A Genetic Algorithm for Cocyclic Hadamard Matrices*

V. Álvarez, J.A. Armario, M.D. Frau, and P. Real

Dpto. Matemática Aplicada I, Universidad de Sevilla,
Avda. Reina Mercedes s/n 41012 Sevilla, Spain
{valvarez, armario, mdfrau, real}@us.es

Abstract. A genetic algorithm for finding cocyclic Hadamard matrices is described. Though we focus on the case of dihedral groups, the algorithm may be easily extended to cover any group. Some executions and examples are also included, with aid of MATHEMATICA 4.0.

1 Introduction

Over the past decade considerable effort has been devoted to computations of cocycles and (Hadamard) cocyclic matrices. On one hand, using classical methods involving the Universal Coefficient Theorem, Schur multipliers, inflation and transgression, two algorithms for finding 2-cocycles representing 2-dimensional cohomology classes and their correspondent cocyclic matrices have been worked out. The first one constitutes the foundational work on the subject [6, 7], and is applied over abelian groups. The second one [8] is applied over groups G for which the word problem is solvable.

On the other hand, Homological Perturbation Theory [12, 13, 16] provides computational short-cuts in a straightforward manner, by means of the so-called *(co)homological models*. This technique has been exploited in [11] from a cohomological point of view and more recently in [1, 2, 3] from a homological point of view.

From past literature it is evident that the search of Hadamard (cocyclic) matrices inherits high computational difficulty. In fact, though the use of the cocyclic construction of Hadamard matrices has permitted cut-downs in the search time, the search space still grows exponentially.

The work in [4] attempts to make an adaptation of image-processing techniques for the restoration of damaged images for the purpose of sampling the search space systematically. As a matter of fact, this approximation reveals to work whenever enough Hadamard cocyclic matrices are already known, from which the performance is then feasible.

Our aim is to provide a tool for generating Hadamard cocyclic matrices in an easy and (hopefully) fast way, which will complement in turn the work in [4]. Here

* All authors are partially supported by the PAICYT research project FQM–296 from Junta de Andalucía (Spain).

M. Fossorier et al. (Eds.): AAECC 2006, LNCS 3857, pp. 144–153, 2006.
© Springer-Verlag Berlin Heidelberg 2006

we look at an adaptation of classical Genetic Algorithms [14, 15] for this purpose. Though it actually works on any group, we will focus on an improved version running on dihedral groups. Both of the genetic algorithms and all the executions and examples of the last section have been worked out with aid of MATHEMATICA 4.0, running on a *Pentium IV 2.400 Mhz DIMM DDR266 512 MB*. It is a remarkable fact that, as far as we know, none of the algorithms already known has produced some Hadamard cocyclic matrices of large order (say $4t \geq 40$). The examples in Section 6 include some Hadamard cocyclic matrices of order 52. This way, our method seems to provide some cocyclic Hadamard matrices of larger order than those previously obtained with other algorithms.

We organize the paper as follows. Section 2 summarizes the classical methods already known for finding (Hadamard) cocyclic matrices. Section 3 is a brief introduction to Genetic Algorithms. The genetic algorithm itself for finding Hadamard cocyclic matrices is described in Section 4. The following section includes an improved version of the algorithm for the case of dihedral groups. As a matter of fact, both the search time and the search space are substantially optimized thanks to the work of the authors in [2, 3]. The last section is devoted to some examples.

2 Generating Cocyclic Matrices

The foundational work on cocyclic matrices is [6, 7], where a basis for 2-cocycles is codified in terms of a *development table*. Horadam and de Launey's method is based on an explicit version of the well-known Universal Coefficient Theorem, which provides a decomposition of the second cohomology group of G into the direct sum of two summands,

$$H^2(G, C) \cong Ext(G/[G, G], C) \oplus Hom(H_2(G), C).$$

The $Ext(G/[G, G], C)$ factor is referred as the *symmetric part*, and is completely determined from a presentation of G and the primary invariant decomposition of the abelianization $G/[G, G]$. The $Hom(H_2(G), C)$ factor is referred as the *commutator part* and turns out to be the difficult one to compute. The case of abelian groups is described in [6, 7]. Once a set of generators for both the symmetric and commutator parts is determined, it suffices to add a basis for 2-coboundaries of G, so that a basis for 2-cocycles is finally achieved.

