Algorithms for computing preimages of cellular automata configurations

Iztok Jeras, Andrej Dobnikar

University of Ljubljana, Faculty of Computer and Information Science, Trzaska cesta 25, SI-1001 Ljubljana, Slovenia

Abstract

This paper investigates *preimages* (ancestors or past configurations) of specified configurations of one-dimensional cellular automata. Both *counting* and *listing* of preimages are discussed. The main graphical tools used are the *de Bruijn diagram*, and its extension the *preimage network*, which is created by concatenating de Bruijn diagrams. The counting of preimages is performed as multiplication of topological matrices of de Bruijn diagrams. Listing of preimages is described using two algorithms. The first algorithm traces paths in the preimage network and focuses on local knowledge of the network. The second performs a complete analysis of the network before proceeding with listing.

Key words: cellular automata, counting and listing preimages, ancestors, algorithms
PACS: 89.70.+c, 89.75.-k

1 Introduction

Preimages of one-dimensional cellular automata (CA) have been discussed by many authors with different focuses. Earlier studies questioned reversibility and the existence of "Garden of Eden" states, and most observed statistical properties of CA. The focus of this paper is first to count (compute the number) the preimages (ancestors or past configurations) of a known present

Email addresses: iztok.jeras@rattus.info (Iztok Jeras), andrej.dobnikar@fri.uni-lj.si (Andrej Dobnikar).
URLs: http://www.rattus.info (Iztok Jeras), http://laspp.fri.uni-lj.si/ (Andrej Dobnikar).

configuration and second to list the preimages. The theory is graphically presented using a preimage network, a graph derived from the de Bruijn diagram.

Counting of preimages of one-dimensional CA has been studied by Jen [1] in around 1989 and Voorhees [2] in 1993. They both refer to the de Bruijn diagram, but the problem with their methods is that they use complicated equations like recurrence relations to compute the number of preimages. McIntosh [5] in 1993 simplified the methods to matrix and vector operations. Counting of preimages in this paper uses equations defined by McIntosh with a modified formalization to bring them closer to the formal language and automata theory, and to generalize them to arbitrary CA parameters.

The de Bruijn diagram describes the overlapping of strings and is used in different ways by different authors. This paper uses the diagram as described by McIntosh but introduces a new graphical representation for it: the preimage diagram. Such diagrams can be concatenated to form a preimage network, where all distinct preimages are represented as paths in the network. The preimage network is analyzed using network analysis theory by Batagelj [7] combined with de Bruijn diagram theory by McIntosh.

The listing of preimages is less studied than their counting. For small configurations, preimages can be generated by brute force methods, but since the number of configurations grows exponentially with configuration size this is not possible for large CA. In 1992 Wuensche [3] informally described an algorithm for listing preimages of finite configurations and later implemented it in his DDLab software. In 2004 Mora, Juárez and McIntosh [4] described a different algorithm that uses the subset diagram of the de Bruijn diagram described by McIntosh [5]. This paper presents two algorithms using the preimage network as a tool to describe them graphically. The two algorithms can be seen as extremes, with all the known algorithms placed between them. The trace and backtrack (TB) algorithm uses only local knowledge of the preimage network and lists preimages during tracing, while the count and list (CL) algorithm performs a complete analysis of the whole network before it proceeds to listing. The TB algorithm is almost identical to the one described by Wuensche, while the algorithm by Mora is closer to the CL algorithm. At the end of the paper, the computational complexity and memory consumption of the described algorithms are compared. For a short explanation and comparison of algorithms see [6].

The theory in this paper describes one-dimensional CA that are general in terms of cell values, neighborhood size and whether the configuration is cyclic or bounded. To improve intelligibility of the paper, most figures do not represent general CA but rather the commonly known decimal rule 110. Some examples using this rule are presented at the end of the paper.

The paper begins with a formal definition of one-dimensional CA and a definition of preimages. Then, the text can be roughly divided into two sections: the first about counting preimages and the second about listing preimages. The section on counting preimages begins with the transformation of de Bruijn diagrams into preimage diagrams and further into a preimage network. The matrix representation of diagrams and use of such matrices to count preimages is defined next. The section about listing preimages describes two listing algorithms. Both algorithms are compared to each other and to other known algorithms. In the appendix at the end of the paper there is a collection of examples that show the usage of the theory described in the paper.

2 Nomenclature

Basic symbols	
S	the set of cell states
k	the number of cell states
c	cell state (value)
α, β	strings of cell values
C	configuration
N	configuration length
x	position index
r	neighborhood radius
n	neighborhood or link
o	overlap or node
m	neighborhood size
f	local transition function
F	global transition function
t	time index (present)

Additional symbols	
$\boldsymbol{D}$	preimage matrix
d	elements of $\boldsymbol{D}$
$\boldsymbol{b}$	preimage vector
$\boldsymbol{b}_{\mathrm{L}}$	left boundary vector
$\boldsymbol{b}_{\mathrm{R}}$	right boundary vector
$\boldsymbol{b}_{\mathrm{u}}$	unrestricted boundary v.
p	number of preimages
w_n	link (neighborhood) weight
w_o	node (overlap) weight

Abbreviations	
CA	cellular automata
TB	trace and backtrack alg.
CL	count and list algorithm

3 Formal definition of cellular automata

One-dimensional CA are finite or infinite one-dimensional arrays of cells arranged on a discrete lattice. Each cell can have one of the k available *cell values* or *cell states* $c \in S = \{0, 1, \ldots, k-1\}$. The state of an array of cells is represented by a *string* $\alpha = \ldots c_{x-1}c_xc_{x+1}\ldots$. The string of states of all cells of a finite CA of length N is a *configuration* $C = c_0c_1\ldots c_{N-1}$. The state of the CA at time t is represented by the configuration C^t.

The *neighborhood* $n_x = c_{x-r}\ldots c_{x+r}$ of the cell c_x is the string of cells in a radius r around the observed cell (Figure 1). The size of the neighborhood is $m = 2r + 1$.

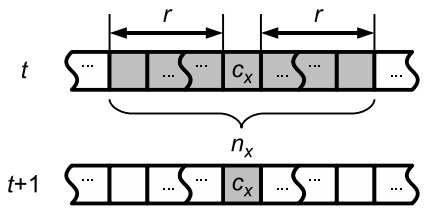

Fig. 1. Neighborhood n_x of the observed cell c_x at the center ($m = 2r + 1$)

At each CA evolution step, all cells from the current configuration C^t synchronously evolve into the future configuration C^{t+1}. The evolution of every single cell c_x^t into its future value c_x^{t+1} is defined as the output of the *local transition function* f that takes as input the present neighborhood n_x^t of the observed cell at position x. The local transition function is commonly called the *rule*.

$$c_x^{t+1} = \mathrm{f}(n_x^t) = \mathrm{f}(c_{x-r}^t \dots c_{x+r}^t)$$

The evolution step of the whole CA is defined as the *global transition function* $C^{t+1} = \mathrm{F}(C^t)$. It is the application of the local transition function to all cells in the configuration simultaneously.

$$\mathrm{F}(c_0 c_1 \dots c_{N-1}) = \mathrm{f}(n_0)\mathrm{f}(n_1) \dots \mathrm{f}(n_{N-1})$$

Neighborhoods of any pair of adjacent cells $c_{x-1}c_x$ overlap over a length of $2r$ cells. The overlap $o_x = c_{x-r}c_{x-r+1} \dots c_{x+r-1}$ uses the position index x from the right cell in the adjacent pair (Figure 2). Another way to think about overlaps is to observe a cell c_x. Then the left overlap is $o_x = c_{x-r}c_{x-r+1} \dots c_{x+r-1}$ and the right overlap is $o_{x+1} = c_{x-r+1}c_{x-r+2} \dots c_{x+r}$. Both the left and the right overlap are parts of the neighborhood n_x.

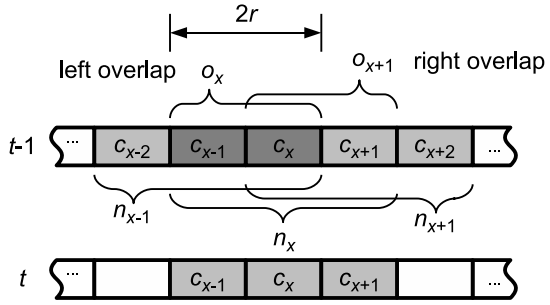

Fig. 2. Overlap o_x at the left and overlap o_{x+1} at the right of the observed cell c_x, as part of the neighborhood n_x ($r = 1$, $2r = 2$)

A *cyclic configuration* is created by joining the left and the right boundary of a finite configuration length N. The position index x becomes a cyclic group of length N.

$$x_{\circlearrowleft} = x \mod N$$

The *cyclic boundary condition* is commonly used since it avoids the explicit definition of the boundary.

A *bounded configuration* is created by cutting a finite configuration length N from an infinite lattice. At the left and the right boundary, neighborhoods overstep the configuration by r cells (Figure 3). To calculate a future configuration C^{t+1}, the $2r$ overstepping cells at both boundaries must be defined. The same problem occurs when calculating preimages, so the preimage C^{t-1} is $2r$ cells longer then the present configuration C^t. The left boundary $B_{\mathrm{L}} = c_{-r} \dots c_{-1}$ and the right boundary $B_{\mathrm{R}} = c_N \dots c_{N-1+r}$ are parts of the preimage $C^{t-1} = c_{-r} \dots c_{N-1+r}$. It is usually assumed that the overstepping cells in the past can have any value. Such boundaries are called unrestricted boundaries.

In this paper the boundary condition is assumed to be cyclic, unrestricted or a specified set of overstepping cell values. In the last case the infinite string outside the boundaries of the observed finite configuration may be implicitly taken into account.

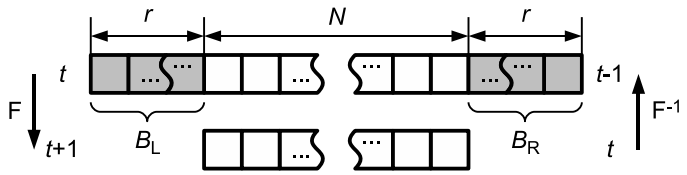

Fig. 3. Neighborhoods overstepping the left and the right boundary on a bounded configuration (forward time direction indexes at the left and backward time direction indexes at the right)

Example 1 *Most of the examples in this paper are based on the elementary ($k = 2$, $r = 1$) rule 110_{10} (Wolfram notation). Each cell can be in two different states. The neighborhood consists of the observed cell and two neighbors $m = 2r + 1 = 3$. The local transition function is defined by the decimal rule 110_{10} (01101110_2 binary). The binary representation of the rule is constructed from outputs of the local transition function for all distinct neighborhoods ($k^m = 8$) as inputs. The transition function is often represented as a transition table:*

$$\underbrace{111_2}_{0}\ \underbrace{110_2}_{1}\ \underbrace{101_2}_{1}\ \underbrace{100_2}_{0}\ \underbrace{011_2}_{1}\ \underbrace{010_2}_{1}\ \underbrace{001_2}_{1}\ \underbrace{000_2}_{0}$$

3.1 Reversing the direction of time

The local transition function and the global transition function define the evolution of the cellular automaton in the forward time direction. To calculate

preimages of the present configuration, inverses of the forward functions must be defined.

The preimages of a single cell c_x^t are *locally valid neighborhoods* n_x^{t-1} that are mapped into the observed cell value by the local transition function (Figure 1). The inverse of the local transition function is defined as:

$$\mathrm{f}^{-1}(c_x^t) = \{n_x^{t-1} \in S^m \mid \mathrm{f}(n_x^{t-1}) = c_x^t\}$$

Preimages C^{t-1} of the present configuration C^t are past configurations that are mapped into the present configuration by the global transition function (Figure 4). The inverse of the global transition function for cyclic configurations is:

$$\mathrm{F}^{-1}(C^t) = \{C^{t-1} \in S^N \mid \mathrm{F}(C^{t-1}) = C^t\} \ ,$$

and for bounded configurations:

$$\mathrm{F}^{-1}(C^t) = \{C^{t-1} \in S^{N+2r} \mid \mathrm{F}(C^{t-1}) = C^t\} \ .$$

The relation between local and global preimages is described later in the paper.

