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. Traditional t-way covering arrays aim to cover all t-way combinations of option settings in a minimum number of configurations. By doing so, they assume that the testing cost of a configuration is the same for all configurations. In my thesis work, we however argue that, in practice, the actual testing cost may differ from one configuration to another and that accounting for these differences can improve the cost-effectiveness of covering arrays. To this end, we introduced a new novel combinatorial object, called a **cost-aware covering array** where a t-way cost-aware covering array is a t-way covering array that minimizes a given cost function.

As part of progress, we developed an algorithm for a simple, yet important scenario, and the results of our empirical studies suggest that cost-aware covering arrays can greatly reduce the actual cost of testing compared to traditional covering arrays. We also defined a framework for defining the cost function but then we observed that manually creating these cost models is impractical. Hence our first future goal is to develop an approach for automatically discovering cost models for complex configuration spaces. Our second future goal is then to develop algorithms to generate cost-aware covering arrays for more general cost scenarios. Our focus is currently on meta-heuristic search algorithms such as simulated annealing and genetic algorithms to construct cost-aware covering arrays. Another goal is to expand the cost framework to be test-case aware where not every test case is valid for a configuration, hence the cost of running the test suite is actually different for each configuration.

Keywords

Combinatorial Interaction Testing, Covering Arrays, Test Cost Reduction

1. INTRODUCTION

The configuration spaces of configurable software systems are often too large to test exhaustively. The number of possible configurations is often far beyond the available resources to test the entire configuration space in a timely manner.

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**. The model includes a set of configuration options, each of which can take on a small number of option settings together with inter-option constraints that implicitly or explicitly invalidate some option setting combinations (i.e., not all configurations may be valid). In effect, the model implicitly defines the valid configuration space for the system under test. Given a configuration space model, the sampling is carried out by computing a combinatorial object, called a **covering array**. Given a configuration space model, 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.

To reduce the actual cost of testing, existing CIT approaches construct a t-way covering array in such a way that all valid t-way combinations of option settings are covered by using a minimum number of configurations. By doing so, these approaches implicitly assume a simple cost model in which the cost of configuring the system under test is the same for all configurations.

In my thesis work, we argue that this cost model is not always valid in practice. First, we observe that the configuration cost often varies from one configuration to other in practice. 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 core component of the system) with its default configuration took about 6 minutes on average (on an 8-core Intel Xeon 2.53GHz CPU with 32 GB of RAM). On the other hand, configuring the system with NDB cluster storage support – a feature that enables clustering of in-memory databases, took about 9 minutes (i.e., a 50% increase in the configuration cost), as these features need to be compiled into the system. Note that it is not just about testing the NDB feature in isolation, but about testing the interactions between this feature and the rest of the features. Furthermore, NDB is not the only feature that increases
the cost of configuring the MySQL server; there are other costly combinations of features. Consequently, reducing the number of configurations that include the NDB feature in a covering array, without adversely affecting the coverage of option setting combinations, can significantly reduce the amount of time required for testing. However, existing approaches do not take actual testing costs into account when computing covering arrays.

Second, we observe that highly configurable systems often have reusable components, which, once configured, can be used in other configurations with no or very little additional cost. One simple example is the presence of compile-time and runtime configuration options. Compile-time options need to be set before the system can be built. The system is then configured as a part of the build process. Therefore, changing the setting of a compile-time option requires a partial or a full rebuild of the system. On the other hand, given a build of the system, runtime options are set when the system is running and the system is configured on the fly. A build of the system is a reusable component. Once the system is built for a given compile-time configuration, the same build can be used with different runtime configurations without any additional cost. Runtime configurations, on the other hand, are not reusable. Even for the same build (i.e., for the same compile-time configuration) they need to be reconfigured every time the program is executed, unless the program state is saved for future use.

Figure 1(a) and 1(b) illustrate the effect of reusable components on testing cost in a hypothetical scenario. In this scenario, we have 7 configuration options \( o_1, \ldots, o_7 \), each of which can take on a binary value (i.e., 0 or 1). The first 3 options \( o_1, o_2, \) and \( o_3 \) are compile-time options, whereas the remaining options \( o_4, o_5, o_6, \) and \( o_7 \) are runtime options. There are no system-wide inter-option constraints and the system is to be tested with a 2-way covering array. Two covering arrays are created for comparison.

The 2-way covering array in Figure 1(a) includes 8 unique combinations of compile-time option settings, requiring to build the system 8 times. On the other hand, the 2-way covering array in Figure 1(b) requires to build the system only 4 times, as it includes 4 unique compile-time configurations. For example, once the system is built for \( o_1=0, o_2=0, \) and \( o_3=0 \), the same binaries are reused without any additional cost in 3 more configurations included in the covering array. Assuming that the runtime configuration cost is negligible compared to the compile-time configuration cost and that the compile-time configuration cost is the same for all configurations, the 2-way covering array in Figure 1(b) tests all 2-way option setting combinations at half of the cost compared to the 2-way covering array in Figure 1(a).