Another method is described in [8], whenever the word problem is solvable in G. This method has already been implemented in [10], using the symbolic computational system MAGMA. Flannery calculates $H^2(G; C) \cong I \oplus T$ as the images of certain embeddings (called *inflation,* I, and *transgression*, T) which are complementary. Calculation of representative 2-cocycles associated to $Ext(G/[G, G], C)$ (inflation) is again canonical. However, calculation of a complement of the image by the embeddings of inflation I in $H^2(G, C)$ as the image of transgression is usually not canonical. As a matter of fact, it depends on the choice of a Schur complement of I in $H^2(G; C)$. If $|I|$ and $|T|$ are not coprime,

there will be more than one complement of I in $H^2(G;C)$. This is a potential source of difficulties in computation of representative 2-cocycles associated with elements of $Hom(H_2(G),C)$. The case of dihedral groups and central extensions is described in [8, 9, 10].

Using the proper techniques of Homological Perturbation Theory [12, 13, 16], Grabmeier and Lambe present in [11] alternate methods for calculating representative 2-cocycles for all finite p–groups. They compute $H^2(G;C)$ straightforwardly from a *cohomological model* K of G. That is, a structure K such that $H^2(G;C) \cong H^2(K;C)$ and the computation of $H^2(K;C)$ is much simpler than that of $H^2(G;C)$.

One more approximation to this question, the so-called *homological reduction method*, is developed in another work of the authors [1, 2, 3]. Here *homological models* K for G are determined instead of cohomological models, in the sense that $H_*(K) \cong H_*(G)$ and $H_*(K)$ is computed substantially more easily than $H_*(G)$. The method developed in these papers covers any iterated product of central extensions and semidirect product of groups, so that dihedral groups D_{4t} are included. The genetic algorithm to be described in Section 4 is performed upon the calculations that the homological reduction method provides when it is applied over D_{4t}.

3 Preliminaries in Genetic Algorithms

Genetic algorithms (more briefly, GAs in the sequel) are appropriate for searching through large spaces, where exhaustive methods cannot be employed.

The father of the original Genetic Algorithm was John Holland who invented it in the early 1970's [14]. We next include a brief introduction to the subject. The interested reader is referred to [15] for more extensive background on GAs.

The aim of GAs is to mimic the principle of evolution in order to find an optimum solution for solving a given optimization problem. More concretely, starting from an initial "population" of potential solutions to the problem (traditionally termed *chromosomes*), some transformations are applied (may be just to some individuals or even to the whole population), as images of the "mutation" and "crossover" mechanisms in natural evolution. Mutation consists in modifying a "gene" of a chromosome. Crossover interchanges the information of some genes of two chromosomes.

Only some of these individuals will move on to the next generation (the more fit individuals, according to the optimization problem, in terms of the measure of an "evaluation function"). Here "generation" is a synonymous of iteration. The mutation and crossover transformations are applied generation through generation, and individuals go on striving for survival. After some number of iterations, the evaluation function is expected to measure an optimum solution, which solves the given problem. Although no bounds are known on the number of iterations which are needed to produce the fittest individual, it is a remarkable fact that GAs usually converge to an optimum solution significantly faster than exhaustive methods do. Indeed, GAs need not to explore the whole space.

4 Finding Hadamard Cocyclic Matrices by Means of GAs

We now set the directives for a genetic algorithm looking for Hadamard cocyclic matrices over a group G. In the following section we improve the design of the algorithm in the particular case of the dihedral group D_{4t}.

Let G be a group and $\mathcal{B} = \{\delta_1, \ldots, \delta_c, \beta_1, \ldots, \beta_s, \gamma_1, \ldots, \gamma_t\}$ a basis for 2-cocycles (according to the Hadamard pointwise product $\bullet$ of matrices). Here δ_i denote 2-coboundaries, β_i denote representative symmetric 2-cocycles (coming from inflation, i.e. factor $Ext(G/[G,G], \mathbb{Z}_2)$) and γ_i denote representative not symmetric 2-cocycles (coming from transgression, i.e. factor $Hom(H_2(G), \mathbb{Z}_2)$). Notice that in these circumstances the whole space of 2-cocycles consists of 2^{c+s+t} elements, and precisely 2^c of them are 2-coboundaries. Moreover, every 2-cocycle f may be uniquely expressed as a binary $(c+s+t)$-tuple $(f_1, \ldots, f_{c+s+t})_{\mathcal{B}}$ such that

$$f = \delta_1^{f_1} \bullet \ldots \bullet \delta_c^{f_c} \bullet \beta_1^{f_{c+1}} \bullet \ldots \bullet \beta_s^{f_{c+s}} \bullet \gamma_1^{f_{c+s+1}} \bullet \ldots \bullet \gamma_t^{f_t}$$

A genetic algorithm for finding Hadamard cocyclic matrices may be designed as follows.

