Inversion of Mutually Orthogonal Cellular Automata

Luca Mariot(✉) and Alberto Leporati

Dipartimento di Informatica, Sistemistica e Comunicazione,
Università degli Studi Milano-Bicocca, Viale Sarca 336, 20126 Milan, Italy
{luca.mariot,alberto.leporati}@unimib.it

Abstract. Mutually Orthogonal Cellular Automata (MOCA) are sets of bipermutive CA which can be used to construct pairwise orthogonal Latin squares. In this work, we consider the inversion problem of pairs of configurations in MOCA. In particular, we design an algorithm based on coupled de Bruijn graphs which solves this problem for generic MOCA, without assuming any linearity on the underlying bipermutive rules. Next, we analyze the computational complexity of this algorithm, remarking that it runs in exponential time with respect to the diameter of the CA rule, but that it can be straightforwardly parallelized to yield a linear time complexity. As a cryptographic application of this algorithm, we finally show how to design a $(2, n)$ threshold Secret Sharing Scheme (SSS) based on MOCA where any combination of two players can reconstruct the secret by applying our inversion algorithm.

Keywords: Cellular automata · Latin squares
Secret sharing schemes · de Bruijn graph

1 Introduction

The *inversion problem* is one of the oldest research questions investigated in the field of *Cellular Automata* (CA). Indeed, the first results in this aspect of CA theory dates back at least to Hedlund [4] and Richardson [14]. Stated informally, the inversion problem consists in determining a preimage of a given configuration under the action of a surjective CA. When dealing with the specific class of *reversible* CA, one can compute such unique preimage in parallel by applying an *inverse CA* to the desired configuration.

However, the general case of surjective CA usually requires the specification of an *inversion algorithm* which computes a preimage in a *sequential* way, starting from the knowledge of the states of some of its cells. Sutner [17] was among the first to describe this inversion algorithm using the *de Bruijn graph* representation of CA. More specifically, he showed that a preimage of a configuration corresponds to a *path* on the vertices of the de Bruijn graph associated to the CA, where the edges are labeled by the cells of the configuration. The existence of such a path is guaranteed under the assumption that the CA global rule is surjective.

G. Mauri et al. (Eds.): ACRI 2018, LNCS 11115, pp. 364–376, 2018.
https://doi.org/10.1007/978-3-319-99813-8_33

De Bruijn graphs turned out to be a very useful tool to address several interesting questions related to the inversion problem, such as studying the spatial periods of surjective CA preimages [10] and solving the parity problem through CA [2].

A recent research thread involving the inversion problem concerns *Mutually Orthogonal Latin Squares* (MOLS) generated by CA. In particular, it has been shown in [7] that CA with *bipermutive* local rules can be used to define Latin squares, and pairs of linear bipermutive rules whose associated polynomials are coprime generate orthogonal Latin squares. The idea of the construction is to split the CA initial configuration in two parts, in order to index the rows and the columns of the squares. Then, the final configurations obtained by applying two linear bipermutive rules with coprime polynomials are used to fill the two entries in the square at the coordinates specified by the initial configuration. In what follows we refer to a pair of bipermutive CA generating orthogonal Latin squares as *Orthogonal Cellular Automata* (OCA), and to a set of pairwise OCA as *Mutually Orthogonal Cellular Automata* (MOCA).

It can be remarked that any pair of OLS defines a permutation between the Cartesian product of the rows/columns sets and the overlapped entries. Hence, starting from a pair of final configurations generated by two OCA, an interesting problem is to reconstruct the unique preimage (i.e. the row and column coordinates) which generated them.

The aim of this paper is to investigate the inversion problem in MOCA, without assuming any linearity of the underlying local rules. As a matter of fact, the inversion of OCA defined by *linear* rules has already been settled in [7], and it basically amounts to inverting a *Sylvester matrix*. Consequently, in this work we focus on pairs of OCA defined by general bipermutive rules, whose exhaustive and heuristic constructions have already been addressed in [8,11].

We leverage on the de Bruijn graph representation to solve the inversion problem. In particular, we design an algorithm which, given as inputs the *coupled de Bruijn graph* of two nonlinear OCA and a pair of final configurations, computes their unique preimage by using a variant of *Depth-First Search* (DFS). We remark in particular that the computational complexity of this algorithm is exponential in the diameter of the OCA rules. Nonetheless, we also show that this algorithm can be straightforwardly parallelized with respect to the initial DFS calls, thus yielding an overall linear time complexity.

As an application of our inversion algorithm, we design a perfect *secret sharing scheme* based on MOCA where every pair of players can reconstruct the secret, while any single player cannot gain any information about it. More specifically, we show that the reconstruction phase consists in the application of the inversion algorithm on the two shares of the players, using the coupled de Bruijn graph of the OCA that the dealer used to compute such shares.

The rest of this paper is organized as follows. Section 2 covers all basic definitions and results concerning cellular automata, orthogonal Latin squares and secret sharing schemes used to prove the results of the paper, addressing the inversion problem in the case of MOCA defined by nonlinear bipermutive rules.

