Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 897027, 19 pages
doi:10.1155/2012/897027

Research Article

New Bounds for Ternary Covering Arrays Using a Parallel Simulated Annealing

Himer Avila-George,[1] Jose Torres-Jimenez,[2] and Vicente Hernández[3]

[1] *Instituto Tecnológico Superior de Salvatierra, Madero 303, 38900 Salvatierra, Guanajuato, Mexico*
[2] *Information Technology Laboratory, Cinvestav Tamaulipas, Km. 5.5 Carretera Victoria-Soto La Marina, 87130 Victoria, TAMPS, Mexico*
[3] *Instituto de Instrumentación para Imagen Molecular (I3M), Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain*

Correspondence should be addressed to Himer Avila-George, hiavgeo@posgrado.upv.es

Received 24 February 2012; Revised 15 June 2012; Accepted 7 July 2012

Academic Editor: John Gunnar Carlsson

A covering array (CA) is a combinatorial structure specified as a matrix of N rows and k columns over an alphabet on v symbols such that for each set of t columns every t-tuple of symbols is covered at least once. Given the values of t, k, and v, the optimal covering array construction problem (CAC) consists in constructing a CA $(N; t, k, v)$ with the minimum possible value of N. There are several reported methods to attend the CAC problem, among them are direct methods, recursive methods, greedy methods, and metaheuristics methods. In this paper, There are three parallel approaches for simulated annealing: the independent, semi-independent, and cooperative searches are applied to the CAC problem. The empirical evidence supported by statistical analysis indicates that cooperative approach offers the best execution times and the same bounds as the independent and semi-independent approaches. Extensive experimentation was carried out, using 182 well-known benchmark instances of ternary covering arrays, for assessing its performance with respect to the best-known bounds reported previously. The results show that cooperative approach attains 134 new bounds and equals the solutions for other 29 instances.

1. Introduction

A covering array, denoted by $\mathrm{CA}(N; t, k, v)$, is a matrix M of size $N \times k$ which takes values from the set of symbols $\{0, 1, 2, \ldots, v-1\}$ (called the alphabet), and every submatrix of size $N \times t$ contains each tuple of symbols of size t (or t-tuple), at least once. The value N is the number of rows of M, k is the number of parameters, where each parameter can take v values, and the interaction degree between parameters is described by the strength t. Each

combination of t columns must cover all the v^t t-tuples. Given that there are $\binom{k}{t}$ sets of t columns in M, the total number of t-tuples in M must be $v^t\binom{k}{t}$. When a t-tuple is missing in a specific set of t columns, we will refer to it as a missing t-wise combination. Then, M is a covering array if the number of missing t-wise combinations is zero.

When a matrix has the minimum possible value of N to be a $\mathrm{CA}(N;t,k,v)$, the value N is known as the covering array number. The covering array number is formally defined as $\mathrm{CAN}(t,k,v) = \min\{N : \exists\ \mathrm{CA}(N;t,k,v)\}$. Given the values of t, k, and v, the optimal covering array construction problem (CAC) consists in constructing a $\mathrm{CA}(N;t,k,v)$ such that the value of N is minimized.

A major application of covering arrays (CAs) arises in software interaction testing [1], where a covering array can be used to represent an interaction test suite as follows. In a software test, we have k components or factors. Each of these parameters has v values or levels. A test suite is an $N \times k$ array where each row is a test case. Each column represents a component, and a value in the column is the particular configuration. By mapping a software test problem to a covering array of strength t, we can guarantee that we have tested, at least once, all t-way combinations of component values [2]. Thus, software testing costs can be substantially reduced by minimizing the number of test cases N in the covering array. Please observe that software interaction testing is a black-box testing technique, and thus it exhibits weaknesses that should be addressed by employing white-box testing techniques. For a detailed example of the use of covering arrays in software interaction testing, the reader is referred to [3].

In this paper, we aim to develop an enhanced sequential simulated annealing (ESSA) algorithm for finding near-optimal covering arrays. Simulated annealing algorithm is a general-purpose stochastic optimization technique that has proved to be an effective tool for approximating globally optimal solutions to many optimization problems. However, one of the major drawbacks of the technique is the time it requires to obtain good solutions (moreover, when the evaluation function requires too much time). To address this drawback, we propose three parallel simulated annealing approaches to solve the CAC problem. The objective is to find the best bounds to some ternary covering arrays by using parallelism. To our knowledge, the application of parallel simulated annealing to the CAC problem has not been reported in the literature. Some methods of parallelization of simulated annealing are discussed in [4–8].

The remainder of this paper is organized as follows. Some techniques that have been used for constructing covering arrays are presented in Section 2. Section 3 describes the components of our sequential annealing algorithm. In Section 4, three parallel simulated annealing approaches are discussed. Section 5 describes the experimental results. Finally, Section 6 presents the conclusions derived from the research presented in this paper.

2. Review of Covering Arrays Construction Methods

Because of the importance of the construction of (near) optimal covering arrays, much research has been carried out in developing effective methods for constructing them. There are several reported methods for constructing these combinatorial models. Among them are (1) direct methods; (2) recursive methods; (3) greedy methods; (4) metaheuristics methods.

Chateauneuf and Kreher [9] introduced a new method to construct covering arrays of strength three. This construction uses the structure of covering arrays and the repetition in covering arrays. The idea is to construct a covering array from a small array, a *starter vector,*

and a group. This construction builds the covering array column by column by considering the group acting on the columns of the starter vector. Meagher and Stevens [10] extended the idea exhibited in [9], presenting a strategy for obtaining the starter vector by local search and the selection of a group action. Recently, Lobb et al. [11] presented a generalization of this method to permit any number of fixed points, permit an arbitrary group acting on the symbols, and permit an arbitrary group acting on the columns. With all these generalizations were obtained new bounds for covering arrays of strength two.

Tang and Woo [12] used *constant weight vectors* to construct test suites to be applied to logic circuit testing. Martinez-Pena et al. [13] introduced a new method for constructing covering arrays using trinomial coefficients, and they improved the results presented in [12]. Martinez-Pena et al. used the trinomial coefficients for the representation of the search space in the construction of ternary covering arrays. It is clear that any covering array is formed by a row set. In this sense, a trinomial coefficient represents a particular subset of rows which may belong to a ternary covering array.

Hartman [3] presented a recursive construction which gives a method of squaring the number k of columns in a covering array of strength t while multiplying the rows N by a factor dependent only on t and v, but independent of k. This factor is related to the Turan numbers $T(t;v)$ that are defined to be the number of edges in the Turan graph.

