The final publication is available at Springer via https://doi.org/10.1007/978-3-030-04070-3_12

The Design of (Almost) Disjunct Matrices by Evolutionary Algorithms

Karlo Knezevic[1], Stjepan Picek[2], Luca Mariot[3], Domagoj Jakobovic[2], and Alberto Leporati[3]

[1] Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
{karlo.knezevic, domagoj.jakobovic}@fer.hr
[2] Cyber Security Research Group, Delft University of Technology, Delft, The Netherlands
S.Picek@tudelft.nl
[3] Department of Informatics, Systems, and Communication, University of Milano-Bicocca, Milan, Italy
{luca.mariot, alberto.leporati}@unimib.it

Abstract. Disjunct Matrices (DM) are a particular kind of binary matrices which have been especially applied to solve the Non-Adaptive Group Testing (NAGT) problem, where the task is to detect any configuration of t defectives out of a population of N items. Traditionally, the methods used to construct DM leverage on error-correcting codes and other related algebraic techniques. Here, we investigate the use of Evolutionary Algorithms to design DM and two of their generalizations, namely Resolvable Matrices (RM) and Almost Disjunct Matrices (ADM). After discussing the basic encoding used to represent the candidate solutions of our optimization problems, we define three fitness functions, each measuring the deviation of a generic binary matrix from being respectively a DM, an RM or an ADM. Next, we employ Estimation of Distribution Algorithms (EDA), Genetic Algorithms (GA), and Genetic Programming (GP) to optimize these fitness functions. The results show that GP achieves the best performances among the three heuristics, converging to an optimal solution on a wider range of problem instances. Although these results do not match those obtained by other state-of-the-art methods in the literature, we argue that our heuristic approach can generate solutions that are not expressible by currently known algebraic techniques, and sketch some possible ideas to further improve its performance.

Keywords: Evolutionary Computing, Disjunct Matrices, Resolvable Matrices, Almost Disjunct Matrices, Group Testing, Estimation of Distribution Algorithms, Genetic Algorithms, Genetic Programming

1 Introduction

The *group testing problem* deals with identifying a set of at most t *defectives* among a population of N *items*. More specifically, in *non-adaptive group testing*

(NAGT) one has a set of M tests which are performed in parallel over subsets of the population, and each test reports a positive result if at least one of the elements in its subset is defective. The goal is to detect all possible combinations of at most t defective items, trying to minimize the number of tests M [7].

From the combinatorial designs perspective, *Disjunct Matrices* (DM) turn out to be the combinatorial objects related to NAGT [4]. Practically speaking, a DM is an $M \times N$ binary matrix, whose rows and columns respectively represent the tests to perform and the items to be tested. An entry at the position (i, j) is set to 1 if the i-th test includes the j-th item, and 0 otherwise. A DM is arranged in such a way that the *support* (i.e., the set of non-zero entries) of any subset of t of its columns does not contain the support of any remaining column. This property guarantees the detection of any configuration of up to t defectives.

In the *error correction/detection* version of the group testing problem, one also allows the possibility to detect some *false positives*, i.e., a test can report that a subset of items contains a defective even if this is not the case. Of course, one wishes to keep the proportion of false positives the smallest possible for the sake of testing efficiency. In this relaxed version of NAGT, the corresponding combinatorial designs involved are a generalization of DM, namely *Resolvable Matrices* (RM) and *Almost Disjunct Matrices* (ADM). Here, the disjunctness constraint is loosened by allowing any t-subset of columns to contain the supports of other columns, up to a specified proportion. Resolvable and Almost Disjunct Matrices are also useful in several other domains beyond group testing: applications include *key distribution patterns* and *frameproof codes* in cryptography [21], *topology transparent scheduling* in shared-channel networks [5], as well as the design of *bloom filters* and *perfect hash families* [6,22], which are in turn used, for example, in bioinformatics for pattern matching [3].

In general, constructions for DM, RM, and ADM focus on *algebraic methods*, usually employing error-correcting codes techniques. Kautz and Singleton [9] pioneered this approach by proposing a construction of DM based on *Reed-Solomon Codes*. Porat and Rothschild [19] provided another construction based on the same approach laid out in [9], but leveraging on a different breed of error-correcting codes reaching the *Gilbert-Varshamov bound*. More recently, Mazumdar [14] and Barg and Mazumdar [2] extended this investigation to ADM, respectively exploiting the average distance analysis and the dual distance of *constant-weight codes*.

