Inversion of Cellular Automata Iterations

Ç. K. Koç [*]
Electrical and Computer Engineering
Oregon State University, ECE 220
Corvallis, Oregon 97331, USA

A. M. Apohan
TUBITAK MAM Research Centre
Department of Electronics, PK 21
Gebze, Kocaeli 41470, TURKEY

Abstract

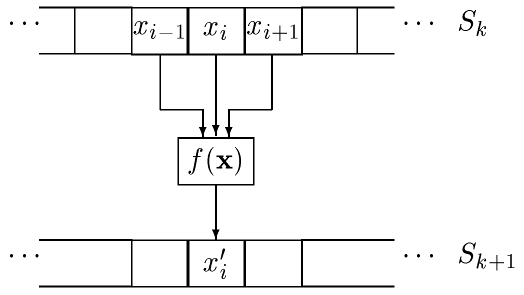
We describe an algorithm for inverting an iteration of the one-dimensional cellular automaton. The algorithm is based on the linear approximation of the updating function, and requires less than exponential time for particular classes of updating functions and seed values. For example, an n-cell cellular automaton based on the updating function CA30 can be inverted in $O(n)$ time for certain seed values, and at most $2^{n/2}$ trials are required for arbitrary seed values. The inversion algorithm requires at most $2^{(q-1)(1-\alpha)n}$ trials for arbitrary nonlinear functions and seed values, where q is the number of variables of the updating function, and α is the probability of agreement between the function and its best affine approximation. The inversion algorithm coupled with the method of Meier and Staffelbach [6] becomes a powerful tool to cryptanalyze the random number generators based on one-dimensional cellular automata, showing that these random number generators provide less amount of security than their state size would imply.

Key Words: Random number generation, best affine approximation, cryptanalysis.

1 Introduction

A one-dimensional cellular automaton consists of a linearly connected array of n cells, each of which takes the value of 0 or 1, and a boolean function $f(\mathbf{x})$ with q variables. The value of the cell x_i is updated in parallel (synchronously) using this function in discrete time steps as $x'_i = f(\mathbf{x})$ for $i = 1, 2, \ldots, n$. The boundary conditions are usually handled by taking the index values modulo n, i.e., the linearly connected array is actually a circular register. A feedback shift register model of the one-dimensional cellular automata is also known [3]. The parameter q is usually an odd integer, i.e., $q = 2r + 1$, where r is often named the radius of the function $f(\mathbf{x})$; the new value of the ith cell is calculated using the value of the ith cell itself and the values of r neighboring cells to the right and left of the ith cell. The cellular automaton for $q = 3$ is illustrated in Figure 1.

Figure 1: The one-dimensional cellular automaton with $q = 3$.



[*]Supported in part by the NSF under grant ECS–9312240 and by Intel Corporation.

Since there are n cells, each of which takes the values of 0 or 1, there are 2^n possible state vectors. Let S_k denote the state vector at the automaton moves to the states S_1, S_2, S_3, etc., at time steps $k = 1, 2, 3$, etc. The state vector S_k takes values from the set of n-bit binary vectors as k advances, and the state machine will eventually cycle, i.e., it will reach a state S_{k+P} which was visited earlier $S_k = S_{k+P}$. The period P is a function of the initial state, the updating function, and the number of cells.

Cellular automata are generally considered as discrete dynamical systems, or discrete approximations to partial differential equations modeling a variety of natural systems. We refer the reader to [11] and the references therein for further information about the properties, dynamics, and applications of cellular automata.

A random sequence generator based on the one-dimensional cellular automaton with $q = 3$ and the so-called CA30 updating function

$$f(x_{i-1}, x_i, x_{i+1}) = x_{i-1} \text{ XOR } (x_i \text{ OR } x_{i+1}) \tag{1}$$

was proposed by Wolfram in [9]. The state vectors produced by this cellular automaton seem to have randomness properties, e.g., the time sequence values of the central cell shows no statistical regularities under the usual randomness tests [5]. In order to use such a generator for cryptographic purposes, we must also ensure that the seed value (the initial state vector S_0) is difficult to construct given a sequence of state vectors. It was stated in [9] that

> This problem is in the class NP. No systematic algorithm for its solution is currently known that takes a time less than exponential in n.

