How to Use Boltzmann Machines and Neural Networks for Covering Array Generation

Ludwig Kampel[1], Michael Wagner[1], Ilias S. Kotsireas[2], and Dimitris E. Simos[1](✉)

[1] SBA Research, 1040 Vienna, Austria
{lkampel,mwagner,dsimos}@sba-research.org
[2] Wilfrid Laurier University, Waterloo, ON, Canada
ikotsire@wlu.ca

Abstract. In the past, combinatorial structures have been used only to tune parameters of neural networks. In this paper, we employ for the first time, neural networks and Boltzmann machines for the construction of covering arrays (CAs). In past works, Boltzmann machines were successfully used to solve set cover instances. For the construction of CAs, we consider the equivalent set cover instances and use Boltzmann machines to solve these instances. We adapt an existing algorithm for solving general set cover instances, which is based on Boltzmann machines and apply it for CA construction. Furthermore, we consider newly designed versions of this algorithm, where we consider structural changes of the underlying Boltzmann machine, as well as a version with an additional feedback loop, modifying the Boltzmann machine. Last, one variant of this algorithm employs learning techniques based on neural networks to adjust the various connections encountered in the graph representation of the considered set cover instances. Supported by an experimental evaluation our findings can act as a beacon for future applications of neural networks in the field of covering array generation and related discrete structures.

Keywords: Neural networks · Boltzmann machines · Covering arrays

1 Introduction

Various approaches have been applied for the construction and optimization of CAs, for a survey see for example [11]. Artificial Neural Networks (ANN) have been applied successfully in various fields of computer science, especially in optimization [10], and recently significant effort has been spent to replicate the decisions of human experts using artificial intelligence [9]. The covering array generation problem is known to be tightly coupled with hard combinatorial optimization problems, see [5] and references therein.

In this paper, to the best of our knowledge, we employ for the very first time neural network models based on Boltzmann machines (BM) to optimization of

CA generation, i.e. the construction of CAs with a small number of rows. We would like to note that constructing a BM architecture corresponding to a particular optimization problem is a difficult task, since one has to construct the topology of the architecture to reflect the particular nature of a given problem. This difficulty has been first noted in [1], where the authors tried to combine neural computation with combinatorial optimization for a different class of problems.

This paper is structured as follows. In Sect. 2 we give the preliminaries needed for this paper. Next, in Sect. 3 we describe how Boltzmann machines can be used for the construction of covering arrays. In Sect. 4 this approach is extended by means of allowing for different learning strategies. Finally, in Sect. 5 the results of our experiments are documented while Sect. 6 concludes our work.

2 Preliminaries

In this section we give the definitions needed in this paper. Related notions for covering arrays and set covers can be also found in [4] while the needed definitions regarding graphs can be also found in [12].

Definition 1. *A* covering array $\mathsf{CA}(N;t,k,v)$ *is an* $N \times k$ *array* $(\mathbf{c}_1,\dots,\mathbf{c}_k)$ *with the properties that for all* $j \in \{1,\dots,k\}$ *the values in the* j*-th column* $\mathbf{c}_j$ *belong to the set* $\{0,\dots,v-1\}$*, and for each selection* $\{\mathbf{c}_{j_1},\dots,\mathbf{c}_{j_t}\} \subseteq \{\mathbf{c}_1,\dots,\mathbf{c}_k\}$ *of* t *different columns, the subarray that is comprised by the columns* $\mathbf{c}_{j_1},\dots,\mathbf{c}_{j_t}$*, has the property that every* t*-tuple in* $\{0,\dots,v-1\}^t$ *appears at least once as a row. The smallest integer* N *for which a* $\mathsf{CA}(N;t,k,v)$ *exists is called the* covering array number *for* t,k,v *and is denoted as* $\mathsf{CAN}(t,k,v)$*. Covering Arrays achieving this bound are called* optimal.

In this work we only consider binary CAs, i.e. CAs over the alphabet $\{0,1\}$, which we denote as $\mathsf{CA}(N;t,k)$. For given t and k we also say we are given a *CA instance*, when we want to construct a $\mathsf{CA}(N;t,k)$. An example of a $\mathsf{CA}(4;2,3,2)$ is given by the array A in relation (1).

Definition 2. *For positive integers* t*,* k *and* v*, a* t*-way interaction is a set* $\{(p_1,x_1),\dots,(p_t,x_t)\}$ *with the property that* $x_i \in \{0,\dots,v-1\}$*,* $\forall i \in \{1,\dots,t\}$ *and* $1 \le p_1 < \dots < p_t \le k$.

We represent t-way interactions as vectors of length k with t positions specified and the others unspecified (see Example 1). With this notion CAs can be characterized as arrays which rows cover all t-way interactions for given t and k.

Definition 3. *A* set cover *(SC) of a finite set* U *is a set* $\mathcal{S}$ *of non-empty subsets of* U *whose union is* U*. In this context,* U *is called the* universe *and the elements of* $\mathcal{S}$ *the* blocks.

A typical optimization problem for set covers is the *minimal set cover problem*. That is, for given $(U,\mathcal{S})$, to find a subset $\mathcal{C} \subseteq \mathcal{S}$ of minimal cardinality, such that $\bigcup \mathcal{C} = U$. We call $(U,\mathcal{S})$ also an *SC instance*.

Definition 4. *For a graph $G = (V, E)$ (in this work we only consider undirected graphs) with vertex set V and edges $E \subseteq V \times V$, a* vertex cover *is a subset C of V, such that each edge in E is incident to at least one vertex of C. An* independent set *of G is a set of vertices $I \subseteq V$, such that no two vertices in I are adjacent, i.e. $I \times I \cap E = \emptyset$.*

Finally, we also consider the concept of Boltzmann machines, see also [1].