The population consists of the whole space of *normalized* cocyclic matrices over G, $M_f = (f(g_i, g_j))$, f being a 2-cocycle. The term "normalized" means that the first row is formed all by 1. Each of the individuals f of the population (i.e. potential solutions to the problem) is identified to a binary $(c+s+t)$-tuple $(f_1, \ldots, f_{c+s+t})_{\mathcal{B}}$, the coordinates of the 2-cocycle f with regards to the basis $\mathcal{B}$. This way, the coordinates f_k are the genes of the individual $f = (f_1, \ldots, f_{c+s+t})_{\mathcal{B}}$.

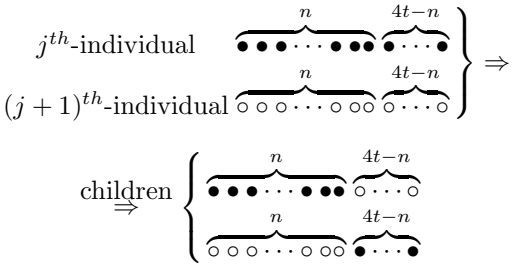
The initial population P_0 is formed by some binary $(c+s+t)$-tuples randomly generated. Assuming that $|G| = 4t$ (remember that only 2×2 Hadamard matrices exist whose sizes are not multiple of 4), we consider $4t$ individuals for instance. Special care must be taken in generating the population, so that the population space does not grow exponentially with the order of the group G.

The population is expected to evolve generation through generation until an optimum individual (i.e. a Hadamard cocyclic matrix) is located. We now describe how to form a new generation P_{i+1} from an old one P_i:

1. Firstly, we must evaluate the fitness of every individual (i.e. 2-cocycle f) of P_i. It is common-knowledge that a computationally cheap test exists [7] to check if f gives rise to a Hadamard cocyclic matrix M_f. Concretely, it suffices to check whether the sum of every row in M_f but the first is zero. Define a *Hadamard row* to be a row whose summation is zero.

   From the property above an evaluation function for individuals is derived immediately: the fitness of M_f grows with the number of its Hadamard rows. Thus, the more fit an individual is, the more Hadamard rows it possess, and vice versa. The optimum is reached when all the rows but the first (i.e. rows from 2 to $4t$) are Hadamard rows. That is, whenever M_f reveals to be a Hadamard cocyclic matrix itself.

2. Once the evaluation is finished, the crossover comes into play. All individuals are paired at random, so that crossover combines the features of two parent chromosomes to form two similar offspring by swapping corresponding segments of the parents. Each time, the break point n is chosen at random, so that two couples of different parents are swapped with possibly different break points.



3. Next we apply the mutation operator. Mutation arbitrarily alters just one gene of a selected individual (i.e. just one coordinate of the corresponding $(c+s+t)$-tuple, swapping 0 to 1 or 1 to 0, as it is the case), by a random change with a probability equal to the mutation rate (for instance, 1%).
4. Now individuals strive for survival: a selection scheme, biased towards fitter individuals (according to the number of their hadamard rows), selects the next generation. In case that an optimum individual exists, the algorithm stops. Otherwise the population P_{i+1} is constructed from a selection of $4t$ of the fittest individuals, in the following sense. Assume that n_k indicates the number of individuals in P_i which consists of exactly k Hadamard rows. Furthermore, assume that the fittest individuals in P_i consist of precisely r Hadamard rows (so that every individual in P_i possess at most r Hadamard rows, possibly less). The selection scheme firstly selects the n_r individuals with r Hadamard rows. If $n_r < 4t$, then all n_{r-1} individuals with exactly $r-1$ Hadamard rows are selected. And so on. This process continues until at least $4t$ individuals have been selected. Eventually, if the number of selected individuals exceeds from $4t$, some of the last individuals to be incorporated must be randomly deleted, in order to keep exactly $4t$ individuals in generation P_{i+1}.

The process goes on generation through generation until an optimum is reached. In spite of its simplicity, the method has surprisingly shown to work over several groups, though the number of required generations grows significantly with the size of the matrices. We next discuss the case of dihedral groups, where some significant improvements are introduced.