We show in this paper that the number of trials may be much fewer than 2^n for particular classes of updating functions and seed values. We give an algorithm for computing S_k given S_{k+1}, whose running time depends on the linearity of the updating function $f(\mathbf{x})$. If $f(\mathbf{x})$ is a q-variable affine function, then the running time of the algorithm is only $O(nq^2)$. Furthermore, the running time of the inversion algorithm may still be $O(nq^2)$ for a nonlinear updating function with certain seed values. In general, the inversion algorithm requires much fewer than 2^n trials for nonlinear updating functions and arbitrary seed values. For example, the number of trials required to invert the cellular automaton based on the updating function CA30 is at most $2^{n/2}$ for an arbitrary seed value.

It has also been shown by Meier and Staffelbach that the sequence of the central bits of the states, which is named as the temporal sequence, can be used to determine the seed using the partial linearity of the CA30 updating function [6]. Given the temporal sequence, their method successfully finds the seed vector when its size is up to 500 bits. The inversion algorithm differs from the method of Meier and Staffelbach in the sense that it computes the predecessor of a given state vector. However, the method of Meier and Staffelbach can be coupled with the inversion algorithm: First we compute a state vector given the central bit values using their method. Then we use the inversion algorithm to compute the predecessors of this state. Thus, we can map the entire evolution of the state machine which generates the random numbers. The method of Meier and Staffelbach coupled with the inversion algorithm becomes a powerful tool to cryptanalyze the random number generators based on one-dimensional cellular automata. This shows that these random number generators provide less amount of security than their state size would imply; they are unsuitable for cryptographic purposes when n is small.

2 Description of the Inversion Algorithm

The proposed inversion algorithm is based on the best affine approximation of the q-variable updating function $f(\mathbf{x})$. Using tools from the spectral analysis of boolean functions [1, 2, 7], we construct a q-variable linear function $g(\mathbf{x})$ which is the best affine approximation to $f(\mathbf{x})$. In other words, $g(\mathbf{x})$ has the minimum Hamming distance to $f(\mathbf{x})$ among all linear functions. The Hamming distance between two functions is defined as the Hamming distance between the binary vectors of length 2^q produced by the application of all possible input values $\mathbf{x} \in \mathcal{Z}_2^q$ to these functions.

The main idea of the inversion algorithm is illustrated in Figure 2. The state vector S_{k+1} is computed by applying the function $f(\mathbf{x})$ to the state vector S_k. We calculate an approximation to S_k by finding a vector S_k^* which maps to S_{k+1} under the linear function $g(\mathbf{x})$. The vector S_k^* is computed by solving a set of linear equations whose matrix is determined by the parameters of the linear function $g(\mathbf{x})$. We then apply $f(\mathbf{x})$ to S_k^* and calculate S_{k+1}^* which is an approximation to the state vector S_{k+1}. Since S_{k+1} is known, we compare S_{k+1} and S_{k+1}^*, and make corrections to the state vector S_k^* until we obtain the state vector S_k^* which results in the equality $S_{k+1}^* = S_{k+1}$. The inversion algorithm calculates one of the predecessors of the state vector S_{k+1}. Repeated application of the inversion with different $g(\mathbf{x})$, whenever applicable, may yield other predecessors.

Figure 2: The main idea of the inversion algorithm.

$$\begin{tikzcd}
\boxed{S_k} \arrow[d, "f(\mathbf{x})"] & & \boxed{S_k^*} \arrow[dll, "g(\mathbf{x})"'] \arrow[d, "f(\mathbf{x})"] \\
\boxed{S_{k+1}} & & \boxed{S_{k+1}^*}
\end{tikzcd}$$

We are assuming that the n-dimensional binary vector S_{k+1} and the q-variable updating function $f(\mathbf{x})$ are given. Our objective is to devise an algorithm for computing S_k as shown in Figure 2. The details of the inversion algorithm are described below.

The Inversion Algorithm

Input: The n-dimensional vector S_{k+1} and the q-variable updating function $f(\mathbf{x})$.

Output: The n-dimensional vector S_k.

1. Obtain the best affine approximation $g(\mathbf{x})$ to the updating function $f(\mathbf{x})$, and determine α which is the probability of agreement between $f(\mathbf{x})$ and $g(\mathbf{x})$.