Definition 5. *A Boltzmann machine (BM) is a (stochastic) neural network, with an underlying undirected graph $G = (V, E)$. The neurons correspond to the vertex set V and can be in two states, either* on *or* off*. The edges of the graph correspond to the connections (synapses) between the neurons, i.e. the edge $\{u_i, u_j\} \in E$ represents the symmetric connection between neuron u_i and u_j. A Boltzmann machine M is now defined as a pair (G, Ω), where $\Omega \subseteq \mathbb{R}^E \times \{0,1\}^V$ is a set of allowable states. For $\omega = (w_{e_1}, w_{e_2}, \ldots, w_{e_{|E|}}, u_{v_1}, \ldots, u_{v_{|V|}})$ the vector $w = (w_{e_1}, w_{e_2}, \ldots, w_{e_{|E|}})$ describes the weight w_e of each edge $e \in E$, and $\kappa = (u_{v_1}, \ldots, u_{v_{|V|}})$ describes for all neurons $v_i \in V$ if it is* on *($u_{v_i} = 1$), or* off *($u_{v_i} = 0$). The* consensus function *of the Boltzmann machine M is the function $F : \Omega \to \mathbb{R}$, defined by $F(\kappa) = \sum_{\{i,j\} \in E} w_{\{i,j\}} u_i u_j$.*

3 A Boltzmann Machine for Covering Arrays

In this section we set up a Boltzmann machine that reflects a given CA instance. To this extent, we first explain how the problem of generating a CA can be interpreted as a set cover problem, following [4]. In a second step we recapitulate the work of [3], where Boltzmann machines where successfully used to compute solutions to set cover problems. Our aim in this work is to combine these works so that we can use Boltzmann machines to compute covering arrays.

3.1 Encoding CA Problems as Set Cover Problems

Next, we explain how to interpret the problem of computing a CA as an SC problem. This connection has been explained in an extensive way, for example, in [4], where the interested reader is referred to for the details. Here we content ourself with repeating the key ideas, guided by means of an example. When we want to construct a CA for given strength t and number of columns k, this can be interpreted as an SC instance $(U, \mathcal{S})$, where the universe U consists of all t-way interactions. Each block in $\mathcal{S}$ corresponds to a row that can appear in a CA and is defined as the set of t-way interactions this row covers. To make this connection more explicit, we review Example 3.3 of [4]:

Example 1. Assume we want to construct a $\mathsf{CA}(N; 2, 3, 2)$ with minimal N. We translate this problem into a minimal set cover problem. Each 2-way interaction needs to be covered, thus $U = \{(0,0,-), (0,1,-), (1,0,-), (1,1,-), (0,-,0), (0,-,1), (1,-,0), (1,-,1), (-,0,0), (-,0,1), (-,1,0), (-,1,1)\}$. Each vector of $\{0,1\}^3$ which can appear as a row in a $\mathsf{CA}(N; 2, 3, 2)$ is identified with the set of 2-way interactions it covers, e.g. the row $(0, 0, 1)$ is mapped to the block

$\{(0,0,-),(0,-,1),(-,0,1,)\}$. Thus we get the SC instance $(U,\mathcal{S})$ corresponding to the CA instance with parameters $t=2$ and $k=3$, where

$$\begin{aligned}\mathcal{S}=\{&\{(0,0,-),(0,-,0),(-,0,0,)\},\{(0,0,-),(0,-,1),(-,0,1,)\},\\&\{(0,1,-),(0,-,0),(-,1,0,)\},\{(0,1,-),(0,-,1),(-,1,1,)\},\\&\{(1,0,-),(1,-,0),(-,0,0,)\},\{(1,0,-),(1,-,1),(-,0,1,)\},\\&\{(1,1,-),(1,-,0),(-,1,0,)\},\{(1,1,-),(1,-,1),(-,1,1,)\}\}.\end{aligned}$$

Provided this correspondence, it is therefore possible to map the minimal set cover $\mathcal{C}=\{\{(0,0,-),(0,-,0),(-,0,0,)\},\{(0,1,-),(0,-,1),(-,1,1,)\},\{(1,0,-),(1,-,1),(-,0,1,)\},\{(1,1,-),(1,-,0),(-,1,0,)\}\}$ of $(U,\mathcal{S})$ to the optimal $\mathsf{CA}(4;2,3,2)$

$$A=\begin{pmatrix}0&0&0\\0&1&1\\1&0&1\\1&1&0\end{pmatrix}. \tag{1}$$

3.2 Boltzmann Machines for Set Cover Problems

We now give an overview of how SC instances can be solved with Boltzmann machines, following the work presented in [3], which serves us as a point of reference. A high level view of the procedure we follow can be found in Fig. 1.

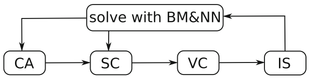


Fig. 1. An overview.

In the following paragraphs we make explicit the connections between set covers, vertex covers and independent sets. Consider a given SC instance $(U,\mathcal{S})$, we can construct an edge labelled graph $G_S=(V,E,\ell)$ representing this instance, as follows. The node set V is defined to be the set of blocks $\mathcal{S}$, such that each block is represented by a node of the graph. For the set of (undirected) edges E, we define $E:=\{\{S_i,S_j\}|S_i\cap S_j\neq\emptyset\}$ and the labelling function of the edges $\ell:E\to\mathcal{P}(U):\{S_i,S_j\}\mapsto S_i\cap S_j$, i.e. we label each edge with the set of elements of U its adjacent vertices cover in common. We call these labels also *label sets*. At this point we would like to remark, that we can assume without loss of generality, that each element of the universe U appears in at least two blocks and hence in at least one label set.