The goal of this paper is to tackle the construction of DM, RM, and ADM through *evolutionary algorithms* (EAs), namely *Estimation of Distribution Algorithms* (EDA), *Genetic Algorithms* (GA), and *Genetic Programming* (GP). In particular, instead of considering the construction of disjunct matrices from previously known combinatorial objects with specified parameters (such as the aforementioned error-correcting codes), we formulate a combinatorial optimization problem over the set of binary matrices, where the optimal solutions correspond either to DM, RM or ADM.

The main reasons motivating this research can be summarized as follows:

1. EA can represent an interesting alternative for the design of DM without any assumption on their underlying structure. In fact, even if algebraic techniques are already able to provide a wide range of disjunct matrices of various sizes, they always rely on additional hypotheses, beside the bare definition and properties of DM. As such, algebraic constructions only yield a subset among all possible disjunct matrices. On the contrary, since EAs can explore the whole set of binary matrices, they can in principle discover DM which are beyond the reach of currently known algebraic constructions.
2. Symmetrically, due to the close connection between DM and codes, any disjunct matrix found by EA which cannot be expressed by one of the current algebraic constructions could give hints on new classes of error-correcting codes, and maybe suggest how to construct them.
3. Finally, the construction of combinatorial designs is a research line that has received relatively little attention in the EA literature, and disjunct matrices play no exception: as far as we know, our paper is the first addressing the design of DM through EAs. As a consequence, we deem this problem also interesting from the benchmarking point of view, to assess how difficult it is for an evolutionary-based optimization heuristic to construct a DM. In this respect, this paper follows the same line of our previous works, where we investigated the construction of other kinds of combinatorial designs via EAs, namely *Orthogonal Latin Squares* [12] and *Binary Orthogonal Arrays* [13].

After formally introducing the three combinatorial optimization problems of our interest, we describe the encoding for the candidate solutions, which is based on a *multiploid genome* where each bitstring represents a column of a binary matrix. Subsequently, we describe three fitness functions, one for each optimization task. The general idea underlying each fitness function is to count the number of columns whose support is contained in the union of the supports of a subset of t columns, and then use this quantity to measure the deviation of a binary matrix from being respectively a DM, an RM or an ADM. Next, we apply EDA, GA, and GP to minimize these three fitness functions.

The rest of this paper is organized as follows. Section 2 gathers all necessary background and basic notions about disjunct matrices, resolvable matrices, and almost disjunct matrices. Section 3 formally states the combinatorial optimization problems addressed in this paper, describes the encoding adopted for the candidate solutions, and defines the three fitness functions to optimize. Section 4 outlines the experimental settings and parameters which we adopted for EDA, GA, and GP, and discusses the obtained results. Finally, Section 5 recaps the main contributions of the paper, and sketches several avenues for further research.

2 Disjunct Matrices

Before formally defining disjunct matrices and their generalizations, let us introduce notation which we will use in the rest of the paper. Given $n \in \mathbb{N}$, let $[n] = \{1, 2, \cdots, n\}$. Moreover, given a binary vector $x \in \{0,1\}^n$, the *support* of x

is the set of non-zero coordinates of x, that is, $supp(x) = \{i \in [n] : x_i \neq 0\}$. For any vector $x \in \{0,1\}^n$, $x^\top$ denotes the transpose of x as a column vector. We represent an $M \times N$ binary matrix A as $A = (x_1^\top, x_2^\top, \cdots, x_N^\top)$, where $x_i \in \{0,1\}^M$ for all $i \in [N]$. We now give the formal definition of a t-disjunct matrix:

Definition 1. *Let $A = (x_1^\top, x_2^\top, \cdots, x_N^\top)$ be an $M \times N$ binary matrix. Then, A is called* t-disjunct *if, for all subsets of t columns $S = \{x_{i_1}, \cdots, x_{i_t}\}$, and for all remaining columns $x_j \notin S$, it holds that*

$$supp(x_j) \not\subseteq \bigcup_{k=1}^{t} supp(x_{i_k}) \ . \tag{1}$$

In other words, a matrix A is t-disjunct if for every subset S of t columns the support of any other column is not contained in the union of the supports of the columns in S.