Note that the relationship between the cost of compile-time and that of runtime configuration options may vary in practice, e.g., runtime options may be more costly to configure than compile-time options. However, this does not affect the validity of the ideas discussed in this paper.

In my thesis work, we have so far introduced a new combinatorial object, called a cost-aware covering array. In a nutshell, a t-way cost-aware covering array is a t-way covering array that minimizes a given cost function [6]. We then developed a mathematical framework for defining the cost function. We conducted a feasibility study to evaluate our approach on a simple, yet important scenario. The results strongly suggested that cost-aware covering arrays can significantly reduce testing cost, compared to traditional covering arrays [6]. As future work, we plan to develop frameworks, algorithms, and tools to efficiently and effectively compute cost-aware covering arrays and evaluate their cost effectiveness by conducting industrial-strength experiments.

![Figure 1: (a) A traditional 2-way covering array. (b) A 2-way cost-aware covering array.](image-url)

Figure 1: (a) A traditional 2-way covering array. (b) A 2-way cost-aware covering array.

2. RELATED WORK

The problem of generating traditional covering arrays is NP-hard [13]. Nie et al. classify the methods for generating covering arrays into 4 main categories [13]: random search-based methods [15], heuristic search-based methods [4, 9], greedy methods [3, 12], and mathematical methods [17].

Random search-based methods employ a random selection without replacement strategy [15]. Valid configurations are randomly selected from the configuration space in an iterative fashion until all the required t-tuples have been covered by the configurations selected.

Heuristic search-based methods, on the other hand, employ heuristic search techniques, such as hill climbing [5], and simulated annealing [4], or AI-based search techniques, such as genetic algorithms [9], to compute covering arrays. These methods typically maintain a set of configurations at a given time and iteratively apply a series of transformations to the set until the set constitutes a t-way covering array.

Greedy algorithms also work in an iterative manner [3, 12]. At each iteration, among the sets of configurations examined as candidates, the one covering the most previously
uncovered $t$-tuples is included in the covering array and the newly covered $t$-tuples are marked as covered. The iterations end when all the required $t$-tuples have been covered.

All of these existing approaches aim to minimize the number of configurations included in a covering array. Our approach, on the other hand, aims to minimize the actual testing cost of the covering array. One of the closest work to our work concerns test prioritization in covering arrays, where the goal is to prioritize the order in which configurations are tested [1]. Elbaum et al. use the code coverage capabilities of test cases as a criterion for the prioritization so that the test cases that are likely to obtain higher code coverage are tested first [8]. Qu et al. use fault detection capabilities of configurations for the prioritization [14]. Do et al. [7] present various test case prioritization criteria. Furthermore, Srikanth et al. [16] and Kimoto et al. [11] use configuration switching costs for the prioritization. Our work differs in that we aim to minimize the total cost of testing, rather than minimizing the configuration switching cost which solely depends on the previously chosen configuration. However, these approaches are clearly complementary in the sense that while computing cost-aware covering arrays, configuration switching costs can also be taken into account in addition to all other types of costs.

3. RESEARCH PROBLEM

Our observations and hypotheses leading to our research problem are: (1) in practice, the testing cost may not be the same for all configurations, and (2) minimizing the number of configurations as is the case in traditional covering arrays does not necessarily minimize the cost of testing, and (3) cost-aware covering arrays, depending on the configuration space model used, can greatly reduce the cost of testing compared to traditional covering arrays. Hence in my thesis, our aim is to reduce the actual testing cost of CIT approaches without adversely affecting their coverage properties.

3.1 Challenges and Potential Approaches

The approach we use to solve our problem is to make covering arrays to be aware of testing costs. As this new cost-aware covering array aims to reduce the cost function given, we first augment traditional configuration space models with a cost function. We then define a new combinatorial object, called a cost-aware covering array. Given the augmented model and a value of $t$, a $t$-way cost-aware covering array is a $t$-way covering array that minimizes the cost function.

Our first challenge is to develop an effective and efficient way to define the cost function. We observe that manually creating these cost models is cumbersome and error-prone, thus impractical. There are many reasons for this. First, the knowledge of the entire configuration space, such as how configuration options interact and affect the cost of testing, continuously evolves and is distributed among the stakeholders within a project, especially for highly configurable large-scale software systems. Therefore, it is generally hard to reliably consolidate this knowledge in a model [19]. Second, manually defining the cost at the level of option setting combinations is typically infeasible as the number of combinations grows exponentially with the number of options included in the combinations. Third, even if only a small fraction of all combinations affects the testing cost, determining these combinations can still be a non-trivial task for practitioners as it may require knowledge of conducting formal experiments and performing statistical analysis. Finally, even if the costly combinations are known a priori, it can still be hard for practitioners to express relative costs of these combinations in an accurate and precise manner.

We plan to develop an approach for automatically discovering cost models for complex configuration spaces. Given a configuration space model, we plan to use a class of highly efficient experimental designs called screening designs [2] to identify combinations of option settings that significantly affect the cost. We will then quantify the effects of these important combinations in the form of coefficients and compute the cost model by fitting a linear equation to the observed costs using the coefficient estimates. Given a configuration, which is a single set of settings, one for each configuration option, the cost model will then estimate the time it takes to perform the testing task of interest in the configuration.