Assume now we are given a vertex cover $\mathcal{V}=\{S_1,\ldots,S_r\}$ of G_S, then $\mathcal{V}$ represents already a set cover of U. This holds, since the vertices (i.e. sets) $S_1,\ldots,S_r$ cover all edges of G_S, the labels of which are subsets of the S_i and already cover the whole universe U. Further in [3] reduced graphs G'_S are considered, where for each element $u\in U$ *exactly one* edge of $E(G_S)$ is allowed to keep u in its label set. Hence, a vertex cover of the reduced graph still constitutes a set cover of $(U,\mathcal{S})$, see Proposition 1 of [3]. Generalizing this approach, in our work we consider reduced graphs $G'_S=(V,E(G'_S))$, where for each $u\in U$ *at least* one edge of $E(G'_S)$ has u in its label set. We thus maintain the property that a vertex cover of a reduced graph G'_S constitutes a set cover of U. We give an example of these two different types of reduced graphs in Example 3 and Fig. 3. Considering that a vertex cover C of a graph $G = (V, E)$ is exactly the complement of an independent set $V \setminus C$ of G (see also Remark 1 of [3]), the analogue of Proposition 1 of [3] also holds for reduced graphs as considered in this work:

Proposition 1. *The complement of a maximal independent set of a reduced graph G'_S (where each element of U appears in the label of at least one edge) is a set cover for $(U, \mathcal{S})$.*

Sketch of Proof. The complement of an independent set is a vertex cover. A vertex cover contains vertices such that each edge is incident to at least one vertex. The label set of an edge is a subset of the sets corresponding to its adjacent vertices. Since each element of U appears in at least one label of an edge of the reduced graph G'_S, the sets corresponding to the nodes of the vertex cover are a set cover of U. $\square$

Before we describe how Boltzmann machines can be used to find independent sets of graphs which yield set covers, respectively CAs for our purpose, we fix some notations and consider some examples. We use the notation $G_{t,k}$ for the graph that corresponds to the set cover instance $(U, \mathcal{S})$, which corresponds again to a CA instance for given t and k, and call it the *underlying graph* of the CA instance.

Example 2. Continuing Example 1 the graph $G_{2,3}$ corresponding to the set cover $(U, \mathcal{S})$ is depicted in Fig. 2. Although self-contained, graph $G_{2,3}$ is not very representative for the general CA instances we have to deal with, since there are exactly two rows that share a 2-way interaction, there is a one-to-one correspondence between edges and 2-way interactions. This is also the reason why we omitted the set notation for the labels, as label sets are singletons in this case.

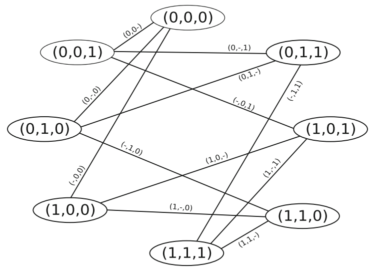


Fig. 2. $G_{2,3}$ underlying the CA instance $t = 2$, $k = 3$.

Example 3. To give a better impression on the instances we have to deal with, we also give an example of a partial graph in Fig. 3, where a subgraph of the graph $G_{2,5}$ is given. Labels occur right next to the edges they belong to. The green coloured 2-way interactions are selected to reside in the label set of the edges. In the middle of Fig. 3 we depict the reduced graph $G'_{2,5}$ as it is described in [3]. Rightmost we show the reduced graph $G'_{2,5}$ as we consider them in this work.

3.3 Computing CAs with Boltzmann Machines

Following [3], the neural network is constructed isomorphic to the graph G'_S, where neurons correspond to vertices and synapses correspond to edges. Each neuron S_i is in state u_i, which can be either *on* or *off*, represented by $u_i \in \{0,1\}$. Synapses, i.e. edges of G'_S, are assigned weights according to Eq. (2). At any time the configuration of the network is defined by the vector of all states $(u_i)_{i \in V}$ of the neurons. For any configuration $\kappa = (u_i)_{i \in V}$ the consensus $F(\kappa)$ is defined as the following quadratic function of the states of the neurons, and the weights of the synapses: $F(\kappa) = \sum_{i,j} \omega(e_{i,j}) u_i u_j$. The defined weights (loops $e_{i,i}$ have positive, other edges have negative weights) and the consensus function F reflect the target of finding a maximal set of vertices, that are not adjacent.[1] Respectively, we want to activate as many non-connected neurons as possible. In fact local maxima of F correspond to maximal (not necessary *maximum*) independent sets of the graph G'_S, which in turn yield set covers, considering the complement on G_S. We refer the interested reader to [3] for more details.

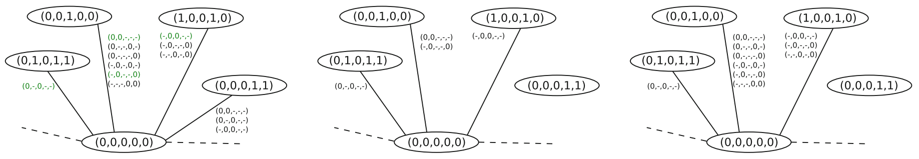


Fig. 3. From left to right: subgraph of the underlying graph of the CA instance $t = 2$, $k = 5$; a subgraph of a reduced graph as it can occur due to the method described in [3]; a subgraph of the reduced graph as consider in our work.