   The q-variable function $f(\mathbf{x})$ may be given in algebraic form, however, we only need its values at 2^q points in order to construct its best affine approximation. The best affine approximation of $f(\mathbf{x})$ is the function

$$g(\mathbf{x}) = w_1 x_1 \oplus w_2 x_2 \oplus \cdots \oplus w_q x_q \oplus c \ , \tag{2}$$

   where $w_i \in \mathcal{Z}_2$ for $i = 1, 2, \ldots, q$, and $c \in \mathcal{Z}_2$, such that the summation

$$\sum_{\mathbf{x} \in \mathcal{Z}_2^q} f(\mathbf{x}) \oplus g(\mathbf{x}) \tag{3}$$

   achieves its minimal value [7, 2]. The function $g(\mathbf{x})$ is determined by calculating the vector $\mathbf{w} = (w_1, w_2, \ldots, w_q)$ and the constant c, which can be computed by exhaustive search. However, there is an efficient method for constructing $g(\mathbf{x})$ based on the Walsh transform [1, 2, 7, 8]. The functions $f(\mathbf{x})$ and $g(\mathbf{x})$ will agree at a portion of 2^q points. Let A be the number of points at which $f(\mathbf{x}) = g(\mathbf{x})$. Note that $A \geq 2^q/2$, since, otherwise, $f(\mathbf{x})$ will agree with $g(\mathbf{x}) \oplus 1$ (the complement of $g(\mathbf{x})$) in more than $2^q/2$ points, and thus, we can take $g(\mathbf{x}) \oplus 1$ as the best affine approximation. The probability of agreement between $f(\mathbf{x})$ and $g(\mathbf{x})$ is defined as $\alpha = A/2^q$. In general, we have $\alpha \geq \frac{1}{2}$. If $f(\mathbf{x})$ is linear, then $\alpha = 1$.

2. Obtain the state vector S_k^* using the affine function $g(\mathbf{x})$, and by solving a set of linear equations of dimension n in $GF(2)$.

   Let $S_k^* = (x_1, x_2, \ldots, x_n)$ and $S_{k+1} = (y_1, y_2, \ldots, y_n)$. The state vector S_{k+1} is obtained by applying the function $g(\mathbf{x})$ to the state vector S_k^*. Instead of computing the inverse of the function $g(\mathbf{x})$, we solve a set of linear equations of dimension n in the field $GF(2)$ involving the known quantities y_i in order to compute the unknown quantities x_i for $i = 1, 2, \ldots, n$. The matrix of the linear system of equations is a band matrix with a bandwidth of q. For example, the linear system of equations of dimension $n = 8$ for the 3-variable ($q = 3$) approximating affine function

$$g(x_1, x_2, x_3) = w_1x_1 \oplus w_2x_2 \oplus w_3x_3 \oplus c \tag{4}$$

   is given as

$$\begin{bmatrix} w_2 & w_3 & & & & & & w_1 \\ w_1 & w_2 & w_3 & & & & & \\ & w_1 & w_2 & w_3 & & & & \\ & & w_1 & w_2 & w_3 & & & \\ & & & w_1 & w_2 & w_3 & & \\ & & & & w_1 & w_2 & w_3 & \\ & & & & & w_1 & w_2 & w_3 \\ w_3 & & & & & & w_1 & w_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} + \begin{bmatrix} c \\ c \\ c \\ c \\ c \\ c \\ c \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{bmatrix} . \tag{5}$$

3. Apply the function $f(\mathbf{x})$ to the state vector S_k^* to obtain the state vector S_{k+1}^*, as shown in Figure 2.

4. If $S_{k+1} = S_{k+1}^*$, then stop. The algorithm found a predecessor to the state vector S_{k+1}. If $S_{k+1} \neq S_{k+1}^*$, then apply 'corrections' to the state vector S_k^* which gives the equality $S_{k+1} = S_{k+1}^*$. The correction heuristics are in Steps 5 through 12.

5. Compare S_{k+1} and S_{k+1}^*, and mark all single bits which are different.

6. Let x_j and x_j^* be two bit values in S_{k+1} and S_{k+1}^*, respectively. We have either $x_j^* = x_j$ or $x_j^* = \bar{x}_j$. If the latter is true, we can obtain the correct bit value by complementing, which requires that the inputs to the function $f(\mathbf{x})$ giving the value of x_j^* need to be changed. We examine the truth table of the function $f(\mathbf{x})$, and make a template consisting of all possible input values which produce the complemented output. There are two kinds of templates for every incorrect bit: we either have to make a change from 0 to 1, or from 1 to 0.

7. Construct as many templates as the number of incorrect bits in the state vector S_{k+1}^*. In other words, the number of templates is equal to the Hamming distance between S_{k+1} and S_{k+1}^*, i.e., $H(S_{k+1}, S_{k+1}^*)$. The length of a template is equal to the number of 1s in the function $f(\mathbf{x})$ if the change is to be made from 0 to 1. Similarly, if the change is to be made from 1 to 0, the length of the template will be equal to the number of 0s in $f(\mathbf{x})$.