The number p of preimages C^{t-1} can vary from none to k^N or k^{N+2r}, depending on the rule, the present configuration C^t and boundary conditions. This raises questions about the injectivity, surjectivity and reversibility of the rule; however, these topics are not discussed in this paper.

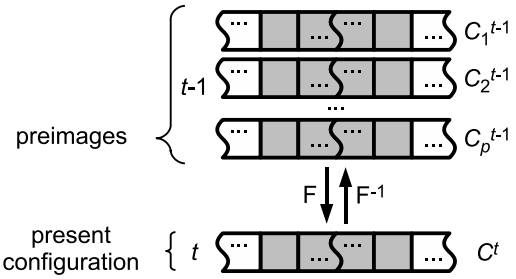

Fig. 4. The forward and inverse global transition functions

3.2 The number notation for strings

The neighborhood and the overlap are strings of cells, but it is often useful to see them as numbers. In this case, k is used as the base of the number, and cell values in the string are digits. For example, when $k = 2$, the strings are binary numbers. The number notation for a string $\alpha = c_0 \dots c_{N-1}$ length N is:

$$\alpha = \sum_{i=0}^{N-1} k^{N-1-i} c_i = k^{N-1} c_0 + \dots + k^1 c_{N-2} + k^0 c_{N-1}$$

The number notation is used to represent and order strings as neighborhoods n, overlaps o and configurations C. When a string of more than one digit is written as a constant, the base of the number is added as a subscript.

4 Counting preimages

De Bruijn diagrams are used to describe how strings can overlap and are the basis of most papers about preimages. Strings of equal length are represented by nodes. Directed links between nodes associate strings that can be overlapped. The source string $a\alpha$ overlaps with the drain string βb if $\alpha = \beta$.

Different methods use the de Bruijn diagram to represent preimages. In this paper the method described by McIntosh [5] is used. To facilitate the description of algorithms for counting and listing preimages, this paper introduces a different graphical representation of the de Bruijn diagram (Figure 5), and names it *preimage diagram* (Figure 8). The preimage diagram represents preimages of a single cell. They can be concatenated into a *preimage network* to represent preimages of a string of cells.

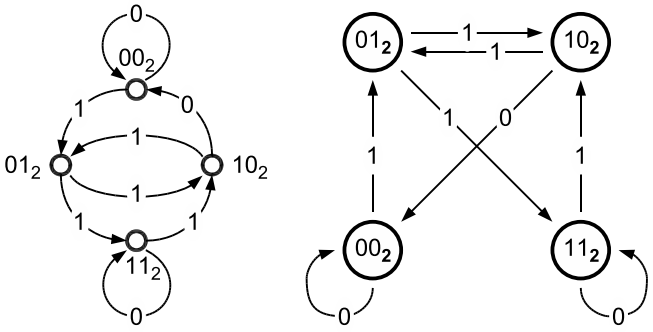

Fig. 5. de Bruijn diagrams for rule 110 as drawn by Jen (left) and McIntosh (right)

4.1 Preimages of a single cell and the preimage diagram

The preimage diagram represents preimages of a single present cell c^t. This preimages are past neighborhoods n^{t-1}. The construction of the preimage diagram will be described in three steps. It starts with the de Bruijn diagram (step I), proceeds with the adaptation for the CA rule (step II) and terminates with decomposition into preimage diagrams (step III).

Step I The size of the de Bruijn diagram (Figure 6) depends on the number of distinct neighborhoods. It is composed of k^{m-1} nodes, one for each of the

overlaps, and k^m directed links, one for each of the neighborhoods. Nodes are drawn twice and arranged into two identical columns (from overlap 0 at the top to overlap $k^{m-1}-1$ at the bottom). The two columns can be seen as overlaps at the left and right side of an observed cell (Figure 2). Directed links connect source nodes o_{s}^{t-1} (left overlaps) to drain nodes o_{d}^{t-1} (right overlaps). Links represent neighborhoods n_{sd}^{t-1}, that contain the source overlap at the left $n_{\mathrm{sd}}^{t-1} = o_{\mathrm{s}}^{t-1} c_{\mathrm{d}}^{t-1}$ and the drain overlap at the right side of the string $n_{\mathrm{sd}}^{t-1} = c_{\mathrm{s}}^{t-1} o_{\mathrm{d}}^{t-1}$ (Figure 7). In the first step links are labeled with neighborhood string values.

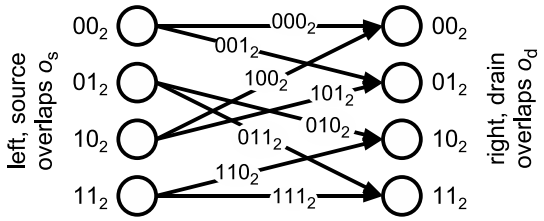

Fig. 6. Step I: de Bruijn diagram adopted for cellular automata (the example represents elementary 1D rules with two cell values $c \in \{0,1\}$, a $m = 3$-cell neighborhood and a $2r = 2$-cell overlap)

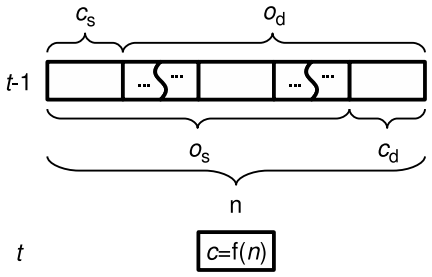

Fig. 7. Decomposition of the neighborhoods into two cell-overlap pairs

Step II To adapt the de Bruijn diagram to the CA rule, link labels are mapped by the local transition function from past neighborhood values n^{t-1} to present cell values $c^t = \mathrm{f}(n^{t-1})$ (Figure 8).

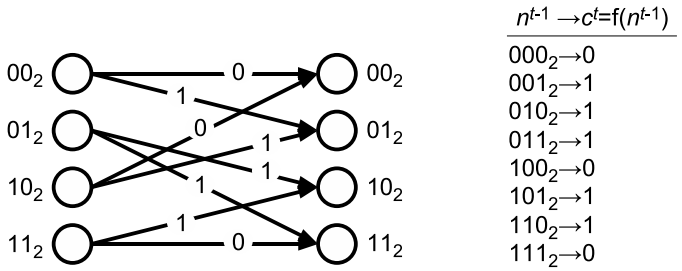

Fig. 8. Step II: links in the de Bruijn diagram are labeled according to the local transition function (the example represents the elementary rule 110)

Step III At the end, the de Bruijn diagram is decomposed into k preimage diagrams (Figure 9), one for each of the available cell values. The preimage diagram represents the history of a single present cell with value c^t. Only locally valid neighborhoods that map into the observed cell by the local transition function $c^t = \mathrm{f}(n^{t-1})$ are allowed in the preimage diagram. All invalid links are removed. Arrows in the diagrams can be omitted, since they always point in the increasing direction of the position index x (from left to right).

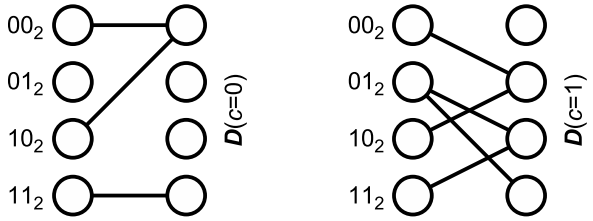

Fig. 9. Step III: the de Bruijn diagram is decomposed into k preimage diagrams (the example represents the elementary rule 110)

Topological matrices $D(c)$ of preimage diagrams for each cell value c are used for counting preimages. Time indexes are omitted in the definition.

Definition 2 *The* single cell preimage matrix $\boldsymbol{D}(c)$ *represents preimages of an observed single present cell with value* c*. It is a square of* $k^{m-1} \times k^{m-1}$ *elements one for each overlap pair* $(o_\mathrm{s}, o_\mathrm{d})$*. The matrix element* $d_{o_\mathrm{s},o_\mathrm{d}}$ *is* 1 *if first, a past neighborhood* $n = o_\mathrm{s}c_\mathrm{d} = c_\mathrm{s}o_\mathrm{d}$ *exists that can be constructed as a link from the source overlap* o_s *to the drain overlap* o_d *and second,* n *is a locally valid neighborhood* $\mathrm{f}(n) = c$*. Else the matrix element is* 0*.*

$$\boldsymbol{D}(c) = \begin{bmatrix} d_{0,0} & d_{0,1} & \cdots \\ d_{1,0} & d_{o_\mathrm{s},o_\mathrm{d}} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \qquad o_\mathrm{s}, o_\mathrm{d} = 0, 1, 2, \ldots, k^{m-1} - 1$$

$$d_{o_\mathrm{s},o_\mathrm{d}} = \begin{cases} 1: & \exists n, c_\mathrm{s}, c_\mathrm{d} : \; o_\mathrm{s}c_\mathrm{d} = c_\mathrm{s}o_\mathrm{d} = n \text{ and } \mathrm{f}(n) = c \\ 0: & \text{else} \end{cases}$$

The above definition for the single cell preimage matrix is a special case of the definition for the cell string preimage matrix, where the string is a single cell long ($|\alpha| = 1$).

Example 3 *For rule 110 there are two cell values* $k = 2$ *and consequently two different single cell preimage matrices (*$\boldsymbol{D}(0)$ *and* $\boldsymbol{D}(1)$*) representing two single cell preimage diagrams (Figure 9).*

$$\boldsymbol{D}(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{D}(1) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

4.2 Preimages of a string of cells and the preimage network

Both the graphical representation and the matrix notation can be extended from single cells into cell strings.

4.2.1 Preimage network

The *preimage network* is an extension of the single cell diagram to a string diagram. As cells are aligned into a string $\alpha = c_0 c_1 \ldots c_{N-1}$, preimage diagrams for each of the cells can also be aligned to form a preimage network (Figure 10). The network is composed of nodes o_x representing overlaps and links n_x representing neighborhoods.

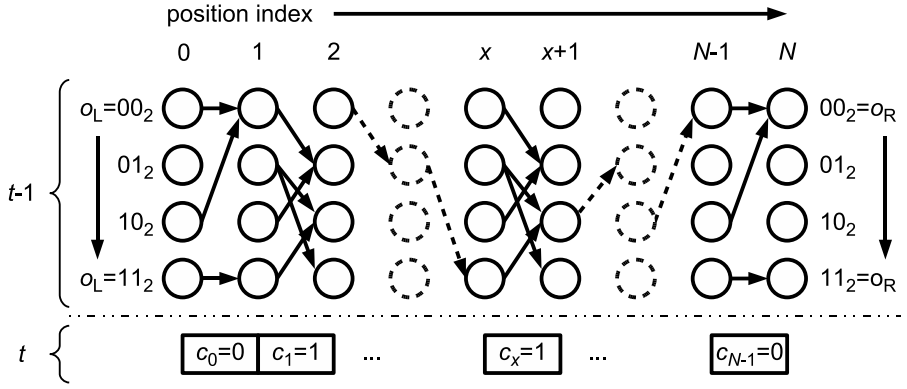

Fig. 10. Preimage network for a string of cells (the example is based on rule 110 CA and the present configuration string $\alpha = 01 \ldots 1 \ldots 0$)

A node o_x is defined by the pair $\langle o, x \rangle$. The overlap value o specifies the node's vertical position. The position index x specifies the node's horizontal position. The maximum fan-in and fan-out of each node is k.