Remark 1. To better illustrate the connections between the different structures, we give an overview of the introduced concepts and notions as follows:

- Rows of CAs correspond to blocks of SCs, which are further mapped to vertices of BMs. These serve as neurons for the devised neural network.
- Analogue t-way interactions correspond to elements of the universe in terms of set covers. These serve as labels of edges that define the weight of the synapses of the devised neural network.

In Algorithm 2 we give a high level description of the algorithm developed in [3], modified in this work in order to be applied to CA instances. The initial weights of the edges (line 3) are set according to Eq. (2). Thereafter a simulated annealing (SA) algorithm is run on the Boltzmann Machine to find a local maximum of the consensus F, where initially all synapses are in state *off* (i.e. the state vector κ equals the all zero vector). A pseudo code of such a simulated annealing algorithm is given in Algorithm 1, taking as input a graph G with weights for the edges, denoted by $\omega(G)$. Further, a starting temperature T_0, a final temperature T_f and a factor α for the cooling schedule is required. In each step a random neuron is selected to change its state. In case the change in the consensus function $\Delta F(\kappa) = (1 - 2u_i)(\omega(e_{ii}) + \sum_j \omega(e_{ij}) u_j)$ is positive the change in state u_i is accepted, otherwise it is refuted with probability $(1 - 1/(1 + \exp(-\Delta F/T))$.

[1] Note that we do not consider vertices as being adjacent to themselves by their loops. In this work we rather use loops to represent the weight of vertices.

The cooling schedule of the SA algorithm is based on the schedule developed by Lundy and Mees [6]. In this cooling schedule it is required that only one iteration is executed at each temperature. In particular, we implemented the cooling schedule (line 11) according to the recursive function $T_{n+1} = T_n/(1 + \alpha T_n)$ where n is the iteration number and α, a small positive value close to zero depending on the instance, that allows for fast convergence.

We describe our algorithmic design using a variety of building blocks. In this way, a compact presentation of the devised algorithms is ensured and also flexibility in terms of their evaluation which is presented in Sect. 5.

The first building block introduced is that of InitialGraph, which is a procedure that transforms the underlying graph $G_{t,k}$ to a subgraph $G'_{t,k}$, to which the Boltzmann machine is reduced and on which the simulated annealing algorithm runs. In all presented versions we instantiate this building block with a randomized procedure, which selects for each t-way interaction a random edge of $E(G_{t,k})$, such that the t-way resides in the label set of the edge. Edges selected this way reside in $G'_{t,k}$ and keep all their labels, where Edges that get not selected are deleted. See also Example 3.

The second building block used to devise our algorithms is that of InitialWeight, which is a procedure that assigns a weight to each edge of G'_S. One way considered to instantiate this building block is via BMweight, assigning the weights as described in [3]

$$\omega(e_{ij}) = \begin{cases} -(\max\{1/|S_i|, 1/|S_j|\} + \epsilon), & i \neq j \\ 1/|S_i|, & i = j \end{cases} \tag{2}$$

Note, that this weighting comes down to a uniform weighting, as $|S_i| = |S_j|$ for all i, j when considering CA instances.

Using these algorithmic building blocks, we can describe the algorithm of [3] applied to CA instances as an instance of Algorithm 2, Instantiating InitialGraph with RandomGraph and InitialWeights according to (2). The simulated annealing algorithm is run once, to find a maximal independent set I on $G'_{t,k}$, the complement of which is returned and constitutes a CA.

Combining the reductions of CAs to SCs and SCs to independent sets on reduced graphs G'_S (Fig. 1), it is possible to state the following corollary of the works presented in [4] and [3] (Theorem 1), which proves the correctness of the previously described algorithm.

Corollary 1. *Maxima of F induce configurations of the BM-network corresponding to Covering Arrays.*

Algorithm 2 serves as a base line for the development of our own learning algorithms, which we describe in the next section.

4 Finding Covering Arrays with Neural Networks

In this section, we describe the algorithms we have devised to find CAs with neural networks. We start with the algorithm described in [3], which serves as a starting point for the development of the algorithms presented in this work, where we incrementally built upon this baseline algorithm and extend it with various features. In detail, as a first extension we consider a weight update for the underlying BM and a second extension introduces a notion of graph update for the underlying graph (which is part of the BM).

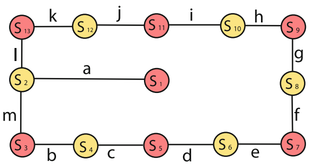
Algorithm 1. SA

```
1: INPUT: $G, \omega(G)$
Require: $T_0, T_f, \alpha$
2: $T \leftarrow T_0$, $\kappa \leftarrow \mathbf{0}$                ▷ $\kappa$ is the vector of states
3: while $T > T_f$ do
4:     randomly choose neuron $S_i$
5:     change state: $u_i \leftarrow u_i + 1(mod\ 2)$
6:     if $\Delta F(\kappa) > 0$ then
7:         keep $\kappa$
8:     else
9:         with probability $1/(1 + \exp(-\Delta F/T))$ keep $\kappa$
10:    end if
11:    $T \leftarrow T/(1 + \alpha T)$
12: end while
13: return $\kappa$
```

Algorithm 2. BMforCA

```
1: INPUT: $t, k$
Require: $\epsilon$
2: $G'_{t,k} \leftarrow$ InitialGraph$(G_{t,k})$
3: $\omega(G'_{t,k}) \leftarrow$ InitialWeight$(G'_{t,k}, \epsilon)$      ▷ Assign weights
4: $I \leftarrow$ SA$(G'_{t,k}, \omega(G'_{t,k}))$
5: return CA$(|V| - |I|; t, k, 2) = V \setminus I$
```

