The Journal of Systems and Software 98 (2014) 191–207

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Generating combinatorial test suite using combinatorial optimization

Zhiqiang Zhang[a,b,d,*], Jun Yan[c,d], Yong Zhao[e], Jian Zhang[a]

[a] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[b] University of Chinese Academy of Sciences, China
[c] Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, China
[d] State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, China
[e] Department of Computer Science, College of Engineering and Applied Science, University of Colorado at Colorado Springs, United States

ARTICLE INFO

Article history:
Received 8 November 2013
Received in revised form 13 April 2014
Accepted 1 September 2014
Available online 16 September 2014

Keywords:
Combinatorial testing
Test generation
Combinatorial optimization

ABSTRACT

Combinatorial testing (CT) is an effective technique to test software with multiple configurable parameters. It is used to detect interaction faults caused by the combination effect of parameters. CT test generation aims at generating covering arrays that cover all t-way parameter combinations, where t is a given covering strength. In practical CT usage scenarios, there are usually constraints between parameters, and the performance of existing constraint-handling methods degrades fast when the number of constraints increases.

The contributions of this paper are (1) we propose a new one-test-at-a-time algorithm for CT test generation, which uses pseudo-Boolean optimization to generate each new test case; (2) we have found that pursuing the maximum coverage for each test case is uneconomic, and a possible balance point is to keep the approximation ratio in [0.8,0.9]; (3) we propose a new self-adaptive mechanism to stop the optimization process at a proper time when generating each test case; (4) extensive experimental results show that our algorithm works fine on existing benchmarks, and the constraint-handling ability is better than existing approaches when the number of constraints is large; and (5) we propose a method to translate shielding parameters (a common type of constraints) into normal constraints.

1. Introduction

Combinatorial testing (CT) is used to test software systems with multiple configurable parameters. For real software systems, the number of configurable parameters may be large, and testing all possible configurations is not possible due to limited testing resource. CT enables the tester to execute a small set of test cases on the system, while achieving very high fault coverage.

The first step to apply CT is to build a parameterized model of the system under test (SUT). The tester should first identify the input parameters related to the test goal, i.e. parameters affecting the system behavior which he or she is interested in during the testing process. These parameters may include but not limited to the following:

- Parameters of method calls.
- Parameters in system settings.
- A selection of replaceable system components installed in a test environment, such as hardware devices, system libraries and applications. These components are usually provided by multiple third-party providers, and can have multiple versions.

After the input parameters are identified, the tester needs to identify the value domain of each parameter. CT requires the value domain of each input parameter to be a finite set of discrete values. Typically, the number of possible values should not be too large, or else the test suite size will be very large. If a parameter has too many possible values, the tester should first select several representative values using techniques such as equivalence partitioning, and boundary value analysis.

Sometimes the parameters are not obvious. In these cases, the tester needs to investigate the input domain, and build some *abstract parameters*. Abstract parameters could be some properties of the program input, such as the length of the input file, or the type of a triangle (e.g. is it a right triangle, acute triangle or obtuse triangle, etc.). Then the test cases will be *abstract test cases*, and the tester needs to translate the abstract test cases into *concrete test cases* before execution.

* Corresponding author at: State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China. Tel.: +86 10 62661625.
E-mail addresses: zhangzq@ios.ac.cn (Z. Zhang), yanjun@otcaix.iscas.ac.cn (J. Yan), yzhao@uccs.edu (Y. Zhao), zj@ios.ac.cn (J. Zhang).

http://dx.doi.org/10.1016/j.jss.2014.09.001
0164-1212/

After the modeling process, the SUT is abstracted as a parameterized black-box model consisting of several parameters, and each parameter has a domain of several possible values.

From a black-box perspective, the CT fault model assumes that failures are caused by parameter combinations. These failures are the consequence of *interaction faults* inside the black-box. When several parameters take specific value combinations, a failure occurs. An investigation by Kuhn and Michael (2002) shows that in some systems, failures are usually caused by combinations of size 1–6 in some systems, and about 90% of the failures are caused by parameter combinations of size no more than 3.[1]

The key idea of CT is that since most failures are caused by small parameter combinations, if we have tested all small parameter combinations, then most of the interaction faults can be detected. In CT, we usually use a *covering array* (CA) as the test suite, which covers these small parameter combinations.

There are a lot of researches on covering array generation. The most popular three test generation strategies are the following:

- The one-test-at-a-time strategy first introduced by Cohen et al. (1997) in the Automatic Efficient Test Case Generator (AETG).
- The In-Parameter-Order (IPO) strategy introduced by Lei et al. (2008).
- The strategy of searching for a whole covering array satisfying the coverage criteria, which is used in methods such as EXACT (Yan and Zhang, 2008) and CASA (Garvin et al., 2009, 2011).

In practical CT applications, SUT models usually have constraints between parameters. Every test case must satisfy all parameter constraints, or else it will be invalid and cannot be executed. Ignoring these constraints will make some test cases invalid, and the parameter combinations covered by the invalid test cases may not be tested. Thus some interaction faults supposed to be detected by the test suite may not be detected. Handling constraints is a difficult problem in CT generation. The performance of existing methods degrades fast when the number of constraints increases. Thus better constraint-handling techniques are in great need.