Section 4 describes the application of our inversion algorithm to the design of a $(2, n)$ threshold secret sharing scheme. Finally, Sect. 5 summarizes the key findings of the paper and puts them into perspective.

2 Preliminary Definitions

In this section, we recall the basic definitions and notions which we will use in the rest of the paper. In particular, Sect. 2.1 covers all necessary background about CA and their representation based on de Bruijn graphs. Section 2.2 gives the basic definitions regarding orthogonal Latin squares and how they can be used to construct perfect $(2, n)$ secret sharing schemes. Section 2.3 briefly reviews the construction of OLS by means of linear OCA and the exhaustive and heuristic search of OLS by nonlinear OCA.

2.1 Cellular Automata

Throughout this work, we focus on one-dimensional *No Boundary Cellular Automata* (NBCA), formally defined as follows:

Definition 1. *Let Σ be a finite alphabet and $n, d \in \mathbb{N}$ with $n \geq d$. Additionally, let the function $f : \Sigma^d \to \Sigma$ be a* local rule *of* diameter *d. The* No Boundary Cellular Automaton *(NBCA) $F : \Sigma^n \to \Sigma^{n-d+1}$ is the vectorial function defined for all $x \in \Sigma^n$ as*

$$F(x_1, \cdots, x_n) = (f(x_1, \cdots, x_d), f(x_2, \cdots, x_{d+1}), \cdots, f(x_{n-d+1}, \cdots, x_n)). \quad (1)$$

Function F is also called the CA global rule*.*

In other words, an NBCA can be viewed as an array of $n \geq d$ cells, where each of the leftmost $n - d + 1$ cells computes its next state by evaluating rule f on the neighborhood formed by itself and the $d - 1$ cells to its right. In particular, the rightmost $d - 1$ cells of the array are ignored, so that the size of the CA "shrinks" by $d - 1$ cells upon application of the global rule F.

In the rest of this paper, we assume that the state alphabet Σ is the *finite field* with two elements $\mathbb{F}_2 = \{0, 1\}$. In this case, a NBCA can be interpreted as a particular kind of *vectorial Boolean function* $F : \mathbb{F}_2^n \to \mathbb{F}_2^{n-d+1}$, where each *coordinate function* $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ defining the i-th output value corresponds to the local rule applied to the neighborhood of the i-th cell. Since in this case the local rule is a single-output d-variable Boolean function $f : \mathbb{F}_2^d \to \mathbb{F}_2$, it can be uniquely represented by the 2^d-bit output column of its *truth table*, which we denote by Ω_f. In the CA literature it is customary to identify a local rule f by its *Wolfram code*, which is the decimal encoding of its truth table Ω_f.

A local rule $f : \mathbb{F}_q^d \to \mathbb{F}_2$ is called *right* (respectively, *left*) *permutive* if, by fixing the values of the leftmost (respectively, rightmost) $d - 1$ cells to any value $\tilde{x} \in \Sigma^{d-1}$, the resulting restriction $f_{\tilde{x}} : \Sigma \to \Sigma$ is a permutation over Σ. Moreover, f is called *bipermutive* if it is both left and right permutive. When

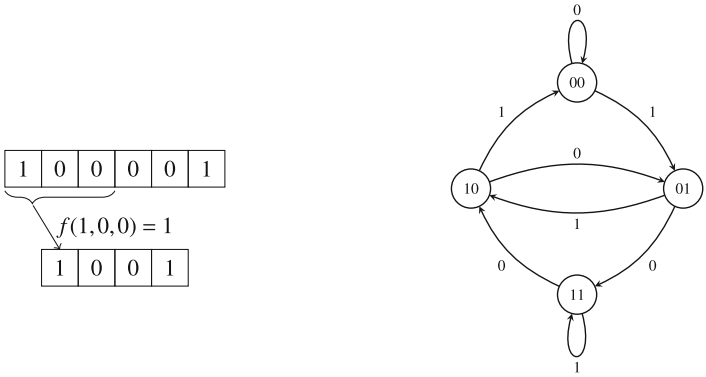

Fig. 1. Example of NBCA defined by rule 150, together with its de Bruijn graph.

$\Sigma = \mathbb{F}_2$, a bipermutive rule $f : \mathbb{F}_2^d \to \mathbb{F}_2$ is defined for all $x = (x_1, \cdots, x_d) \in \mathbb{F}_2^d$ as:

$$f(x_1, \cdots, x_d) = x_1 \oplus g(x_2, \cdots, x_{d-1}) \oplus x_d, \tag{2}$$

where $g : \mathbb{F}_2^{d-2} \to \mathbb{F}_2$ is a $(d-2)$-variable Boolean function.