8. Group the templates in such a way that each group contains neighboring templates. Two templates are considered neighbors if the difference of their indices is strictly less than q. For example, if $q = 3$ and the indices of incorrect bits are 6, 7, 8, 11, 13, then the templates corresponding to 6, 7, 8 are placed in one group, while the templates corresponding to 11, 13 are placed in another group.

9. Each group is then represented by a single reduced template which enumerates all possible input values producing the correct bits within the group. This operation is performed by examining the templates, and removing the conflicts. The objective is to find input configurations which correct all bits in the group at the same time. An input configuration which corrects one bit while destroying one ore more bits is considered a conflict, and is removed from the set. This removal reduces the number of rows in each template.

10. After each group is represented by a single template, we remove any rows within the template, which destroy the bit values outside the group. The input values correcting the bit values within a group should not be allowed to corrupt any bits which have not been placed in a group because they were correct to begin with.

11. The remaining templates give all possible bit configurations giving the correct state vector S_k^*. We place these bit configurations in S_k^* one by one, apply the function $f(\mathbf{x})$, and check if $S_{k+1}^* = S_{k+1}$. If there is a bit configuration which produces this equality, then we have found a predecessor to the state S_{k+1}.

12. If none of these bit configurations produces the correct state vector, then we conclude that certain bits which have not been placed in the templates need to be changed. We exhaustively enumerate all bit values outside the templates together with the bit values in the reduced templates in order to calculate the correct state vector S_k^*.

3 Analysis of the Inversion Algorithm

Given the n-dimensional binary vector S_{k+1} and the updating function $f(\mathbf{x})$, we can try all possible n-dimensional binary vectors to find the one which gives the state vector S_{k+1}. Since there are 2^n binary vectors of length n, the number of trials required by the exhaustive search algorithm is equal to 2^n in the worst case. The running time of the inversion algorithm is to be contrasted to that of the exhaustive search algorithm.

The construction of the best affine approximation of the updating function $f(\mathbf{x})$ is performed using the Walsh transform. The running time of this operation is a function of q, the number of variables in the function $f(\mathbf{x})$. Since q is usually very small compared to n, we ignore the time required to calculate all $g(\mathbf{x})$ functions which are the best affine approximations to $f(\mathbf{x})$.

In Step 2, we select one of these affine functions, and solve a linear system of equations of dimension n in the field $GF(2)$. This requires implementation of $GF(2)$ arithmetic and the Gaussian elimination algorithm. As was mentioned, the matrix of these linear equations is band matrix with a bandwith of q. It is known [4] that the Gaussian elimination algorithm requires $O(nq^2)$ arithmetic operations to solve a linear system of equations with size n and bandwith q. If $f(\mathbf{x})$ happens to be a linear function, the inversion algorithm will be successful in Step 4 since in this case $f(\mathbf{x})$ will be equal to $g(\mathbf{x})$ at all points, and the application of $f(\mathbf{x})$ to the state vector S_k^* will produce S_{k+1}^* which will be equal to S_{k+1}.

Even if $f(\mathbf{x})$ is not linear, the algorithm may still be successful in Step 4 for certain seed values. This is due to the fact that the nonlinear function $f(\mathbf{x})$ and the linear function $g(\mathbf{x})$ agree at $\alpha 2^q$ points; these two functions are indistinguishable from one another if these q-tuples are used as inputs. If S_k consists of those q-tuples at which $f(\mathbf{x}) = g(\mathbf{x})$, then the application of $f(\mathbf{x})$ to S_k, and $g(\mathbf{x})$ to S_k^* will result in the equality $S_{k+1} = S_{k+1}^*$. Thus, we will obtain the solution immediately after solving the set of linear equation. We refer the reader to Example 1 to clarify our arguments on this issue.

If S_{k+1} is different from S^*_{k+1}, we apply the correction steps (5 through 12). We construct as many templates as the number of bits which differ in state vectors S_{k+1} and S^*_{k+1}. Let p be the Hamming distance between S_{k+1} and S^*_{k+1}, i.e., we construct exactly p templates. Now, we need to make changes in S^*_k in order to find the correct S^*_{k+1}. We will also assume that $f(\mathbf{x})$ is a nearly balanced function, i.e., the number of 1s is nearly equal to the number of 0s. Since $f(\mathbf{x})$ is a q-variable function, the length (the number of rows) of a template is approximately $2^q/2 = 2^{q-1}$. The number of rows in a template is the number of possible inputs (q-tuples) to substitute in S^*_k in order to have the output bit complemented in the state vector S^*_{k+1}. The trial substitutions are performed to complement all incorrect bits in S^*_{k+1} so that S^*_{k+1} will be equal to S_{k+1}. Since there are p incorrect bits and 2^{q-1} possible replacements for each one of them, the maximum number of trials will be

$$\underbrace{2^{q-1} \times 2^{q-1} \times \cdots 2^{q-1}}_{p} \; = \; 2^{(q-1)p} \; . \tag{6}$$