5 Genetic Algorithm on Dihedral Groups

Denote by D_{4t} the dihedral group $\mathbb{Z}_{2t} \times_\chi \mathbb{Z}_2$ of order $4t$, $t \geq 1$, given by the presentation

$$< a, b | a^{2t} = b^2 = (ab)^2 = 1 >$$

and ordering

$$\{1=(0,0), a=(1,0), \ldots, a^{2t-1}=(2t-1,0), b=(0,1), \ldots, a^{2t-1}b=(2t-1,1)\}$$

In [9] a representative 2-cocycle f of $[f] \in H^2(D_{4t}, \mathbb{Z}_2) \cong \mathbb{Z}_2^3$ is written interchangeably as a triple (A, B, K), where A and B are the inflation variables and K is the transgression variable. All variables take values ± 1. Explicitly,

$$f(a^i, a^j b^k) = \begin{cases} A^{ij}, & i+j < 2t, \\ A^{ij}K, & i+j \geq 2t, \end{cases}$$

$$f(a^i b, a^j b^k) = \begin{cases} A^{ij}B^k, & i \geq j, \\ A^{ij}B^k K, & i < j, \end{cases}$$

Let β_1, β_2 and γ denote the representative 2-cocycles related to $(A, B, K) = (1, -1, 1), (-1, 1, 1), (1, 1, -1)$ respectively.

A basis for 2-coboundaries is described in [3]. Let $\partial_x : D_{4t} \to \mathbb{F}_2$ denote the characteristic set map associated to x, such that $\partial_x(y) = 1$ for $y \neq x$ and $\partial_x(x) = -1$. Let δ_x denote the 2-coboundary naturally associated to ∂_x, such that $\delta_x(s,t) = \partial_x(s)\partial_x(t)\partial_x(s \cdot t)$. According to the ordering above, a basis for 2-coboundaries may be constructed straightforwardly. It suffices to drop coboundaries $\delta_1, \delta_{a^{2t-2}b}, \delta_{a^{2t-1}b}$ from the whole set of coboundaries naturally associated to the elements in D_{4t}, as it is shown in [3]. Consequently, there are 2^{4t-3} different 2-coboundaries. Furthermore, there are 2^{4t} different 2-cocycles, and $\mathcal{B} = \{\delta_a, \ldots, \delta_{a^{2t-3}b}, \beta_1, \beta_2, \gamma\}$ is a basis for 2-cocycles.

Once a basis for 2-cocycles over D_{4t} has been determined, we turn towards cocyclic Hadamard matrices.

A condition for the existence of a cocyclic Hadamard matrix over D_{4t} is detailed in [9]. Cocyclic Hadamard matrices developed over D_{4t} can exist only in the cases $(A, B, K) = (1, 1, 1), (1, -1, 1), (1, -1, -1), (-1, 1, 1)$ for t odd. We focus in the case $(A, B, K) = (1, -1, -1)$, since computational results in [9, 3] suggest that this case contains a large density of cocyclic Hadamard matrices. Anyway, the techniques presented in this paper can be adapted easily for other cases of (A, B, K), or even other finite groups rather than D_{4t}, as the examples in Section 6 illustrate.

At this time, we may assume that individuals of our population consists of binary $(4t-3)$-tuples (better than $4t$-tuples), corresponding to generators from the basis for 2-coboundaries. Furthermore, computational results in [3] suggest that tuples formed from $2t-1$ to $2t+1$ ones gives rise to a significantly large density of cocyclic Hadamard matrices.

So that we may assume that individuals of our initial population consists of tuples that meet these bounds. That is, tuples of length $4t-3$ which consists of k ones and $4t-3-k$ zeros, for $2t-1 \leq k \leq 2t+1$. Consequently, the search space reduces in turn, from $2^{4t-3} = \sum_{i=0}^{4t-3} \binom{4t-3}{i}$ individuals to precisely $\binom{4t-3}{2t-1} + \binom{4t-3}{2t} + \binom{4t-3}{2t+1}$ individuals. We do not care about

missing many potential solutions (from among those tuples which do not meet the impossed bounds). Computational evidences of this fact are discussed in [3]. Anyway, crossover and mutation operators will eventually introduce individuals out from the impossed bounds, which will also strive for survival, so that good behaviuors outside the reduced search space may be ocassionally incorporated.