Another common way for representing a CA is through its *de Bruijn graph*. Let us assume that $u, v \in \Sigma^n$ are two strings over the alphabet Σ of length n such that $u = u_1 x$ and $v = x v_1$, where $u_1, v_1 \in \Sigma$ and $x \in \Sigma^{n-1}$ is a string of length $n-1$. In other words, u and v *overlap* respectively on the rightmost and leftmost $n-1$ symbols. The *fusion* between u and v is the string $z = u \odot v$ of length $n+1$ obtained by adding to u the last symbol of v [17]. Then, one can formally define the de Bruijn graph associated to a CA as follows:

Definition 2. *Let $F : \Sigma^{\mathbb{Z}} \to \Sigma^{\mathbb{Z}}$ be a CA defined by a local rule $f : \Sigma^d \to \Sigma$ of diameter d. The* de Bruijn graph *associated to F is the directed labeled graph $G_{DB}(f) = (V, E, l)$ where $V = \Sigma^{d-1}$ and such that for any $v_1, v_2 \in V$, one has $(v_1, v_2) \in E$ if and only if there exists $z \in \Sigma^d$ such that $z = v_1 \odot v_2$. The label function $l : E \to \Sigma$ on the edges is defined for all $(v_1, v_2) \in E$ as $l(v_1, v_2) = f(v_1 \odot v_2)$.*

Stated otherwise, the vertices of the de Bruijn graph correspond to all possible blocks of $d-1$ cells. Two vertices v_1 and v_2 are connected by an edge if and only if they overlap respectively on the rightmost and leftmost $d-1$ cells, and the label on this edge is obtained by computing the CA local rule on the fusion of v_1 and v_2. Figure 1 depicts an example of binary NBCA $F : \mathbb{F}_2^6 \to \mathbb{F}_2^4$ induced by the local rule $f(x_i, x_{i+1}, x_{i+2}) = x_i \oplus x_{i+1} \oplus x_{i+2}$, whose Wolfram code is 150, together with its de Bruijn graph.

2.2 Orthogonal Latin Squares and Secret Sharing Schemes

Given $N \in \mathbb{N}$, let us denote by $[N]$ the set $\{1, \cdots, N\}$. Then, one can formally define orthogonal Latin squares as follows:

Definition 3. *A* Latin square *L of order $N \in \mathbb{N}$ is a $N \times N$ matrix whose rows and columns are permutations of $[N]$, i.e. every element of $[N]$ occurs exactly once in each row and each column. Two Latin squares L_1, L_2 of order N are called* orthogonal *if for all distinct pairs of coordinates $(i_1, j_1), (i_2, j_2) \in [N] \times [N]$ one has*

$$(L_1(i_1, j_1), L_2(i_1, j_1)) \neq (L_1(i_2, j_2), L_2(i_2, j_2)), \tag{3}$$

that is, the superposition *of L_1 and L_2 yields all possible pairs in the Cartesian product $[N] \times [N]$.*

Remark 1. Two orthogonal Latin squares L_1, L_2 of order $N \in \mathbb{N}$ induce a *permutation* $\pi : [N] \times [N] \to [N] \times [N]$ over the Cartesian product $[N] \times [N]$, which is defined as

$$\pi(i, j) = (L_1(i, j), L_2(i, j)) \tag{4}$$

for all $(i, j) \in [N] \times [N]$.

A set n pairwise orthogonal Latin squares of order $[N]$ is denoted as $n-$ MOLS (*Mutually Orthogonal Latin Squares*). Figure 2 reports an example of orthogonal Latin squares of order $N = 4$, together with their superposition.

1	3	4	2
4	2	1	3
2	4	3	1
3	1	2	4

1	4	2	3
3	2	4	1
4	1	3	2
2	3	1	4

1,1	3,4	4,2	2,3
4,3	2,2	1,4	3,1
2,4	4,1	3,3	1,2
3,2	1,3	2,1	4,4

Fig. 2. Orthogonal Latin squares of order $N = 4$.

Orthogonal Latin squares turn out to have several applications in cryptography and coding theory [5,16], one of the most interesting being *secret sharing schemes* (SSS). Informally speaking, a SSS is a procedure which enables a trusted party (called the *dealer*) to share a *secret* S among a set of n players. In particular, the players receive *shares* of the secret from the dealer, and only certain *authorized subsets* of players specified in an *access structure* can reconstruct the secret by combining together their shares. A SSS is called *perfect* if any subset not belonging to the access structure cannot determine the secret (in an information-theoretic sense).

In this work we focus mainly on perfect $(k, n)-$ *threshold* SSS, where the authorized subsets are those having cardinality at least k. Hence, any combination of k shares is enough to uniquely determine the secret, while knowing $k - 1$ or less shares keeps any value of the secret equally likely.