Colbourn et al. [14] presented a product construction for $t = 2$. In general, the product of two covering arrays where $t = 2$ consists in obtaining a new covering array where the number of columns is equal to the product of the columns of the ingredients, and the number of rows is equal to the sum of the rows of each ingredient.

The majority of commercial and open-source test data generating tools use greedy algorithms for covering arrays construction (AETG, TCG, ACTS, DDA, among others).

Cohen et al. [1] presented a strategy called AETG. In AETG, covering arrays are constructed *one row at a time*. To generate a row, the first t-tuple is selected based on the one involved in most uncovered pairs. Remaining factors are assigned levels in a random order. Levels are selected based on the one that covers the most new t-tuples. For each row that is actually added to the covering array, there are a number of, M, candidate rows that are generated, and only a candidate that covers the most new t-tuples is added to the covering array. Once a covering is constructed, a number, R, of test suites are generated and the smallest test suite generated is reported. This process continues until all pairs are covered.

Tung and Aldiwan [15] proposed a tool called TCG. In TCG, one row is added at a time to a covering array until all pairs are covered. Before each row is added, a number of up to M candidate rows are generated, and the best candidate (covering the most new pairs) is added. M is defined to be the maximum cardinality of factors (the maximum number of levels associated with any factor). To construct each row, factors are assigned levels in an order based on a nonascending order of the cardinality of each factor. Each level for the factor is evaluated, and a count of the number of pairs that are covered is used to determine whether or not to select a level for a factor.

Bryce and Colbourn [16] presented an algorithm called DDA. The DDA constructs one row of a covering array at a time using a steepest ascent approach. Factors are dynamically fixed one at a time in an order based on density. New rows are continually added until all interactions have been covered.

Lei and Tai [17] introduced a new algorithm called IPO, for pairwise testing. For a system with two or more input parameters, the IPO strategy generates a pairwise test set for the first two parameters, extends the test set to generate a pairwise test set for the first three parameters, and continues to do so for each additional parameter. Contrary to many other

algorithms that build covering arrays *one row at a time*, the IPO strategy constructs them *one column at a time*. Lei et al. [18] introduced an algorithm for the efficient production of covering arrays, called IPOG, which generalizes the IPO strategy from pairwise testing to multiway testing. The main idea is that covering arrays of $k-1$ columns can be used to efficiently build a covering array with degree k.

Ronneseth and Colbourn [19] introduced a new algorithm for constructing covering arrays, the BBA. The BBA's fundamental idea is to combine smaller covering arrays by reordering the rows and then to append additional rows for the remaining uncovered pairs.

Some stochastic algorithms in artificial intelligence, such as *tabu search* [20, 21], *simulated annealing* [22], *generic algorithms,* and *ant colony optimization algorithm* [23], provide an effective way to find approximated solutions. In these algorithms, the optimization focuses on one value of N at a time, attempting to find a covering array for that size.

A simulated annealing metaheuristic (henceforth called SAC) has been applied by Cohen et al. in [22] for constructing covering arrays. SAC starts with a randomly generated initial solution M whose cost $c(M)$ is measured as the number of uncovered t-tuples. In their implementation, Cohen et al. use a simple geometric function $T_n = 0.9998T_{n-1}$ with an initial temperature fixed at $T_i = 0.20$. At each temperature, 2000 neighboring solutions are generated. The algorithm stops either if a valid covering array is found, or if no change in the cost of the current solution is observed after 500 trials. The authors justify their choice of these parameter values based on some experimental tuning. They conclude that their simulated annealing implementation is able to produce smaller covering arrays than other computational methods, sometimes improving upon algebraic constructions.

Torres-Jimenez and Rodriguez-Tello [24] introduced a new simulated annealing implementation for constructing binary covering arrays.

3. Sequential Simulated Annealing

In this section, we briefly review simulated annealing (SA) algorithm and propose an enhanced sequential simulated annealing (ESSA) implementation to solve the CAC problem. ESSA is an extension of the simulated annealing presented in [24] for constructing covering arrays for $v > 2$ and mixed-level covering arrays.

Simulated annealing is a randomized local search method based on the simulation of annealing of metal. The acceptance probability of a trial solution is given by (3.1), where T is the *temperature* of the system, ΔC is the difference of the costs between the trial and the current solutions (the cost change due to the perturbation), and (3.1) means that the trial solution is accepted by nonzero probability $e^{-\Delta C/T}$ even though the solution deteriorates (*uphill move*)

$$(\mathbb{P}) = \begin{cases} 1 & \text{if } \Delta C < 0, \\ e^{-\Delta C/T} & \text{otherwise.} \end{cases} \tag{3.1}$$

Uphill moves enable the system to escape from the local minima; without them, the system would be trapped into a local minimum. Too high of a probability for the occurrence of uphill moves, however, prevents the system from converging. In simulated annealing, the probability is controlled by temperature in such a manner that at the beginning of the procedure the temperature is sufficiently high, in which a high probability is available, and

as the calculation proceeds, the temperature is gradually decreased, lowering the probability [25].

The following paragraphs will describe each of the components of the implementation of our simulated annealing implementation. The description is done given the matrix representation of a covering array.

3.1. Internal Representation

Let M be a potential solution in the search space $\mathcal{M}$, that is, a covering array $\mathrm{CA}(N; t, k, v)$ of size N, strength t, degree k, and order v. Then M is represented as an $N \times k$ array on v symbols, in which the element $m_{i,j}$ denotes the symbol assigned in the test configuration i to the parameter j. The size of the search space $\mathcal{M}$ is then given by

$$|\mathcal{M}| = v^{Nk}. \tag{3.2}$$

3.2. Initial Solution

The *initial solution* M is constructed by generating M as a matrix with maximum Hamming distance. The Hamming distance $d(x, y)$ between two rows $x, y \in M$ is the number of elements in which they differ. Let r_i be a row of the matrix M. To generate a random matrix M of maximum Hamming distance, the following steps are performed:

(1) generate the first row r_1 at random,

(2) generate s rows $c_1, c_2, \ldots, c_s$ at random, which will be candidate rows,

(3) select the candidate row c_i that maximizes the Hamming distance according to (3.3) and added to the ith row of the matrix M,

(4) repeat from step 2 until M is completed;

$$g(r_i) = \sum_{s=1}^{i-1} \sum_{v=1}^{k} d(m_{s,v}, m_{i,v}), \quad \text{where } d(m_{s,v}, m_{i,v}) = \begin{cases} 1 & \text{if } m_{s,v} \neq m_{i,v}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

An example is shown in (3.4); the number of symbols different between rows r_1 and c_1 is 4 and between r_2 and c_1 is 3 summing up 7. Then, the Hamming distance for the candidate row c_1 is 7:

$$\text{Rows} \begin{cases} r_1 = \{2\ 1\ 0\ 1\} \\ r_2 = \{1\ 2\ 0\ 1\} \\ c_1 = \{0\ 2\ 1\ 0\} \end{cases}, \qquad \text{Distances} \begin{cases} d(r_1, c_1) = 4 \\ d(r_2, c_1) = 3 \\ g(c_1) = 7 \end{cases}, \tag{3.4}$$

Equation (3.4) is an example of the Hamming distance between two rows r_1, r_2 that are already in the matrix M and a candidate row c_1.