The contributions of this paper are listed as follows: (1) we propose a new one-test-at-a-time algorithm for CT test generation, which uses pseudo-Boolean optimization to generate each new test case; (2) we have found that pursuing the maximum coverage for each test case is uneconomic, and a possible balance point is to keep the approximation ratio in [0.8,0.9]; (3) we propose a new self-adaptive mechanism to stop the optimization process at a proper time when generating each test case; (4) extensive experimental results show that our algorithm works fine on existing benchmarks, and the constraint-handling ability is better than existing approaches when the number of constraints is large; and (5) we propose a method to translate shielding parameters (a common type of constraints) into normal constraints.

This paper is organized as follows: In Section 2, we introduce some background knowledge of CT. In Section 3, we introduce our CT test generation algorithm. In Section 4, we perform experiments to evaluate the algorithm's performance and make discussions. In Section 5 we discuss the related works. Finally, we conclude our work in Section 6.

[1] A recent case study by Ghandehari et al. (2013) revealed that there are some cases where failures are caused by parameter combinations of size greater than 6. The size of the failure-causing parameter combinations depends a lot on the model and the nature of the SUT. However, CT is still applicable in many situations and has been used by a lot of industrial practitioners.

2. Basic concepts and notations

We first introduce some basic concepts used in this paper. An SUT model is defined as follows:

Definition 1. An *SUT model* $SUT(P, D)$ consists of a set of input parameters $P = \{p_1, p_2, \ldots, p_k\}$, and a function D mapping each parameter p_i to its value domain $D(p_i)$. k is called the *number of parameters*, and $s_i = |D(p_i)|$ is called the *level* of parameter p_i.

Definition 2. A *test case* $t = (v_1, v_2, \ldots, v_k)$ is an assignment to all input parameters, such that parameter p_i takes the value of $v_i \in D(p_i)$, for $1 \le i \le k$.

Definition 3. A *combination* $\sigma = \{(p_{i_1}, v_{i_1}), (p_{i_2}, v_{i_2}), \ldots, (p_{i_l}, v_{i_l})\}$ is an assignment to parameters $p_{i_1}, p_{i_2}, \ldots, p_{i_l}$, such that parameter p_{i_j} takes the value of $v_{i_j} \in D(p_{i_j})$, for $1 \le j \le l$. l is called the *size* of the combination.

A test case $t = (v_1, v_2, \ldots, v_k)$ *covers* combination $\sigma = \{(p_{i_1}, v'_{i_1}), (p_{i_2}, v'_{i_2}), \ldots, (p_{i_l}, v'_{i_l})\}$, if and only if for $1 \le j \le l$, $v_{i_j} = v'_{i_j}$, i.e. the values of p_{i_j} in t and in σ are identical.

A covering array is defined as follows:

Definition 4. A *covering array* $CA(SUT(P, D), t)$ is an $N \times k$ array, where $SUT(P, D)$ is an SUT model, and t is the *strength* of the CA. Each row of the array is a test case, and for $1 \le i \le k$, the ith column corresponds to the values of parameter p_i in test cases. For any t columns of the array, the $N \times t$ sub-array covers all value combinations of the corresponding t parameters.

Note that the original definition of covering arrays requires that all parameters have the same level, however most real SUT models have parameters of different levels. In our paper, a covering array is actually a *mixed-level covering array*, which allows parameters to have different levels. A covering array $CA(SUT(P, D), t)$ can also be denoted as $CA(N; k, (s_1, s_2, \ldots, s_k), t)$, which is a more traditional form, where N is the number of rows (test cases).

The original definition of covering arrays requires all t-way parameter combinations to be covered. However, there are many cases where some parameters interact more (or less) often with each other than with other parameters. If we enforce a global strength, the covering strength needs to be set at the highest interaction level, which will greatly increase the number of test cases, and a lot of resources will be wasted on testing unimportant parameter combinations. Cohen et al. (2003a,b) proposed the concept of *variable strength covering array* (VCA), which allows the tester to specify different covering strengths on different subsets of parameters. Here we use a modified definition:

Definition 5. A variable strength $t^+ = \{(P_1, t_1), (P_2, t_2), \ldots, (P_l, t_l))\}$ is a set of coverage requirements, where P_i is a set of parameters, and t_i is a covering strength on P_i, for $1 \le i \le l$. For $1 \le i \le l$, coverage requirement (P_i, t_i) requires that all t_i-way value combinations of parameters in P_i be covered by the test suite.

When we replace the universal strength t with a variable strength t^+, the covering array will be called a *variable strength covering array*. (Note that the previous definition of the universal strength t can be represented by a variable strength $t^+ = \{(P, t)\}$.) If a variable strength covering array meets covering requirement t^+, then for each $p_i = (P_i, t_i) \in t^+$, the sub-array of parameters in p_i is a covering array of strength t_i.

In the rest of this paper, we use the term *target combinations* to denote parameter combinations which need to be covered as specified by the coverage requirements (covering strength).

Another important concept in CT is *constraints*. Sometimes, some parameters in the SUT model must conform to some restrictions, or else the test case will become invalid. Ignoring parameter