The connection between perfect threshold SSS and orthogonal Latin squares is established by the following result [16]:

Theorem 1. *A perfect* $(2, n)-$ *threshold SSS exists if and only if there exists a set of* n *MOLS of order* N*.*

The *setup phase* of a $(2, n)-$ threshold SSS from a set of n MOLS $L_1, \cdots, L_n$ goes as follows. First, the secret S is represented as a row $i \in [N]$ of the squares, and the dealer randomly chooses a column $j \in [N]$. Then, for each $m \in \{1, \cdots, n\}$, the dealer secretly sends to the m-th player the share $B_m = L_m(i, j)$, i.e. the entry of the m-th Latin square at row i and column j. Finally, the dealer publishes the Latin squares $L_1, \cdots, L_n$.

In the recovery phase, any pair of players p, q respectively holding shares B_p, B_q can recover the secret simply by overlaying the two public Latin squares L_p, L_q. Since L_p and L_q are orthogonal, the pair of shares (B_p, B_q) occurs at a single pair of coordinates (i, j), the row of which is the secret S. Conversely, if p tries to determine the secret on her own without knowing the share B_q, there will be exactly N pairs $(B_p, \cdot)$ in the overlay of the two Latin squares, due to the fact that L_p and L_q are orthogonal. A symmetric argument holds when q tries to determine S by herself without knowing B_p. Hence, the knowledge of a single share leaves the value of the secret completely undetermined, which makes the scheme perfect.

2.3 Construction of OLS by CA

We now describe how CA can be employed to obtain orthogonal Latin squares, briefly recalling the construction reported in [7]. In what follows, given a binary vector $x \in \mathbb{F}_2^n$, we will denote by $\phi(x) \in \{1, \cdots, 2^n\}$ the integer number corresponding to the decimal representation of $x+1$. On the contrary, for any integer number $i \in \{1, \cdots, 2^n\}$, $\psi(i) \in \mathbb{F}_2^n$ will stand for the n-bit binary representation of $i-1$. Notice that $\phi = \psi^{-1}$ and $\psi = \phi^{-1}$.

Let $F : \mathbb{F}_2^{2(d-1)} \to \mathbb{F}_2^{d-1}$ be a CA based on a local rule $f : \mathbb{F}_2^d \to \mathbb{F}_2$ of d variables. This means that F is a vectorial Boolean function mapping binary strings of length $2(d-1)$ to strings of length $d-1$. Setting $N = 2^{d-1}$, one can associate a $N \times N$ square matrix S_F to F as follows: for each $(i, j) \in [N] \times [N]$, the entry of S_F at row i and column j equals

$$S_F(i, j) = \phi(F(\psi(i)||\psi(j))), \tag{5}$$

where $||$ denotes the concatenation operator. Thus, the entry $S_F(i, j)$ is determined by computing the CA on the input vector where the first $d-1$ bits corresponds to the binary representation of row i, while the last $d-1$ are the binary representation of column j.

One may wonder under which conditions the matrix associated to a CA is a Latin square. As shown in the next result [7], this situation happens when the underlying local rule is bipermutive:

Lemma 1. *Let* $F : \mathbb{F}_2^{2(d-1)} \to \mathbb{F}_2^{d-1}$ *be a CA with bipermutive local rule* $f : \mathbb{F}_2^d \to \mathbb{F}_2$*. Then, the square* S_F *induced by* F *is a Latin square of order* $N = 2^{d-1}$*.*

As an example, Fig. 3 depicts the Latin square of order $N = 4$ associated to the CA $F : \mathbb{F}_2^4 \to \mathbb{F}_2^2$ with bipermutive local rule 150. A natural question immediately following from Lemma 1 is when the Latin squares associated to two bipermutive CA F, G are orthogonal. In this case, we call the pair F, G as *Orthogonal Cellular Automata* (OCA), and by analogy a family of bipermutive CA whose associated Latin squares are MOLS is called a set of *Mutually Orthogonal Cellular Automata* (MOCA).

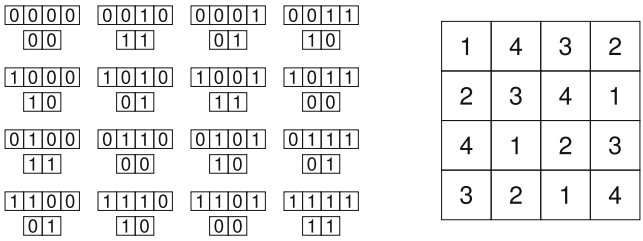

Fig. 3. Example of Latin square to the CA $F : \mathbb{F}_2^4 \to \mathbb{F}_2^2$ with local rule 150.