In these circumstances, the evaluation function for fitness may be redesigned to check precisely rows from 2 to t. This is a straightforward consequence of a result in [3]: a cocyclic matrix over D_{4t} of the type $\left(\prod_{i\in I}\delta_i\right)\beta_1\gamma$ is Hadamard if and only if rows from 2 to t are Hadamard rows. Consequently, the Hadamard test runs 4 times faster each time. This way, when the genetic algorithm runs on D_{4t} we are able to reduce not only the search space but also the search time.

6 Examples

Both of the genetic algorithms and all the executions and examples of this section have been worked out with aid of MATHEMATICA 4.0, running on a *Pentium IV 2.400 Mhz DIMM DDR266 512 MB*. We include here some Hadamard cocyclic matrices of order $4t$ for $6 \leq t \leq 13$. Apparently, our method seems to provide some cocyclic Hadamard matrices of larger order than those previously obtained with other algorithms.

Calculations in [5, 9, 2] suggest that $G_1^t = \mathbb{Z}_t \times \mathbb{Z}_2^2$ and $G_2^t = D_{4t}$ give rise to a large number of Hadamard cocyclic matrices.

This behavior has also been observed on a third family of groups [2],

$$G_3^t = (\mathbb{Z}_t \times_f \mathbb{Z}_2) \times_\chi \mathbb{Z}_2$$

Here f denotes the normalized 2-cocycle $f : \mathbb{Z}_2 \times \mathbb{Z}_2 \to \mathbb{Z}_t$ such that

$$f(-1,-1) = \lceil \frac{t}{2} \rceil + 1$$

And $\chi : \mathbb{Z}_2 \times (\mathbb{Z}_t \times_f \mathbb{Z}_2) \to \mathbb{Z}_t \times_f \mathbb{Z}_2$ denotes the dihedral action, such that $\chi(-1, x) = -x$. Notice that G_3^t is a slight modification of $G_2^t = D_{4t}$, since f becomes a 2-coboundary precisely for odd $t = 2k+1$. Thus $G_3^{2k+1} = G_2^{2k+1}$ and $G_3^{2k} \neq G_2^{2k}$.

However the search space for cocyclic Hadamard matrices over the families G_i^t above grows exponentially with t (according to the dimensions of the basis $\mathcal{B}_i$ for 2-cocycles), so that exhaustive search is only possible in low orders (up to $t = 5$). Each of the matrices is represented as a tuple with regards to some basis $\mathcal{B}_i = \{\delta_k|\beta_j|\gamma_n\}$ for 2-cocycles over G_i^t. At this point, we only indicate how

many generators of each type (coboundaries, inflation and transgression) appear in $\mathcal{B}_i$ (see [2, 3] for details):

- For odd t, $\mathcal{B}_1$ consists of $4t-3$ coboundaries δ_k, 2 cocycles β_j coming from inflation and 1 cocycle γ coming from transgression. For even t, $\mathcal{B}_1$ consists of $4t-4$ coboundaries δ_k, 3 cocycles β_j coming from inflation and 3 cocycles γ_n coming from transgression.
- $\mathcal{B}_2$ is the basis $\mathcal{B}$ described at Section 5, which consists of $4t-3$ coboundaries δ_k, 2 cocycles β_j coming from inflation and 1 cocycle γ coming from transgression.
- $\mathcal{B}_3$ coincides with $\mathcal{B}_2$ for odd t. We have not identified a general behavior for even t, so we analyze the cases $t = 2, 4, 6, 8$ independently:
  - If $t = 2$, $\mathcal{B}_3$ consists of 4 coboundaries δ_k, 3 cocycles β_j coming from inflation and 3 cocycles γ_n coming from transgression.
  - If $t = 4$, $\mathcal{B}_3$ consists of 13 coboundaries δ_k, 2 cocycles β_j coming from inflation and 1 cocycle γ coming from transgression.
  - If $t = 6$, $\mathcal{B}_3$ consists of 20 coboundaries δ_k, 3 cocycles β_j coming from inflation and 3 cocycles γ_n coming from transgression.
  - If $t = 8$, $\mathcal{B}_3$ consists of 29 coboundaries δ_k, 2 cocycles β_j coming from inflation and 1 cocycle γ coming from transgression.

