Applied Soft Computing xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

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

Using simulated annealing for computing cost-aware covering arrays

Q1 Gulsen Demiroz*, Cemal Yilmaz

Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul 34956, Turkey

ARTICLE INFO

Article history:
Received 30 December 2015
Received in revised form 11 August 2016
Accepted 12 August 2016
Available online xxx

Keywords:
Software quality assurance
Combinatorial interaction testing
Covering arrays
Cost-aware testing
Simulated annealing

ABSTRACT

The configuration spaces of software systems are often too large to test exhaustively. Combinatorial interaction testing approaches, such as covering arrays, systematically sample the configuration space and test only the selected configurations. In an attempt to reduce the cost of testing, standard t-way covering arrays aim to cover all t-way combinations of option settings in a minimum number of configurations. By doing so, they simply assume that every configuration costs the same. When the cost varies from one configuration to another, however, minimizing the number of configurations is not necessarily the same as minimizing the cost. To overcome this issue, we have recently introduced cost-aware covering arrays. In a nutshell, a t-way cost-aware covering array is a standard t-way covering array that "minimizes" a given cost function modeling the actual cost of testing. In this work we develop a simulated annealing-based approach to compute cost-aware covering arrays, which takes as input a configuration space model enhanced with a cost function and computes a cost-aware covering array by using two alternating neighboring state generation strategies together with a fitness function expressed as a weighted sum of two objectives: covering all required t-way option setting combinations and minimizing the cost function. To the best of our knowledge, the proposed approach is the first approach that computes cost-aware covering arrays for general, non-additive linear cost functions with multiplicative interaction effects. We evaluate the approach both by conducting controlled experiments, in which we systematically vary the input models to study the sensitivity of the approach to various factors and by conducting experiments using real cost functions for real software systems. We also compare cost-aware covering arrays to standard covering arrays constructed by well-known algorithms and study how fast the construction costs are compensated by the cost reductions provided. Our empirical results suggest that the proposed approach is more effective and efficient than the existing approaches.

1. Introduction

Q3 The configuration spaces of configurable software systems are often too large to test exhaustively. In general, the number of possible configurations grows exponentially with the number of configuration options. Therefore, testing all configurations in a timely manner is often far beyond the available resources.

Combinatorial interaction testing (CIT) approaches systematically sample the configuration space and test only the selected configurations. These approaches take as input a *configuration space model*. In its simplest form, this model includes a set of configuration options, each of which can take on a small number of settings. Given a configuration space model, the sampling is typically carried out by computing a combinatorial object, called a *covering array*. A t-way covering array is a set of configurations in which each valid combination of option settings for every combination of t options appears at least once [1]. In this definition, t is often referred to as the *coverage strength*.

The basic justification for using CIT is that t-way covering arrays can (under certain assumptions) reveal all failures caused by the interactions of t or fewer options [2]. The results of many empirical studies suggest that the majority of option-related failures in practice are caused by the interactions between only a small number of options; thus, t is small in practice, typically $2 \leq t \leq 6$ [1,3–5]. Therefore, covering arrays are an efficient and effective way of revealing such failures [1,3–5].

In an attempt to reduce the actual cost of testing, standard covering arrays aim to cover all required t-way combinations of option settings using a minimum number of configurations. By doing so, they assume that every configuration costs the same. However, we empirically demonstrated that the cost may vary from one configuration to another and when it does, minimizing the number of configurations is not necessarily the same as minimizing the

Q2 * Corresponding author.
E-mail addresses: gulsend@sabanciuniv.edu (G. Demiroz), cyilmaz@sabanciuniv.edu (C. Yilmaz).

http://dx.doi.org/10.1016/j.asoc.2016.08.022
1568-4946/

actual cost of testing [6,7]. For example, in a study we conducted on MySQL, a widely-used and highly-configurable database management system, we observed that the cost of configuring MySQL Community Server (a component of the system) with its default configuration takes about 6 min on average (on an 8-core Intel Xeon 2.53 GHz CPU with 32 GB of RAM). On the other hand, configuring the server with NDB cluster storage support, a feature that enables clustering of in-memory databases, takes about 9 min (50% more), as this feature needs to be compiled into the system [6]. Consequently, reducing the number of configurations that include the NDB feature in a covering array without adversely affecting the coverage properties of the array can greatly reduce the amount of time required for testing.

Standard covering arrays, on the other hand, do not take actual testing costs into account when computing covering arrays. To overcome this shortcoming, we have recently introduced a novel combinatorial object for testing, called a *cost-aware covering array* [6]. A t-way cost-aware covering array is a t-way covering array that "minimizes" a given cost function, which models the actual cost of testing. Fig. 1 illustrates cost-aware covering arrays using an example scenario. In this scenario, the software under test (SUT) has seven binary options (o_1, ..., o_7) with two costly option setting combinations: ($o_1 = 0$) and ($o_2 = 1 \wedge o_3 = 1$). Furthermore, the system is to be tested using a 2-way covering array. Fig. 1a and b presents a standard 2-way covering array and a 2-way cost-aware covering array, respectively. Although both of these arrays are valid 2-way covering arrays for the given scenario, the costly combinations appear three times more frequently in the standard covering array than in the cost-aware covering array; 9 appearances of the costly combinations in the standard covering array vs. 3 appearances in the cost-aware covering array (highlighted as bold in the figure). That is, if we, for example, assume that both costly combinations cost the same and the cost of the remaining combinations is negligible compared to that of the costly combinations, then the cost-aware covering array in Fig. 1b covers all 2-way option setting combinations at one third of the cost compared to the standard covering array in Fig. 1a.