The question has been settled in [7] for *linear* rules. A local rule $f : \mathbb{F}_2^d \to \mathbb{F}_2$ is linear if there exists a vector $a = (a_1, \cdots, a_d) \in \mathbb{F}_2^d$ such that $f(x_1, \cdots, x_d) = a_1 x_1 \oplus \cdots \oplus a_d x_d$ for all $x = (x_1, \cdots, x_d) \in \mathbb{F}_2^d$. In this case, rule f is bipermutive if and only if $a_1 = a_d = 1$. Additionally, one can easily associate to f a *polynomial* $p_f(X) \in \mathbb{F}_2[X]$ of degree $d-1$ by defining it as $p_f(X) = a_1 + a_2 X + \cdots + a_d X^{d-1}$. Using this representation, the authors of [7] proved the following result:

Theorem 2. *Let $F, G : \mathbb{F}_2^{2(d-1)} \to \mathbb{F}_2^{d-1}$ be two CA respectively defined by two linear bipermutive rules $f, g : \mathbb{F}_2^d \to \mathbb{F}_2$. Further, let p_f, p_g denote the two polynomials respectively associated to f and g. Then, F and G are OCA if and only if $\gcd(p_f, p_g) = 1$, that is, if and only if f and g are coprime.*

In [8] the authors performed an exhaustive search for finding all OCA pairs equipped with nonlinear bipermutive rules of diameter up to $d = 6$. Further, the optimization problem of determining nonlinear OCA of diameter $d = 7, 8$ has been addressed in [11]. In particular, since exhaustive search is not feasible for any $d > 6$, the authors resorted to *genetic algorithms* (GA) and *genetic programming* (GP).

3 Computing Preimages of OCA

We can now formally state the *inversion problem for OCA* which we analyze in the rest of this paper:

Problem 1. Let $F, G : \mathbb{F}_2^{2(d-1)} \to \mathbb{F}_2^{d-1}$ be a pair of OCA respectively defined by bipermutive local rules $f, g : \mathbb{F}_2^d \to \mathbb{F}_2$, and let $w, z \in \mathbb{F}_2^{d-1}$ be two $(d-1)-$bit vectors. Then, find the vector $c = x \,||\, y$ with $x, y \in \mathbb{F}_2^{d-1}$ such that $(F(c), G(c)) = (w, z)$.

Using the terminology of Latin squares, Problem 1 requires finding a pair of row/column coordinates $(\phi(x), \phi(y))$ such that the corresponding entry in the superposition of Latin squares S_F and S_G is the pair $(\phi(w), \phi(z))$. Since S_F and S_G are orthogonal, by Remark 1 such pair of coordinates is unique.

Notice that Problem 1 does not assume any linearity on the bipermutive local rules underlying the two OCA, so the inversion algorithm which we develop in this section works both for linear and nonlinear OCA. Before describing it, we first need to introduce some additional data structures and algorithms.

Let $G_{DB}(f) = (V, E, l_f)$ and $G_{DB}(g) = (V, E, l_g)$ be the de Bruijn graphs respectively associated to two CA $F, G : \Sigma^{2(d-1)} \to \Sigma^{d-1}$ equipped with local rules $f, g : \Sigma^d \to \Sigma$ of diameter d. Then, the *coupled de Bruijn graph* induced by F and G is the de Bruijn graph $G_{DB}(f, g) = (V, E, l_{f,g})$ whose edge labeling function $l : E \to \Sigma \times \Sigma$ is defined for all $(v_1, v_2) \in E$ as

$$l(v_1, v_2) = (l_f(v_1, v_2), l_g(v_1, v_2)). \tag{6}$$

Thus, the labeling on the coupled de Bruijn graph is formed setting side by side the edge labels of the de Bruijn graphs of the single CA.

In what follows, we will make use of the variant of *Depth First Search* originally introduced in [10] to compute the *unfolding* of de Bruijn graphs. Given a configuration y of length p and a vertex v of a de Bruijn graph $G_{DB}(f) = (V, E, l)$ associated to a CA, this algorithm visits $G_{DB}(f)$ starting from a single vertex v_1 and following the path on the edges labeled by y. In particular, contrary to the plain version of DFS, this variant does not mark the visited edges, so that in principle they can be visited multiple times. The fusion of the vertices $v_1, \cdots, v_p$ visited during this algorithm determines a preimage x of configuration y. In our case, we will denote by DFS-Mod(V, E, l, v, w, z) a call to this DFS variant on the coupled de Bruijn graph $G_{DB}(f, g) = (V, E, l)$ associated to f and g, starting from vertex v and reading the edge labels determined by juxtaposing the configurations $w, z \in \mathbb{F}_2^{d-1}$. In particular, it is not guaranteed that a preimage of w, z can be found, since for any $i \in \{1, \cdots, d-1\}$ there might be no edges labeled with (w_i, z_i) that exit from vertex v_i visited by the DFS on step $i - 1$. Thus, we will assume that DFS-Mod$(G_{DB}(f, g), l, v, w, z)$ either returns a preimage c of w, z or the value NIL when such preimage cannot be constructed starting from vertex v.