Let $f(S^*_k)$ represent the state vector obtained by applying the function $f(\mathbf{x})$ to the state vector S^*_k. As can be seen from Figure 2, we have

$$p = H(S_{k+1}, S^*_{k+1}) \; = \; H(g(S^*_k), f(S^*_k)) \; . \tag{7}$$

If n is sufficiently large and S^*_k is random, then the Hamming distance between the state vectors $g(S^*_k)$ and $f(S^*_k)$ will be approximately equal to $(1-\alpha)n$ where $1-\alpha$ is the probability of disagreement between the functions $f(\mathbf{x})$ and $g(\mathbf{x})$. If all possible 2^q bit vectors of length q occur equally likely in S_k, S_{k+1}, and S^*_k, then it is safe to say that the Hamming distance between the vectors $g(S^*_k)$ and $f(S^*_k)$ will be very close to $(1-\alpha)n$. In cryptographic applications, one usually starts with random seed values, otherwise they can easily be guessed. Furthermore, n needs to be large since the exhaustive search algorithm can be successfully applied to cryptanalyze the random number generator for small values of n. Thus, both of our assumptions seem reasonable. Therefore, we calculate the expected number of incorrect bits as

$$p = H(g(S^*_k), f(S^*_k)) = (1-\alpha)n \; . \tag{8}$$

This gives the number of trials to compute S_k as

$$2^{(q-1)p} = 2^{(q-1)(1-\alpha)n} \; . \tag{9}$$

However, we note that since we work with reduced templates based on the neighborhood principle, and the number of templates and the average number of rows in each template is expected to be much fewer than $(1-\alpha)n$ and 2^{q-1}, respectively. The number of trials given by the above equation is indeed a loose upper bound.

Unfortunately, the algorithm may still not be successful at the end of Step 11, i.e., none of the tried bit vectors may produce the correct state vector. This implies that bit values other than those placed in the templates need to be changed. Since for each incorrect bit in S^*_{k+1}, we replace approximately q bits in S^*_k, the number of untouched bits is approximately equal to $n - pq$. We exhaustively try each one of these 2^{n-pq} possible inputs with the input rows in the reduced templates. We may have to perform as many trials as

$$2^{n-pq} \times 2^{(q-1)p} \; = \; 2^{n-p} \; = \; 2^{n-(1-\alpha)n} \; = \; 2^{\alpha n} \; , \tag{10}$$

if Step 12 is to be executed. The above value is an upper bound on the expected number of trials since at this stage we use the reduced templates, and not all untouched bits may need to be changed.

4 Examples

In this section, we give several examples illustrating the application of the inversion algorithm to the cellular automaton based on the updating function CA30. These examples are constructed to describe certain properties of the inversion algorithm.

The first step is to construct the best affine approximation of the function CA30. As was mentioned earlier, the best affine approximation of a function may not be unique. It turns out that there are 4 affine functions which are the best approximations to the CA30 updating function. These affine functions are

$$\begin{aligned} g_1(x_1,x_2,x_3) &= x_1 \oplus 1 \ , \\ g_2(x_1,x_2,x_3) &= x_1 \oplus x_2 \ , \\ g_3(x_1,x_2,x_3) &= x_1 \oplus x_3 \ , \\ g_4(x_1,x_2,x_3) &= x_1 \oplus x_2 \oplus x_3 \ . \end{aligned}$$

Any of these affine functions $g_i(\mathbf{x})$ matches the function CA30 in $A = 6$ points out of 8, which gives $\alpha = 6/8 = 0.75$. This value of α is the maximum attainable among all affine functions. We tabulate the function CA30 and its best affine approximations below:

x_{i-1}	x_i	x_{i+1}	CA30	$g_1(\mathbf{x})$	$g_2(\mathbf{x})$	$g_3(\mathbf{x})$	$g_4(\mathbf{x})$
0	0	0	0	1	0	0	0
0	0	1	1	1	0	1	1
0	1	0	1	1	1	0	1
0	1	1	1	1	1	1	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	0	0
1	1	0	0	0	0	1	0
1	1	1	0	0	0	0	1

If the inversion algorithm is successful in Step 4, it will require $O(n)$ arithmetic operations since a linear system of equations with band $q = 3$ is solved. If the inversion algorithm executes until Step 11, it will require at most $2^{(q-1)(1-\alpha)n} = 2^{n/2}$ trials. If it also executes Step 12, then the number of trials will be at most $2^{\alpha n} = 2^{3n/4}$.