Now we show some executions of the genetic algorithm (in its general version) running on these families. The tables below show some Hadamard cocyclic matrices over G_i^t, and the number of iterations and time required (in seconds) as well. Notice that the number of generations is not directly related to the size of the matrices. Do not forget about randomness of the genetic algorithm.

t	iter.	time	product of generators of 2-cocycles over G_1^t
1	0	$0''$	$(1,0,0,0)$
2	0	$0''$	$(1,0,0,1,0,1,0,0,1,1)$
3	1	$0.14''$	$(0,1,1,0,1,0,0,0,1,1,1,1)$
4	7	$1.89''$	$(0,1,1,0,0,1,1,1,1,1,1,0,1,1,1,0,0,1)$
5	30	$17.08''$	$(1,0,1,1,0,0,0,1,0,1,1,1,0,1,1,1,0,1,1,1)$
6	3	$3.69''$	$(0,0,0,1,0,0,0,1,1,1,0,0,1,1,0,1,1,1,0,1,1,0,1,0,0,1)$
7	584	$21'33''$	$(1,0,1,1,0,1,1,0,1,0,1,0,0,0,0,0,1,1,0,0,1,1,0,0,0,1,1,1)$
8	239	$14'33''$	$(0,0,0,1,0,0,0,0,0,1,1,1,1,0,1,0,1,0,1,0,0,0,1,0,1,1,0,1,0,0,1,0,0,0,1)$

t	iter.	time	product of generators of 2-cocycles over G_2^t
1	0	$0''$	$(0,1,1,1)$
2	0	$0''$	$(1,0,1,1,1,1,0,0)$
3	3	$0.25''$	$(1,0,1,1,0,0,1,0,0,1,0,1)$
4	0	$0''$	$(0,1,1,1,1,1,1,0,0,0,1,0,1,1,0,1)$
5	3	$1.42''$	$(0,0,1,0,1,1,0,1,0,1,0,0,0,1,0,1,1,1,0,1)$
6	31	$34.87''$	$(1,1,0,1,1,0,1,0,0,0,0,0,0,1,0,1,0,0,1,1,0,1,0,1)$
7	102	$5'17''$	$(1,0,0,1,0,1,1,0,0,1,1,1,0,1,0,0,0,0,0,0,0,0,1,1,0,1,0,1)$
8	98	$6'27''$	$(0,0,1,0,1,1,0,1,1,0,1,1,1,0,1,1,0,1,1,1,1,0,0,0,0,0,0,1,0,1,0,1)$

t	iter.	time	product of generators of 2-cocycles over G_3^t
1	0	$0''$	$(0,1,1,1)$
2	0	$0''$	$(1,1,0,0,0,0,1,0,0,0)$
3	0	$0''$	$(0,1,1,0,1,1,0,0,1,1,1,1)$
4	6	$1.20''$	$(0,0,1,0,0,0,1,1,1,1,1,0,0,0,1,0)$
5	18	$10.33''$	$(1,0,1,0,1,1,1,0,0,1,1,1,1,1,1,0,0,0,1,0)$
6	15	$19.49''$	$(1,1,1,1,0,0,1,1,1,0,0,0,1,0,1,1,1,1,1,1,0,1,1,1,1,1)$
7	6	$12.39''$	$(0,1,1,0,1,0,1,0,0,1,1,1,0,0,1,1,0,1,1,0,1,0,1,0,0,0,0,1,1,1)$
8	153	$9'45''$	$(1,0,0,1,0,1,1,0,1,1,1,0,1,0,0,0,1,1,1,0,1,1,1,0,1,0,0,1,0,1,0,0)$

As it is expected, the improved version of the genetic algorithm for $G_2^t = D_{4t}$ provides not only faster outputs but also larger sizes on the matrices.

