Information Sciences 460–461 (2018) 172–189

Contents lists available at ScienceDirect

Information Sciences

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

A greedy-metaheuristic 3-stage approach to construct covering arrays

Idelfonso Izquierdo-Marquez[a], Jose Torres-Jimenez[a,*], Brenda Acevedo-Juárez[b], Himer Avila-George[b]

[a] CINVESTAV-Tamaulipas, Cd. Victoria, Tamaulipas, Mexico
[b] Centro Universitario de los Valles, Universidad de Guadalajara. Ameca, Jalisco 46600, Mexico

ARTICLE INFO

Article history:
Received 6 November 2017
Revised 12 April 2018
Accepted 21 May 2018
Available online 29 May 2018

Keywords:
Greedy algorithms
Metaheuristic algorithms
Covering arrays
3-stage approach
Covering perfect hash families
Simulated annealing

ABSTRACT

Covering arrays are combinatorial designs used as test-suites in software and hardware testing. Because of their practical applications, the construction of covering arrays with a smaller number of rows is desirable. In this work we develop a greedy-metaheuristic 3-stage approach to construct covering arrays that improve some of the best-known ones. In the first stage, a covering perfect hash family is created using a metaheuristic approach; this initial array may not be complete, and so the derived covering array may have missing tuples. In the second stage, the covering perfect hash family is converted to a covering array and, in case there are missing tuples, a greedy approach completes the covering array through the addition of some rows. The third stage is an iterative postoptimization stage that combines two greedy algorithms and a metaheuristic algorithm; the greedy algorithms detect and reduce redundancy in the covering array, and the metaheuristic algorithm covers the tuples that may become uncovered after the reduction of redundancy. The effectiveness of our greedy-metaheuristic 3-stage approach is assessed through the construction of covering arrays of order four and strengths 3–6; the main results are the improvement of 9473 covering arrays of strength three, 9303 of strength four, 2150 of strength five, and 291 of strength six. To see how to apply covering arrays to real testing scenarios, the final part of this work presents the use of covering arrays of order four for setting up a composting process.

1. Introduction

A covering array CA(N; t, k, v) is a combinatorial design defined by four parameters N, t, k, and v, where N and k are respectively the number of rows and columns of the array; v is the *order* or the number of distinct symbols in every column of the array; and t is the *strength*, which indicates that every one of the $\binom{k}{t}$ subarrays formed by t columns contain as a row each t-tuple of order v at least once. Fig. 1 shows a CA(12; 2, 7, 3). This covering array has $N = 12$ rows and $k = 7$ columns; the order is $v = 3$ because each column has elements from the alphabet $\mathbb{Z}_3 = \{0, 1, 2\}$; and the strength is $t = 2$ because every one of the $\binom{7}{2} = 21$ subarrays formed by two columns contains as a row the $v^t = 3^2 = 9$ tuples of length two over $\mathbb{Z}_3$,

* Corresponding author.
E-mail addresses: iizquierdo@tamps.cinvestav.mx (I. Izquierdo-Marquez), jtj@cinvestav.mx (J. Torres-Jimenez), brenda.acevedo@academicos.udg.mx (B. Acevedo-Juárez), himer.avila@academicos.udg.mx (H. Avila-George).

https://doi.org/10.1016/j.ins.2018.05.047
0020-0255/

$$\begin{pmatrix} 1 & 2 & 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 2 & 2 & 2 & 1 & 2 & 1 & 2 \\ 2 & 1 & 0 & 1 & 0 & 2 & 1 \\ 0 & 0 & 2 & 1 & 1 & 0 & 1 \\ 0 & 2 & 0 & 2 & 1 & 2 & 2 \\ 2 & 2 & 1 & 2 & 0 & 0 & 0 \\ 1 & 1 & 2 & 0 & 0 & 0 & 2 \\ 1 & 0 & 2 & 2 & 2 & 2 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 & 2 & 2 \\ 0 & 1 & 1 & 2 & 2 & 1 & 1 \end{pmatrix}$$

Fig. 1. A covering array CA(12; 2, 7, 3). Each column has symbols from the alphabet $\mathbb{Z}_3$, and every subarray of $t = 2$ columns contains as a row the nine 2-tuples over $\mathbb{Z}_3$ at least once.

which are the tuples (0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), and (2,2). Some tuples may appear more than once in a subarray, but the requirement is that each tuple appears at least once.

