Contents lists available at ScienceDirect

CAAI Transactions on Intelligence Technology

journal homepage: http://www.journals.elsevier.com/caai-transactions-on-intelligence-technology/

Original article

Metaheuristic post-optimization of the NIST repository of covering arrays

Jose Torres-Jimenez[*], Arturo Rodriguez-Cristerna

CINVESTAV-Tamaulipas, Information Technology Laboratory, Km. 5.5 Carretera Cd., Victoria, Tamaulipas, Mexico

A R T I C L E I N F O

Article history:
Received 19 September 2016
Accepted 25 December 2016
Available online xxx

Keywords:
Covering arrays
NIST repository of covering arrays
Metaheuristic post-processing algorithms

A B S T R A C T

Construction of Covering Arrays (CA) with minimum possible number of rows is challenging. Often the available CA have redundant combinatorial interaction that could be removed to reduce the number of rows. This paper addresses the problem of removing redundancy of CA using a metaheuristic post-optimization (MPO) approach. Our approach consists of three main components: a redundancy detector (RD); a row reducer (RR); and a missing-combinations reducer (MCR). The MCR is a metaheuristic component implemented using a simulated annealing algorithm. MPO was instantiated with 21,964 CA taken from the National Institute of Standards and Technology (NIST) repository. It is a remarkable result that this instantiation of MPO has delivered 349 new upper bounds for these CA.

1. Introduction

The use of software has permeated many areas of human activity, so the reliability of software has become important worldwide. It is estimated that software testing consumes about 50% of the cost of developing a new piece of software. A 2002 NIST report [23] indicates that the cost of an inadequate infrastructure for software testing was in the range of \$22.2 to \$59.5 billion (US dollars). Reducing this cost is not only important but the design and implementation of adequate software testing procedures is critical for the reliability of many electronic and mechanical systems, even more so in complex and important systems, such as space shuttles Lions and et al. [16].

According to Myers et al. [17] functional software testing methods may be divided into two main categories: white-box testing and black-box testing. The design of white-box testing suites requires source code of the software under examination. Some testing strategies based on the white-box approach are: statement coverage, decision coverage, condition coverage, decision-condition coverage and multiple-condition coverage. The building of test suites using white-box strategies is more challenging than for black-box strategies, since white-box strategies are based on knowledge of the internal structure of the system. Furthermore, if the system is modified, then tests must be redesigned to satisfy the new version of the system. On the other hand, the design of black box testing suites does not require source code of the software under examination. It compares actual behaviour against expected behaviour based on the functionality and the specification of the software system under examination. Some black-box testing strategies are: exhaustive input testing, equivalence partitioning, boundary-value analysis, cause-effect graphing, error guessing, and *combinatorial interaction testing.*

It is easy to construct test suites using a random black-box approach, but they rarely cover a large percentage of the functionality of the system under examination. A black-box approach that covers 100 percent of the functionality is the exhaustive approach, but it is impractical in most cases because too many tests are required. As an example: if we need to design a test suite for a system that has 20 parameters and each parameter has 10 possible values, it would require 10^{20} tests; however, using a combinatorial interaction testing approach that covers the combinations of all pairs of parameter values, the test suite will require only 155 tests. The number of tests required with combinatorial interaction testing grows logarithmically according to the number of parameters [11]. Empirical studies in software testing have shown that combinatorial interaction testing is a useful approach Kuhn et al. [14], Bell [4]. The mathematical objects that support combinatorial interaction

[*] Corresponding author.
E-mail addresses: jtj@cinvestav.mx (J. Torres-Jimenez), arodriguez@tamps.cinvestav.mx (A. Rodriguez-Cristerna).
Peer review under responsibility of Chongqing University of Technology.

http://dx.doi.org/10.1016/j.trit.2016.12.006

testing are Covering Arrays (CA) and Mixed Covering Arrays (MCA).

CA and MCA are combinatorial structures that have been used successfully in various areas. The most reported application of CA and MCA is in the design of test suites for software combinatorial interaction testing [7,8] which is based on the concept that software faults are caused by unexpected combinatorial interactions of certain size between components. Another application is found in the field of parameter fine-tuning of metaheuristic algorithms [12,19,21,22].