We can now describe the structure of our inversion procedure for OCA, whose pseudocode is reported in Algorithm 1. The procedure takes as input the coupled de Bruijn graph $G_{DB}(f, g)$ of two OCA $F, G : \mathbb{F}_2^{2(d-1)} \to \mathbb{F}_2^{d-1}$ defined by bipermutive rules $f, g : \mathbb{F}_2^d \to \mathbb{F}_2$ respectively, and two configurations $w, z \in \mathbb{F}_2^{d-1}$. The first three steps of the algorithm simply extract the vertex set, the edge set and the labeling function of the graph, while the fourth step initializes the configuration to be returned to NIL. Then, the `while` loop is performed until there are edges in E labeled with the first symbols of w and z, and c equals NIL. Inside the loop, the only instruction is the call to DFS-Mod starting from the first vertex of the edge. If the DFS visit successfully completes, then a preimage of (w, z) is returned and assigned to c, otherwise c remains NIL. As soon as a

preimage is found or there are no other edges labeled with (w_1, z_1) in the coupled de Bruijn graph, the execution exits the *while* loop and the current value of c is returned.

Algorithm 1. INVERT-OCA$(G_{DB}(f,g), w, z)$

$V :=$ VERTEX$(G_{DB}(f,g))$
$E :=$ EDGES$(G_{DB}(f,g))$
$l :=$ LABELS$(G_{DB}(f,g))$
$c := NIL$
while $e \in \{(v_1, v_2) \in E : \ l(v_1, v_2) = (w_1, z_1)\}$ AND $c = NIL$ **do**
  $c :=$ DFS-MOD(V, E, l, v_1, w, z)
end while
return c

We now prove the correctness and the time complexity of Algorithm 1, under the assumption that F and G are OCA.

Theorem 3. *Let $F, G : \mathbb{F}_2^{2(d-1)} \to \mathbb{F}_2^{d-1}$ be two OCA with bipermutive local rules $f, g : \mathbb{F}_2^d \to \mathbb{F}_2$ and let $G_{DB}(f,g)$ be the coupled de Bruijn graph of F and G. Then, for any pair of final configurations $w, z \in \mathbb{F}_2^{d-1}$, the procedure* INVERT-OCA *correctly returns the unique preimage $c \in \mathbb{F}_2^{2(d-1)}$ such that $(F(c), G(c)) = (w, z)$ in $\mathcal{O}(d \cdot 2^d)$ steps.*

Proof. Correctness. *Let $w, z \in \mathbb{F}_2^{d-1}$ be two configurations of $d-1$ bits, and let $\phi(w), \phi(z)$ be their decimal representations ranging in $[N]$, where $N = 2^{d-1}$. Since the two Latin squares S_F and S_G are orthogonal, the pair $(\phi(w), \phi(z))$ appears exactly once in their superposition. Let $i, j \in [N]$ be respectively the row and column coordinates where such pair occurs. Given the binary representation $\psi(i), \psi(j) \in \mathbb{F}_2^{d-1}$ of i, j and denoting by $c = \psi(i) \,||\, \psi(j)$ their concatenation, this means that*

$$(F(c), G(c)) = (w, z) \tag{7}$$

Algorithm 1 invokes DFS-MOD *on all vertices $v \in V$ which have an outgoing edge labeled by (w_1, z_1). In particular, due to the fact that S_F and S_G are orthogonal, there will be exactly one call which returns a value different from NIL, and this value corresponds to the only preimage c which satisfies Eq. (7).*

Complexity. *To determine the time complexity of* INVERT-OCA*, first remark that a single call to* DFS-MOD *requires at most $d-1$ steps to complete, because the two configurations w, z have length $d-1$ each, and their symbols are pairwise read during the DFS visit. In particular, a DFS visit could return before $d-1$ steps, due to the fact that there are no outgoing edges labeled with the pairs of symbols of w and z. To conclude, we need to determine how many times* DFS-MOD *is invoked. Lemma 3 in [8] shows that the local rules of OCA are* pairwise balanced*, meaning that there are exactly 2^{d-2} edges on the coupled de Bruijn graph labeled with (w_1, z_1). Consequently,* DFS-MOD *is invoked 2^{d-2} times, thus the overall time complexity of* INVERT-OCA *is $\mathcal{O}(d \cdot 2^d)$.* □

One may notice that the time complexity of Algorithm 1 is exponential with respect to the diameter of the CA. However, remark that Algorithm 1 can be straightforwardly parallelized by assigning a processor to each DFS call inside the `while` loop. Hence, by using 2^{d-2} processors in parallel, the time complexity of Invert-OCA can be reduced down to $\mathcal{O}(d)$, which is the number of steps necessary to complete a DFS visit.

4 Application to Secret Sharing Schemes

On account of Theorem 2, a set $\{p_1, \cdots, p_n\}$ of n *pairwise coprime polynomials* of degree $d-1$ is equivalent to a family of n linear MOCA of order $N = 2^{d-1}$, and thus by Theorem 1 it is also equivalent to a perfect $(2, n)$-threshold SSS. However, publishing the whole set of n MOLS is not an efficient way to implement the recovery phase of a SSS, especially if the size of the squares is huge. Thus, one needs to find a compact way to describe the recovery phase of the secret starting from the knowledge of two shares.