3.3. Evaluations Function

The *evaluation function* $\mathcal{C}(M)$ is used to estimate the goodness of a candidate solution. Previously reported metaheuristic algorithms for constructing covering arrays have commonly evaluated the quality of a potential solution (covering array) as the number of combination of symbols missing in the matrix M [20, 22, 23]. Then, the expected solution will be zero missing. In the proposed simulated annealing implementation, this evaluation function definition was used. Its computational complexity is equivalent to $O(N\binom{k}{t})$.

3.4. Neighborhood Function

Given that ESSA is based on local search (LS), then a neighborhood function must be defined. The main objective of the neighborhood function is to identify the set of potential solutions which can be reached from the current solution in a local search (LS) algorithm. In case two or more neighborhoods exhibit complementary characteristics, it is then possible and interesting to create more powerful compound neighborhoods. The advantage of such an approach is well documented in [26]. Following this idea, and based on the results of our preliminary experimentations, a neighborhood structure composed by two different functions is proposed for ESSA.

The neighborhood function $\mathcal{N}_1(s)$ makes a random search of a missing t-tuple and then tries by setting the jth combination of symbols in every row of M. The neighborhood function $\mathcal{N}_2(s)$ randomly chooses a position (i, j) of the matrix M and makes all possible changes of symbol. During the search process, a combination of both $\mathcal{N}_1(s)$ and $\mathcal{N}_2(s)$ neighborhood functions is employed by ESSA algorithm. The former is applied with probability $\mathbb{P}$, while the latter is employed at a $(1 - \mathbb{P})$ rate. This combined neighborhood function $\mathcal{N}_3(s, x)$ is defined in (3.5), where x is a random number in the interval $[0, 1]$:

$$\mathcal{N}_3(s, x) = \begin{cases} \mathcal{N}_1(s) & \text{if } x \leq \mathbb{P}, \\ \mathcal{N}_2(s) & \text{if } x > \mathbb{P}. \end{cases} \tag{3.5}$$

3.5. Cooling Schedule

The *cooling schedule* determines the degree of uphill movement permitted during the search and is thus critical to the ESSA algorithm's performance. The parameters that define a cooling schedule are an initial temperature, a final temperature or a stopping criterion, the maximum number of neighboring solutions that can be generated at each temperature, and a rule for decrementing the temperature. The literature offers a number of different cooling schedules, see, for instance, [4, 27]. ESSA uses a geometrical cooling scheme mainly for its simplicity. It starts at an initial temperature T_i which is decremented at each round by a factor α using the relation $T_k = \alpha T_{k-1}$. For each temperature, the maximum number of visited neighboring solutions is L. It depends directly on the parameters (N, k, and v) of the studied covering array. This is because more moves are required for covering arrays with alphabets of greater cardinality.

```
(1) INITIALIZE(M, T, L)    /* Create the initial solution. */
(2) M* ← M;                /* Memorize the best solution. */
(3) T = T0;                /* Initial temperature of SA. */
(4) repeat
(5)   for i ← 1 to L do
(6)     Mi ← GENERATE (M);            /* Perturb current state. */
(7)     ΔC ← C(Mi) − C(M);            /* Evaluate cost function. */
(8)     x ← random;                   /*x in the range [0,1). */
(9)     if ΔC < 0 or e^(−ΔC/T) > x then
(10)      M ← Mi;                     /* Accept new state.*/
(11)      if C(M) < C(M*) then
(12)        M* ← M;                   /* Memorize the best solution. */
(13)      end if
(14)    end if
(15)  end for
(16)  CALCULATE_CONTROL(T, ϕ)
(17) until termination condition is satisfied;
```

Algorithm 1: Sequential simulated annealing for the CAC problem.

3.6. Termination Condition

The *stop criterion* for ESSA is either when the current temperature reaches T_f, when it ceases to make progress, or when a valid covering array is found. In the proposed implementation, a lack of progress exists if after ϕ (frozen factor) consecutive temperature decrements the best-so-far solution is not improved.

3.7. Simulated Annealing Pseudocode

Algorithm 1 presents the simulated annealing heuristic as described above. The meaning of the three functions is obvious: INITIALIZE computes a start solution and initial values of the parameters T and L; GENERATE selects a solution from the neighborhood of the current solution, using the neighborhood function $\mathcal{N}_3(s, x)$; CALCULATE_CONTROL computes a new value for the parameter T (cooling schedule) and the number of consecutive temperature decrements with no improvement in the solution.

This is unlike the classical method which takes as a solution to the problem the last value obtained in the annealing chain [28]. We memorize the best solution found during the whole annealing process (see lines 2 and 12).

In the next section, it is presented three parallel simulated annealing approaches for solving the CAC problem.

4. Parallel Simulated Annealing

Parallelization is recognized like a powerful strategy to increase algorithms efficiency; however, simulated annealing parallelization is a hard task because it is essentially a sequential process.

In evaluating performance of a parallel simulated annealing (PSA), it needs to consider solution quality as well as execution speed. The execution speed may be quantified in terms of *speed-up* ($\mathcal{S}$) and *efficiency* ($\mathcal{E}$). The $\mathcal{S}$ is defined as the ratio of the execution time (on one processor) by the sequential simulated annealing to that by the PSA (on P processors) for an equivalent solution quality. In the ideal case, $\mathcal{S}$ would be equal to P. The $\mathcal{E}$ is defined as the ratio of the actual $\mathcal{S}$ to the ideal $\mathcal{S}(P)$.