t	iter.	time	product of generators of 2-cocycles over D_{4t} (improved version)
6	0	$0''$	$(1,1,0,1,1,1,1,0,1,1,1,0,0,1,0,1,1,1,0,1,0,1,0,1)$
7	4	$0.69''$	$(0,0,0,1,1,0,0,1,0,1,0,0,1,1,1,1,0,1,0,1,0,0,1,1,0,1,0,1)$
8	3	$1.18''$	$(0,0,1,0,1,0,1,1,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,1,0,1)$
9	7	$5.09''$	$(1,0,0,0,0,1,1,0,1,0,1,0,0,0,1,0,0,1,1,0,0,1,1,0,1,1,1,0,0,0,0,1,$ $0,1,0,1)$
10	43	$48.03''$	$(0,0,1,0,0,0,0,0,1,1,1,1,1,0,1,1,0,1,1,0,1,0,0,1,0,0,0,0,1,1,1,$ $0,1,0,1,0,1,1,1,0,1)$
11	471	$13'15''$	$(1,1,1,1,0,0,0,1,1,1,1,0,1,1,1,1,0,1,1,1,0,0,0,1,0,1,0,1,1,1,$ $1,1,0,1,0,0,1,0,0,1,1,1,0,1)$
12	279	$11'16''$	$(0,0,0,0,1,1,0,1,1,1,0,1,1,0,1,0,0,1,1,1,1,0,1,1,0,0,0,0,1,1,$ $0,1,0,0,0,1,0,1,0,1,1,0,0,0,1,1,0,1)$
13	970	$53'44''$	$(0,1,0,0,0,0,1,1,0,1,0,0,0,1,0,1,1,0,1,0,1,1,0,0,0,0,1,1,1,1,$ $1,0,0,0,0,0,1,0,0,0,1,0,0,0,1,1,0,1,0,1,0,1)$

The authors are convinced that improved versions of the algorithm are still to be implemented, attending to refinements on the crossover operator. We are yet to find a systematic way of doing crossover more suitably for our purposes.

Acknowledgments

The authors want to express their gratitude to the referees for their many valuable advices and suggestions, which have led to a number of improvements of the paper.

References

1. V. Álvarez, J.A. Armario, M.D. Frau and P. Real. An algorithm for computing cocyclic matrices developed over some semidirect products. *Proceedings AAECC'14*. Eds. S. Boztas, I.E. Shparlinski. *Springer Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, (2001).
2. V. Álvarez, J.A. Armario, M.D. Frau and P. Real. Homological reduction method for constructing cocyclic Hadamard matrices. To appear.

3. V. Álvarez, J.A. Armario, M.D. Frau and P. Real. How to bound the search space looking for cocyclic Hadamard matrices. To appear.
4. A. Baliga and J. Chua. Self-dual codes using image resoration techniques. *Proceedings AAECC'14*. Eds. S. Boztas, I.E. Shparlinski. *Springer Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, (2001).
5. A. Baliga and K.J. Horadam. Cocyclic Hadamard matrices over $\mathbb{Z}_t \times \mathbb{Z}_2^2$. *Australas. J. Combin.*, **11**, 123–134, (1995).
6. W. de Launey and K.J. Horadam. Cocyclic development of designs. *J. Algebraic Combin.*, **2** (3), 267–290, 1993. Erratum: *J. Algebraic Combin.*, (1), pp. 129, 1994.
7. W. de Launey and K.J. Horadam. Generation of cocyclic Hadamard matrices. *Computational algebra and number theory* (Sydney, 1992), volume **325** of *Math. Appl.*, 279–290. Kluwer Acad. Publ., Dordrecht, (1995).
8. D.L. Flannery. Calculation of cocyclic matrices. *J. of Pure and Applied Algebra*, **112**, 181–190, (1996).
9. D.L. Flannery. Cocyclic Hadamard matrices and Hadamard groups are equivalent. *J. Algebra*, **192**, 749–779, (1997).
10. D.L. Flannery and E.A. O'Brien. Computing 2-cocycles for central extensions and relative difference sets. *Comm. Algebra*, **28(4)**, 1939–1955, (2000).
11. J. Grabmeier, L.A. Lambe. Computing Resolutions Over Finite p-Groups. *Proceedings ALCOMA'99*. Eds. A. Betten, A. Kohnert, R. Lave, A. Wassermann. *Springer Lecture Notes in Computational Science and Engineering*, Springer-Verlag, Heidelberg, (2000).
12. V.K.A.M. Gugenheim and L.A. Lambe. Perturbation theory in Differential Homological Algebra, I. *Illinois J. Math.,* **33**, 556–582, (1989).
13. V.K.A.M. Gugenheim, L.A. Lambe and J.D. Stasheff. Perturbation theory in Differential Homological Algebra II, *Illinois J. Math.,* **35** (3), 357–373, (1991).
14. J.H. Holland. Adaptation in natural and artificial systems. *University of Michigan Press*, Ann Arbor (1975).
15. Z. Michalewicz. Genetic algorithms + data structures = evolution programs. *Springer-Verlag* (1992).
16. P. Real. Homological Perturbation Theory and Associativity. *Homology, Homotopy and Applications*, **2**, 51–88, (2000).