Before we describe the algorithmic extensions we made to Algorithm 2, we would like to mention that our initial experiments with Algorithm 2 were not satisfactory. Due to the good experimental results in [3], reporting to find smaller SCs than other heuristics, we expected that Algorithm 2 would produce CAs with a small number of rows. Yet, our objective is still not achieved, i.e. having a learning algorithm capable of further reducing the number of rows in a CA. We believe that this is the case, since the approach of finding small set covers as the complements of large independent sets of vertices on corresponding graphs is badly suited for graphs that have a relative high *density*, i.e. on average, vertices are highly connected. It seems that the condition of finding an independent set of nodes on the reduced graph G'_S, is too strong of a sufficient condition to actually find small SCs for such instances. To illustrate this we give the following example of a general SC instance, which is highly connected and the algorithm described in [3] is badly suited.



Example 4. Consider the following set cover instance $(U, \mathcal{S})$, $U = \{a, b, c, \ldots, k, l, m\}$ and $\mathcal{S} = \{S_1, \ldots, S_{13}\}$ where $a \in S_i$ for all $i = 1, \ldots, 13$ and $S_1 = \{a, m\}, S_2 = \{a, m, b, c\}, S_3 = \{a, b, c, d\}, S_4 = \{a, c, d, e\}, S_5 = \{a, d, e, f\}, S_6 = \{a, e, f, g\}, S_7 = \{a, f, g, h\}, S_8 = \{a, g, h, i\}, S_9 = \{a, h, i, j\}, S_{10} = \{a, i, j, k\}, S_{11} = \{a, j, k, l\}, S_{12} = \{a, k, l, m\}, S_{13} = \{a, l, m, b\}$. The graph representing the set cover instance $(U, \mathcal{S})$ is the complete graph with 13 vertices. In the figure above we give an example of a reduced graph of this set cover instance, in the sense of [3], i.e. each element of the universe appears as a label of exactly one edge. A maximal (even maximum) independent set of nodes can be identified as $I = \{S_1, S_3, S_5, S_7, S_9, S_{11}, S_{13}\}$. Then the complement $\mathcal{C} = \{S_2, S_4, S_6, S_8, S_{10}, S_{12}\}$ constitutes a minimal vertex cover of this graph, and hence $\mathcal{C}$ is a cover of the universe U. Though, $\mathcal{C}$ is not a minimal set cover, since $\mathcal{C}' = \{S_2, S_5, S_8, S_{11}\}$ also constitutes a cover of U of smaller size. In fact it is not hard to see that $\mathcal{C}'$ is a minimal set cover of $(U, \mathcal{S})$.

We take this example as a further motivation to modify the approach of [3], towards relaxing the target of finding an independent set on the reduced graph $G'_{\mathcal{S}}$. Finding independent sets is encoded in the consensus function F (as introduced in Subsect. 3.3), that characterizes independent sets through local maxima, as long as weights of vertices are positive, and the weights of the edges are smaller than the negative vertex weights. Using the same consensus function, our approach is to increase the edge weights, such that local maxima of F can originate also from vertex sets containing adjacent vertices. From this we gain, that when maximizing consensus F more neurons are in state *on*, hence the complement, the neurons in state *off* will be less than in the original approach of [3]. On the downside, we might lose the property that these neurons in state *off* translate to a set cover, respectively a CA in our case. We address this issue by evaluating the returned solution, and updating the weights of the edges. Then we maximize the consensus F for the updated instance. The key idea behind this approach is, that the neural network decreases the weights of those edges that carry elements as labels that where not covered in the previous iteration. This modifies the network, such that in the next iteration it is less likely that all neurons connected by such edges are turned *on* and hence some will remain turned *off*, which means they will be part of the suggested solution of the set cover. We detail our edge updates and additional learning features in the next section. The experimental results provided in Sect. 5 fully justify this approach.

4.1 Weight Updates: A First Step Towards Learning

New Initial Weights. The first change we made to the algorithm as it is presented in [3] is that we changed the computation of the edge weights. This is done by assigning the weights as a function of $|S_i \cap S_j|$ instead of $\max(|S_i|, |S_j|)$. The number of t-way interactions two rows S_i and S_j cover in common depends on the number of positions in which these rows are equal, we hence can compute

$|S_i \cap S_j| = \binom{k-d_{ij}}{t}$, where d_{ij} denotes the *hamming distance*[2] of the two rows S_i and S_j. We thus considered two additional instantiations of the building block InitialWeights:

- HDweight$_1$: $\omega(e_{ij}) = -\binom{k-d_{ij}}{t} \cdot 1/\binom{k}{t}$
- HDweight$_1$: $\omega(e_{ij}) = -\frac{\binom{k}{d_{ij}}^{-1}}{\binom{k}{\lfloor k/s \rfloor}}$

for $i \neq j$ in both cases and $\omega(e_{ii}) = 1$ for the loops. In Sect. 5 we will also compare the results when the initial edge weighting HDweight$_1$ and HDweight$_2$ are used in Algorithm 2.

Weight Updates: Learning in Epochs. Another improvement to the algorithm presented in [3] was achieved by extending it by means of epochs in which the weights of the edges connecting neurons get updated. This algorithmic extension was implemented for two reasons: First and foremost we wanted the neural network to be able to adapt to given problem instances. Second, since we gave the neural network more freedom by weakening the consensus function F by assigning larger weights to edges using our newly introduced versions of InitialWeights, we are not guaranteed anymore that the output of the SA algorithm constitutes an independent set and hence its complement must not constitute a CA.