Once a cost model is defined, our next challenge is to develop efficient and effective approaches to compute covering arrays that minimizes the cost function. We will then empirically evaluate their success both in terms of the construction and the actual testing cost of the resulting covering arrays.

One approach could be to use heuristic search techniques such as simulated annealing or genetic algorithms similar to existing approaches for traditional covering arrays in [4, 9] where the search objective is two-fold now: 1) Meet the coverage criteria of the traditional $t$-way covering array, 2) Minimize the cost function. We are evaluating whether to represent this multi-objective problem as a single-objective one or whether this optimization problem should be viewed as a Pareto-optimal search problem with two goals [10]. A second approach could be to solve this problem with a single-objective of minimizing the cost with a hard constraint of coverage criteria [10]. A third approach could be to treat the cost minimization (costly option setting combinations) as a soft constraint with the single-objective being the coverage criteria [1] and use an approach similar to the other greedy approaches to construct traditional covering arrays [3, 12].

To empirically evaluate the cost-aware covering arrays, we will conduct experiments to compare the efficiency (e.g., reductions in testing time) and effectiveness (e.g., fault revealing capabilities) of cost-aware covering arrays to those of existing traditional covering arrays using the same configuration space models. We plan to use configuration spaces of large highly configurable software systems, such as MySQL and Apache servers to perform our experiments.

3.2 Progress

As a feasibility study for our research problem, we have so far developed an algorithm to compute cost-aware covering arrays for a simple, yet quite practical scenario. In this scenario, the configuration space of the system under test was composed of compile-time and runtime options. We defined a cost function, in which the cost of configuring the system with runtime options was negligible compared to that of configuring the system with compile-time options, and the compile-time configuration cost was the same for all configurations. With this cost function, minimizing the cost is the same as minimizing the number of unique compile-time configurations included in the covering array.

We computed the cost-aware covering arrays for this scenario as follows: (1) a traditional $t$-way covering array was generated for only the compile-time options, (2) the compile-time configurations included in this covering array were then
expressed as inter-option constraints, so that no other compile-time configurations can be selected in the next step and (3) finally, a traditional t-way covering array satisfying these constraints, was generated for all the configuration options.

The final output was a t-way cost-aware covering array, aiming to minimize the testing cost, i.e., the number of times the system was required to be built. To carry out the experiments, we implemented our algorithm using a well-known and widely-used covering array generator ACTS [12].

The results of the experiments showed that t-way cost-aware covering arrays, compared to the traditional t-way covering arrays, reduced the actual testing cost by up to 61% when \( t=3 \) and 60% when \( t=4 \) [6]. These results strongly suggest that cost-aware covering arrays can significantly reduce testing cost, compared to traditional covering arrays.

As another progress, we have also been experimenting with screening designs [2], to identify important effects, i.e., combinations of option settings that affect the testing cost most for a given configuration space, a test case and the cost of running that test case. We have then used these important effects identified to fit a cost model to the observations. We empirically evaluated the proposed approach using two different types of formal screening designs with different testing tasks of practical importance on two large highly configurable software systems. The results of our experiments strongly suggest that the proposed approach can efficiently and effectively discover reliable cost models.

4. CONCLUDING REMARKS

Our main motivation for this research problem is that, in practice, the actual testing cost may not be the same for all configurations and minimizing the number of configurations as is the case in traditional covering arrays does not necessarily minimize the actual cost of testing. The fact that the cost-aware covering arrays were generally more cost-effective than the traditional covering arrays in this feasibility study we did in [6], supports our basic motivation and hypotheses.

We believe that this line of research is novel and interesting, but much work remains to be done to compute these cost-aware combinatorial interaction objects. We will first develop approaches to automatically discover the cost function on highly-configurable systems and enhance the cost model framework as needed. We will then develop frameworks, algorithms, and tools to compute cost-aware covering arrays for any given cost function. Our current focus is on metaheuristic search-based multi-objective optimization techniques as well as evaluating the greedy approaches using constraints. Finally, we will empirically evaluate the success of our approach by conducting large scale experiments on highly configurable software systems.

As another future work, we plan to compute not only cost-aware covering arrays, but also cost- and test case-aware covering arrays. Test case-aware covering arrays have been recently introduced in [18]. Unlike traditional covering arrays, test case-aware covering arrays take both system-wide and test case-specific inter-option constraints into account while constructing interaction test suites. In a nutshell, a t-way test case-aware covering array is not just a set of configurations as in traditional covering arrays, but a set of configurations each of which is associated with a set of test cases. Therefore, making test case-aware covering arrays to be aware of actual testing cost, requires us to take both configuration costs and test execution costs into account.

5. ACKNOWLEDGMENTS

This research was supported by the Scientific and Technological Research Council of Turkey (113E546).

6. REFERENCES