In this concluding section, we show how our inversion algorithm Invert-OCA can be used precisely for this purpose. To our knowledge, this is the first time that a full perfect $(2, n)$-threshold SSS based on CA is described in the literature. As a matter of fact, there have been other attempts at designing CA-based secret sharing schemes (such as [9,13]), but the resulting access structures suffered from an additional *adjacency constraint* on the shares, since they actually represent blocks of CA configurations.

Let the secret S be a vector of $\mathbb{F}_2^m$ where $m = d-1$, and assume that there are n players $P_1, \cdots, P_n$. Then, the setup phase of our $(2, n)$-threshold SSS is as follows:

Setup Phase
Initialization:

1. Find n local rules $f_1, \cdots, f_n : \mathbb{F}_2^d \to \mathbb{F}_2$ which give rise to a set of n MOCA of order $N = 2^{d-1}$. By Theorem 2, this can be done for example by picking n relatively prime polynomials $p_{f_1}(x), \cdots, p_{f_n}(x)$ over $\mathbb{F}_2$
2. Concatenate secret S with a random vector $R \in \mathbb{F}_2^m$, thus obtaining a configuration $C \in \mathbb{F}_2^{2m}$ of length $2(d-1)$

Loop: For all $i \in \{1, \cdots n\}$ do:

1. Given $F_i : \mathbb{F}_q^{2m} \to \mathbb{F}_2^m$ the NBCA defined by rule f_i, compute $B_i = F_i(C)$
2. Send share B_i to player P_i

Termination: Publish the n local rules $f_1, \cdots, f_n$ defining the MOCA.

For the recovery phase, suppose that two players P_i and P_j want to determine the secret. Let B_i and B_j respectively denote the share of P_i and P_j. Since the local rules of the MOCA are public, both P_i and P_j know the CA linear rules f_i and f_j used by the dealer to compute their shares. Hence, they adopt the following procedure to recover S:

Recovery Phase
Initialization:
1. Find the CA linear rules f_i and f_j published by the dealer corresponding to players P_i and P_j
2. Compute the coupled de Bruijn graph $G_{DB}(f_i, f_j)$

Reconstruction:
1. Compute configuration C by calling Invert-OCA$(G_{DB}(f_i, f_j), B_i, B_j)$
2. Return the first half of C as the secret S

Hence, the recovery phase of this SSS simply consists in computing the preimage of the pair of configurations represented by the shares B_i, B_j under the action of the two OCA with local rules f_i, f_j. In particular, the whole preimage returned by Invert-OCA contains both secret S in its left half and the random column chosen by the dealer in the second half.

5 Discussion, Conclusions and Directions for Future Work

In this paper, we described an algorithm to invert a pair of configurations under the action of two OCA. Specifically, starting from the coupled de Bruijn graph of the two OCA of diameter d, the algorithm applies a DFS-based search until a valid path labeled with the two configurations is found. The existence of such unique path is guaranteed by the fact that the two OCA define a pair of orthogonal Latin squares, and thus a bijection among pairs of $(d-1)$-bit vectors. Since there are 2^{d-1} vertices in the coupled de Bruijn graph, in the worse case the running time of our algorithm is exponential in the diameter of the CA. However, this algorithm is easily parallelizable, by assigning a DFS call to a separate processor. Hence, using $\mathcal{O}(2^d)$ processors in parallel yields a time complexity which is linear in the CA diameter. As an application of this algorithm, we showed how to implement the recovery phase of a $(2, n)$-threshold secret sharing scheme based on MOCA.

Taking a closer look at the computational complexity of Algorithm 1, one may notice that we did not consider the size of the input in our analysis. As a matter of fact, the de Bruijn graph of a CA is already exponential in the CA diameter, something which apparently hinders the applicability of our inversion algorithm. However, depending on the nature of the underlying local rules, one can find more efficient representations of this algorithm. For instance, if the local rules are linear, then it is possible to adapt the preimage construction procedure described in [9] as follows: first, the leftmost $(d-1)$-cell block of the preimage is randomly guessed. Then, one exploits the right permutivity property of the two local rules to compute the two values for the d-th cell of the preimage. If the two values are equal, then the preimage is consistent up to that point, and the next cell in position $d+1$ can be computed. This process is repeated rightwards, until either a mismatch is found between the two computed values (meaning that one has to start over with a new left block of $d-1$ cells), or the rightmost block

is completed (i.e. the correct preimage mapping to the pair of configurations has been found). Under this procedure, one can compute the two values for the current preimage cell using the *Algebraic Normal Form* (ANF) [3] of the two CA local rules. If the rules are linear, then the size of their ANF is linear in the CA diameter d, since it just corresponds to an XOR of a subset of the input variables. Of course, in the general case of nonlinear bipermutive rules the size of the ANF can still be exponential in the diameter.

