HOSTED BY

ELSEVIER

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: http://www.elsevier.com/locate/jestch

Full Length Article

Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing

Bestoun S. Ahmed [a,b,*]

[a] Istituto “Dalle Molle” di Studi sull'Intelligenza Artificiale (IDSIA), USI/SUPSI, Manno (Lugano), Switzerland
[b] Software Engineering Department, Salahaddin University-Hawler, Erbil, Iraq

A R T I C L E   I N F O

Article history:
Received 21 June 2015
Received in revised form
22 October 2015
Accepted 3 November 2015
Available online 14 December 2015

Keywords:
Combinatorial testing
Test case design
Fault seeding
Software mutation testing
Software structural testing
Cuckoo search algorithm

A B S T R A C T

This paper presents a technique to minimize the number of test cases in configuration-aware structural testing. Combinatorial optimization is used first to generate an optimized test suite by sampling the input configuration. Second, for further optimization, the generated test suite is filtered based on an adaptive mechanism by using a mutation testing technique. The initialized test suite is optimized using cuckoo search (CS) along with combinatorial approach, and mutation testing is used to seed different faults to the software-under-test, as well as to filter the test cases based on the detected faults. To measure the effectiveness of the technique, an empirical study is conducted on a software system. The technique proves its effectiveness through the conducted case study. The paper also shows the application of combinatorial optimization and CS to the software testing.

1. Introduction

Similar to any other engineering process, software development is subjected to cost. Nowadays, software testing (as a process of the software development life cycle) consumes most of the time and cost spent on software development. This cost may decrease rapidly as testing time decreases. Most of the time, a software may be released without being tested sufficiently because of marketing pressure as well as the intention to save time and cut costs. However, releasing low-quality software products to the market is no longer acceptable because it may cause loss of revenue or even loss of life. Thus, software testers should design high-quality test cases that catch most of the faults in the software without taking more than the scheduled time for testing. Thus, test case minimization mechanisms play a major role in reducing the number of test cases without affecting their quality. However, reducing the number of test cases especially in configurable software systems is a major problem.

In recent years, configurable software systems have gained paramount importance in the market because of their ability to alter software behavior through configuration. Traditional test design techniques are useful for fault discovery and prevention but not for fault elimination because of the combinations of input components and configurations [1]. We consider that all configuration combinations lead to exhaustive testing, which is impossible because of time and resource constraints [2,3]. The number of test cases could be minimized by designing effective test cases that have the same effect as exhaustive testing.

Strategies have been developed in the last 20 years to solve the aforementioned problem. Among these strategies, combinatorial testing strategies are the most effective in designing test cases for the problem. These strategies help search and generate a set of tests, thereby forming a complete test suite that covers the required combinations in accordance with the strength or degree of combination. This degree starts from two (i.e., $d = 2$, where d is the degree of combinations).

We consider that all combinations in a minimized test suite is a hard computational optimization problem [4–6] because searching for the optimal set is a nondeterministic polynomial time (NP)-hard problem [5–9]. Thus, searching for an optimum set of test cases can be a difficult task, and finding a unified strategy that generates optimum results is challenging. Two directions can be followed to solve this problem efficiently and to find a near-optimal solution. The first uses computational algorithms with a mathematical arrangement; the other uses nature-inspired meta-heuristic algorithms [10].

Using nature-inspired meta-heuristic algorithms can generate more efficient results than computational algorithms with a

* Tel.: +41 779158530.
E-mail address: bestoun@idsia.ch.
Peer review under responsibility of Karabuk University.

http://dx.doi.org/10.1016/j.jestch.2015.11.006
2215-0986/© 2016, The Authors. Publishing services by Elsevier B.V. on behalf of Karabuk University

mathematical arrangement [10,11]. In addition, this approach is more flexible than others because it can construct combinatorial sets with different input factors and levels. Thus, its outcome is more applicable because most real-world systems have different input factors and levels.

Developed by Xin-She Yang and Suash Deb [12], the cuckoo search (CS) algorithm is a new algorithm that can be used to efficiently solve global optimization problems [13]. CS can solve NP-hard problems that cannot be solved by exact solution methods [14]. This algorithm is applicable and efficient in various practical applications [9,13,15–17]. Recent evidence shows that CS is superior to other meta-heuristic algorithms in solving NP-complete problems [9,13,16,17].

Although combinatorial testing proves its effectiveness in many researches in the literature, evidence showed that there are weak points in this testing technique [18]. It supposes that the input factors have the same impact of the system. However, practically the test cases have different impact and some of the test cases may not detect any fault. In other words, most of the faults may be detected by a fraction of the test suite. Hence, there should be a mechanism to filter the test suite based on the fault detection strength of each test case. Success to do so will lead to further optimize the generated test suite by the combinatorial strategy. This paper presents a technique to overcome this problem systematically. It should be noted that there could be constraints among the input configuration of the software-under-test. This is out of the scope of this paper; however, the method could be applicable for this issue too.

The rest of the paper is organized as follows: Section 2 presents the mathematical notations, definitions, and theories behind the combinatorial testing. Section 3 illustrates a practical model of the problem using a real-world case study. Section 4 summarizes recent related works and reviews the existing literature. Section 5 discusses the methodology of the research and implementation. The section reviews CS in detail and discusses how the combinatorial test suites are generated using such an algorithm. Section 6 contains the evaluation results. Section 7 gives threats to validity for the experiments and case study. Finally, Section 8 concludes the paper.