In an earlier work [6], we presented a greedy, proof-of-concept algorithm to compute cost-aware covering arrays and empirically demonstrated that they could reduce testing costs without adversely affecting the coverage properties of covering arrays. However, the proposed algorithm addressed only a simple and quite specific scenario, in which it is assumed that: (1) the configuration space is composed of compile-time and runtime options; (2) all compile-time configurations cost the same; and (3) the rest of the costs are negligible. For this scenario, minimizing the cost was effectively the same as minimizing the number of times the system under test was built (i.e., minimizing the number of unique compile-time configurations).

In this work, however, we present a simulated annealing-based algorithm to compute cost-aware covering arrays for non-additive linear cost functions with multiplicative interaction effects, where the cost may vary from one option setting combination to another. To the best of our knowledge, this work is the first work that computes cost-aware covering arrays for such general cost functions.

Given a standard configuration space model enhanced with a cost function, the approach starts with a standard t-way covering array computed as the initial state and proceeds by using two alternating neighboring state generation strategies guided by a fitness function expressed as a weighted sum of two objectives: covering all required t-way combinations of option settings and minimizing the cost function. Simulated annealing has been used to compute standard covering arrays [8–11]. Our work differs in that we use it to compute cost-aware covering arrays.

We evaluate the proposed approach both (1) by conducting controlled experiments, in which we systematically vary the configuration space models and the cost functions to study the sensitivity of the approach to various factors, including the coverage strength, the number of configuration options, the percentage of costly options, the number of costly option setting combinations, the cardinality of costly combinations, and their cost impact ratio and (2) by conducting experiments using real cost functions created for two widely-used highly-configurable software systems, namely Apache (an HTTP server) and MySQL (a database management system). Furthermore, we compare the cost-effectiveness of cost-aware covering arrays to standard covering arrays constructed by well-known algorithms, namely IPOG [12,13], IPOF [14], and IPOF2 [14], and study how fast the construction costs of cost-aware covering arrays are compensated by the cost reductions they provide.

Overall, compared to standard covering arrays, the cost-aware covering arrays generated by the proposed approach were 26.18%, 20.81%, and 19.81% less costly when $t = 2$, 3, and 4, respectively. Furthermore, the average compensation rate, i.e., the number of times a cost-aware covering array is to be used for testing before its construction time can be compensated, was 0.07 with a maximum rate of 2.52. That is, cost-aware covering arrays were on average more cost-effective than standard covering arrays even during the first use. And in the worst case, at most 3 uses were required before cost-aware covering arrays became more cost-effective.

The remainder of the paper is organized as follows: Section 2 provides background information; Section 3 describes the proposed approach; Section 4 presents the experimental studies; Section 5 discusses potential threats to validity; Section 6 summarizes related work; and Section 7 presents concluding remarks and possible directions for future work.

2. Background information

In this section we provide background information on standard covering arrays, cost-aware covering arrays, and simulated annealing.

2.1. Covering arrays

Covering arrays take as input a configuration space model, which implicitly defines the valid configuration space for the system under test. In its simplest form, the model $M = \langle O, V \rangle$ includes a set of configuration options $O = \{o_1, o_2, \ldots, o_n\}$ and their possible values $V = \{V_1, V_2, \ldots, V_n\}$, where each configuration option o_i ($1 \le i \le n$) takes a value from a finite set of $|V_i|$ distinct values $V_i = \{v_{i1}, v_{i2}, \ldots, v_{i|V_i|}\}$.

Given a configuration space model $M = \rangle O, V \langle$:

Definition 1. A tuple $\phi = \{\langle o_{i_1}, v_{j_1} \rangle, \langle o_{i_2}, v_{j_2} \rangle, \ldots, \langle o_{i_k}, v_{j_k} \rangle\}$ is a set of option-value pairs for a combination of k distinct options, such that $1 \le k \le n$, $1 \le i_1 < i_2 < \cdots < i_k \le n$, and $v_{j_p} \in V_{i_p}$ for $p = 1, 2, \ldots, k$.

Definition 2. A t-tuple ϕ_t is a tuple comprised of t configuration options, where $1 \le t \le n$.

Let Φ_t be the set of all t-tuples.

Definition 3. A configuration c is an n-tuple, i.e., $c \in \Phi_n$, where $n = |O|$.

Definition 4. The configuration space $C = \{c : c \in \Phi_n\}$ is the set of all configurations.

Definition 5. A t-way covering array $CA(t, M = \langle O, V \rangle)$ is a set of configurations, in which each t-tuple appears at least once, i.e., $CA(t, M = \langle O, V \rangle) = \{c_1, c_2, \ldots, c_N\}$, such that $\forall \phi_t \in \Phi_t \, \exists c_i \supseteq \phi_t$, where $c_i \in C$ for $i = 1, 2, \ldots, N$.

Please cite this article in press as: G. Demiroz, C. Yilmaz, Using simulated annealing for computing cost-aware covering arrays, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.08.022