Application of the 0-1 Programming Model for Cost-Effective Regression Test

Hirohisa Aman
Graduate School of Science and Engineering,
Ehime University
Matsuyama, Japan
Email: aman@cs.ehime-u.ac.jp

Manami Sasaki, Kei Kureishi and Hideto Ogasawara
Corporate Software Engineering Center,
Toshiba Corporation
Kawasaki, Japan
Email: {manami.sasaki, kei.kureishi, hideto.ogasawara}@toshiba.co.jp

***Abstract*—This paper reports an application of the 0-1 programming model to the regression testing plan for an industrial software. The key idea is to formulate a testing plan as a 0-1 programming problem (Knapsack problem). The empirical study shows that the 0-1 programming method can produce a cost-effective testing plan in which all potential regressions are found at only 22% of the cost of running all test cases.**

***Keywords*-testing; regression; 0-1 programming model; cost-effectiveness**

I. Introduction

System testing—testing behaviors of software—is a crucial activity to develop high-quality software. Any bugs overlooked during system testing may cause failures in the operational phase. Test engineers usually prepare many test cases to check various functions and conditions.

During system testing, many minor upgrades to fix bugs may appear. Then, test engineers should run test cases again to check whether they have regressions (bugs) or not, i.e., regression testing [1]. While it is ideal to run all test cases again for each minor upgrade, such a thorough regression test would be difficult due to realistic limitations such as lack of time and/or manpower. Thus, test engineers have to plan a cost-effective regression test under realistic limitations.

Test engineers often assign some values (worth to be tested) to their test cases in order to prioritize them. Then, those test cases can be selected in descending order of worth. Such a strategy is called the "greedy method" in the field of algorithm theory. While the greedy method is simple and easy-to-perform, it cannot consider testing costs properly. To take into account both the testing worth and the testing cost, Aman [2] proposed to formulate such a testing plan as a 0-1 programming problem (Knapsack problem [3]). The contribution of this paper is to report the application of Aman's method to industrial software.

II. History-Based Worth to Run Regression Test

A. Test History Data

In a system testing for a software, we suppose that we have N test cases T_i (for $i = 1, \ldots, N$) and M versions V_j (for $j = 1, \ldots, M$) to be tested. Then, let $r(i, j)$ to be the test result of running T_i for version V_j:

$$r(i,j) = \begin{cases} 0 & \text{(failure)}, \\ 1 & \text{(success)}, \\ \text{NA} & \text{(non-running)}. \end{cases}$$

These results can form a $N \times M$ matrix $\boldsymbol{R} = (\ r(i,j)\ )$, and the matrix is our test history data. It is ideal that all elements of the matrix are either 0 or 1; that is to say, all test cases were executed for all versions. In reality, however, some tests are omitted ($r(i,j) = \text{NA}$) if their tests already passed in the previous version, e.g., $r(i, j-1) = 1$.

B. Worth to Run Regression Test

The test history of T_i is given as the i-th row of $\boldsymbol{R}$:

$$r(i,1),\ r(i,2),\ \ldots,\ r(i,M)\ .$$

Using the above history data, we will evaluate the worth to run T_i again (perform the regression test).

Now we will focus on consecutive NAs in the test history, and we will consider that a test case having a long series of NAs as needing to be tested again. We define T_i's worth to be run again (to perform the regression test) as follows:

Suppose that T_i was run at version V_k, but it had never been executed after that version, i.e., $r(i,j) = \text{NA}$ (for $j = k+1, \ldots, M$), so there are "$M - k$" consecutive NAs until the latest version. Then, define T_i's worth to run the regression test, w_i, as

$$w_i = M - k\ .$$

□

The test case T_i with larger w_i would have a higher risk of overlooking regressions because it has not been run for a longer time.

Table I shows an example of $\boldsymbol{R}$ with $N = 3$ and $M = 5$. In the table, we obtain $w_1 = 2$, $w_2 = 3$ and $w_3 = 0$. T_1 and T_2 have not been run since V_4 and V_3, respectively. That is to say, there are some possibilities of potential regressions in the latest version. T_3 has little or no worth to run again in the latest version because it was just run in the same version.

Table I
SIMPLE EXAMPLE OF $\boldsymbol{R}$, WHERE $N = 3$, $M = 5$.