2. Combinatorial optimization and its mathematical representation

A future move toward combinatorial testing involves the use of a sampling strategy derived from a mathematical object called covering array (CA) [19]. In combinatorial testing, a CA can be demonstrated simply through a table that contains the designed test cases. Each row of the table represents a test case, and each column is an input factor for the software-under-test.

This mathematical object originates essentially from another object called orthogonal array (OA) [20]. An OA_λ (N; d, k, v) is an $N \times k$ array, where for every $N \times d$ sub-array, each d-tuple occurs exactly λ times, where $\lambda = N/v^d$; d is the combination strength; k is the number of factors ($k \geq d$); and v is the number of symbols or levels associated with each factor. In covering all the combinations, each d-tuple must occur at least once in the final test suite [21]. When each d-tuple occurs exactly once, $\lambda = 1$, and it can be unmentioned in the mathematical syntax, that is, OA (N; d, k, v). As an example, OA (9; 2, 4, 3) contains three levels of value (v) with a combination degree (d) equal to two, and four factors (k) can be generated by nine rows. Fig. 1(a) illustrates the arrangement of this array.

The main drawback of OA is its limited usefulness in this application because it requires the factors and levels to be uniform, and it is more suitable for small-sized test suites [22,23]. To address this limitation, CA has been introduced.

CA is another mathematical notation that is more flexible in representing test suites with larger sizes of different parameters and

OA (9; 2, 4, 3)			
k_1	k_2	k_3	k_4
1	1	1	1
2	2	2	1
3	3	3	1
1	2	3	2
2	3	1	2
3	1	2	2
1	3	2	3
2	1	3	3
3	2	1	3

(a)

CA (9; 2, 4, 3)			
k_1	k_2	k_3	k_4
1	3	3	3
3	2	3	1
1	1	2	1
1	2	1	2
3	1	1	3
2	1	3	2
3	3	2	2
2	3	1	1
2	2	2	3

(b)

MCA (9; 2, 4, $3^2 2^2$)			
k_1	k_2	k_3	k_4
2	1	1	2
2	2	2	1
3	3	2	2
1	3	1	1
1	1	2	1
1	2	1	2
3	2	1	1
3	1	1	1
2	3	1	2

(c)

Fig. 1. Three different examples to illustrate OA, CA, and MCA.

values. In general, CA uses the mathematical expression CA_λ (N; d, k, v) [24]. A CA_λ (N; d, k, v) is an $N \times k$ array over $(0, \ldots, v-1)$ such that every $B = \{b_0, \ldots, b_{d-1}\} \in \binom{\{0, \ldots, k-1\}d}{d}$ is λ-covered and every $N \times d$ sub-array contains all ordered subsets from v values of size d at least λ times [25], where the set of column $B = \{b_0, \ldots, b_{d-1}\} \supseteq \{0, \ldots, k-1\}$.To ensure optimality, we normally want d-tuples to occur at least once. Thus, we consider the value of $\lambda = 1$, which is often omitted. The notation becomes CA (N; d, k, v) [26]. We assume that the array has size N, combination degree d, k factors, v levels, and index λ. Given d, k, v, and λ, the smallest N for which a CA_λ (N; t, k, v) exists is denoted as CAN_λ (d, k, v). A CA_λ (N; d, k, v) with $N = CAN_\lambda$ (d, k, v) is said to be optimal as shown in Eq. 1 [27]. Fig. 1(b) shows a CA with size 9, which has 4 factors each having 3 levels with a combination degree equal to 2.

$$CAN(d, k, v) = \min\{N : \ni CA(N, d, k, v)\} \tag{1}$$

CA is suitable when the number of levels v is equal for each factor in the array. When factors have different numbers of levels, mixed covering array (MCA) is used. MCA is notated as MCA (N, d, k, (v_1, v_2, v_3. . .. . v_k)). It is an $N \times k$ array on v levels and k factors, where the rows of each $N \times d$ sub-array cover all d-tuples of values from the d columns at least once [2]. For more flexibility in the notation, MCA can be represented as MCA (N; d, v^k)) and can be used for a fixed-level CA such as CA (N; d, v^k) [8]. Fig. 1(c) illustrates an MCA with size 9 that has 4 factors: 2 of them having 3 levels each and the other 2 having 2 values each. In addition, Fig. 2 shows an example CA (4; 2, 2^3) of the way the d-tuples are generated and covered using CA.

3. Problem definition through a practical example

With the development of communication systems, mobile phones are among the latest industry innovations and a common mode of communication among humans. Various operating systems have been developed for these devices as platforms for performing basic tasks, such as recognizing inputs, sending outputs, keeping track of files, and controlling peripheral devices. This development has paved the way for the emergence of smart phones. Smart phone applications or “apps” installed on mobile platforms perform useful tasks. Android is an important platform that includes a specialized operating system and an open-source development environment.

In controlling the behavior of a smart phone, many configuration options must be adjusted in the Android unit. In executing the running apps on a variety of hardware and software platforms, this adjustment of options plays an important role. Some smart phones, for example, have a physical keyboard, whereas others have a soft keyboard. Fig. 3 shows a sample of the resource configuration file