It is easy to see that a t-disjunct matrix is equivalent to a NAGT which is able to detect all possible combination of t defective out of N objects: in particular, the M rows represent the test, and an entry (i, j) set to 1 indicates that the i-th test probes the j-th item.

We will be also interested in the relaxed versions of disjunct matrices, namely *resolvable matrices* and *almost disjunct matrices*, which are related to the error correction/detection variant of NAGT with false positives. We formally define them below.

Definition 2. *An $M \times N$ binary matrix $A = (x_1^\top, x_2^\top, \cdots, x_N^\top)$ is called* (t, f)-resolvable *if, for all subsets of t columns $S = \{x_{i_1}, \cdots, x_{i_t}\}$, the set D defined as:*

$$D = \{x_j \notin S : supp(x_j) \subseteq \bigcup_{k=1}^{t} supp(x_{i_k})\}$$

has cardinality at most f.

Definition 3. *An $M \times N$ binary matrix $A = (x_1^\top, x_2^\top, \cdots, x_N^\top)$ is called* (t, ε)-disjunct *if, for all subsets of t columns $S = \{x_{i_1}, \cdots, x_{i_t}\}$, the probability that the support of a random column $x_j \notin S$ is contained in S is at most ε.*

Stated otherwise, for (t, f)-resolvable matrix we allow for each possible subset S of t columns to have at most f remaining columns whose support is contained in the union of the supports of the vector of S. On the other hand, with (t, ε)-*disjunct* matrices, we require that the support of a random column sampled among the remaining $N - t$ ones is contained in the union of the supports of S with probability at most ε. In what follows, we will also refer to (t, f)-resolvable matrices and (t, ε)-*disjunct* matrices respectively as *Resolvable Matrices* (RM) and *Almost Disjunct Matrices* (ADM).

Remark 4. The following relations are straightforward:

– A t-disjunct matrix is also a (t, f)-resolvable matrix, with $f = 0$.

– A (t, f)-resolvable matrix is also a (t, ε)-disjunct matrix with $\varepsilon = \frac{f}{N-t}$. Notice however that the converse is not true: in a (t, ε)-disjunct matrix, the frequency of columns whose support is contained in the union is averaged over all possible subsets of t columns.

To conclude this section, we now formally define the combinatorial optimization problem which we will address in the remainder of this paper:

Problem 5. Let $M, N \in \mathbb{N}$. Then:

– Given $t < N$, find a t-disjunct $M \times N$ matrix.
– Given $t, f < N$, find a (t, f)-resolvable $M \times N$ matrix.
– Given $t < N$ and $\varepsilon \in [0, 1]$, find a (t, ε)-disjunct matrix.

3 Optimization Problem Structure

3.1 Solutions encoding

Since we are interested in using *evolutionary algorithms* (EAs) to solve Problem 5, we must first define a suitable encoding for the feasible solutions of the problem.

Given the underlying matrix structure of the objects we want to optimize, and since the disjunctness properties are checked on the columns of such matrix, the most natural way to encode the genotype of a candidate solution is by means of a *multiploid genome*, i.e., a set of N binary string, each of length M, that when put one next to the other form the columns of a binary $M \times N$ matrix. More formally, the genotype of an individual will be a sequence $G = (x_1, x_2, \cdots x_N)$ such that $x_i \in \{0, 1\}^M$ for all $i \in [N]$. The phenotype, on the other hand, will simply correspond to the same sequence by transposing the strings as column vectors, i.e., $P = (x_1^\top, \cdots, x_N^\top)$.

Using *Genetic Algorithms* (GA) or *Estimation of Distribution Algorithms* (EDA) does not put any constraint on the length M of the chromosomes. On the other hand, since *Genetic Programming* (GP) evolves Boolean trees which are then mapped to n-variable Boolean functions, one has either to restrain the length of the chromosome to be equal to the size of their truth table, i.e., $M = 2^n$, or else to truncate them at a certain length. In our experiments, we adopted the latter approach.

3.2 Fitness Functions

We now define a fitness function for each of the three optimization problems set forth in Problem 5. In what follows, given an $M \times N$ binary matrix $A = (x_1^\top, \cdots, x_N^\top)$, we denote by $X = \{x_1, \cdots x_N\}$ its support set containing the column vectors. Additionally, we define $\mathcal{S}_t = \{S \subseteq X : |S| = t\}$ as the family of all subsets of t columns of A. Then, for all $S \in \mathcal{S}$, let us define the *deviation* $\delta(S)$ as follows:

$$\delta(S) = |\{x_j \in X \setminus S : supp(x_j) \subseteq \bigcup_{x_i \in S} supp(x_i)\}| \ , \tag{2}$$

that is, $\delta(S)$ is the number of columns in A that does not belong to S and such that their support is included in the union of supports in S.