A link n_x is defined by the pair $\langle n, x \rangle$. The neighborhood value n specifies the vertical position of the link's source and drain nodes. The position index x specifies the link's horizontal position.

There are no cycles in the preimage network, although preimages of cyclic configurations can be seen as cycles in the network with joined boundaries.

Nodes at the boundaries have a special notation (Figure 11). Nodes at the left boundary are $o_{\mathrm{L}} = \langle o, x = 0 \rangle$ and nodes at the right boundary are $o_{\mathrm{R}} = \langle o, x = N \rangle$. In cyclic configurations both boundaries represent the same nodes $(0 = N \mod N)$.

4.2.2 Preimages of a string of cells

Distinct preimages have distinct representations as paths in the network. Each preimage C^{t-1} of a present configuration C^t is a *globally valid path* that must begin at one of the overlaps o_{L} at the left boundary, then pass $N-1$ overlaps in between the boundaries, and end at one of the overlaps o_{R} at the right boundary. For cyclic configurations the boundaries are connected, so paths must begin and end at the same overlap $o_{\mathrm{L}} = o_{\mathrm{R}}$. The idea can be extended to configurations of infinite size. Preimages become paths of infinite length.

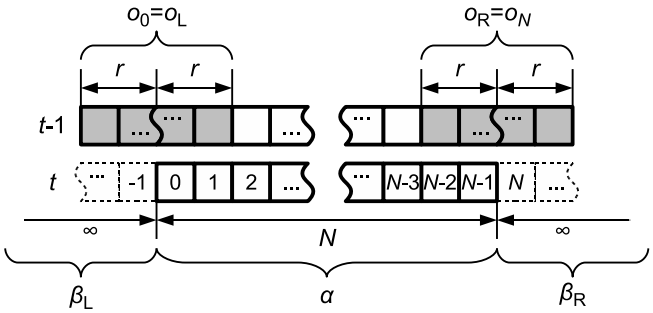

Fig. 11. Overlaps at the configuration boundaries

A preimage matrix can be defined to describe the preimage network in a similar way as for the preimage diagram.

Definition 4 $\boldsymbol{D}(\alpha)$ *is the* cell string preimage matrix *of the observed present string* α *of length* $|\alpha| = N \geq 0$. *Elements* d_{o_L,o_R} *in the preimage matrix represent the number of preimages (distinct paths in the network) that begin at an overlap* o_L *at the left boundary and end at an overlap* o_R *at the right boundary (Figures 10 and 11).*

$$\boldsymbol{D}(\alpha) = \begin{bmatrix} d_{0,0} & d_{0,1} & \cdots \\ d_{1,0} & d_{o_{\mathrm{L}},o_{\mathrm{R}}} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \qquad o_{\mathrm{L}}, o_{\mathrm{R}} = 0, 1, 2, \ldots, k^{m-1} - 1$$

$$d_{o_{\mathrm{L}},o_{\mathrm{R}}}(\alpha) = \begin{cases} \text{the number of distinct paths from } o_{\mathrm{L}} \text{ to } o_{\mathrm{R}} \\ \text{in the preimage network of the string } \alpha \end{cases}$$

Definition 5 *The preimage matrix of an empty string* $\alpha = \varepsilon$ *is an identity matrix.*

$$\boldsymbol{D}(\varepsilon) = I$$

Theorem 6 *The cell string preimage matrix* $\boldsymbol{D}(\alpha)$ *of the string* $\alpha = c_0 c_1 \ldots c_{N-1}$ *is the product of the chain of single cell preimage matrices* $\boldsymbol{D}(c_x)$.

$$\boldsymbol{D}(\alpha) = \prod_{x=0}^{N-1} \boldsymbol{D}(c_x) = \boldsymbol{D}(c_0)\boldsymbol{D}(c_1)\cdots\boldsymbol{D}(c_{N-1})$$

Proof (sketch) The proof is a simple induction on the length of the string. Matrices $\boldsymbol{D}(\varepsilon)$ and $\boldsymbol{D}(c)$ for strings of length 0 and 1 are used for the basis. The inductive step states that from $\boldsymbol{D}(\alpha)$ follows $\boldsymbol{D}(\alpha c) = \boldsymbol{D}(\alpha)\boldsymbol{D}(c)$.

4.3 Boundary conditions

Preimage vector $\boldsymbol{b}$ is a common name in this paper for column vectors that describe a certain property of overlaps. Each preimage vector of k^{m-1} elements refers to a column of overlaps in the preimage network. The described overlap property depends on the usage of the vector; therefore, some preimage vectors are given names that distinguish them. This sections defines boundary vectors $\boldsymbol{b}_\mathrm{L}$ and $\boldsymbol{b}_\mathrm{R}$.

The left preimage boundary B_L and the right preimage boundary B_R are strings of cells length r. There are k^r possible values for this strings. Boundary conditions must state which of these strings can be used to construct preimages and which cannot. In the context of preimages as paths in the preimage network, overstepping strings are seen as part of boundary overlaps (Figures 10 and 11). Boolean boundary vectors ($\boldsymbol{b}_\mathrm{L}$ for the left boundary and $\boldsymbol{b}_\mathrm{R}$ for the right boundary) specify which boundary overlaps can be used to trace preimage paths and which cannot.

Definition 7 $\boldsymbol{b}_\mathrm{L}$ *is the left* Boolean boundary vector. *If* $\boldsymbol{b}_\mathrm{L}(o_\mathrm{L})$ *(the* o_L*-th element of the vector* $\boldsymbol{b}_\mathrm{L}$*) is* 1*, then the left boundary overlap* o_L *is valid and* can *be used to trace preimages; if the element is* 0*, the overlap is invalid and* cannot *be used to trace preimage paths. The definition for the right Boolean boundary vector* $\boldsymbol{b}_\mathrm{R}$ *is similar.*

$$\boldsymbol{b}_\mathrm{L} = \Big[\boldsymbol{b}_\mathrm{L}(0), \boldsymbol{b}_\mathrm{L}(1), \ldots, \boldsymbol{b}_\mathrm{L}(o_\mathrm{L}), \ldots, \boldsymbol{b}_\mathrm{L}(k^{m-1}-1)\Big]^\mathrm{T}$$

$$\boldsymbol{b}_\mathrm{L}(o_\mathrm{L}) = \begin{cases} 1 & : \text{the overlap } o_\mathrm{L} \textit{ can} \text{ be used to trace preimages} \\ 0 & : \text{the overlap } o_\mathrm{L} \textit{ cannot} \text{ be used to trace preimages} \end{cases}$$

Unrestricted boundary vectors are used to get preimages for all distinct boundary strings B_L and B_R.

Definition 8 *For the* unrestricted boundary vector b_u *it is assumed that all boundary overlaps can be used to trace preimages. Therefore, the unrestricted boundary vector contains all ones.*

$$\boldsymbol{b}_\mathrm{u} = [1, 1, \ldots, 1]^\mathrm{T}$$

4.4 The number of preimages

The number of preimages of a bounded configuration defined by the cell string α is computed by applying boundary conditions $\boldsymbol{b}_{\mathrm{L}}$ and $\boldsymbol{b}_{\mathrm{R}}$ to the preimage matrix of the string.

$$p_{\mathrm{L}\leftrightarrow\mathrm{R}}(\alpha) = \boldsymbol{b}_{\mathrm{L}}^{\mathrm{T}}\, \boldsymbol{D}(\alpha)\, \boldsymbol{b}_{\mathrm{R}}$$

Unrestricted boundaries $\boldsymbol{b}_{\mathrm{u}}$ are used to get all distinct preimages.

$$p_{\mathrm{u}\leftrightarrow\mathrm{u}}(\alpha) = \boldsymbol{b}_{\mathrm{u}}^{\mathrm{T}}\, \boldsymbol{D}(\alpha)\, \boldsymbol{b}_{\mathrm{u}}$$

In the case of cyclic configurations there is no outside information, so there is no need to define boundaries. But there is a restriction that a preimage must begin and end with the same overlap. Such preimages can be found on the diagonal of the preimage matrix. A cyclic preimage vector $\boldsymbol{b}_{\circlearrowleft}$ is constructed from the diagonal.

$$\boldsymbol{b}_{\circlearrowleft}(\alpha) = \left[d_{0,0}(\alpha), d_{1,1}(\alpha), \ldots, d_{k^{m-1}-1,k^{m-1}-1}(\alpha)\right]^{\mathrm{T}}$$

The number of all preimages is now the scalar product of an unrestricted boundary vector $\boldsymbol{b}_{\mathrm{u}}$ and the cyclic preimage vector $\boldsymbol{b}_{\circlearrowleft}$.

$$p_{\circlearrowleft}(\alpha) = \boldsymbol{b}_{\mathrm{u}}^{\mathrm{T}} \cdot \boldsymbol{b}_{\circlearrowleft}(\alpha)$$

4.5 Weighted preimage network

In a weighted preimage network each node and link is given a weight. The primary function of weights is to distinguish globally valid nodes and links from those that are only locally valid. The preimage network is constructed from preimage diagrams of a string of cells. By definition of the preimage diagram, all links in the preimage network *are* locally valid. But *not* all links are part of a globally valid path representing a preimage. The knowledge of whether a link is globally valid or not is useful in algorithms for listing preimages.

Node and link weights specify the number of preimages that can be traced through a node or a link in the network. A link or a node is *globally valid* if at least one preimage can be traced through it, or in other words if it can be reached from the left *and* from the right boundary. The last idea is used to compute the weights. Two different methods are given for computing weights for bounded and cyclic configurations.

4.5.1 Preimage network weights for cyclic configurations

Path counters that are needed to compute weights are calculated with a forward and backward counting pass. In the case of a cyclic configuration, overlaps inside the boundary have to be distinguished, since each preimage must start and end with the same overlap. Therefore, counters are computed as preimage matrices $\boldsymbol{D}_{x,\mathrm{f}}$ (forward pass) and $\boldsymbol{D}_{x,\mathrm{b}}$ (backward pass).

The number of paths $p_{o_\mathrm{L}\to o_x}$ between a left boundary overlap o_L and an observed overlap o_x is computed as the element $d_{o_\mathrm{L}o_x,\mathrm{f}}$ that can be found at the o_L-th row and o_x-th column in the matrix $\boldsymbol{D}_{x,\mathrm{f}}$, while the number of paths $p_{o_x\leftarrow o_\mathrm{R}}$ between an observed overlap o_x and a right boundary overlap o_R is computed as the element $d_{o_x o_\mathrm{R},\mathrm{b}}$ that can be found at the o_x-th row and o_R-th column in the matrix $\boldsymbol{D}_{x,\mathrm{b}}$.

$$d_{o_\mathrm{L}o_x,\mathrm{f}} = p_{o_\mathrm{L}\to o_x} \qquad d_{o_x o_\mathrm{R},\mathrm{b}} = p_{o_x\leftarrow o_\mathrm{R}} \qquad o_\mathrm{L}, o_x, o_\mathrm{R} = 0, 1, \ldots, k^{m-1} - 1$$

Counting starts at one of the boundaries, and the path length is incremented cell by cell until it reaches the opposite boundary. Two counting passes are needed. The forward pass begins at the left boundary; it computes counter matrices for each overlap position index $x = 0, 1, \ldots, N$ and ends at the right boundary. The backward pass begins at the right boundary; it computes counter matrices for each overlap position index $x = N, N-1, \ldots, 1, 0$ and ends at the left boundary.