Covering arrays are the primary objects used in combinatorial testing. Combinatorial testing is an important testing strategy that has proven to be effective to detect failures in software and hardware components [18]. This technique is based on the idea that failures occur due to parameter interactions. Covering arrays are employed because they ensure the coverage of all interactions among any t parameters, where t is the strength of the covering array. The covering array of Fig. 1 can be used as a test-suite to test interactions of size $t = 2$ in a component with $k = 7$ input parameters, each of which has $v = 3$ distinct values. Every row of the covering array is a test-case; for example, the first row is the test-case where the first parameter takes its second value, the second parameter takes its third value, the third and fourth parameters take their first value, and so on. If a failure in the component occurs when parameter p_i takes value x and parameter p_j takes value y ($i < j$), then such failure will be detected by the test-suite defined by CA(12; 2, 7, 3), because the tuple (x, y) appears in any subarray of two columns.

When $t = 2$ the combinatorial technique is called *pairwise testing*, and it is the basic technique. However, to detect more complex failures the size of the parameter interactions (the strength of the covering array) must be incremented. Kuhn et al. [19] studied some software systems, and they found that failures are triggered by interactions of at most six parameters. This result gives an estimated value of the size of the parameter interactions required to ensure certain reliability of a software or hardware component. At present, the research in the construction of covering arrays is devoted mainly to covering arrays of strengths two to six.

The problem of constructing covering arrays can be stated as follows: for t, k, v given, find the smallest value of N such that a CA(N; t, k, v) exists. The smallest N is denoted by CAN(t, k, v), and it is called the *covering array number* of t, k, v. Currently, there is no polynomial-time algorithm that can find the exact value of CAN(t, k, v) for general values of t, k, and v; only specific cases have been solved optimally, for examples see [6,16,17]. Thus, the research in the construction of covering arrays is mainly focused on improving the current upper bounds of CAN(t, k, v). The search for covering arrays better than the best-known ones is of practical importance because we can test all interactions of a certain size using fewer test cases.

To search for better covering arrays, we have developed a greedy-metaheuristic 3-stage approach. The first stage uses a metaheuristic algorithm to construct an initial covering perfect hash family (CPHF). CPHFs are arrays that represent certain covering arrays in a compact way. The elements of a CPHF are t-tuples over the finite field with v elements. Every one of these t-tuples represents a column vector of length v^t, and when the elements of a CPHF are replaced by their corresponding column vectors the result is a covering array. The reason to use CPHFs in the first stage is that they allow the construction of large covering arrays with small computational effort. The first stage was instantiated in this paper with a simulated annealing (SA) algorithm. The CPHF is allowed to be incomplete, that is, to have some combinations of t columns that do not generate a subarray covering the v^t tuples over $\mathbb{Z}_v$ in the domain of covering arrays.

The second stage of our approach moves the CPHF resulting from the first stage to the domain of covering arrays. In case there are some missing tuples in the resulting array, a greedy approach is used to complete the covering array through the addition of rows.

The third stage is an iterative postoptimization approach that makes a mixture of two greedy algorithms to detect and reduce redundancy and a metaheuristic algorithm to cover missing tuples. The first greedy algorithm detects redundant elements (elements that do not affect the coverage of the covering array when they are changed). The second greedy algorithm eliminates a row from the covering array by copying the non-redundant elements of the row to the redundant elements of the other rows; this operation may produce an array that is not a complete covering array. The metaheuristic algorithm of the third stage is a SA algorithm whose objective is to cover the missing tuples introduced by the row deletion. If the SA algorithm completes the covering array, then a new iteration of the postoptimization stage is executed.