However, we remark that this issue is mainly a matter of trade-off between the required amount of nonlinearity of the CA local rules and their ANF sizes, which highly depends on the specific application domain of our inversion algorithm. Returning to our secret sharing scheme example, most of the existing protocols used in practice are actually linear. Thus, plugging linear rules into our example described in Sect. 4 would yield another linear threshold scheme with a recovery phase that can be performed in $\mathcal{O}(d)$ steps using $\mathcal{O}(2^d)$ processors in parallel. As a consequence, it would be interesting to compare the complexity of our scheme with those of other well-established linear SSS, such as Shamir's scheme [15]. Further, as pointed out in [7], the inversion problem of two linear OCA actually amounts to the inversion of a *Sylvester matrix*. Hence, another direction worth exploring for further research is to investigate the computational complexity of inverting this kind of matrices, in order to verify if a faster inversion algorithm can be designed.

Under a different perspective, for certain applications there is the need for *nonlinear* secret sharing schemes. An example are *cheater-immune secret sharing schemes* based on nonlinear constructions, which are robust towards dishonest players who submit fake shares during the reconstruction phase [18]. In this case, it would be interesting to analyze the trade-off between the amount of nonlinearity that the local rules must have to achieve cheater-immunity and the size of their ANF. A possible strategy could be to cast this question in terms of an optimization problem, and then solve it through heuristic techniques such as *Genetic Programming* (GP), which already proved to be successful in the design of S-boxes with good cryptographic properties and small implementation costs [12].

As a closing remark, we note that determining how large a family of MOCA can be is still an open problem, even in the linear case. As shown in [7], verifying whether a set of linear bipermutive CA of diameter d form a family of MOCA is equivalent to check that the polynomials associated to the local rules are pairwise coprime. However, despite the enumeration of coprime polynomials over finite fields is a well-developed research topic (see e.g. [1]), as far as we know there are no works in the literature addressing coprimality of monic polynomials with nonzero constant term, which is exactly the subclass corresponding to linear bipermutive local rules. Very recently, the first author showed a construction of a family of pairwise coprime polynomials of this kind in his PhD thesis [6], thus providing a first lower bound on its size. Nonetheless, optimality of this construction is still open.

References

1. Benjamin, A.T., Bennett, C.D.: The probability of relatively prime polynomials. Math. Mag. **80**(3), 196–202 (2007)
2. Betel, H., de Oliveira, P.P.B., Flocchini, P.: Solving the parity problem in one-dimensional cellular automata. Nat. Comput. **12**(3), 323–337 (2013)
3. Carlet, C.: Boolean functions for cryptography and error correcting codes. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, 1st edn, pp. 257–397. Cambridge University Press, New York. (2010)
4. Hedlund, G.A.: Endomorphisms and automorphisms of the shift dynamical systems. Math. Syst. Theory **3**(4), 320–375 (1969)
5. Keedwell, A.D., Dénes, J.: Latin Squares and Their Applications. Elsevier, New York City (2015)
6. Mariot, L.: Cellular automata, Boolean functions and combinatorial designs (2018). https://boa.unimib.it/bitstream/10281/199011/2/phd_unimib_701962.pdf
7. Mariot, L., Formenti, E., Leporati, A.: Constructing orthogonal Latin squares from linear cellular automata. CoRR abs/1610.00139 (2016)
8. Mariot, L., Formenti, E., Leporati, A.: Enumerating orthogonal Latin squares generated by bipermutive cellular automata. In: Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A.E. (eds.) AUTOMATA 2017. LNCS, vol. 10248, pp. 151–164. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58631-1_12
9. Mariot, L., Leporati, A.: Sharing secrets by computing preimages of bipermutive cellular automata. In: Wąs, J., Sirakoulis, G.C., Bandini, S. (eds.) ACRI 2014. LNCS, vol. 8751, pp. 417–426. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11520-7_43
10. Mariot, L., Leporati, A., Dennunzio, A., Formenti, E.: Computing the periods of preimages in surjective cellular automata. Nat. Comput. **16**(3), 367–381 (2017)
11. 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, 15–19 July 2017, Berlin, Germany, pp. 306–313 (2017)
12. Mariot, L., Picek, S., Leporati, A., Jakobovic, D.: Cellular automata based s-boxes. Cryptogr. Commun. (2018). https://doi.org/10.1007/s12095-018-0311-8
13. del Rey, Á.M., Mateus, J.P., Sánchez, G.R.: A secret sharing scheme based on cellular automata. Appl. Math. Comput. **170**(2), 1356–1364 (2005)
14. Richardson, D.: Tessellations with local transformations. J. Comput. Syst. Sci. **6**(5), 373–388 (1972)
15. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
16. Stinson, D.R.: Combinatorial Designs - Constructions and Analysis. Springer, New York (2004). https://doi.org/10.1007/b97564
17. Sutner, K.: De Bruijn graphs and linear cellular automata. Complex Syst. **5**(1), 19–30 (1991)
18. Tompa, M., Woll, H.: How to share a secret with cheaters. J. Cryptol. **1**(2), 133–138 (1988)