Next, we propose three parallel implementations of the simulated annealing algorithm described in Section 3. For these cases, let P denote the number of processors and L the length of Markov chain.

4.1. Independent Search Approach

A common approach to parallelizing simulated annealing is the ISA [4, 6, 29]. In this approach, each processor independently perturbs the configuration, evaluates the cost, and decides on the perturbation. The processors P_i, $i = 0, 1, \ldots, P-1$ carry out the independent annealing searches using the same initial solution and cooling schedule as in the sequential algorithm. At each temperature, P_i executes $N \times k \times v^2$ annealing steps. When each processor finishes, it sends its results to processor P_0. Finally, processor P_0 chooses the final solution among the local solutions.

We have implemented a simulated annealing algorithm using ISA approach for constructing covering arrays. In the developed implementation, the processors do not interact during individual annealing processes until all processors find their final solution. Then, the best of the solutions is saved and the others are discarded.

4.2. Semi-Independent Search Approach

Aarts and van Laarhoven [4] introduced a new parallel simulated annealing algorithm named *division algorithm*. In the division algorithm, the number of iterations at each temperature is divided equally between the processors. After a change in temperature, each processor may simply start from the final solution obtained by that processor at the previous temperature. The best solution from all the processors is then taken to be the final solution. Another variant of this approach is to communicate the best solution from all the processors to each processor every time the temperature changes. Aarts and van Laarhoven [4] found no significant differences in the performance of these two variants.

We have developed an implementation of division algorithm; we named the implementation SSA. In SSA, parallelism is obtained by dividing the effort of generation a Markov chain over the available processors. A Markov chain is divided into P subchains of the length $\lfloor L/P \rfloor$. In this approach, the processors exchange local information including intermediate solutions and their costs. Then, each processor restarts from the best intermediate ones.

Compared to the ISA, communication overhead in this SSA approach would be increased. However, each processor can utilize the information from other processors such that the decrease in computations and idle times can be greater than the increase in communication overhead. For instance, a certain processor which is trapped in an inferior solution can recognize its state by comparing it with others and may accelerate the annealing procedure. That is, processors may collectively converge to a better solution.

4.3. Cooperative Search Approach

In order to improve the performance of the SSA approach, we propose the cooperative search approach (CSA); it used asynchronous communication among processors accessing the global state to eliminate the idle times. Each processor follows a separate search path, accesses the global state which consists of the current best solution and its cost whenever it finished a Markov subchain, and updates the state if necessary. Once a processor gets the global state, it proceeds to the next Markov subchain with any delay.

Unlike SSA, CSA has the following characteristics:

(i) idle times can be reduced since asynchronous communications overlap a part of the computation;

(ii) less communication overhead, an isolated access to the global state is needed by each processor at the end of each Markov subchain;

(iii) the probability of being trapped in a local optimum can be smaller. This is because not all the processors start from the same state in each Markov subchain.

5. Experimental Results

This section presents an experimental design and results derived from testing the parallel ISA, SSA, and CSA algorithms described in the Section 4. In order to show the performance of these approaches, three experiments were developed. The first experiment had as purpose to fine-tune the probabilities of the neighborhood functions to be selected. The second experiment had as purpose to compare the three approaches in terms of parallel execution time. Among the three approaches, the CSA approach was the fastest. The third experiment evaluated the quality of the solutions of the CSA approach over a new benchmark proposed in this paper. The results were compared against the best-known solutions reported in the literature to construct covering arrays [30].

The three parallel approaches were implemented using C language and the message passing interface (MPI) library. The implementations were run on the *Tirant supercomputer* (The Tirant supercomputer: http://www.uv.es/siuv/cas/zcalculo/res/informa.wiki). Tirant comprises 256 JS20 compute nodes (blades) and 5 p515 servers. Every blade has two IBM Power4 processors at 2.0 GHz running Linux operating system with 4 GB of memory RAM and 36 GB of local disk storage. All the servers provide a total of nearly 10 TB of disk storage accessible from every blade through GPFS (global parallel file system). Tirant has in total 512 processors, 1 TB of memory RAM, and 19 TB of disk storage. The following parameters were used for all simulated annealing implementations:

(1) Initial temperature $T_i = 4.0$,

(2) Final temperature $T_f = 1.0E - 10$,

(3) Cooling factor $\alpha = 0.99$,

(4) Maximum neighboring solutions per temperature $L = N \times k \times v^2$,

(5) Frozen factor $\phi = 11$;

(6) The neighborhood function $\mathcal{N}_3(s, x)$ is applied using a probability $\mathbb{P} = 0.3$.

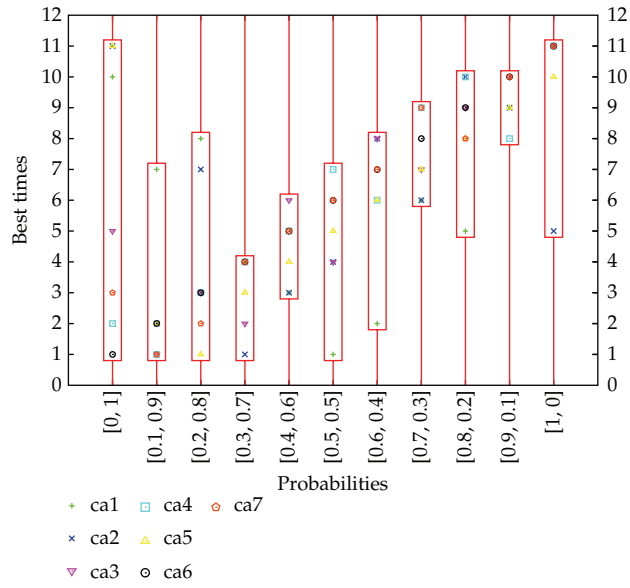

Figure 1: Performance of ESSA with the 11 combinations of probabilities.