4.1 Example 1

Let $n = 11$ and $S_k = 01001000010$, which gives $S_{k+1} = 11111100111$ under the updating function CA30. Now, assuming that S_k is unknown and S_{k+1} is given, we can calculate S_k using the proposed inversion algorithm. First we select the affine function $g_4(\mathbf{x})$, and then solve a set of linear equations of dimension 11 in order to compute S_k^* as explained in Step 2. The state vector obtained from this step is found as $S_k^* = 01001000010$. We then apply the updating function CA30 to S_k^* to obtain S_{k+1}^*, which is found as $S_{k+1}^* = 11111100111$.

	1	2	3	4	5	6	7	8	9	A	B	1	2	3	4	5	6	7	8	9	A	B	
S_k	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	S_k^*
S_{k+1}	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1	S_{k+1}^*

Since $S_{k+1} = S_{k+1}^*$, we conclude that the algorithm successfully found the solution vector S_k at the end of Step 4. Even though the updating function $f(\mathbf{x})$ is not linear, we have found a solution immediately after solving a set of linear equations. As was explained in Section 3, this will happen when the seed vector (S_k) entirely consists of those q-tuples at which $f(\mathbf{x})$ and its best affine approximation $g(\mathbf{x})$ agree. We inspect the above table and notice that $f(\mathbf{x})$ and $g_4(\mathbf{x})$ are equal at the following set of 3-tuples: $\mathcal{R}_1 = \{000, 001, 010, 100, 101, 110\}$. On the other hand, the set of all 3-tuples which make up S_k is found as $\mathcal{R}_2 = \{000, 001, 010, 100\}$. Since $\mathcal{R}_2 \subset \mathcal{R}_1$, the application of $f(\mathbf{x})$ to S_k will produce the same S_{k+1} as the application of $g_4(\mathbf{x})$ to S_k.

4.2 Example 2

If there are several predecessors of S_{k+1}, the proposed inversion algorithm will find one of them, which may not be the original S_k used to obtain S_{k+1}. For example, let $n = 8$ and $S_k = 11100100$ which gives $S_{k+1} = 10011111$ under CA30. After solving the linear equations for the affine function $g_4(\mathbf{x})$, we obtain $S_k^* = 00001001$. We then apply CA30 to S_k^*, and calculate $S_{k+1}^* = 10011111$.

	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	
S_k	1	1	1	0	0	1	0	0	0	0	0	0	1	0	0	1	S_k^*
S_{k+1}	1	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	S_{k+1}^*

Since $S_{k+1} = S_{k+1}^*$, we conclude that the algorithm has indeed found a predecessor of S_{k+1}. However, this predecessor is not the original S_k from which we computed S_{k+1}. The application of the other best affine approximations of $f(\mathbf{x})$ may yield other predecessors of S_{k+1}.

4.3 Example 3

In this example, we take $n = 11$ and $S_k = 11001101001$ which gives $S_{k+1} = 00111001111$ under the updating function CA30. We select the affine function $g_1(\mathbf{x})$, and then solve a set of linear equations, and calculate $S_k* = 10001100001$, and finally apply CA30 to S_k^* to calculate $S_{k+1}^* = 01011010011$.

	1	2	3	4	5	6	7	8	9	A	B	1	2	3	4	5	6	7	8	9	A	B	
S_k	1	1	0	0	1	1	0	1	0	0	1	1	0	0	0	1	1	0	0	0	0	1	S_k^*
S_{k+1}	0	0	1	1	1	0	0	1	1	1	1	0	1	0	1	1	0	1	0	0	1	1	S_{k+1}^*

As can be seen from above the state vector S_{k+1}^* is not equal to S_{k+1}, and thus we need to execute Steps 5 through 12 of the inversion algorithm. The state vectors S_{k+1} and S_{k+1}^* differ in bit positions 2, 3, 7, 8, and 9. Thus, we construct five templates as shown below.