In short, we lose the guarantee of a feasible solution as it was guaranteed by Corollary 1. Therefore we enabled the neural network with the capability to increase or decrease the weight of edges, depending on whether the elements in their label sets were covered in the solution returned in the previous epoch or not. This new algorithmic building block WeightUpdate can be described as procedure that modifies the weight of the edges of the underlying graph $G_{t,k}$, in the following way. Whenever a t-way interaction is covered more than twice in the solution of the previous epoch, all edges that have this interaction in its label set get an increment of $1/cov$ in weight (recall that edge weights are initialized negative), where cov is the total number of covered t-way interactions. Opposite, every edge carrying an interaction that was not covered by the solution returned in the previous epoch gets a proportional decrement in weight. The weights of some edges get smaller and in the next epoch it is less likely that both vertices adjacent to such an edge will be in the independent set to be constructed. This in turn means, that at least one of the vertices will be in the complement, i.e. the return of the next epoch.

We present Algorithm 3 in terms of a pseudocode. First a reduced graph $G'_{t,k}$ is constructed and initial weights are assigned. Further a global best solution is recorded in I_{max}, which is initially set empty. Then a number e of epochs is run, where in each epoch x runs of SA are executed, where we keep the solution I maximizing the consensus F over these x runs. The weight update CoverWeight is based on this solution I. If I is larger than I_{max} we store it accordingly, before entering the next epoch. Finally if $V \setminus I$ covers all t-way interactions, a CA is found and returned.

[2] The hamming distance of two vectors is defined as the number of positions in which these two disagree.

4.2 Graph Updates: An Additional Layer for Learning

In our experiments we recognized that the quality of the solution produced by Algorithm 3 highly depends on the random graph that is chosen in the initialization step.

Algorithm 3. BMforCAlearning

```
1: INPUT: $t, k$
Require: $e$, $x$
2: $G'_{t,k} \leftarrow$ InitialGraph($G_{t,k}$), $\omega(G'_{t,k}) \leftarrow$ InitialWeights($G'_{t,k}$)    ▷ Initialization
3: $I_{max} \leftarrow \emptyset$
4: while epoch count $\leq$ e do
5:     run SA on $G'_{t,k}$ $x$ times, store $I$ maximizing consensus
6:     $\omega(G'_{t,k}) \leftarrow$ WeightUpdate($G'_{t,k}, I$)
7:     if $|I_{max}| < |I|$ then
8:         $I_{max} \leftarrow I$
9:     end if
10: end while
11: if $V \setminus I$ covers all $t$-way interactions then return CA($|V| - |I_{max}|; t, k, 2) = V \setminus I_{max}$
12: else return $V \setminus I_{max}$ with additional coverage information
13: end if
```

Thus we strived to enhance the learning rate of the neural network with a functionality capable to update the reduced graph that the Boltzmann machine runs on. We describe this additional layer of learning next. A pseudo code description can be seen in Algorithm 4. The graph updates essentially happen in an additional layer of learning phases, built upon Algorithm 3. Therefore, the initialization as well as lines 6–12 of Algorithm 4 are the same as the initialization and the lines 4–10 of Algorithm 3 where variable I_{max} gets renamed to I_{learn}). Around these epochs n learning phases are run, where at the beginning of each learning phase the I_{learn} parameter is reset to the empty set. At the end of each learning phase a graph update based on I_{max} occurs and a bias update based to the best solution I_{learn} found during this learning phase. Both procedures act on the underlying graph $G_{t,k}$ and are explained more detailed as follows.

For the key procedure GraphUpdate we introduce the following instances:

– RandomGraph: This procedure selects a new random graph, just as in the random initialization.
– BestEdges: In each learning phase a subset L of the nodes (respectively rows) of $V \setminus I_{max}$ is randomly selected. For each row in L we flip a random position to create a second row, to which we draw an edge in the graph. By only flipping one position we generate a row that shares the maximal number of t-way interactions with the original row. The edge thus constructed has a large label set. Thereafter for each t-way interaction that is not covered by any of the rows in L, we generate a random edge having this interaction as a label, just as in InitialGraph. With this strategy the neural network can reduce the number of edges in the new reduced graph.

To guide the neural network and enable it to learn from solutions previously found, we added the additional functionality of BiasUpdate. The bias update acts on the neurons, rather than on the synapses of the neural network. In our encoding it can be realized as a weight update, acting exclusively on the loops, by adding a certain, relatively small, δ to the weight of the loops. The bias update is a way to reward vertices that were part of previous solutions, so that the Boltzmann network has a larger tendency to include them in future solutions. This is due to the structure of the consensus function F (see Subsect. 3.3) which value increases whenever a vertex with an increased weight $\omega(e_{ii})+\delta$ is activated, instead of a vertex with edge weight $\omega(e_{ii})$. Vertices being part of I_{learn} in several learning phases are incrementally rewarded through this bias update.

Remark 2. Note that due to bias updates, and also updates of edge weights the cumulative weight in the whole network is not constant over several learning phases. Adopting our weight updates such that the total weight of the network is constant over time is considered as part of future work.