5.1. Tuning of the Parameters of Parallel Simulated Annealing

It is well known that the performance of a simulated annealing algorithm is sensitive to parameter tuning. In this sense, we follow a methodology for a fine-tuning of the two neighborhood functions used in ESSA. The fine-tuning was based on the linear Diophantine equation (5.1), where x_i represents a neighborhood function, and its value set to 1, $\mathbb{P}_i$ is a value in $\{0.0, 0.1, \ldots, 1.0\}$ that represents the probability of executing x_i, and q is set to 1.0 which is the maximum probability of executing any x_i:

$$\mathbb{P}_1 x_1 + \mathbb{P}_2 x_2 = q. \tag{5.1}$$

A solution to the given linear Diophantine equation must satisfy (5.2). This equation has 11 solutions, each solution is an experiment that tests the degree of participation of each neighborhood function in ESSA to accomplish the construction of a covering array. Every combination of the probabilities was applied by ESSA to construct the set of covering arrays shows in Table I(a), and each experiment was run 31 times; with the data obtained for each experiment, we calculate the median. A summary of the performance of ESSA with the probabilities that solved the 100% of the runs is shown in Table I(b):

$$\sum_{i=1}^{2} \mathbb{P}_i x_i = 1.0 \tag{5.2}$$

Finally, given the results shown in Figure 1, the best configuration of probabilities was $\mathbb{P}_1 = 0.3$ and $\mathbb{P}_2 = 0.7$ because it found the covering arrays in smaller time (median value). The values $\mathbb{P}_1 = 0.3$ and $\mathbb{P}_2 = 0.7$ were kept fixed in the second experiment.

Table 1: (a) A set of 7 covering arrays configurations. (b) Performance of ESSA. Columns 3–9 show the time taken to construct each of the covering arrays.

(a)

Id	CA description
ca_1	CA(19; 2, 30, 3)
ca_2	CA(35; 3, 5, 3)
ca_3	CA(58; 3, 10, 3)
ca_4	CA(86; 4, 5, 3)
ca_5	CA(204; 4, 10, 3)
ca_6	CA(243; 5, 5, 3)
ca_7	CA(1040; 5, 15, 3)

(b)

$\mathbb{P}_1$	$\mathbb{P}_2$	Covering arrays						
		ca_1	ca_2	ca_3	ca_4	ca_5	ca_6	ca_7
0	1	4789.763	3.072	46.989	12.544	3700.038	167.901	0.102
0.1	0.9	1024.635	0.098	0.299	0.236	344.341	3.583	0.008
0.2	0.8	182.479	0.254	0.184	0.241	173.752	1.904	0.016
0.3	0.7	224.786	0.137	0.119	0.222	42.950	1.713	0.020
0.4	0.6	563.857	0.177	0.123	0.186	92.616	3.351	0.020
0.5	0.5	378.399	0.115	0.233	0.260	40.443	1.258	0.035
0.6	0.4	272.056	0.153	0.136	0.178	69.311	2.524	0.033
0.7	0.3	651.585	0.124	0.188	0.238	94.553	2.127	0.033
0.8	0.2	103.399	0.156	0.267	0.314	81.611	5.469	0.042
0.9	0.1	131.483	0.274	0.353	0.549	76.379	4.967	0.110
1	0	7623.546	15.905	18.285	23.927	1507.369	289.104	2.297

5.2. Comparison of the ISA, SSA, and CSA Approaches

To test the performance of the ISA, SSA, and CSA approaches, we propose the construction of a covering array with $N = 80$, $t = 3$, $k = 22$, and $v = 3$. Each approach was executed 31 times (for providing statistical validity to experiment) using $P = \{4, 8, 16, 32\}$.

The performance of the algorithms has been compared based on median speedup as a function of the number of processors; the results are shown in Figure 2.

The ISA approach; had difficulty in handling the large problem instances, it does not scale well. The SSA approach provides reasonable results; however, because it is a synchronous algorithm, the idle and communication times are inevitable. The CSA approach is the one which offers the best results; it reduces the execution time of the SSA approach by employing asynchronous information exchange.

In the next subsection, it is presented the third experiment of this work, and the purpose is to measure the performance of the CSA algorithm against the best-known solutions reported in the literature.

5.3. Comparison with State-of-the-Art Procedures

The purpose of this experiment is to carry out a performance comparison of the bounds achieved by the CSA approach with respect to the best-known solutions reported in

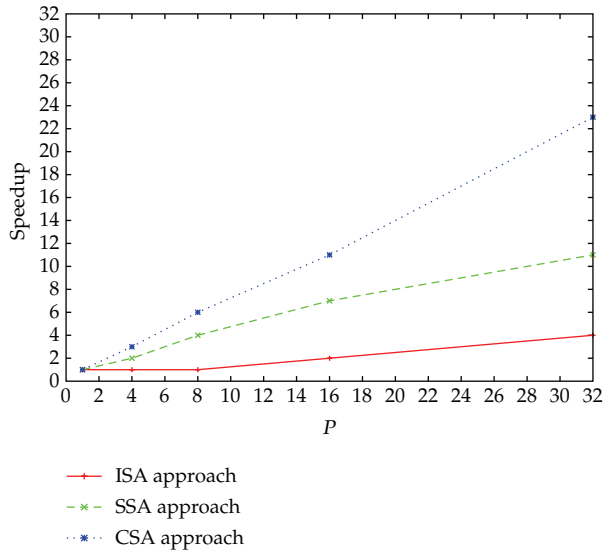

Figure 2: Median speedup for the ISA, SSA, and CSA approaches.

the literature [30], which were produced using the following state-of-the-art procedures: orthogonal array construction, Roux type constructions, doubling constructions, algebraic constructions, deterministic density algorithm (DDA), Tabu search, and IPOG-F.

For this experiment, we have fixed the maximum computational time expended by our PSA for constructing a covering array to 72 hours and 50 processors. We create a new benchmark composed of 182 covering arrays distributed as follows:

(i) 47 covering arrays with strength $t = 3$, degree $4 \leq k < 50$, and order $v = 3$,

(ii) 46 covering arrays with strength $t = 4$, degree $5 \leq k < 50$, and order $v = 3$,

(iii) 45 covering arrays with strength $t = 5$, degree $6 \leq k < 50$, and order $v = 3$,

(iv) 44 covering arrays with strength $t = 6$, degree $7 \leq k < 50$, and order $v = 3$,

The detailed results produced by this experiment are listed in Table 2. The first two columns in each subtable indicate the strength t and the degree k of the selected instances. Next two columns show, in terms of the size N of the covering arrays, the best-known solution reported in the literature and the improved bounds produced by the CSA approach. Last column depicts the difference between the best result produced by CSA approach and the best-known solution ($\Delta = \beta - \vartheta$).

Figure 3 compares the results shown in Table 2 involving the CSA algorithm and the best-known solutions. The analysis of the data presented led us to the following observation. The solutions quality attained by the CSA approach is very competitive with respect to that produced by the state-of-the-art procedures summarized in column 3 (ϑ). In fact, it is able to improve on *134* previous best-known solutions.