Template 2

	1	2	3
a	0	0	0
b	1	0	1
c	1	1	0
d	1	1	1

Template 3

	2	3	4
e	0	0	1
f	0	1	0
g	0	1	1
h	1	0	0

Template 7

	6	7	8
i	0	0	0
j	1	0	1
k	1	1	0
l	1	1	1

Template 8

	7	8	9
m	0	0	1
n	0	1	0
o	0	1	1
p	1	0	0

Template 9

	8	9	A
q	0	0	1
r	0	1	0
s	0	1	1
t	1	0	0

There are two groups of templates, based on the neighborhood principle. The first group contains the templates 2 and 3, while the second group consists of the templates 7, 8, 9. We combine the templates in the first group by retaining those rows of the templates which have the same bit values. For example, the row 'a' in the template 2 and the row 'e' in the template 3 are combined making a row corresponding to the bit positions 1, 2, 3, and 4. The reduced templates are given below:

Template 2, 3

	1	2	3	4
ae	0	0	0	1
bf	1	0	1	0
bg	1	0	1	1
ch	1	1	0	0

Template 7, 8, 9

	6	7	8	9	A
imr	0	0	0	1	0
ims	0	0	0	1	1
jnt	1	0	1	0	0
kpq	1	1	0	0	1

We then exhaustively check all rows in the first reduced template and keep only those rows which complement bit values in positions 2 and 3, while not altering the other bits. The only row in the first reduced template is the row 'ch'. Similarly, in the second reduced template, the row 'jnt' complements the bit values in positions 7, 8, and 9, while not altering the other bit values. Therefore, we found the predecessor of the state S_{k+1}, which is given as

1	2	3	4	5	6	7	8	9	A	B
1	1	0	0	1	1	0	1	0	0	1

where the bit values in positions 1, 2, 3, 4 come from the row 'ch', the bit values in positions 6, 7, 8, 9, A come from the row 'jnt', while the bit values 5 and B are the original values of S_k^*.

4.4 Example 4

Let $n = 11$ and $S_k = 01101000010$ which gives $S_{k+1} = 11001100111$ using the updating function CA30. In this example, we also select the affine function $g_1(\mathbf{x})$ and calculate $S_k^* = 01100110000$ by solving a set of linear equations. We then apply CA30 to S_k^* and compute $S_{k+1}^* = 11011101000$.

	1	2	3	4	5	6	7	8	9	A	B	1	2	3	4	5	6	7	8	9	A	B	
S_k	0	1	1	0	1	0	0	0	0	1	0	0	1	1	0	0	1	1	0	0	0	0	S_k^*
S_{k+1}	1	1	0	0	1	1	0	0	1	1	1	1	1	0	1	1	1	0	1	0	0	0	S_{k+1}^*

By inspection we determine that S_{k+1} and S_{k+1}^* differ in bit positions 4, 8, 9, A, and B. Therefore, the templates are found as

Template 4

	3	4	5
a	0	0	0
b	1	0	1
c	1	1	0
d	1	1	1

Template 8

	7	8	9
e	0	0	0
f	1	0	1
g	1	1	0
h	1	1	1

Template 9

	8	9	A
i	0	0	1
j	0	1	0
k	0	1	1
l	1	0	1

Template A

	9	A	B
m	0	0	1
n	0	1	0
o	0	1	1
p	1	0	0

Template B

	A	B	1
q	0	0	1
r	0	1	0
s	0	1	1
t	1	0	0

The template 4 is put into the first group by itself, while the second group consists of the templates 8, 9, A, and B. The row 'b' is the only row in the first reduced template, which complements the 4th bit while not altering the other bit values. The reduced templates of these two groups are given below:

Template 4

	3	4	5
b	1	0	1

Template 8, 9, A, B

	7	8	9	A	B	1
eint	0	0	0	1	0	0
fjpq	1	0	1	0	0	1
glmr	1	1	0	0	1	0
glms	1	1	0	0	1	1

However, when we try each one of these rows in the above reduced templates in S_k^*, we notice that the correct S_{k+1}^* is not being produced. We conclude that the bit values outside the templates may also need to be changed. Thus, we need to execute Step 12 of the inversion algorithm; in the worst case, all untouched bits (bits 2 and 6) may need to be altered. It turns out that we need to complement bit 6, and keep bit 2 the same in order to obtain the correct solution vector.

1	2	3	4	5	6	7	8	9	A	B
0	1	1	0	1	0	0	0	0	1	0

Here, bits 3, 4, 5 come from the first reduced template (row 'b'), bits 7, 8, 9, A, B, 1 come from the second reduced template (row 'eint'), and bit 6 is the complement of the original bit 6.