Since for the first optimization problem we are interested in obtaining matrices such that $\delta(S) = 0$ for every subset of t columns, we define the corresponding fitness function simply as the sum of the deviations of all subsets $S \in \mathcal{S}_t$. In particular, given an $M \times N$ binary matrix A, its fitness function is defined as:

$$fit_1(A) = \sum_{S \in \mathcal{S}_t} \delta(S) \ . \tag{3}$$

Clearly, the optimization objective is to minimize fit_1, and an optimal solution corresponds to a binary matrix A such that $fit_1(A) = 0$, i.e., a t-disjunct matrix.

For the second optimization problem concerning (t, f)-resolvable matrices, we have to take into account that the union of the supports of any t columns in A can include up to f supports of the remaining columns. Hence, the idea here is to minimize the number of t-subsets of columns that do not satisfy this requirement. In particular, given a matrix A, the second fitness function is defined as:

$$fit_2(A) = |\{S \in \mathcal{S}_t : \delta(S) > f\}| \ , \tag{4}$$

where the objective is to minimize fit_2, with $fit_2(A) = 0$ representing the optimal value.

Finally, in the third optimization problem we consider the most relaxed definition of disjunctness, since the event that the support of a column is contained in the union of supports of a random t-subset of other columns must be less than or equal to ε. To address this case, we considered a third fitness function where the sum of the deviations of all subsets of t columns (i.e., fit_1) is averaged over all possible choices with which one can select one of these subsets and all remaining columns. In particular, given an $M \times N$ matrix A, the number of subsets of t columns is $\binom{N}{t}$, while the number of remaining columns is $N - t$. Thus, the third fitness function of A is defined as:

$$fit_3(A) = \frac{fit_1(A)}{\binom{N}{t} \cdot (N - t)} = \frac{\sum_{S \in \mathcal{S}_t} \delta(S)}{\binom{N}{t} \cdot (N - t)} \ . \tag{5}$$

As in the previous two cases, the optimization goal is to minimize fit_3. In particular, remark that the range of fit_3 is the interval $[0, 1]$, and effectively represents the probability that the support of a random column is contained in the union of supports of a random subset of t distinct columns. Hence, an optimal solution for the third optimization problem is an $M \times N$ matrix A such that $fit_3(A) \leq \varepsilon$.

Algorithm 1 Estimation of distribution algorithm

Set $t \Leftarrow 0$. Generate N solutions randomly;
repeat
  Evaluate the solutions using the fitness function;
  Select a population D_t^S of $K \leq N$ solutions according to a selection method;
  Calculate a probabilistic model of D_t^S;
  Generate N new solutions sampling from the distribution represented in the model;

  $t \Leftarrow t + 1$;
until Termination criteria are met

4 Experimental Setting and Results

In this section, we introduce the algorithms we use, common parameters, and the obtained results. Note that the parameters used in our experiments are selected after a tuning phase.

4.1 Estimation of Distribution Algorithms

Estimation of distribution algorithms (EDAs) work by extracting patterns shared by the best solutions and representing these patterns using a probabilistic graphical model (PGM) [17] in order to generate new solutions from this model [15,10]. Differing from GAs, EDAs apply learning and sampling of distributions instead of classical crossover and mutation operators. Modeling the dependencies between the variables of the problem serves to efficiently orient the search to more promising areas of the search space by explicitly capturing and exploiting potential relationships between the problem variables. We give a pseudocode of an EDA in Algorithm 1.

In our experiments, we use the Univariate Marginal Distribution Algorithm (UMDA) [16]. UMDA is a type of EDA that uses operator α to estimate the marginal distributions from a selected population $S(D)$. By assuming that selected population contains λ elements, α produces probabilities:

$$p_{t+1}(X_i) = \frac{1}{\lambda} \sum_{x \in S(D)} x_i, \forall i \in 1, 2, \ldots, N. \tag{6}$$

4.2 Genetic Algorithm

The GA represents the individuals of an optimization problem as strings of bits. We use a 3-tournament selection, where the worst from the 3 randomly selected individuals is eliminated [8]. A new individual is created by applying crossover to the remaining two and then a mutation with given probability (Algorithm 2).

