations and the size of pareto front. The original row reports the results of running the default test suite where the random row shows the results using random generated sequence as a baseline. In the experiments involving random generation, the time spent (recorded in the sixth column) is that time on the fitness function calculation and elite sorting of randomly generated individuals.

The results of our experiments suggest that prioritisation can be very effective for Guava, finding test sequences that achieve coverage of both test adequacy criteria with only a tiny fraction of the budget required by the entire test suite. That is, both 100% APCC and APSC can be achieved with 0.2% of test cases and 0.45% of total execution time for the entire suite. For the Guava developer this highlights the value of prioritising test cases in order to maximise early coverage of both changed and unchanged statements.

Table 3: The $p-value$ of Mann-Whitney-Wilcoxon test and Vargha and Delaney $\widehat{A}_{12}$ between two different set of TCP experiment groups

Group	100 VS. 200		100 VS. 500		200 VS. 500	
	MWW	VDA	MWW	VDA	MWW	VDA
G1	3.24E-08	0.7263	2.71E-20	0.8778	6.10E-10	0.7533
G2	2.32E-30	0.9687	9.67E-33	0.9877	1.66E-07	0.7142
G3	1.67E-13	0.8018	1.39E-23	0.9097	7.09E-07	0.7030

The 100, 200, 500 rows show the overall results of three experiments running with population size of 100,200 and 500 respectively. We used the $Mann-Whitney-Wilcoxon$ test with *Bonferroni correction* to check the *hypervolume* distribution of pareto front sets in different population size and then we compare the significants between these results by using Vargha and Delaney $\widehat{A}_{12}$ effect. The results are in Table 3. In Table 3, MWW and VDA show the p-value and the

$\widehat{A}_{12}$ value. The results show that the hypervolume of the pareto front generated from three different population settings has significant differences with a high effect size.

We also calculated the number of similar test cases used by the different sequences as shown in Table 4. In order to avoid double counting, we combined the test cases with the same statement coverage which reduced the number of test cases considered from 26,815 to 14,516 different test cases. In Table 4, the number on each cell denotes the number of test cases in the intersection of the two different test sequences (row and column values) considered and the last column is the total number of those test cases in the sequences. Again the the results are total number over all 100 runs.

The results indicate that, as we might expect, covering all objectives (G3) is similar to covering statements with minimal execution time (G1). This is because there are relatively few changed statements, so covering all statements and all changed statements is very much similar to covering all statements; the former subsumes the latter. However, targeting the coverage of *only* the changed statements at minimal cost, yields very different test suites that either attempting statement coverage alone or both statement coverage and changed statement coverage.

Table 4: The total number of test cases in one group and intersection number between two different groups

		100			200			500			Total
		G1	G2	G3	G1	G2	G3	G1	G2	G3	
100	G1	-	430	2060	624	71	2221	489	64	571	2591
	G2		-	1008	295	59	1101	237	52	265	1260
	G3			-	1223	140	8889	823	134	945	10758
200	G1				-	65	1337	420	59	483	1486
	G2					-	142	63	30	66	167
	G3						-	859	142	995	11933
500	G1							-	49	448	935
	G2								-	49	165
	G3									-	1066

In the future we will consider to include the fault information based on Guava project to verify which combine of objectives is more effective for testing the errors in Guava project. Also we need to include more objective to satisfy the requirements of industrial needed, for example, adding the mutation testing to measure the errors detection ability between test sequences or considering to give a higher the coverage value for the test sequence that can quickly coverage the most important classes in project. At last we are also considering to use GPGPU technology to accelerate the TCP process which will obviously improve the efficiency of regression testing.

4 Conclusions and Actionable Findings

In our experiments, we extended multi-objective test case prioritisation process to consider coverage of changed statements and applied it to test prioritisation for Guava's collection package. Our experiments revealed that prioritisation can dramatically reduce the size of test sequences required to achieve early coverage of changed statements (and all statements) for Guava developer. Also targeting only coverage of the changed statements yields very different test sequences to targeting coverage of all statements.

References

1. Yoo, S., Harman, M. (2012). Regression testing minimization, selection and prioritisation: a survey. Software Testing, Verification and Reliability, 22(2), 67-120.
2. Rothermel, G., Untch, R. H., Chu, C., Harrold, M. J. (2001). Prioritizing test cases for regression testing. Software Engineering, IEEE Transactions on, 27(10), 929-948.
3. Huang, P., Ma, X., Shen, D., Zhou, Y. (2014, May). Performance regression testing target prioritisation via performance risk analysis. In Proceedings of the 36th International Conference on Software Engineering (pp. 60-71). ACM.
4. Guava Project Web Site, https://github.com/google/guava
5. http://blog.takipi.com/google-guava-5-things-you-never-knew-it-can-do/
6. Jiang B, Chan W K. On the integration of test adequacy, test case prioritisation, and statistical fault localization[C]//Quality Software (QSIC), 2010 10th International Conference on. IEEE, 2010: 377-384.
7. Elbaum S, Malishevsky A G, Rothermel G. Test case prioritisation: A family of empirical studies[J]//Software Engineering, IEEE Transactions on, 2002, 28(2): 159-182.
8. Li Z, Harman M, Hierons R M. Search algorithms for regression test case prioritisation[J]. Software Engineering, IEEE Transactions on, 2007, 33(4): 225-237.
9. Yoo S, Harman M. Pareto efficient multi-objective test case selection[C]//Proceedings of the 2007 international symposium on Software testing and analysis. ACM, 2007: 140-150.
10. Deb K, Pratap A, Agarwal S, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II[J]. Evolutionary Computation, IEEE Transactions on, 2002, 6(2): 182-197.
11. Harman M. Making the Case for MORTO: Multi Objective Regression Test Optimization[C]//ICST Workshops. 2011: 111-114.
12. Sun W, Gao Z, Yang W, et al. Multi-objective test case prioritization for GUI applications[C]//Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, 2013: 1074-1079.
13. Snchez A B, Segura S, Ruiz-Corts A. A comparison of test case prioritization criteria for software product lines[C]//Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on. IEEE, 2014: 41-50.
14. Li Z, Bian Y, Zhao R, et al. A fine-grained parallel multi-objective test case prioritisation on GPU[M]//Search Based Software Engineering. Springer Berlin Heidelberg, 2013: 111-125.
15. Micheal Epitropakis, Shin Yoo, Mark Harman, Edmund Burke, Empirical Evaluation of Pareto Efficient Multi Objective Regression Test Case Prioritisation[C]//ISSTA 2015, To appear.