The *forward counting pass* starts at $x = 0$ with an identity matrix $\boldsymbol{D}_{0,\mathrm{f}} = \boldsymbol{I}$. The matrix at position x is calculated from the matrix at position $x - 1$.

$$\boldsymbol{D}_{0,\mathrm{f}} = \boldsymbol{I} \qquad \boldsymbol{D}_{x,\mathrm{f}} = \boldsymbol{D}(c_0 \ldots c_{x-1}) = \boldsymbol{D}_{x-1,\mathrm{f}}\,\boldsymbol{D}(c_{x-1}) \qquad \boldsymbol{D}_{N,\mathrm{f}} = \boldsymbol{D}(\alpha)$$

The *backward counting pass* starts at $x = N$ with an identity matrix $\boldsymbol{D}_{N,\mathrm{b}} = \boldsymbol{I}$. The matrix at position x is calculated from the matrix at position $x + 1$.

$$\boldsymbol{D}_{N,\mathrm{b}} = \boldsymbol{I} \qquad \boldsymbol{D}_{x,\mathrm{b}} = \boldsymbol{D}(c_x \ldots c_{N-1}) = \boldsymbol{D}(c_x)\,\boldsymbol{D}_{x+1,\mathrm{b}} \qquad \boldsymbol{D}_{0,\mathrm{b}} = \boldsymbol{D}(\alpha)$$

At the end of each counting pass the last computed counter matrix is the preimage matrix of the whole string. Both passes are needed to compute node and link weights, but it is clear that one pass is enough to compute the configuration's preimage matrix and consequently the number of cyclic preimages.

The partial node weight $w_o(o_x, o_\mathrm{L} = o_\mathrm{R})$ of the observed overlap o_x and one of the boundary nodes $o_\mathrm{L} = o_\mathrm{R}$ is the product of the number of paths $p_{o_\mathrm{L}\to o_x}$ from an overlap o_L in the left boundary to the node o_x and the number of paths $p_{o_x\leftarrow o_\mathrm{R}}$ from an overlap o_R in the right boundary to the node o_x (Figure

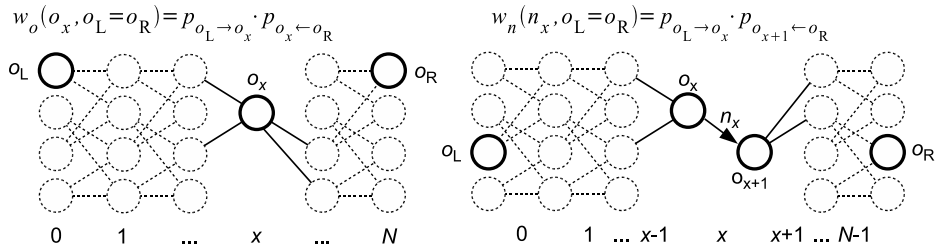

Fig. 12. Node (left) and link (right) weights for cyclic configurations; each boundary overlap is treated separately

12).

$$w_o(o_x, o_\mathrm{L} = o_\mathrm{R}) = p_{o_\mathrm{L}\to o_x} \cdot p_{o_x\leftarrow o_\mathrm{R}}$$

The full node weight $w_o(o_x)$ is the sum of partial node weights over all the boundary nodes.

$$w_o(o_x) = \sum_{o_\mathrm{L}=0}^{k^{m-1}-1} w_o(o_x, o_\mathrm{L} = o_\mathrm{R}) \qquad \begin{matrix} o_x = 0, 1, \ldots, k^{m-1} - 1 \\ x = 0, 1, \ldots, N \end{matrix}$$

The partial link weight $w_n(n_x, o_\mathrm{L} = o_\mathrm{R})$ of the observed locally valid neighborhood $n_x = o_x c_{x+r} = c_{x-r} o_{x+1}$ and one of the boundary nodes $o_\mathrm{L} = o_\mathrm{R}$ is the product of the number of paths $p_{o_\mathrm{L}\to o_x}$ from an overlap o_L in the left boundary to the link's source node o_x and the number of paths $p_{o_{x+1}\leftarrow o_\mathrm{R}}$ from an overlap o_R in the right boundary to the link's drain node o_{x+1} (Figure 12). The weights of locally invalid links are zero.

$$w_n(n_x, o_\mathrm{L} = o_\mathrm{R}) = \begin{cases} p_{o_\mathrm{L}\to o_x} \cdot p_{o_{x+1}\leftarrow o_\mathrm{R}} & : \mathrm{f}(n_x^{t-1}) = c_x^t \\ 0 & : \mathrm{f}(n_x^{t-1}) \neq c_x^t \end{cases}$$

The full link weight $w_n(n_x)$ is the sum of partial link weights over the boundary nodes.

$$w_n(n_x) = \sum_{o_\mathrm{L}=0}^{k^{m-1}-1} w_n(n_x, o_\mathrm{L} = o_\mathrm{R}) \qquad \begin{matrix} n_x = 0, 1, \ldots, k^m - 1 \\ x = 0, 1, \ldots, N - 1 \end{matrix}$$

4.5.2 Preimage network weights for bounded configurations

In the case of a bounded configuration there is no need to distinguish between overlaps inside the boundary, since boundaries can be specified independently. Forward counters $p_{b_\mathrm{L}\to o_x}$ and backward counters $p_{o_x\leftarrow b_\mathrm{R}}$ contain the number of preimages from each overlap o_x to all the overlaps at one of the boundaries.

$$p_{b_\mathrm{L}\to o_x} = \sum_{o_\mathrm{L}=0}^{k^{m-1}-1} p_{o_\mathrm{L}\to o_x} \qquad p_{o_x\leftarrow b_\mathrm{R}} = \sum_{o_\mathrm{R}=0}^{k^{m-1}-1} p_{o_x\leftarrow o_\mathrm{R}}$$

These are stored in counter vectors $\boldsymbol{b}_{x,\mathrm{f}}$ (forward pass) and $\boldsymbol{b}_{x,\mathrm{b}}$ (backward pass).

$$\begin{aligned}\boldsymbol{b}_{x,\mathrm{f}} &= [p_{b_\mathrm{L}\to 0}, p_{b_\mathrm{L}\to 1}, \ldots, p_{b_\mathrm{L}\to o_x}, \ldots, p_{b_\mathrm{L}\to k^{m-1}-1}]^\mathrm{T} \\ \boldsymbol{b}_{x,\mathrm{b}} &= [p_{0\leftarrow b_\mathrm{R}}, p_{1\leftarrow b_\mathrm{R}}, \ldots, p_{o_x\leftarrow b_\mathrm{R}}, \ldots, p_{k^{m-1}-1\leftarrow b_\mathrm{R}}]^\mathrm{T}\end{aligned} \qquad x = 0, 1, \ldots, N$$

The *forward counting pass* starts at $x = 0$ with the left boundary vector $\boldsymbol{b}_{0,\mathrm{f}} = \boldsymbol{b}_\mathrm{L}$. The vector at position x is calculated from the vector at position $x - 1$.

$$\boldsymbol{b}_{0,\mathrm{f}}^\mathrm{T} = \boldsymbol{b}_\mathrm{L}^\mathrm{T} \qquad \boldsymbol{b}_{x,\mathrm{f}}^\mathrm{T} = \boldsymbol{b}_\mathrm{L}^\mathrm{T}\,\boldsymbol{D}(c_0 \ldots c_{x-1}) = \boldsymbol{b}_{x-1,\mathrm{f}}^\mathrm{T}\,\boldsymbol{D}(c_{x-1}) \qquad \boldsymbol{b}_{N,\mathrm{f}}^\mathrm{T} = \boldsymbol{b}_\mathrm{L}^\mathrm{T}\,\boldsymbol{D}(\alpha)$$

Forward counter vectors can be computed from forward counter matrices by summing each matrix column into a single element.

$$\boldsymbol{b}_{x,\mathrm{f}}^\mathrm{T} = \boldsymbol{b}_\mathrm{L}^\mathrm{T}\,\boldsymbol{D}_{x,\mathrm{f}}$$

The *backward counting pass* starts at $x = N$ with the right boundary vector $\boldsymbol{b}_{N,\mathrm{b}} = \boldsymbol{b}_\mathrm{R}$. The vector at position x is calculated from the vector at position $x + 1$.

$$\boldsymbol{b}_{N,\mathrm{b}} = \boldsymbol{b}_\mathrm{R} \qquad \boldsymbol{b}_{x,\mathrm{b}} = \boldsymbol{D}(c_x \ldots c_{N-1})\,\boldsymbol{b}_\mathrm{R} = \boldsymbol{D}(c_x)\,\boldsymbol{b}_{x+1,\mathrm{b}} \qquad \boldsymbol{b}_{0,\mathrm{b}} = \boldsymbol{D}(\alpha)\,\boldsymbol{b}_\mathrm{R}$$

Backward counter vectors can be computed from backward counter matrices by summing each matrix row into a single element.

$$\boldsymbol{b}_{x,\mathrm{b}} = \boldsymbol{D}_{x,\mathrm{b}}\,\boldsymbol{b}_\mathrm{R}$$

At the end of each counting pass the last computed counter vector contains the information about all full-length paths in the network. Since one of the boundaries has been applied at the beginning of the pass, only the opposite boundary has to be applied to compute the number of preimages.

$$p_{\mathrm{L}\leftrightarrow\mathrm{R}}(\alpha) = \boldsymbol{b}_{N,\mathrm{f}}^\mathrm{T} \cdot \boldsymbol{b}_\mathrm{R} = \boldsymbol{b}_\mathrm{L}^\mathrm{T} \cdot \boldsymbol{b}_{0,\mathrm{b}}$$

Both passes are needed to compute weights, but it is clear that one pass is enough to compute the number of preimages.

The node weight $w_o(o_x)$ of the observed overlap o_x is the product of the number of paths $p_{b_\mathrm{L}\to o_x}$ from the left boundary $\boldsymbol{b}_\mathrm{L}$ to the node o_x and the number of paths $p_{o_x\leftarrow b_\mathrm{R}}$ from the right boundary $\boldsymbol{b}_\mathrm{R}$ to the node o_x (Figure 13).

$$w_o(o_x) = p_{b_\mathrm{L}\to o_x} \cdot p_{o_x\leftarrow b_\mathrm{R}} \qquad \begin{matrix} o_x = 0, 1, \ldots, k^{m-1} - 1 \\ x = 0, 1, \ldots, N \end{matrix}$$

The link weight $w_n(n_x)$ of the observed locally valid neighborhood $n_x = o_x c_{x+r} = c_{x-r} o_{x+1}$ is the product of the number of paths $p_{b_\mathrm{L}\to o_x}$ from the left

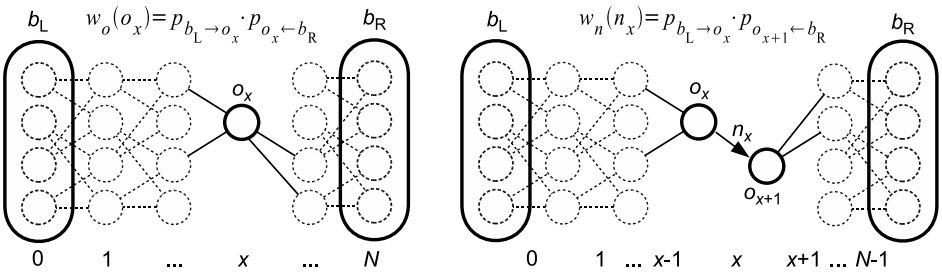

Fig. 13. Node (left) and link (right) weights for bounded configurations; each boundary is treated as a whole

boundary $\boldsymbol{b}_\mathrm{L}$ to the link's source node o_x and the number of paths $p_{o_{x+1}\leftarrow b_\mathrm{R}}$ from the right boundary $\boldsymbol{b}_\mathrm{L}$ to the link's drain node o_{x+1} (Figure 13). The weights of locally invalid links are zero.

$$w_n(n_x) = \begin{cases} p_{b_\mathrm{L}\to o_x} \cdot p_{o_{x+1}\leftarrow b_\mathrm{R}} & : \mathrm{f}(n_x^{t-1}) = c_x^t \\ 0 & : \mathrm{f}(n_x^{t-1}) \neq c_x^t \end{cases} \qquad \begin{matrix} n_x = 0, 1, \ldots, k^m - 1 \\ x = 0, 1, \ldots, N-1 \end{matrix}$$

4.5.3 Graphical representation of the weighted preimage network

The main purpose of preimage network weights is to see globally valid links. In the graphical representation of the preimage network, link weights can be represented by the line's thickness. For large networks, the weights may be too big to be used directly; in such a case the logarithm of the weight may be used for line thickness. Some examples of weighted preimage networks are given in the appendix.