Mutation is selected uniformly at random between a simple mutation, where a single bit is inverted, and a mixed mutation, which randomly shuffles the bits in a randomly selected subset. The crossover operators are one-point and

Algorithm 2 Steady-state tournament selection

randomly select k individuals;
remove the worst of k individuals;
child = crossover (best two of the tournament);
perform mutation on *child*, with given individual mutation probability;
insert *child* into population;

uniform crossover, performed uniformly at random for each new offspring. We use population size of 100 and individual mutation probability of 0.3. The mutation probability is used to select whether an individual would be mutated or not, and the mutation operator is executed only once on a given individual.

4.3 Genetic Programming

Genetic Programming (GP) uses a representation where individuals are trees of Boolean primitives which are then evaluated according to the truth table they produce. The function set for GP in all experiments is OR, XOR, AND, XNOR, and AND with one input inverted. Terminals correspond to n Boolean variables. GP uses the same selection as presented in Algorithm 2 with a tournament size 3. The crossover is performed with five different tree-based crossover operators selected at random: a simple tree crossover with 90% bias for functional nodes, uniform crossover, size fair, one-point, and context preserving crossover [18]. We use the subtree mutation applied with 30% probability and the maximum tree depth size of 6. The population size equals 100.

4.4 Common Parameters

In all the experiments the number of independent trials for each configuration is 30 and the stopping criterion for all algorithms equals 500 000 evaluations or achieving the optimum value for the corresponding fitness function.

Regarding the matrices parameters, we decided to experiment with $t = 2$ and $t = 3$ for disjunct matrices, in order to have a baseline of comparison with the results of [1], which reports the best known examples obtained through algebraic techniques. For resolvable matrices, we set f equal to the 30% of the $N - t$ remaining columns of a matrix, while for almost disjunct we set $\varepsilon = 10^{-4}$ after a preliminary phase of parameter tuning.

4.5 Results

In Table 1, we give results for all optimization techniques and dimensions we considered. We experimented with various matrix sizes according to the number of rows and columns. The first column shows the number of rows and maximum number of columns per row of the matrices considered. Then, each entry indicates the maximum number of columns of the best solution found by the associated

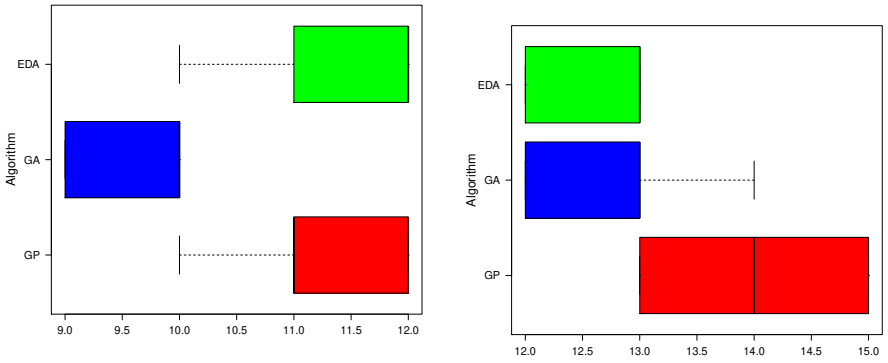
heuristic under the corresponding fitness function. Values in bold are the optimal ones, i.e., the maximum number of columns of the best known examples produced by algebraic techniques, as reported in [1].

For fit_1 GP achieves the best results when compared to other algorithms. In fact, GP found the maximum number disjunct matrices for smaller dimensions when $t = 2$ and $t = 3$. As the relaxation of the fit_1, GP achieves the best results for fit_2. For higher dimensions, all three algorithms find more disjunct matrices but not the maximum. Finally, for the third fitness, GP again achieves the best results. Note that here, we reach optimal values for more dimensions since fit_3 is a relaxed version of fit_2. According to the given values, we can conclude that GP obtained the best results, followed by EDA and finally by GA.