	V_1	V_2	V_3	V_4	V_5
T_1	NA	0	1	NA	NA
T_2	1	1	NA	NA	NA
T_3	0	1	NA	0	1

III. 0-1 PROGRAMMING MODEL

We have introduced the notion of worth to run regression tests in the previous section. Using such worth, we can select test cases in descending order; such a scheme is referred to as the "greedy method." While the greedy method is simple and easy-to-perform, it would not produce the optimal solution when the available time or effort is limited. To consider not only the testing worth but also the testing cost, Aman [2] proposed to formulate the testing plan as the 0-1 programming problem as follows:

$$\begin{aligned} &\text{maximize} && \sum_{i=1}^{N} w_i \cdot x_i \ , \\ &\text{subject to} && \sum_{i=1}^{N} c_i \cdot x_i \leq L, \quad x_i \in \{0,1\}, \\ &\text{and} && x_i \leq x_j \quad \text{if } T_i \text{ requires to run } T_j \ , \end{aligned}$$

where c_i is the cost needed to run T_i, and x_i is the 0-1 variable showing whether we run T_i or not; $x_i = 1$ if we run it, otherwise $x_i = 0$. L denotes our upper limit of total cost available for our regression testing.

□

The above inequality $x_i \leq x_j$ represents a causal connection between T_i and T_j. Whenever T_i is selected to be run, T_j is also selected since $x_i = 1$ and $x_i \leq x_j$ imply $x_j = 1$. Test engineers can make any causal connections depending on their situation. The authors have empirically used the following co-occurrence relation to make such a causal connection: whenever T_i is run, T_j is also run, we consider that T_i requires running T_j. For example, T_3 is always run when T_2 is run in Table I, so we have $x_2 \leq x_3$ as one of our constraints.

The key contribution of the 0-1 programming model is to take into account "testing costs" reasonably. Such a notion of cost can always be crucial matter in the development of industry software.

IV. EMPIRICAL STUDY

To evaluate the effectiveness of the above 0-1 model, we conducted an empirical study with a software system which has been developed and maintained by the authors' company.

Using $300(= N)$ test cases, we tested $13(= M)$ versions of the software during system testing. Now we consider the test cost to be the testing time (the unit is minute), and conducted the following comparison experiment:

Table II
NUMBER OF REGRESSIONS FOUND IN EMPIRICAL WORK.

	method	
L	greedy	0-1 model
1h (60min)	8	16
2h (120min)	14	22
3h (180min)	14	22

- Method to select test cases:
  1) greedy method[1],
  2) 0-1 programming method (0-1 model).
- Cost Limit (L):
  1) 1 hour (60 min),
  2) 2 hours (120 min),
  3) 3 hours (180 min).

□

Table II shows the number of regressions found by running the selected test cases. For all L, the 0-1 model could find more regressions than the greedy method. Therefore, the 0-1 model would be more useful in planning cost-effective regression tests.

In order to discuss the efficiency of the above methods, we also performed all 300 test cases in 554 minutes. As a result, 22 regressions were found. Thus, the 0-1 model could find all regressions in only two hours, which corresponds to about 22% of total costs needed to run all test cases[2]. While the greedy method is a simple and effective method, the 0-1 model seems to be much more effective.

V. CONCLUSION

This paper reported an application of 0-1 model to the regression testing plan for an industrial software. The 0-1 model could produce a more cost-effective testing plan than the conventional greedy method, and could find all regressions at only 22% of the costs to run all test cases again. Our future work includes further data collection and experiments for comparing the 0-1 model with other intelligent methods (not only greedy one).

REFERENCES

[1] J. Richardson and W. Gwaltney, Jr., *Ship It!: A Practical Guide to Successful Software Projects*, Pragmatic Bookshelf: O'Reilly Media, CA, 2005.

[2] H. Aman, *Planning of Priority Review using 0-1 Programming Model with Logical Constraints*, Computer Software, vol.29, no.3, pp.115–120, Iwanami Shoten, Tokyo, 2012 (in Japanese).

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of Np-Completeness*, W.H.Freeman and Company, NY, 1979.

[1] The co-occurrence relation is considered as well. That is to say, if T_i requires running T_j, then both T_i and T_j are selected whenever T_i is selected.

[2] $120/554 \simeq 0.22$