5 Algorithms for listing preimages

Two algorithms are described in detail: the *trace and backtrack* (TB) algorithm and the *count and list* (CL) algorithm. Both algorithms are graphically described using the preimage network, but most of the tools described earlier are only used in the second algorithm. The network can be traveled in the forward and the backward direction. It is a choice that the forward direction is from left to right and the backward direction from right to left. The direction is chosen so that the preimages of bounded configurations are sorted in ascending order with the most significant digit (cell) at the left.

The TB algorithm is similar to the *wall follower* algorithm for solving mazes. The idea is that a person searching for an exit must always stick to the left wall. If he comes to a fork, he takes the first passage on the left. If he gets into a deadend, he must backtrack the walked path to the last fork. There he takes the next passage on the left. This way all the paths in the whole labyrinth can be searched for an exit. To remember the solution, he must write down the passage choice at each fork on the successful path through the maze.

In the TB algorithm the preimage network replaces the maze. Forks are nodes and passages are links in the network. The algorithm searches for all exit paths where left boundary overlaps are start-points and right boundary overlaps are exit-points. A path buffer C_B of the size of a single preimage is used to store the current path. Each time the current path reaches the right boundary, a copy of the path buffer is stored as a preimage and the counter of preimages is incremented. Then the algorithm backtracks and searches for other solutions, until the whole network has been explored and all the preimages have been found.

The TB algorithm uses only knowledge about the local validity of links when choosing which link to trace. Another important aspect of the TB algorithm is low memory consumption; this is because it is sufficient to store a single path in memory. There is a detailed example explaining the TB algorithm in the appendix.

The description of the algorithm is divided into five parts, as shown in the flowchart (Figure 14).

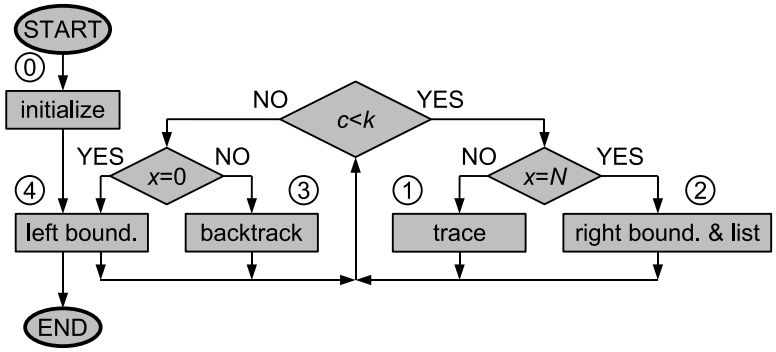

Fig. 14. A simplified flowchart of the trace and backtrack algorithm

(0) Initialize: Tracing starts at the first overlap $o_\mathrm{L} = 0$ at the left boundary $x = 0$. The overlap string is written into the path buffer C_B (Figure 15). The preimage counter is initialized to zero $p = 0$.

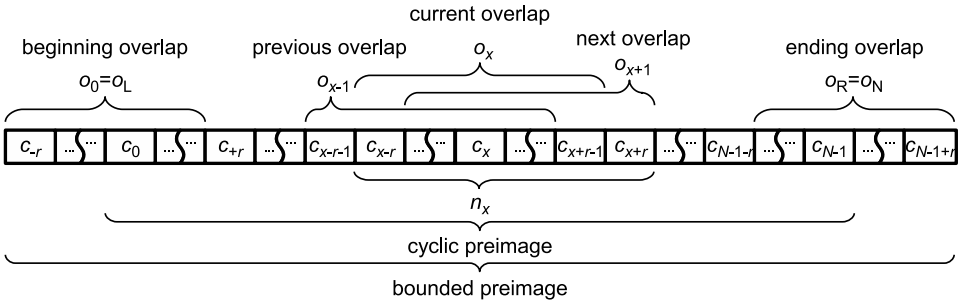

Fig. 15. Path buffer

(1) Trace: Tracing follows a neighborhood (link) n_x from the current overlap o_x to the next overlap o_{x+1} (Figures 15 and 16). In the process the cell position index x is incremented. There are k neighborhoods that can be traced from each overlap. The selected neighborhood $n_x = o_x c_{x+r}$ is constructed from the current overlap o_x by adding a cell at the right. The added cell c_{x+r} is used to label the link to be traced. If the current overlap o_x was reached by tracing (not backtracking), then the first link to be traced is $c_{x+r} = 0$. The local validity of the neighborhood to be traced n_x must be tested with the local transition function $\mathrm{f}(n_x^{t-1}) = c_x^t$ against the observed cell at the present time c_x^t. If the neighborhood is valid, the path is traced and the added cell value is stored in the path buffer $C_\mathrm{B}(x + r) = c_{x+r}$, else the next neighborhood is tried $c_{x+r} = c_{x+r} + 1$. If the value of the incremented link label is less than the number of available links $c_{x+r} < k$, then the algorithm continues with tracing, otherwise it continues with backtracking.

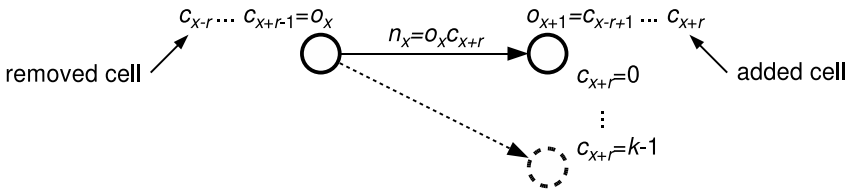

Fig. 16. Tracing step from the current to the next node

(3) Backtrack: When all the available links forward have been consumed $c_{x+r} = k$, the algorithm must backtrack to the last visited overlap o_{x-1} on the path buffer (Figures 15 and 17). The selected neighborhood $n_{x-1} = c_{x-r-1} o_x$ is constructed from the current overlap o_x by adding at the left a cell from the path buffer $C_\mathrm{B}(x - r - 1)$. The new overlap o_{x-1} is constructed from the backtracked neighborhood $n_{x-1} = o_{x-1} c_{x+r-1}$ by removing the rightmost cell. The removed cell c_{x+r-1} labels the last traced link from the overlap o_{x-1}. The cell value c_{x+r-1} is incremented so that the algorithm can continue with tracing an unvisited link. If the value of the incremented link label is less than the number of available links $c_{x+r-1} < k$, then the algorithm continues with tracing, otherwise it continues with backtracking.

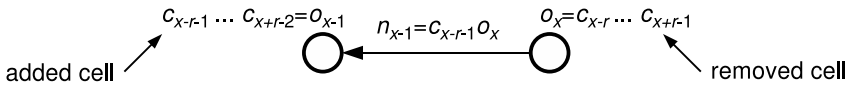

Fig. 17. Backtracking from the current node to the previous node

(2) Right boundary & list: Whenever the right boundary is reached $x = N$, the starting overlap $o_0 = o_\mathrm{L}$ and the current overlap $o_x = o_\mathrm{R}$ must be checked against boundary conditions before a copy of the current path buffer C_B can be stored into the list of preimages (Figure 15). For *cyclic configurations*, starting and ending overlaps must be equal $o_\mathrm{R} = o_\mathrm{L}$ and the preimage has the same length as the present configuration. For *bounded configurations* both overlaps must be valid $\boldsymbol{b}_\mathrm{L}(o_\mathrm{L}), \boldsymbol{b}_\mathrm{R}(o_\mathrm{R}) > 0$ and the preimage is $2r$ cells longer than the present configuration. The preimage counter p is incremented. Since there are no links forward, the algorithm continues with backtracking.

(4) Left boundary: When the left boundary has been reached and the links forward have been consumed $c_{x+r} = k$, the next left overlap $o_\mathrm{L} = o_\mathrm{L} + 1$ is used as a new start-point. When all left boundary overlaps o_L have been consumed ($o_\mathrm{L} = k^{m-1}$), the algorithm ends.

5.1.1 Algorithm pseudo code

initialize, start with the first overlap at the left **(0)**
WHILE (there are available start-points) DO
    IF (there are available links to trace forward) THEN
        IF (*right* boundary not yet reached) THEN **(1)**
            trace the first available *locally valid neighborhood*
        ELSE IF (**right boundary** reached) **(2)**
            apply *boundary conditions* and
            list the path buffer as a preimage
        END IF
    ELSE IF (there are no available links to trace forward)
        IF (*left* boundary not yet reached) THEN **(3)**
            backtrack the current path (stored in the buffer)
        ELSE IF (**left boundary** reached) THEN **(4)**
            move on to the next start-point
        END IF
    END IF
END WHILE

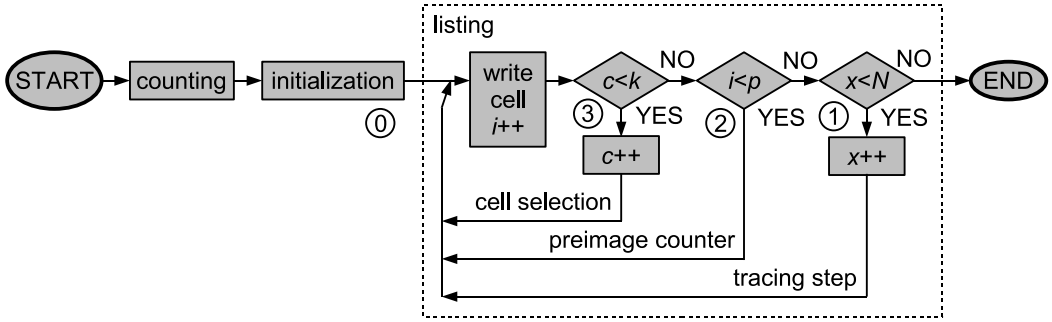

Fig. 18. A simplified flowchart of the count and list algorithm

5.2 Count and list (CL) algorithm

The main difference between the TB and the CL algorithm is that the latter avoids deadends. To accomplish this the algorithm must first analyze the network, to know which overlaps will lead to an endpoint at the right boundary and which will lead to a deadend. This network analysis is performed by the *backward counting pass* (the same direction as backtracking), which was described in the section on weighted preimage networks. The counting analysis not only answers the question of whether overlaps will lead to an endpoint, but it also computes the number of paths that can be traced forward from each overlap in the network, and at the end (left boundary) leads to the number of all preimages.

The data produced by the counting pass is used for *initialization* and the *listing pass* (Figure 18). Initialization prepares an empty structure for the list of preimages and as many tracing start-points (overlaps at the left boundary) as there are preimages. Then the listing pass begins at the start-points and adds cells to preimages column by column (traces them quasi-simultaneously, Figure 19) until the right boundary is reached. Multiple start-points can be seen as thick path roots. These paths are then branched into thinner paths until the endpoints with thickness 1 are reached.

The listing pass is composed of three nested loops (Figures 18 and 19). The *tracing step* (outermost loop) runs position indexes from the left to the right boundary. The preimage counter (middle loop) only counts the modified preimages. The cell selection (innermost loop) provides the cell to be added to each preimage.