Table 1: Best solutions found by GP, GA, and EDA for each fitness

(M/max. N)	GP	GA	EDA	GP	GA	EDA	GP	GA	EDA
$t = 2$		fitness 1			fitness 2			fitness 3	
8/8	**8**	5	7	**8**	5	**8**	**8**	6	7
9/12	**12**	10	**12**	**12**	11	**12**	**12**	11	**12**
10/13	12	10	11	12	11	11	**13**	11	**13**
11/18	**17**	15	16	**17**	16	**17**	**17**	16	**17**
12/20	16	13	13	17	13	14	18	13	14
13/26	20	17	18	20	18	19	21	19	21
14/28	21	19	20	22	21	22	22	21	22
15/35	24	20	22	25	24	24	25	24	24
$t = 3$		fitness 1			fitness 2			fitness 3	
13/13	**13**	11	11	**13**	**13**	**13**	**13**	**13**	**13**
14/14	**14**	12	12	**14**	**14**	**14**	**14**	**14**	**14**
15/15	**15**	14	13	**15**	14	**15**	**15**	**15**	**15**
16/20	17	14	14	19	17	18	**20**	18	18
17/21	18	15	16	20	16	16	20	16	17
18/22	19	16	18	21	17	19	21	17	19
19/28	19	16	17	23	17	18	23	17	18
20/30	22	19	21	24	19	22	25	20	22
21/31	23	20	21	26	22	23	26	24	24

In Figures 1a and 1b, we present results for $t = 2$ and dimension 9/12 and for $t = 3$ and dimension 15/15, respectively. Note that the values given are average values over all experimental runs. We can see that in the first case, GP and EDA behave similarly while GA exhibits much worse performance. For the second case, GP outperforms by far both EDA and GA. These results are in accordance with other scenarios where at least one algorithm reached optimal result.



(a) $t = 2$, scenario 9/12

c‘

(b) $t = 3$, scenario 15/15

Fig. 1: Boxplot results for 2 test scenarios

5 Conclusions

In this paper, we addressed the construction of disjunct matrices, resolvable matrices, and almost disjunct matrices through evolutionary algorithms. To the best of our knowledge, our work is the first attempt at solving this combinatorial design problem using an evolutionary optimization approach, since all methods described in the existing literature are based on coding-theoretic and algebraic techniques. We encoded the genotype of a candidate solution as a *multiploid genome*, where each chromosome is a binary string representing a column of a binary matrix. We then defined three fitness functions, one for each kind of matrix we were interested in; the idea underlying all fitness functions was to count the number of columns whose support is contained in the union of the supports of other t-subsets of columns. Next, we applied EDA, GA, and GP to minimize these three fitness functions. The results showed that GP is the best heuristic among the three considered, since it managed to converge to an optimal solution on the widest range of problem instances we considered. This finding corroborates the hypothesis that GP is in general a better heuristic for constructing combinatorial designs, as we found similar results also in the case of orthogonal Latin squares [12].

Still, GP results are not as good as those achieved by algebraic methods, which are able to construct larger DM. However, our heuristic approach do not require any assumption beyond the bare definitions of DM, RM or ADM set forth in Section 2, as opposed to algebraic techniques, which usually puts additional constraints on the structure of the optimal solutions. An example is given by the Kautz-Singleton construction [9], which relies on the adoption of *constant-weight codes*. This means that the columns of the matrices all have the same Hamming weight, i.e., the same number of ones. Thus, the solutions produced by algebraic methods are actually a subset of the space of all DM. This observation leads

us to the following ideas for further improving our heuristic approach in future research:

- Investigate whether the solutions produced by GP (as well as by EDA and GA) can be generated by algebraic techniques. Due to the connection between disjunct matrices and codes, even finding a single DM produced by one of our heuristics which cannot be expressed through any known algebraic method could shed light on new classes of error-correcting codes.
- Restrict the search space explored by EDA, GA, and GP by putting additional constraints on the encoding of the candidate solutions. Taking inspiration from the Kautz-Singleton construction, a possibility could be to enforce a constant number of ones on the columns of the matrices, either at the fitness function level (i.e., as an additional property to optimize) or by using heuristic specifically designed for evolving balanced Boolean functions, such as the discrete Particle Swarm Optimizer described in [11].

Among other possible avenues for further research, we expect that experimenting with other variants of EDA such as 1-order Markov and tree models could yield better performances on this problem. Finally, the note that the optimization approach laid out in this paper tries to construct DM, RM or ADM starting from matrices with the same number of columns as those of the desired optimal solutions. An alternative method worth exploring is to use an *incremental approach* similar to the one proposed in [20], where the authors constructed new orthogonal arrays from old ones by incrementally adding columns.