6. Conclusions

The long execution time of simulated annealing due to its sequential nature hinders its application to realistically sized problems, in this case, the CAC problem when $t > 3$,

Table 2: It shows the improved bounds produced by CSA approach. Column ϑ represents the best-known solution reported in the literature [30]. Column β represents the best solution in terms of N produced by CSA approach. Last column (Δ) depicts the difference between the best result produced by CSA approach and the best-known solution ($\Delta = \beta - \vartheta$). (a) Improved bounds on CAN$(3, k, 3)$; (b) improved bounds on CAN$(4, k, 3)$; (c) improved bounds on CAN$(5, k, 3)$; (d) improved bounds on CAN$(6, k, 3)$.

(a)

t	k	ϑ	β	Δ
3	4	27	27	0
3	5	33	33	0
3	6	33	33	0
3	7	40	**39**	−1
3	8	42	42	0
3	9	45	45	0
3	10	45	45	0
3	11	45	45	0
3	12	45	45	0
3	13	50	**49**	−1
3	14	51	**50**	−1
3	15	57	57	0
3	16	60	**59**	−1
3	17	60	**59**	−1
3	18	60	**59**	−1
3	19	60	**59**	−1
3	20	60	**59**	−1
3	21	66	67	1
3	22	66	71	5
3	23	69	71	2
3	24	72	**71**	−1
3	25	75	**72**	−3
3	26	78	**72**	−6
3	27	81	**79**	−2
3	28	81	**79**	−2
3	29	87	**84**	−3
3	30	87	**84**	−3
3	31	90	**88**	−2
3	32	90	**89**	−1
3	33	90	**89**	−1
3	34	90	**89**	−1
3	35	90	**89**	−1
3	36	90	**89**	−1
3	37	90	**89**	−1
3	38	90	**89**	−1
3	39	90	**89**	−1
3	40	90	**89**	−1
3	41	98	**94**	−4
3	42	98	**94**	−4
3	43	100	**99**	−1
3	44	100	**99**	−1
3	45	103	**99**	−4
3	46	103	**101**	−2
3	47	106	**101**	−5
3	48	106	**101**	−5
3	49	108	**101**	−7
3	50	108	**102**	−6

(b)

t	k	ϑ	β	Δ
4	5	81	81	0
4	6	111	111	0
4	7	123	123	0
4	8	141	**135**	−6
4	9	159	**135**	−24
4	10	159	164	5
4	11	183	183	0
4	12	201	201	0
4	13	219	219	0
4	14	237	249	12
4	15	237	277	40
4	16	237	277	40
4	17	300	**287**	−13
4	18	307	**300**	−7
4	19	313	313	0
4	20	315	321	6
4	21	315	338	23
4	22	315	347	32
4	23	315	359	44
4	24	377	**375**	−2
4	25	384	**375**	−9
4	26	393	**387**	−6
4	27	393	**387**	−6
4	28	393	**392**	−1
4	29	393	406	13
4	30	393	401	8
4	31	440	**424**	−16
4	32	445	**431**	−14
4	33	454	**438**	−16
4	34	462	**447**	−15
4	35	471	**440**	−31
4	36	471	**456**	−15
4	37	471	**460**	−11
4	38	471	**465**	−6
4	39	471	**468**	−3
4	40	499	**472**	−27
4	41	506	**484**	−22
4	42	509	**488**	−21
4	43	518	**494**	−24
4	44	522	**497**	−25
4	45	526	**497**	−29
4	46	530	**506**	−24
4	47	534	**510**	−24
4	48	542	**516**	−26
4	49	549	**523**	−26
4	50	549	**525**	−24

(c)

t	k	ϑ	β	Δ
5	6	243	243	0
5	7	351	351	0
5	8	405	405	0
5	9	483	**405**	−78
5	10	483	**405**	−78
5	11	705	**550**	−155
5	12	723	**600**	−123
5	13	723	828	105
5	14	922	**890**	−32
5	15	963	**944**	−19
5	16	963	1025	62
5	17	1117	1117	0
5	18	1167	**1165**	−2
5	19	1197	**1190**	−7
5	20	1266	**1257**	−9
5	21	1317	**1312**	−5
5	22	1346	**1319**	−27
5	23	1405	**1387**	−18
5	24	1447	**1420**	−27
5	25	1486	**1440**	−46
5	26	1521	**1493**	−28
5	27	1538	**1527**	−11
5	28	1579	**1555**	−24
5	29	1615	**1585**	−30
5	30	1647	**1616**	−31
5	31	1681	**1643**	−38
5	32	1724	**1671**	−53
5	33	1783	**1702**	−81
5	34	1783	**1724**	−59
5	35	1851	**1748**	−103
5	36	1882	**1778**	−104
5	37	1909	**1800**	−109
5	38	1937	**1829**	−108
5	39	1960	**1851**	−109
5	40	1986	**1866**	−120
5	41	2023	**1896**	−127
5	42	2046	**1923**	−123
5	43	2069	**1940**	−129
5	44	2091	**2089**	−2
5	45	2112	**2111**	−1
5	46	2130	**2129**	−1
5	47	2150	**2149**	−1
5	48	2174	**2168**	−6
5	49	2191	**2189**	−2
5	50	2213	**2211**	−2

(d)

t	k	ϑ	β	Δ
6	7	729	729	0
6	8	1152	1152	0
6	9	1431	1600	169
6	10	1449	1849	400
6	11	1449	2136	687
6	12	2181	2482	301
6	13	2734	2744	10
6	14	2907	3220	313
6	15	3234	3338	104
6	16	3443	3672	229
6	17	3658	3882	224
6	18	3846	4098	252
6	19	4054	4256	202
6	20	4486	**4400**	−86
6	21	4678	**4600**	−78
6	22	4853	**4732**	−121
6	23	4942	**4941**	−1
6	24	5193	**5100**	−93
6	25	5257	**5238**	−19
6	26	5709	**5380**	−329
6	27	5853	**5810**	−43
6	28	6003	**5965**	−38
6	29	6150	**6110**	−40
6	30	6281	**6250**	−31
6	31	6413	**6393**	−20
6	32	6535	**6518**	−17
6	33	6656	**6642**	−14
6	34	6772	**6760**	−12
6	35	6877	**6871**	−6
6	36	6989	**6978**	−11
6	37	7092	**7086**	−6
6	38	7194	**7187**	−7
6	39	7293	**7284**	−9
6	40	7391	**7385**	−6
6	41	7490	**7478**	−12
6	42	7574	**7569**	−5
6	43	7672	**7661**	−11
6	44	7757	**7748**	−9
6	45	7845	**7836**	−9
6	46	7938	**7928**	−10
6	47	8013	**8005**	−8
6	48	8092	**8089**	−3
6	49	8179	**8176**	−3
6	50	8256	**8253**	−3