(0) Initialization: As with the TB algorithm, listing begins with overlaps at the left boundary. The difference is that only globally valid overlaps are used and that tracing is started simultaneously for all preimages. Where the TB algorithm remembers a single current overlap o_x, the CL algorithm needs an array A_{o_x} of all current overlaps ($|A_{o_x}| = p$). At the beginning of listing,

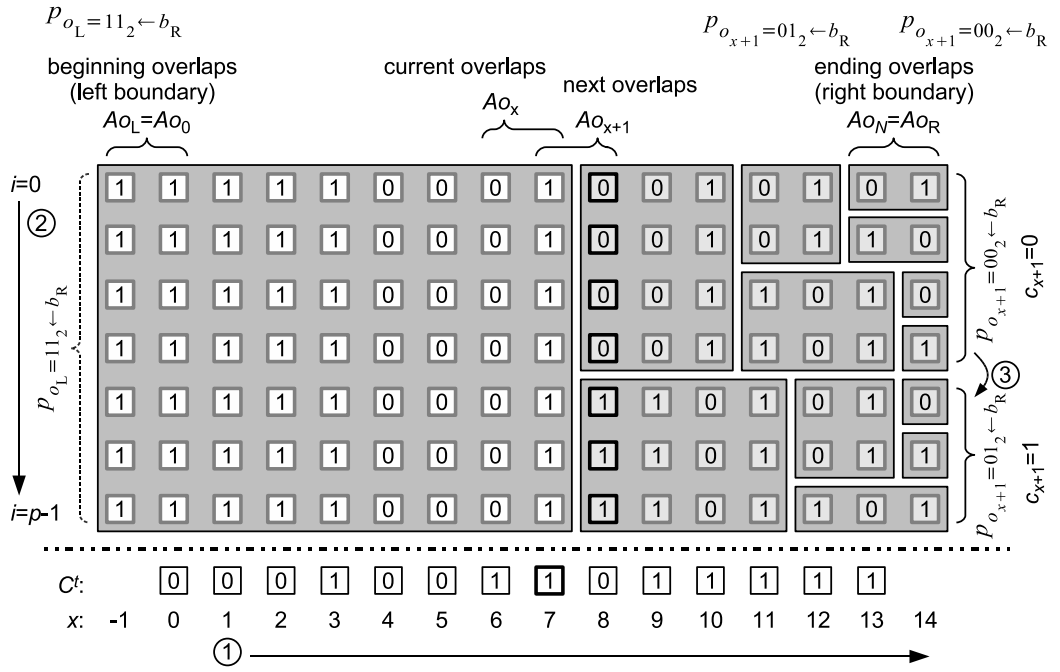

Fig. 19. Computing the list of preimages (rule 110, $k = 2$, $r = 1$, bounded configuration $\alpha = 00010011011111$), observed when $x = 7$ is the observed cell in the present configuration and $x + r = 8$ is the column of cells currently being added to preimages

the array is initialized to overlaps at the left boundary $A_{o_x} = A_{o_\mathrm{L}}$.

For *cyclic configurations* the number $p_{o_L \leftarrow o_\mathrm{R}=o_\mathrm{L}}$ of preimages starting and ending with overlap o_L is the o_L-th element on the diagonal of the backward counter matrix at the left boundary $\boldsymbol{D}_{0,\mathrm{b}} = \boldsymbol{D}(\alpha)$.

$$p_{o_\mathrm{L} \leftarrow o_\mathrm{R}=o_\mathrm{L}} = \boldsymbol{b}_{\circlearrowleft}(o_\mathrm{L}) = d_{o_\mathrm{L},o_\mathrm{L}}(\alpha) \qquad o_\mathrm{L} = 0, 1, \ldots, k^{m-1} - 1$$

$$A_{o_\mathrm{L}} = [\underbrace{0, 0, \ldots, 0}_{p_{0 \leftarrow 0}}, \underbrace{1, 1, \ldots, 1}_{p_{1 \leftarrow 1}}, \ldots, \underbrace{o_\mathrm{L}, \ldots, o_\mathrm{L}}_{p_{o_\mathrm{L} \leftarrow o_\mathrm{R}=o_\mathrm{L}}}, \ldots, \underbrace{k^{m-1} - 1, \ldots, k^{m-1} - 1}_{p_{k^{m-1}-1 \leftarrow k^{m-1}-1}}]^\mathrm{T}$$

Since for cyclic configurations tracing must end with the same overlap it began with, start-points must be remembered. This is achieved by making a copy of the initialized array of current overlaps into a separate array of left boundary overlaps.

For *bounded configurations* the number $p_{o_\mathrm{L} \leftarrow b_\mathrm{R}}$ of traces starting with overlap o_L is the o_L-th element in the backward counter vector at the left boundary $\boldsymbol{b}_{0,\mathrm{b}}$ multiplied by the o_L-th element in the left boundary vector.

$$p_{o_\mathrm{L} \leftarrow b_\mathrm{R}} = \boldsymbol{b}_{0,\mathrm{b}}(o_\mathrm{L}) \cdot \boldsymbol{b}_\mathrm{L}(o_\mathrm{L}) \qquad o_\mathrm{L} = 0, 1, \ldots, k^{m-1} - 1$$

$$A_{o_\mathrm{L}} = [\underbrace{0, 0, \ldots, 0}_{p_{0 \leftarrow b_\mathrm{R}}}, \underbrace{1, 1, \ldots, 1}_{p_{1 \leftarrow b_\mathrm{R}}}, \ldots, \underbrace{o_\mathrm{L}, \ldots, o_\mathrm{L}}_{p_{o_\mathrm{L} \leftarrow b_\mathrm{R}}}, \ldots, \underbrace{k^{m-1} - 1, \ldots, k^{m-1} - 1}_{p_{k^{m-1}-1 \leftarrow b_\mathrm{R}}}]^\mathrm{T}$$

Preimages of bounded configurations begin with a left boundary overlap; therefore, the initialized array of current overlaps A_{o_L} is written into the list of preimages as strings (Figures 15 and 19).

(1) Tracing step: In the main loop of the CL algorithm, the tracing step is performed (Figure 19). At each step another cell value at position $x + r$ is added to all preimages (a column in the structure of preimages) and the cell position index x is incremented. The algorithm ends when all preimages are fully listed, that is, after N steps.

(2) Preimage counter: At the beginning of each tracing step, the column of current preimage cells c_{x+r} is empty. Cells are written into preimages one by one, from the first preimage at the top $i = 0$ to the last at the bottom $i = p-1$ (Figure 19). Each time a cell is written into a preimage, the preimage counter is incremented $i = i + 1$. This is implemented as the middle loop.

(3) Cell selection: In the TB algorithm a cell c_{x+r} is selected and added to the path buffer each time a locally valid link $n_x = o_x c_{x+r}$ is traced from the current overlap o_x to the next overlap o_{x+1}. In the process the next overlap becomes the current overlap. The same happens in the CL algorithm. The current overlap is stored in the array of current overlaps A_{o_x}; when a cell is written to the i-th preimage, the i-th overlap in the array is updated to the new value.

The algorithm tries all the k possible neighborhoods n_x that can be constructed from the current overlap o_x (i-th element of A_{o_x}). If a neighborhood is locally valid $\mathrm{f}(n_x^{t-1}) = c_x^t$ the cell used to construct it c_{x+r} is written into the number of preimages computed by the counting pass. This represents branching of preimage paths (width p_{o_x}) into k subbranches (widths $p_{o_{x+1}}$, where c_{x+r} and consequently o_{x+1} is different for each subbranch).

$$A_{o_x} = [\ldots, \overbrace{o_x, o_x, \ldots \qquad\qquad \ldots, o_x}^{p_{o_x}}, \ldots]^\mathrm{T}$$

$$A_{o_{x+1}} = [\ldots, \overbrace{\underbrace{o_{x+1}, \ldots, o_{x+1}}_{\substack{p_{o_{x+1}} \\ c_{x+r}=0}}, \underbrace{o'_{x+1}, \ldots, o'_{x+1}}_{\substack{p_{o'_{x+1}} \\ c'_{x+r}=1}}, \ldots, \underbrace{o''_{x+1}, \ldots, o''_{x+1}}_{\substack{p_{o''_{x+1}} \\ c''_{x+r}=k-1}}}^{p_{o_x}}, \ldots]^\mathrm{T}$$

For *bounded configurations* the subbranch width is $p_{o_{x+1}} = p_{o_{x+1} \leftarrow b_\mathrm{R}}$ the o_{x+1}-th element of the counter vector $\boldsymbol{b}_{x+1,\mathrm{b}}$. For *cyclic configurations* the subbranch width is $p_{o_{x+1}} = p_{o_{x+1} \leftarrow o_\mathrm{R} = o_\mathrm{L}}$ the element $d_{o_{x+1}, o_\mathrm{L}}$ of the counter matrix $\boldsymbol{D}_{x+1,\mathrm{b}}$, where o_L is the starting overlap for the observed branch of preimages.

When all k neighborhoods following from the current overlap have been tried, a new current overlap is read from the array and processed. This loop continues until $i = p$.

5.2.1 Algorithm pseudo code

counting:
Cyclic: set the counter matrix to the identity matrix
Bounded: set the counter vector to the right boundary vector
FOR (position indexes x from N to 0) DO
    compute the next counter matrix (or vector) and store each of them
END FOR
compute the *number of preimages p*

initialization: (0)
create a list of p empty preimages each having N empty cells
create an array A_{o_x} of current overlaps with p empty entries
FOR (all overlaps o from 0 to $k^{m-1} - 1$) DO
    Cyclic: write the overlap o into $p_{o_{\mathrm{L}}=o\leftarrow o_{\mathrm{R}}=o_{\mathrm{L}}}$ array entries
    Bounded: write the overlap o into $p_{o_{\mathrm{L}}=o\leftarrow b_{\mathrm{R}}}$ array entries
END FOR
Cyclic: copy A_{o_x} into the array of start-point overlaps $A_{o_{\mathrm{L}}}$
Bounded: write A_{o_x} as overlap strings into the list of preimages

listing:
FOR (position indexes x from 0 to $N - 1$) DO **(1)**
    WHILE (there are empty cells at the current position index) DO **(2)**
        FOR (all distinct cell values c from 0 to $k - 1$) DO **(3)**
            IF ($n = o_x c$ is a locally valid neighborhood) THEN
                construct the next overlap o_{x+1} from o_x
                Writing directly into the list of preimages:
                Cyclic: write c into $p_{o_{x+1}\leftarrow o_{\mathrm{L}}}$ empty cells at pos. $x + r \mod N$
                Bounded: write c into $p_{o_{x+1}\leftarrow b_{\mathrm{R}}}$ empty cells at pos. $x + r$
            END IF
        END FOR
    END WHILE
END FOR

5.3 Computation complexity and memory consumption

In both the TB and the CL algorithm the complexity of searching and listing of paths can be treated separately.

Listing complexity is the same for both algorithms. It is a linear function of the size of the lattice N and the number of preimages p. Only the constant C_{list} may be improved.

$$T_{\text{list}} = C_{\text{list}} \cdot N \cdot p$$

Memory consumption for storing preimages is a simple product of the preimages size N or $N + 2r$, their number p and the memory consumption of a single cell C_{cell}. The constant depends on the implementation and is at least $C_{\text{cell}} \geq \log_2 k$ (a cell value can be stored as a single bit or as an integer).

$$M_{\text{list,cyclic}} = C_{\text{cell}} \cdot N \cdot p \qquad M_{\text{list,bounded}} = C_{\text{cell}} \cdot (N + 2r) \cdot p$$

The maximum number of preimages $p_{\max}$ is equal to the maximum number of configurations in a finite CA. The average number of preimages $\overline{p}$ (over all configurations in a finite CA) is one.

$$p_{\max} = k^N \qquad \overline{p} = 1$$

In the TB algorithm searching complexity heavily depends on the present configuration due to different network structures. Since every traced link is later backtracked, it is sufficient to observe tracing. If all links are valid than the upper complexity limit is reached. The number of traced links is the number of branches in a full k-ary tree of depth N. The same equation holds for bounded and cyclic lattices.

$$T_{\text{TB}} \leq C_{\text{trace}} \cdot \sum_{x=0}^{N-1} k^{m-1} \cdot k^x = C_{\text{trace}} \cdot k^{m-1} \cdot \frac{k^N - 1}{k - 1}$$

The maximum complexity is very high for the TB algorithm, but the average complexity is much lower and probably approaches a linear relation to the problem size N.