Algorithm 4. BMforCAlearningGraph

```
 1: INPUT: $t, k$
Require: $e$, $x$, $n$, $\delta$
 2: $G'_{t,k} \leftarrow$ InitialGraph($G_{t,k}$), $\omega(G'_{t,k}) \leftarrow$ InitialWeights($G'_{t,k}$)    ▷ Initialization
 3: $I_{max} \leftarrow \emptyset$
 4: while learning phases $\leq n$ do
 5:     $I_{learn} \leftarrow \emptyset$
 6:     while epoch count $\leq$ e do
 7:         run SA on $G'_{t,k}$ $x$ times, store $I$ maximizing consensus
 8:         $\omega(G'_{t,k}) \leftarrow$ WeightUpdate($G'_{t,k}, I$)
 9:         if $|I_{learn}| < |I|$ then
10:             $I_{learn} \leftarrow I$
11:         end if
12:     end while
13:     if $|I_{max}| < |I_{learn}|$ then
14:         $I_{max} \leftarrow I_{learn}$
15:     end if
16:     $G'_{t,k} \leftarrow$ GraphUpdate($G_{t,k}, I_{max}$)
17:     $G_{t,k} \leftarrow$ BiasUpdate($I_{learn}, \delta$)
18: end while
19: return $V \setminus I_{max}$ with coverage information
```

5 Experimental Evaluation

In this section we report experimental results for different configurations of our algorithms which serves as a proof of concept for their validity and efficiency. Tuning the parameters of neural networks for search problems has been subject to a number of related works (e.g. with *genetic algorithms* [2] or combinatorial approaches [7,8]) but an evaluation in that direction is beyond the scope of this paper, and is left for future work. Once, again here we want to demonstrate the premise of our approach especially when compared to past works related with BMs and SCs. We implemented the algorithms in C# and performed the experiments in a workstation equipped with 8 GB of RAM and an i5-Core. In the experiments conducted we used the following settings regarding the simulated annealing algorithm. Temperatures and the factor α were set as $T_0 = 1$, $T_f = 0.001$, $\alpha = 0.005$. The number of inner SA cycles x for configurations of Algorithm 3 was set to 5. One SA cycle takes around 20 ms of execution time. Finally we would like to remark that although the numbers seem very small, bear in mind that for a CA instance (t, k), the underlying graphs grow exponentially in k, e.g. for the CA instance $(2, 10)$ the underlying graph $G_{2,10}$ has 1024 vertices, each having 1012 edges to other vertices.

5.1 Tested Configurations of Algorithm 2

With our first experiments we compare different Configurations of Algorithm 2, using different instantiations of InitialWeights. Configuration 2.1 uses BMweight, Configuration 2.2 uses HDweight$_1$ and Configuration 2.3 uses HDweight$_2$. Table 1 documents the results of our experiments. In the first column we specify the CA instance, the second column headed by CAN lists the number of rows of the respective optimal CAs. In the remaining columns, headed by *min, avg, max* we document the smallest, average and maximal size of the generated *covering arrays* for each configuration. Additionally in the column headed by *avg % cov* we document the average percentage of t-way interactions covered over all generated arrays. Since we deal with randomized algorithms (recall that the procedure InitialGraph is randomized), we executed 100 individual runs of each configuration. Note that Configuration 2.1 always returns a CA, due to Corollary 1. As discussed in Sect. 4 we abandoned the conceptions of [3] with our weight initialization, which is the reason why the other two configurations also return arrays with less than 100% coverage of t-way interactions. Nevertheless considering the sizes of the generated CAs, we can see that the algorithms with the initial weights computed with HDweight$_1$ and HDweight$_2$ generate much smaller covering arrays. Especially Configuration 2.2, producing the smallest covering arrays of these three versions, seems to prioritize smaller size over coverage of t-way interactions, having also the smallest percentage in t-way interactions covered over all returned CAs. In our evaluation of Configuration 2.1 we could not produce amazing results as documented in [3] achieved for general set cover instances. We believe this is mostly due to the graphs underlying the CA instances, being very dense compared to underlying graphs of general SC instances.

Summarizing these experiments we can see that the initial weighting of edges in the graph, respectively of synapses in the neural network, is crucial for the quality of the output of the tested algorithms.

Table 1. Results of the experiments with configurations of Algorithm 2.

(t, k)	CAN	Configuration 2.1				Configuration 2.2				Configuration 2.3			
		min	*avg*	*max*	*avg % cov*	*min*	*avg*	*max*	*avg % cov*	*min*	*avg*	*max*	*avg % cov*
(2,3)	4	4	4	4	100	4	4	4	51.5	4	4	4	100
(2.4)	4	7	8.3	10	100	5	6.14	7	85.04	5	6.15	7	93.71
(2.5)	5	13	15.16	18	100	6	8.22	10	94.75	8	10.25	12	98.83
(2.6)	6	23	26.9	31	100	9	11.90	15	98.42	12	15.22	19	99.63
(2.7)	6	41	45.62	51	100	11	13.63	17	98.33	18	23.39	29	99.95
(2.8)	6	65	72.37	81	100	11	14.60	19	97.26	23	28.69	34	99.98
(2.9)	6	97	106.04	116	100	12	13.77	15	94.74	24	31.75	39	99.99
(2.10)	6	138	146.36	158	100	12	12.67	14	86.23	25	33.22	42	99.99

5.2 Tested Configurations of Algorithm 3

In this subsection we document the results of our experiments with different configurations of Algorithm 3, to evaluate the efficiency of the introduced weight update, in combination with the different weight initializations. Thus we compared Configuration 3.1 using BMWEIGHT, Configuration 3.2 using HDWEIGHT$_1$ and Configuration 3.3 using HDWEIGHT$_2$, each using the WEIGHTUPDATE. The results can be found in Table 2, which have been generated over 10 individual runs for each configuration. Summarizing, first and foremost it is remarkable, that the deployed learning in form of weight updates nullified the severe difference in the number of rows of the generated CAs, as it is witnessed in the versions of Algorithm 2. Further it is notable that due to the weight update the number of rows of the smallest CAs generated decreases, even when comparing to Configuration 2.2 which scaled the best in the previous subsection. Additionally all Configurations always returned CAs (attained 100% coverage of t-way interactions). Hence, we omit the column with the average coverage information.