A CA, denoted by CA(N; t, k, v), is an N $\times$ k array, where every entry of the array takes values from a set of symbols of size v, such that every N $\times$ t sub-array contains at least once all possible v^t t-tuples of symbols. An MCA is a generalization of a CA where the alphabets of the columns could have different cardinalities. The test cases are represented by the rows, the parameters are represented by the columns, the parameter values are taken from the set $\{0, 1\ldots, v-1\}$ which is called the alphabet, and t is the strength or combinatorial interaction degree between parameters covered by the CA. Fig. 1 shows an example of a CA(9; 2,4,3), and an MCA(6; 2,4,$3^1 2^3$) is shown in Fig. 2.

The Covering Array Number (CAN) is the minimum N such that for fixed k, v, and t a CA exists. The CAN is denoted by CAN(t,k,v). The construction of CA with N=CAN(t,k,v) is a challenging problem whether we use mathematical structures or metaheuristic algorithms.

When we have non-optimal CA (i.e. a CA with N > CAN(t,k,v)), it usually has many t-tuples that are covered more than once. This fact presents the opportunity to reduce number of rows of CA, given that it may then be possible to identify redundant rows [18] that can be removed.

In this paper we introduce a Metaheuristic Post-Optimization (MPO) approach to reduce the size of a CA by exploiting redundant elements in CA. MPO is composed of three main components: a) a redundancy detector (RD); a row reducer (RR); and a missing-combination reducer (MCR) implemented using a simulated annealing algorithm (the metaheuristic component of our approach). MPO was extensively tested using 21,964 CA (taken from the CA NIST repository). We have improved almost all 21,964 CA, but the most remarkable result is that MPO has set 349 new upper bounds for these CA.

The remaining of the paper is structured in three sections. In section 2 we present in detail MPO approach giving details of the redundancy detector, row reducer and missing-combination reducer components. In section 3 we present the results of instantiating the MPO with the whole National Institute of Standards and Technology repository of covering arrays. Finally in section 4 we present some conclusions.

0	1	2	2	0	1	1	2	0
0	1	2	1	2	0	2	0	1
0	1	2	0	1	2	0	1	2
0	0	0	1	1	1	2	2	2

Fig. 1. Transposed matrix of a $CA(9; 2, 4, 3)$.

0	1	1	0	0	1
0	1	0	1	1	0
0	1	0	1	0	1
0	0	1	1	2	2

Fig. 2. Transposed matrix of an $MCA(6; 2, 4, 3^1 2^3)$.

2. Metaheuristic post-optimization (MPO)

In this section we present implementation details of the MPO approach. We firstly give an overview of the whole process of MPO, secondly, we present details of each of the components RD, RR, MCR.

2.1. Design of MPO approach

The design and implementation of MPO approach is briefly described in algorithm 1, where it can be observed that it has three components and two main loops. The inner loop executes the components: *Redundancy Detector* (RD) and *Row Reducer* (RR). After the inner loop is executed, the *Missing-Combinations Reducer* (MCR) runs. When the MCR (implemented with a simulated annealing (SA) algorithm) is not able to make the number of missing combinations equal to zero, MPO ends.

MPO (algorithm 1) receives as input $\mathscr{A}$ = CA(N; t,k,v) and gives as output $\mathscr{B}$ = CA(N'; t, k, v) with N' $\leq$ N and no missing t-wise combinations. The function τ computes the number of missing t-wise combinations of the parameter passed to it. τ has temporal complexity $O\left(N\binom{k}{t}\right)$ (a more detailed description of how to compute the missing interactions for CA was presented by Avila-George et al. [2].

2.2. Redundancy detector (RD)

The goal of the Redundancy Detector (RD) algorithm is to find a

Algorithm 1 Metaheuristic Post-Optimization (MPO) algorithm.

```
input  : $\mathcal{A} = CA(N; t, k, v)$
output: $\mathcal{B} = CA(N'; t, k, v) | N' \leq N$
1 begin
2     $\mathcal{B}' \leftarrow \mathcal{A}$
      repeat
3         repeat
4             $\mathcal{B}' \leftarrow$ Redundancy Detector $(\mathcal{B}')$;  if $(\mathcal{B}' = 0)$ then $\mathcal{B} \leftarrow \mathcal{B}'$
              $\mathcal{B}' \leftarrow$ Row Reducer$(\mathcal{B}')$;           if $(\mathcal{B}' = 0)$ then $\mathcal{B} \leftarrow \mathcal{B}'$
5         until $\tau(\mathcal{B}') > 0$;
6         $\mathcal{B}' \leftarrow$ Simulated Annealing $(\mathcal{B}')$;      if $(\mathcal{B}' = 0)$ then $\mathcal{B} \leftarrow \mathcal{B}'$
7     until $\tau(\mathcal{B}') > 0$;
8     return $\mathcal{B}$
9 end
```