Memory consumption of the TB algorithm is low; only a buffer for a single preimage is needed.

$$M_{\text{TB}} = C_{\text{cell}} \cdot (N + 2r)$$

The CL algorithm performs searching using the counting pass. Its complexity does not depend on the present configuration, which defines the network structure. Counting is defined as matrix multiplication, but is less complex, because regular and sparse matrices are used. The maximum number of nonzero elements in the single cell preimage matrix $D(c)$ is k^m: at most k elements (overlap fan-out) in each of the k^{m-1} rows. For cyclic configurations, a string preimage matrix (all elements can be nonzero) is multiplied by a cell preimage matrix (sparse). There are k scalar multiplications for each of the $k^{m-1} \times k^{m-1}$ elements in the output matrix.

$$T_{\text{counting, cyclic}} = C_{\text{counting, cyclic}} \cdot N \cdot k^{2m-1}$$

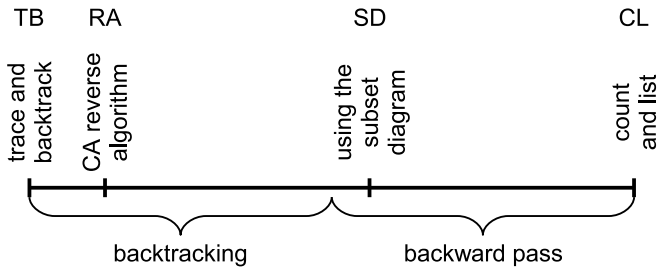

Fig. 20. Overview of known algorithms on a scale (from only local network knowledge on the left to full network knowledge on the right)

For bounded configurations, a preimage vector (all elements can be nonzero) is multiplied by a cell preimage matrix (sparse). There are k scalar multiplications for each of the k^{m-1} elements in the output vector.

$$T_{\text{counting, bounded}} = C_{\text{counting, bounded}} \cdot N \cdot k^m$$

Memory consumption of the CL algorithm is high and is the product of the present configuration size N, the size of the preimage vector or matrix and memory consumption of a single counter element C_{counter} (usually a 32-bit integer).

$$M_{\text{counting, cyclic}} = C_{\text{counter}} \cdot N \cdot k^{(m-1)\cdot 2}$$
$$M_{\text{counting, bounded}} = C_{\text{counter}} \cdot N \cdot k^{m-1}$$

The TB algorithm can be improved to a lower average complexity; on the other hand the CL algorithm is very simple and can be optimized for a low constant factor.

5.4 Comparison of known algorithms

In addition to the two algorithms described in this paper, there are two other algorithms for listing preimages: the *CA reverse algorithm* (RA) (as described by Wuensche in [3c]) and *calculating ancestors using a subset diagram* (SD) (as described by Mora, Juárez and McIntosh [4]). The four algorithms can be sorted depending on how much insight into the structure of the preimage network they have (Figure 20). Two groups can be formed, depending on whether the algorithms need backtracking or they first perform a backward pass of the network to analyze it.

The TB and RA algorithms have only local knowledge of the preimage network, and therefore they sometimes get lost in dead-end paths and have to return to the last fork by backtracking. Many small modifications can be made to the TB algorithm to reduce average complexity, but most would increase

the constant C_{trace} and memory consumption. For example; the RA algorithm stores data about all forks with unused links into a stack, so it can jump back to the last fork in a single backtracking step.

The CL and SD algorithms perform an analysis of the network (first backward pass) before tracing preimages (second forward pass). The usage of the subset diagram in the SD algorithm can be seen as a Boolean version of the counting pass. A lot of memory can be saved, but the total number of preimages is not known until the end of listing. The SD algorithm must still trace each preimage separately, but it never gets lost. Another advantage of the CL algorithm is linear code instead of the recursion used by the SD algorithm. Because of this there is less overhead for function calls and memory allocation.

Another idea described by Mora is calculating preimages two or more steps into the past. This can be done by constructing a local transition function that computes n time steps.

$$c_x^{t+n} = \mathrm{f}^n(c_{x-nr}^t \dots c_{x+nr}^t)$$

But this approach means the growth of the neighborhood size from $m = 2r+1$ to $m' = 2nr + 1$ and the subsequent exponential growth of the de Bruijn diagram and the preimage matrix. This adds a lot to memory consumption, and the profit due to a deeper look into the history is probably equal to or less than the increase in complexity.

6 Concluding remarks

The idea of the article was to focus on counting and listing preimages. A new graphical representation of the de Bruijn diagram is introduced, which can be extended to the preimage network. A graph-theoretical approach has already been used to analyze de Bruijn diagrams, but the preimage network needs a different approach. To this end the analysis of paths in large citation networks as described by Batagelj is used.

Two algorithms for listing preimages are described and compared with the two other known algorithms. It seems there is not much room for algorithmic improvement, since the complexity of all algorithms is already a linear function of configuration size. All the mentioned algorithms can be parallelized, but at the moment there seems to be no need for it.

Weighted preimage networks may be useful for observing local reversibility of configurations. There are localized areas in the network, where multiple paths coincide (locally reversible substrings) and areas where paths diverge into distinct pasts (locally irreversible substrings, causing information loses)

(Figure A.1). Observing local reversibility may lead us to the understanding of information dynamics for CA. This was the main impetus behind this article, but it still lies far ahead.

Information is commonly introduced into deterministic CA as the initial configuration (initial condition or time boundary). Another option is to use space boundaries, but sometimes introducing information into the CA needs to be avoided, which can be done with cyclic configurations or by placing a finite configuration on an infinite inert background (for example, all zero). While running the CA in reverse, boundary vectors can be used to describe the information about the background outside the CA boundaries. The observed string α can be seen as part of an infinite string $\beta_{\mathrm{L}}\alpha\beta_{\mathrm{R}}$ (Figure 11). Boundary vector $\boldsymbol{b}_{\mathrm{L}}$ describes preimages of the left semi-infinite string β_{L} and boundary vector $\boldsymbol{b}_{\mathrm{R}}$ describes preimages of the right semi-infinite string β_{R}. Such boundary vectors can be generalized to contain positive integers or probabilities.

Preimage vectors with Boolean elements are used by McIntosh, Mora and Juárez to construct the subset diagram. The subset diagram can be used as a finite automaton to define the regular language of all Garden of Eden configurations.

Another obvious further development would be to modify the theory for computing preimages in two or more dimensions. This could be used in searching for Garden of Eden configurations in the "Game of Life".

A Examples

For an example of implementation of the algorithms in C language see the source code in [8].

A.1 Rule 110 and characteristic configurations

All the examples here are performed on the elementary rule 110, as defined in Example 1 at the end of Section 3 and Example 3 at the end of Section 4.1.

Two characteristic configurations of rule 110 CA will be observed: the *ether background* and the *quiescent background*.

If the CA evolution of rule 110 from a random initial condition is observed, after some time most of the space is covered with the *ether background*. This is a pattern periodic in space and time. The space period $\alpha = 00010011011111_2$ is 14 cells long and the time period is 7 steps.

The *quiescent background* is, on the other hand, used for simple initial conditions. The future configuration of the quiescent background is again the quiescent background. This pattern is again periodic in space and time. The space period $\alpha = 0_2$ is a single cell long and the time period is a single step.

A.2 The preimage network, counting and listing preimages

The weighted preimage network for the ether background on a bounded configuration can be seen in Figure A.2 and the cyclic version in Figure A.1. Link weights are represented by the thickness of the links. To compute the link weights, the forward and backward counting passes are needed.

A.3 Cyclic configuration

For cyclic configurations, intermediate counters are stored as preimage matrices (Tables A.1 and A.2). The results of the backward count can be used by the CL algorithm. The last computed matrix is the preimage matrix of the whole observed string.

Table A.1
Forward counters for the cyclic example

o	counter matrices
00	1000 1000 1000 1000 0100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
01	0100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
10	0010 1000 1000 1000 0100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
11	0001 0001 0001 0001 0010 1000 1000 0100 0011 1001 0110 0111 0121 0221 0232
α	0 0 0 1 0 0 1 1 0 1 1 1 1 1

Table A.2
Backward counters for the cyclic example

o	counter matrices
00	0000 0000 0000 0000 0232 0232 0232 0000 0121 0121 0111 0110 0011 0100 1000
01	0000 0000 0000 0232 0000 0000 0121 0232 0000 0221 0121 0111 0110 0011 0100
10	0000 0000 0000 0000 0232 0232 0232 0000 0121 0121 0111 0110 0011 0100 0010
11	0232 0232 0232 0232 0000 0000 0000 0121 0111 0111 0110 0011 0100 0010 0001
α	0 0 0 1 0 0 1 1 0 1 1 1 1 1

Node and link weights (Table A.3) are calculated from counters, and a weighted preimage network for the cyclic ether configuration can be drawn (Figure A.1).

Table A.3
Network weights for the cyclic ether configuration

(a) Node (overlap) weights

o	weights
00	0 0 0 0 0 2 2 0 0 1 0 0 0 0 0
01	0 0 0 0 0 0 0 2 0 0 1 1 0 2 0
10	0 0 0 0 2 0 0 0 1 0 1 0 2 0 0
11	2 2 2 2 0 0 0 0 1 1 0 1 0 0 2
α	0 0 0 1 0 0 1 1 0 1 1 1 1 1

(b) Link (neighborhood) weights

n	weights
000	0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0
001	0 0 0 0 0 0 0 2 0 0 1 0 0 0 0
010	0 0 0 0 0 0 0 0 1 0 0 0 1 0 0
011	0 0 0 0 0 0 0 0 1 0 0 1 0 0 2
100	0 0 0 0 2 0 0 0 1 0 0 0 0 0
101	0 0 0 0 0 0 0 0 0 0 0 1 0 2 0
110	0 0 0 2 0 0 0 0 0 0 1 0 1 0 0
111	2 2 2 0 0 0 0 0 0 1 0 0 0 0 0
α	0 0 0 1 0 0 1 1 0 1 1 1 1 1

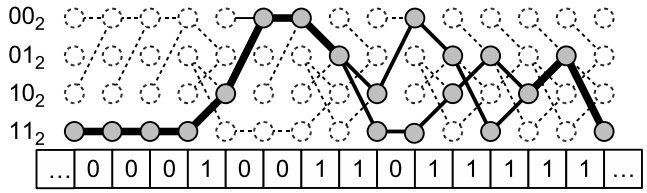

Fig. A.1. Preimage network for the cyclic ether background for rule 110

A.3.1 Counting preimages

The diagonal of the preimage matrix $\boldsymbol{D}(\alpha)$ of the observed string shows $p_{\circlearrowleft} = 2$ preimages for a cyclic configuration.

$$\boldsymbol{D}(\alpha) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 3 & 2 \end{bmatrix} \qquad p_{\circlearrowleft} = \boldsymbol{b}_{\mathrm{u}}^{\mathrm{T}} \boldsymbol{b}_{\circlearrowleft} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 2 \end{bmatrix} = 2$$

A.3.2 The list of preimages

The length of preimages for cyclic configurations is equal to the length of the present string $N = 14$ (Table A.4).