Table 2. Results of experiments with configurations of Algorithm 3.

(t, k)	CAN	Configuration 3.1			Configuration 3.2			Configuration 3.3		
		min	*avg*	*max*	*min*	*avg*	*max*	*min*	*avg*	*max*
(2,3)	4	4	4	4	4	4	4	4	4	4
(2.4)	4	5	6.2	7	5	5.9	6	5	6	7
(2.5)	5	6	6.8	8	6	6.2	7	6	6.5	8
(2.6)	6	6	7.9	9	6	7.8	9	6	7.6	9
(2.7)	6	8	9.6	11	6	9.2	12	7	9.9	12
(2.8)	6	10	11.2	13	10	11.1	12	9	10.8	12
(2.9)	6	9	11.6	13	11	13	15	11	12.6	14
(2.10)	6	11	13.1	15	11	12.9	14	11	13.1	14

5.3 Tested Configurations of Algorithm 4

We evaluated the two configurations of Algorithm 4 as given by Table 3, Configuration 4.1 uses the graph update BestEdges and Configuration 4.2 uses the graph update RandomGraph. We ran them for the CA instances $(t = 2, k = 6)$ and $(t = 3, k = 5)$, where we limited them to 60 and 70 learning phases respectively. Each learning phase contains 20 epochs, and a bias update at the end, where a the weight of the vertices (i.e. rows) in I_{learn} is increased by 0.01. For the BestEdges graph update 50% of the rows in $V \setminus I_{max}$ were used to construct the new graph. The graphs in Figs. 4a and b depict the evaluation of the best found solution after each learning phase which is normalized to the ideal solution. These experiments show that the Configuration using BestEdge as graph update converges faster towards the ideal solution.

Table 3. Benchmark configurations for Algorithm 4.

Building block	Configuration	
	Config. 4.1	Config. 4.2
InitialWeight	HDweight$_1$	HDweight$_1$
WeightUpdate	On	On
GraphUpdate	BestEdges	RandomGraph
BiasUpdate	On	On

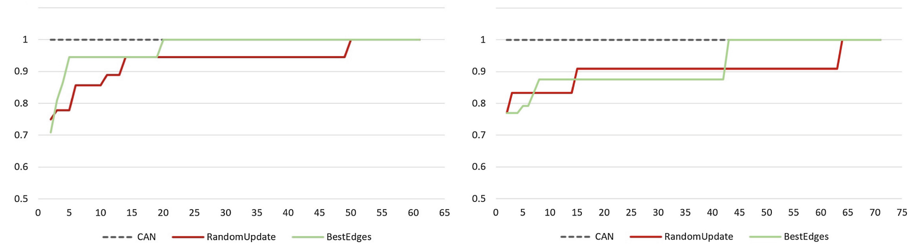


Fig. 4. Configuration 4.1 (green) and 4.2 (red) for the CA instances $(t = 2, k = 6)$ (left) and $(t = 3, k = 5)$ (right). (Color figure online)

6 Conclusion

The cornerstone of this paper is the use of artificial neural networks for CA problems, where we presented for the first time neural models for the construction of covering arrays. Combining the works of [4] and [3], we were able to devise Boltzmann machines for the construction of covering arrays and enhance them with learning capabilities in form of weight updates, graph updates and bias updates. The first experiment results confirm that the application of neural networks to the CA generation problem can lead to optimal solutions and pave the way for future applications.

Acknowledgements. This research was carried out as part of the Austrian COMET K1 program (FFG).

References

1. Aarts, E., Korst, J.: Simulated Annealing and Boltzmann Machines. Wiley, Hoboken (1988)
2. Bashiri, M., Geranmayeh, A.F.: Tuning the parameters of an artificial neural network using central composite design and genetic algorithm. Scientia Iranica **18**(6), 1600–1608 (2011)
3. Hifi, M., Paschos, V.T., Zissimopoulos, V.: A neural network for the minimum set covering problem. Chaos, Solitons Fractals **11**(13), 2079–2089 (2000)
4. Kampel, L., Garn, B., Simos, D.E.: Covering arrays via set covers. Electron. Notes Discrete Math. **65**, 11–16 (2018)
5. Kampel, L., Simos, D.E.: A survey on the state of the art of complexity problems for covering arrays. Theoret. Comput. Sci. **800**, 107–124 (2019)
6. Lundy, M., Mees, A.: Convergence of an annealing algorithm. Math. Program. **34**(1), 111–124 (1986)
7. Ma, L., et al.: Combinatorial testing for deep learning systems. arXiv preprint arXiv:1806.07723 (2018)
8. Pérez-Espinosa, H., Avila-George, H., Rodriguez-Jacobo, J., Cruz-Mendoza, H.A., Martínez-Miranda, J., Espinosa-Curiel, I.: Tuning the parameters of a convolutional artificial neural network by using covering arrays. Res. Comput. Sci. **121**, 69–81 (2016)
9. Silver, D., et al.: Mastering the game of Go without human knowledge. Nature **550**(7676), 354 (2017)
10. Smith, K.A.: Neural networks for combinatorial optimization: a review of more than a decade of research. INFORMS J. Comput. **11**(1), 15–34 (1999)
11. Torres-Jimenez, J., Izquierdo-Marquez, I.: Survey of covering arrays. In: 2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 20–27, September 2013
12. Vazirani, V.V.: Approximation Algorithms. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-662-04565-7