Acknowledgments

Parts of our work have been inspired by COST Action CA15140 supported by COST (European Cooperation in Science and Technology).

References

1. Balint, G., Bloch, M., Ellis, R., Monson, G., Schulte, A., Schultz, A., Soule, M., Vaden, D.: An investigation of d-separable, $\bar{d}$-separable, and d-disjunct binary matrices. Tech. rep., San Diego State University (2013), http://www.sci.sdsu.edu/math-reu/2013-2.pdf
2. Barg, A., Mazumdar, A.: Group testing schemes from codes and designs. IEEE Trans. Information Theory **63**(11), 7131–7141 (2017)
3. Belazzougui, D., Gagie, T., Mäkinen, V., Previtali, M.: Fully dynamic de bruijn graphs. In: String Processing and Information Retrieval - 23rd International Symposium, SPIRE 2016, Beppu, Japan, 2016, Proceedings. pp. 145–152 (2016)
4. Colbourn, C.J., Dinitz, J.H.: Combinatorial designs. In: Handbook of Discrete and Combinatorial Mathematics. CRC Press (1999)
5. Colbourn, C.J., Ling, A.C.H., Syrotiuk, V.R.: Cover-free families and topology-transparent scheduling for manets. Des. Codes Cryptography **32**(1-3), 65–95 (2004)

6. Damaschke, P., Schliep, A.: An optimization problem related to bloom filters with bit patterns. In: SOFSEM 2018: Theory and Practice of Computer Science - 44th International Conference on Current Trends in Theory and Practice of Computer Science, Krems, Austria, 2018, Proceedings. pp. 525–538 (2018)
7. Du, D., Hwang, F.K., Hwang, F.: Combinatorial group testing and its applications, vol. 12. World Scientific (2000)
8. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Natural Computing Series, Springer (2015)
9. Kautz, W.H., Singleton, R.C.: Nonrandom binary superimposed codes. IEEE Trans. Information Theory **10**(4), 363–377 (1964)
10. Larrañaga, P., Karshenas, H., Bielza, C., Santana, R.: A review on probabilistic graphical models in evolutionary computation. Journal of Heuristics **18**(5), 795–819 (2012)
11. Mariot, L., Leporati, A.: Heuristic search by particle swarm optimization of boolean functions for cryptographic applications. In: Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings. pp. 1425–1426 (2015)
12. Mariot, L., Picek, S., Jakobovic, D., Leporati, A.: Evolutionary algorithms for the design of orthogonal latin squares based on cellular automata. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017. pp. 306–313 (2017)
13. Mariot, L., Picek, S., Jakobovic, D., Leporati, A.: Evolutionary search of binary orthogonal arrays. In: Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part I. pp. 121–133 (2018)
14. Mazumdar, A.: On almost disjunct matrices for group testing. In: Algorithms and Computation - 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings. pp. 649–658 (2012)
15. Mühlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions I. Binary parameters. In: Parallel Problem Solving from Nature - PPSN IV. Lectures Notes in Computer Science, vol. 1141, pp. 178–187. Springer, Berlin (1996)
16. Mhlenbein, H.: The equation for response to selection and its use for prediction. Evolutionary Computation **5**(3), 303–346 (1997). https://doi.org/10.1162/evco.1997.5.3.303
17. Pearl, J.: Causality: Models, Reasoning and Inference. Cambridge University Press (2000)
18. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk` (2008), (With contributions by J. R. Koza)
19. Porat, E., Rothschild, A.: Explicit nonadaptive combinatorial group testing schemes. IEEE Trans. Information Theory **57**(12), 7982–7989 (2011)
20. Safadi, R., Wang, R.: The use of genetic algorithms in the construction of mixed multilevel orthogonal arrays. Tech. rep., OLIN CORP CHESHIRE CT OLIN RESEARCH CENTER (1992)
21. Stinson, D.R., Van Trung, T., Wei, R.: Secure frameproof codes, key distribution patterns, group testing algorithms and related structures. Journal of Statistical Planning and Inference **86**(2), 595–617 (2000)
22. Stinson, D.R., Wei, R.: Generalized cover-free families. Discrete Mathematics **279**(1-3), 463–477 (2004)