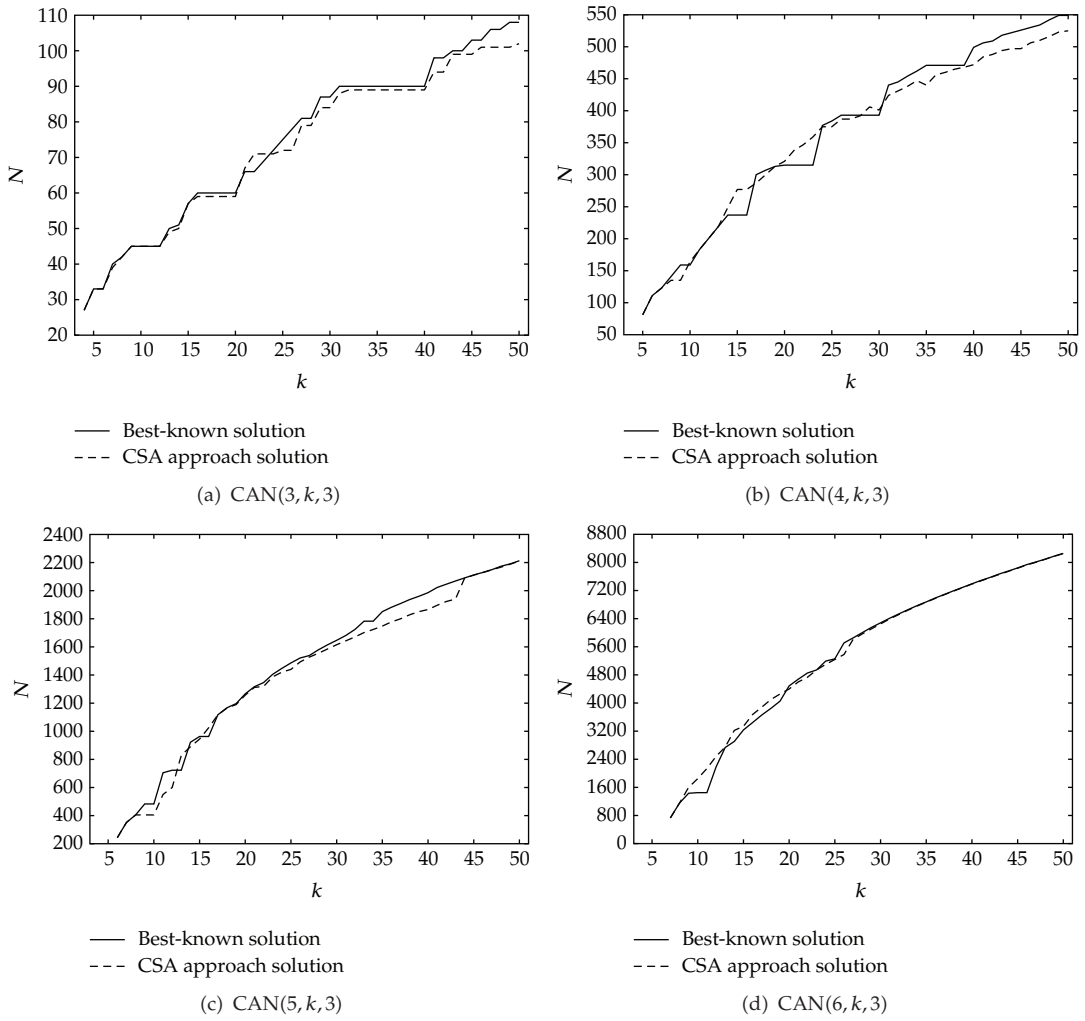

Figure 3: Graphical comparison of the quality solutions between CSA and the state of the art [30].

$5 < k \leq 100$, and $v = 3$. A more efficient way to reduce execution time and make the simulated annealing a more promising method is to parallelize sequential simulated annealing. It is a challenging task. In fact, there are many approaches that may be considered in parallelizing simulated annealing. However, an inappropriate strategy used will likely result in poor performance.

In this paper, we have used three different approaches to do the parallelization of the simulated annealing algorithm. From the experimental results, we found that the ISA approach has the worst performance, and it does not scale well. The SSA approach offers reasonable execution times; compared to the ISA, communication overhead in the SSA approach would be increased when the size of the problem grows. However, each processor can utilize the information from other processors such that the decrease in computations and idle times can be greater than the increase in communication overhead. For instance, a certain processor which is trapped in an inferior solution can recognize its state by comparing it

with others and may accelerate the annealing procedure. That is, processors may collectively converge to a better solution.

The CSA approach is the one which offers the best results; it significantly reduces the execution time of the SSA approach by employing asynchronous information exchange. The quality solutions attained by the CSA approach are very competitive with respect to that produced by the state-of-the-art procedures; in fact, it is able to improve on *134* previous best-known solutions and equals the solutions for other 29 instances.

These experimental results confirm the practical advantages of using CSA algorithm in the software testing area. It is a robust algorithm yielding smaller test suites than other representative state-of-the-art algorithms, which allows reducing software testing costs.

The new bounds are available in Cinvestav covering array repository (CAR), which is available under request at http://www.tamps.cinvestav.mx/~jtj/authentication.php. We have verified all covering arrays described in this paper using the tool described in [31].

Acknowledgments

The authors thankfully acknowledge the computer resources and assistance provided by Spanish Supercomputing Network (TIRANT-UV). This research work was partially funded by the following projects: CONACyT 58554; Cálculo de Covering Arrays; 51623-Fondo Mixto CONACyT; Gobierno del Estado de Tamaulipas.

References

[1] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton, "The combinatorial design approach to automatic test generation," *IEEE Software*, vol. 13, no. 5, pp. 83–88, 1996.

[2] K. Z. Zamli, M. F. J. Klaib, M. I. Younis, N. A. M. Isa, and R. Abdullah, "Design and implementation of a t-way test data generation strategy with automated execution tool support," *Information Sciences*, vol. 181, no. 9, pp. 1741–1758, 2011.