5 Further Comments and Conclusions

We note a couple of possible extensions of the inversion algorithm. First, we can use other types of approximations to $f(\mathbf{x})$ instead of the best affine approximation $g(\mathbf{x})$. If the Hamming distance between S_{k+1} and S^*_{k+1} is small, and the resulting system of equations $g(S^*_k) = S_{k+1}$ can be efficiently solved, then the extended inversion algorithm will be successful. Our incentive in using the best affine approximation is due to the fact that this system of equations is linear, and thus, can be solved in $O(nq^2)$ time. As an example, a quadratic function $g(\mathbf{x})$ which has a small Hamming distance to $f(\mathbf{x})$ can be used if the resulting system of quadratic equations can be efficiently solved.

Another extension of the inversion algorithm is its generalization to several rounds. A t-round inversion can be accomplished in two different ways: The first method performs t inverse iterations by applying the inversion algorithm t times. In this process, we start with S_t, and compute $S_{t-1}, S_{t-2}, \ldots, S_0$. Another idea is to find the best affine approximation to the composite function $f \circ f \circ \cdots \circ f$, where the decomposition operator '$\circ$' is applied t times. We then apply the inversion algorithm using this best affine approximation. We start with S_t, and directly calculate S_0 without computing $S_{t-1}, S_{t-2}, \ldots, S_1$. For example, for $q = 3$ and $t = 2$, we have $f(\mathbf{x}) = f(x_{i-1}, x_i, x_{i-1})$, and

$$f(\mathbf{x}) \circ f(\mathbf{x}) = h(\mathbf{x}) = h(x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}) \ . \tag{11}$$

Thus, we start with S_2, and directly calculate S_0 without computing S_1 by using the best affine approximation to the composite function $h(\mathbf{x})$.

As we illustrated in Example 2, the inversion algorithm will calculate a predecessor to the state vector S_{k+1}, which may not be the original one. If a state has several predecessors, then the inversion algorithm may hit any one of them depending on which of the best affine approximations is used. However, it seems that the proportion of states having several predecessors is very small among all possible state vectors. For example, it has been noted in [9] that only a small fraction of $(\kappa/2)^n \approx 0.85^n$ of states do not have unique predecessors for the CA30 based cellular automaton, where κ is the real root of $4\kappa^3 - 2\kappa^2 - 1 = 0$. Thus, the probability that the inversion algorithm would give an incorrect seed value gets smaller as n grows.

Finally, we note that the number of trials given as $2^{(q-1)(1-\alpha)n}$ is a loose upper bound. Since we work with reduced templates by concatenating the initial templates into groups based on the neighborhood principle, a tight upper bound can be computed by obtaining the average number of reduced templates and the average number of rows in a reduced template, and then multiplying these quantities. Another open problem is the proportion of seed values among all possible 2^n values, for which the inversion algorithm computes a predecessor after solving a set of linear equations.

References

[1] K. G. Beauchamp. *Walsh Functions and Their Applications.* New York, NY: Academic Press, 1975

[2] C. Ding, G. Xiao, and W. Shan. *The Stability Theory of Stream Ciphers.* New York, NY: Springer-Verlag, 1991.

[3] S. W. Golomb. *Shift Register Sequences.* Laguna Hills, CA: Aegean Park Press, 1982.

[4] G. H. Golub and C. F. van Loan. *Matrix Computations.* 2nd Edition, Baltimore, MD: The Johns Hopkins University Press, 1989.

[5] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Reading, MA: Addison-Wesley, Second edition, 1981.

[6] W. Meier and O. Staffelbach. Analysis of pseudo random sequences generated by cellular automata. *Advances in Cryptology – Eurocrypt '91*, Lecture Notes in Computer Science, No. 547, D. W. Davies, editor, pages 186–199, Springer-Verlag, 1991.

[7] R. A. Rueppel. *Analysis and Design of Stream Ciphers.* New York, NY: Springer-Verlag, 1986.

[8] R. A. Rueppel. Stream ciphers. In G. J. Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*, pages 65–134. New York, NY: IEEE Press, 1992.

[9] S. Wolfram. Cryptography with cellular automata. In H. C. Williams, editor, *Advances in Cryptology — CRYPTO 85, Proceedings*, Lecture Notes in Computer Science, No. 218, pages 429–432. New York, NY: Springer-Verlag, 1985.

[10] S. Wolfram. Random sequence generation by cellular automata. *Advances in Applied Mathematics*, 7:123–169, June 1986.

[11] S. Wolfram. *Cellular Automata and Complexity.* Reading, MA: Addison-Wesley, 1994.