[3] A. Hartman, "Software and hardware testing using combinatorial covering suites," in *Graph Theory, Combinatorics and Algorithms*, M. C. Golumbic and I. B. A. Hartman, Eds., vol. 34 of *Operations Research/Computer Science Interfaces*, pp. 237–266, Springer, New York, NY, USA, 2005.

[4] E. H. L. Aarts and P. J. M. van Laarhoven, "Statistical cooling: a general approach to combinatorial optimization problems," *Philips Journal of Research*, vol. 40, no. 4, pp. 193–226, 1985.

[5] D. Janaki Ram, T. H. Sreenivas, and K. G. Subramaniam, "Parallel simulated annealing algorithms," *Journal of Parallel and Distributed Computing*, vol. 37, no. 2, pp. 207–212, 1996.

[6] S. Y. Lee and K. G. Lee, "Synchronous and asynchronous parallel simulated annealing with multiple Markov chains," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 10, pp. 993–1008, 1996.

[7] D. J. Chen, C. Y. Lee, C. H. Park, and P. Mendes, "Parallelizing simulated annealing algorithms based on high-performance computer," *Journal of Global Optimization*, vol. 39, no. 2, pp. 261–289, 2007.

[8] Z. J. Czech, W. Mikanik, and R. Skinderowicz, "Implementing a parallel simulated annealing algorithm," in *Proceedings of the 8th International Conference on Parallel Processing and Applied Mathematics (PPAM'10)*, vol. 6067 of *Lecture Notes in Computer Science*, pp. 146–155, Springer, Berlin, Germany, 2010.

[9] M. Chateauneuf and D. L. Kreher, "On the state of strength-three covering arrays," *Journal of Combinatorial Designs*, vol. 10, no. 4, pp. 217–238, 2002.

[10] K. Meagher and B. Stevens, "Group construction of covering arrays," *Journal of Combinatorial Designs*, vol. 13, no. 1, pp. 70–77, 2005.

[11] J. R. Lobb, C. J. Colbourn, P. Danziger, B. Stevens, and J. Torres-Jimenez, "Cover starters for covering arrays of strength two," *Discrete Mathematics*, vol. 312, no. 5, pp. 943–956, 2012.

[12] D. T. Tang and L. S. Woo, "Exhaustive test pattern generation with constant weight vectors," *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1145–1150, 1983.

[13] J. Martinez-Pena, J. Torres-Jimenez, N. Rangel-Valdez, and H. Avila-George, "A heuristic approach

for constructing ternary covering arrays using trinomial coefficients," in *Proceedings of the 12th Ibero-American Conference on Artificial Intel-Ligence—IBERAMIA*, vol. 6433 of *Lecture Notes in Computer Science*, pp. 572–581, Springer, Berlin, Germany, 2010.

[14] C. J. Colbourn, S. S. Martirosyan, G. L. Mullen, D. Shasha, G. B. Sherwood, and J. L. Yucas, "Products of mixed covering arrays of strength two," *Journal of Combinatorial Designs*, vol. 14, no. 2, pp. 124–138, 2006.

[15] Y. W. Tung and W. S. Aldiwan, "Automating test case generation for the new generation mission software system," in *Proceedings of the IEEE Aerospace Conference*, vol. 1, pp. 431–437, IEEE Press, March 2000.

[16] R. C. Bryce and C. J. Colbourn, "The density algorithm for pairwise interaction testing," *Software Testing Verification and Reliability*, vol. 17, no. 3, pp. 159–182, 2007.

[17] Y. Lei and K. C. Tai, "In-parameter-order: a test generation strategy for pair-wise testing," in *Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering (HASE'98)*, pp. 254–261, IEEE Computer Society, 1998.

[18] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: a general strategy for T-way software testing," in *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, pp. 549–556, IEEE Computer Society, Tucson, Ariz, USA, March 2007.

[19] A. H. Ronneseth and C. J. Colbourn, "Merging covering arrays and compressing multiple sequence alignments," *Discrete Applied Mathematics*, vol. 157, no. 9, pp. 2177–2190, 2009.

[20] K. J. Nurmela, "Upper bounds for covering arrays by tabu search," *Discrete Applied Mathematics*, vol. 138, no. 1-2, pp. 143–152, 2004.

[21] L. Gonzalez-Hernandez, N. Rangel-Valdez, and J. Torres-Jimenez, "Construction of mixed covering arrays of variable strength using a tabu search approach," in *Proceedings of the 4th International Conference on Combinatorial Optimization and Applications (COCOA'10)*, vol. 6508 of *Lecture Notes in Computer Science*, pp. 51–64, Springer, Berlin, Germany, 2010.

[22] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling, "Augmenting simulated annealing to build interaction test suites," in *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE'03)*, pp. 394–405, IEEE Computer Society, 2003.

[23] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, vol. 1, pp. 72–77, IEEE Computer Society, September 2004.

[24] J. Torres-Jimenez and E. Rodriguez-Tello, "New bounds for binary covering arrays using simulated annealing," *Information Sciences*, vol. 185, no. 1, pp. 137–152, 2012.

[25] Y. Jun and S. Mizuta, "Detailed analysis of uphill moves in temperature parallel simulated annealing and enhancement of exchange probabilities," *Complex Systems*, vol. 15, no. 4, pp. 349–358, 2005.

[26] L. Cavique, C. Rego, and I. Themido, "Subgraph ejection chains and tabu search for the crew scheduling problem," *Journal of the Operational Research Society*, vol. 50, no. 6, pp. 608–616, 1999.

[27] M. M. Atiqullah, "An efficient simple cooling schedule for simulated annealing," in *Proceedings of the International Conference on Computational Science and its Applications (ICCSA'04)*, vol. 3045 of *Lecture Notes in Computer Science*, pp. 396–404, Springer, Berlin, Germany, 2004.

[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[29] Z. Czech, "Three parallel algorithms for simulated annealing," in *Parallel Processing and Applied Mathematics*, vol. 2328 of *Lecture Notes in Computer Science*, pp. 210–217, Springer, Berlin, Germany, 2006.

[30] C. J. Colbourn, "Covering array tables for t=2, 3, 4, 5, 6," 2012, http://www.public.asu.edu/~ccolbou/src/tabby/catable.html.

[31] H. Avila-George, J. Torres-Jimenez, N. Rangel-Valdez, A. Carrión, and V. Hernández, "Supercomputing and a Grid computing on the verification of coveringarrays," *The Journal of Supercomputing*. In press.