Table A.4
Preimages of the cyclic ether configuration

t-1	11110001001101
	11110001110101
t	00010011011111

A.4 Bounded configuration

For bounded configurations, intermediate counts are stored as preimage vectors (Table A.5). A simple unrestricted boundary is used on both sides.

$$\boldsymbol{b}_{\mathrm{L}} = \boldsymbol{b}_{\mathrm{R}} = \boldsymbol{b}_{\mathrm{u}} = [1, 1, 1, 1]^{\mathrm{T}}$$

Table A.5
Counters for the bounded ether configuration

(a) Forward counters

o	counter vectors
00	1 2 2 2 0 1 1 0 0 1 0 0 0 0 0
01	1 0 0 0 2 0 0 1 0 0 1 1 1 2 2
10	1 0 0 0 1 0 0 0 1 0 1 1 2 2 3
11	1 1 1 1 0 0 0 0 1 1 0 1 1 1 2
α	0 0 0 1 0 0 1 1 0 1 1 1 1 1

(b) Backward counters

o	counter vectors
00	0 0 0 0 7 7 7 0 4 4 3 2 2 1 1
01	0 0 0 7 0 0 4 7 0 5 4 3 2 2 1
10	0 0 0 0 7 7 7 0 4 4 3 2 2 1 1
11	7 7 7 7 0 0 0 4 3 3 2 2 1 1 1
α	0 0 0 1 0 0 1 1 0 1 1 1 1 1

Table A.6
Network weights for the bounded ether configuration

(a) Node (overlap) weights

o	weights
00	0 0 0 0 0 7 7 0 0 4 0 0 0 0 0
01	0 0 0 0 0 0 0 7 0 0 4 3 2 4 2
10	0 0 0 0 7 0 0 0 4 0 3 2 4 2 3
11	7 7 7 7 0 0 0 0 3 3 0 2 1 1 2
α	0 0 0 1 0 0 1 1 0 1 1 1 1 1

(b) Link (neighborhood) weights

n	weights
000	0 0 0 0 0 7 0 0 0 0 0 0 0 0 0
001	0 0 0 0 0 0 0 7 0 0 4 0 0 0 0
010	0 0 0 0 0 0 0 0 4 0 0 2 2 1 2
011	0 0 0 0 0 0 0 0 3 0 0 2 1 1 2
100	0 0 0 0 7 0 0 0 4 0 0 0 0 0 0
101	0 0 0 0 0 0 0 0 0 0 0 3 2 4 2
110	0 0 0 7 0 0 0 0 0 3 0 2 1 1
111	7 7 7 0 0 0 0 0 3 0 0 0 0 0
α	0 0 0 1 0 0 1 1 0 1 1 1 1 1

Node and link weights (Table A.6) are calculated from counters, and a weighted preimage network for the bounded ether configuration can be drawn (Figure A.2).

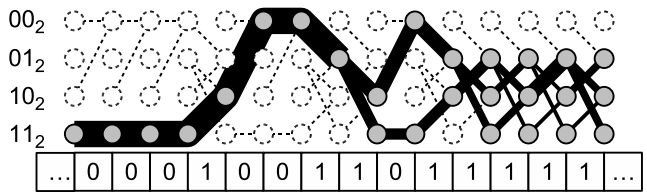

Fig. A.2. Preimage network for the bounded ether background (unrestricted boundaries)

A.4.1 Counting preimages

The number of preimages can be computed from ending count vectors, by applying the ending boundary.

$$p_{\mathrm{L}\leftrightarrow\mathrm{R}} = \boldsymbol{b}_{N,\mathrm{f}}^{\mathrm{T}} \cdot \boldsymbol{b}_{\mathrm{R}} = \begin{bmatrix}0 & 2 & 3 & 2\end{bmatrix} \begin{bmatrix}1\\1\\1\\1\end{bmatrix} = \boldsymbol{b}_{\mathrm{L}}^{\mathrm{T}} \cdot \boldsymbol{b}_{0,\mathrm{b}} = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix} \begin{bmatrix}0\\0\\0\\7\end{bmatrix} = 7$$

Another way to compute the number of preimages is to use the string preimage matrix $\boldsymbol{D}(\alpha)$, which is computed by a counting pass for the cyclic configuration.

$$p_{\mathrm{L}\leftrightarrow\mathrm{R}} = \boldsymbol{b}_{\mathrm{L}}^{\mathrm{T}} \boldsymbol{D}(\alpha) \boldsymbol{b}_{\mathrm{R}} = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix} \begin{bmatrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 2 & 3 & 2\end{bmatrix} \begin{bmatrix}1\\1\\1\\1\end{bmatrix} = 7$$

A.4.2 The list of preimages

Preimages for bounded configurations are two cells longer than the present string $N_{\mathrm{L}\leftrightarrow\mathrm{R}} = N + 2r = 16$ (Table A.7).

Table A.7
Preimages of the bounded ether configuration

t-1	1111100010010101
	1111100010010110
	1111100010011010
	1111100010011011
	1111100011101010
	1111100011101011
	1111100011101101
t	00010011011111

A.5 Boolean boundary conditions

The boundary conditions for the quiescent background $\beta_{\mathrm{L}}, \beta_{\mathrm{R}} = \ldots 000 \ldots_2$ (Figure A.3) can be defined by Boolean boundary vectors.

$$\boldsymbol{b}_{\mathrm{L}} = [1, 0, 0, 1]^{\mathrm{T}} \qquad \boldsymbol{b}_{\mathrm{R}} = [1, 0, 1, 1]^{\mathrm{T}}$$

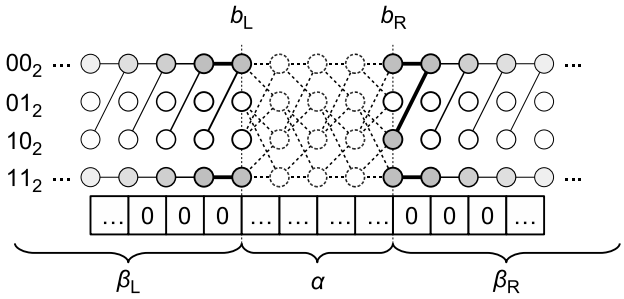

Fig. A.3. Preimage network of a string α on the quiescent background $C = \ldots \beta_{\mathrm{L}} \alpha \beta_{\mathrm{R}} \ldots$

A.6 TB algorithm

The TB algorithm is presented on rule 110 and configuration $C = \alpha = 0011_2$. Figure A.4 shows the preimage network with the path of the trace and backtrack algorithm. Tracing is represented by continuous arrows and backtracking by dashed arrows. The same path is aditionally represented using equations (Table A.8).

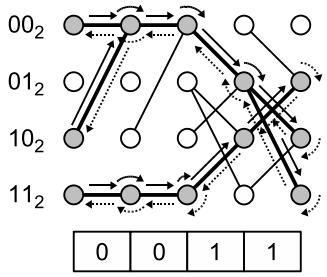

Fig. A.4. Tracing and backtracking through the preimage network of $\alpha = 0011_2$

Every time tracing reaches the right boundary, the starting overlap o_{L} and ending overlap o_{R} are checked against boundary conditions and eventually the traced path is stored as a preimage. There are $p = 5$ preimages for unrestricted boundary conditions $\boldsymbol{b}_{\mathrm{u}}$ and a single $p_{\circlearrowleft} = 1$ preimage for cyclic boundary conditions.

A.7 CL algorithm

The CL algorithm is again presented on rule 110 and configuration $C = \alpha = 0011_2$, and the preimage network is the same as for the previous example (Figure A.4). First, preimage counters must be constructed using the backward counting pass (Table A.9).

For cyclic configurations, backward counting starts at the right boundary with

Table A.8
Steps performed in the TB algorithm

tracing and backtracking path	cyclic pre.	bounded pre.
$o_\mathrm{L} = 00_2 \overset{0}{\to} 00_2 \overset{0}{\to} 00_2 \overset{1}{\to} 01_2 \overset{0}{\to} 10_2 = o_\mathrm{R}$		00 0 0 1 0_2
$01_2 \leftarrow 10_2$		
$01_2 \overset{1}{\to} 11_2 = o_\mathrm{R}$		00 0 0 1 1_2
$00_2 \leftarrow 00_2 \leftarrow 00_2 \leftarrow 01_2 \leftarrow 11_2$		
$o_\mathrm{L} = 01_2$		
$o_\mathrm{L} = 10_2 \overset{0}{\to} 00_2 \overset{0}{\to} 00_2 \overset{1}{\to} 01_2 \overset{0}{\to} 10_2 = o_\mathrm{R}$	0 0 0 1_2	10 0 0 1 0_2
$01_2 \leftarrow 10_2$		
$01_2 \overset{1}{\to} 11_2 = o_\mathrm{R}$		10 0 0 1 1_2
$10_2 \leftarrow 00_2 \leftarrow 00_2 \leftarrow 01_2 \leftarrow 11_2$		
$o_\mathrm{L} = 11_2 \overset{1}{\to} 11_2 \overset{1}{\to} 11_2 \overset{0}{\to} 10_2 \overset{1}{\to} 01_2 = o_\mathrm{R}$		11 1 1 0 1_2
$11_2 \leftarrow 11_2 \leftarrow 11_2 \leftarrow 10_2 \leftarrow 01_2$		

an identity matrix $\boldsymbol{D}_{N=4,\mathrm{b}} = \boldsymbol{I}$ and computes counter matrices until it reaches the left boundary.

For bounded configurations, backward counting begins at the right boundary with an unrestricted boundary vector $\boldsymbol{b}_\mathrm{R} = \boldsymbol{b}_\mathrm{u}$ and computes counter vectors until it reaches the left boundary, which is unrestricted too $\boldsymbol{b}_\mathrm{L} = \boldsymbol{b}_\mathrm{u}$. Counter vectors can be also computed from the counter matrices.

Table A.9
Backward counters for the configuration $\alpha = 0011$

(a) Cyclic configuration

o	counter matrices				
00	0 0 1 1	0 0 1 1	0 0 1 1	0 1 0 0	1 0 0 0
01	0 0 0 0	0 0 0 0	0 1 1 0	0 0 1 1	0 1 0 0
10	0 0 1 1	0 0 1 1	0 0 1 1	0 1 0 0	0 0 1 0
11	0 1 0 0	0 1 0 0	0 1 0 0	0 0 1 0	0 0 0 1
α	0	0	1	1	

(b) Bounded conf.

o	counter vectors				
00	2	2	2	1	1
01	0	0	2	2	1
10	2	2	2	1	1
11	1	1	1	1	1
α	0	0	1	1	

The listing of preimages is described using arrays of current overlaps. There is a single preimage for the cyclic configuration.

$$A_{o_0} = [\ \underbrace{10_2}_{p_{o_\mathrm{L}=10_2 \leftarrow o_\mathrm{R}} = 1}\ ]^\mathrm{T} \quad o_\mathrm{R} = 10_2 \qquad A_{o_1} = [\ \underbrace{00_2}_{p_{o_\mathrm{L}=00_2 \leftarrow o_\mathrm{R}} = 1}\ ]^\mathrm{T} \quad c_1 = 0$$

$$A_{o_2} = [\ \underbrace{00_2}_{p_{o_\mathrm{L}=00_2 \leftarrow o_\mathrm{R}} = 1}\ ]^\mathrm{T} \quad c_2 = 0 \qquad A_{o_3} = [\ \underbrace{01_2}_{p_{o_\mathrm{L}=01_2 \leftarrow o_\mathrm{R}} = 1}\ ]^\mathrm{T} \quad c_3 = 1 \qquad A_{o_4} = [\ \underbrace{10_2}_{p_{o_\mathrm{L}=10_2 \leftarrow o_\mathrm{R}} = 1}\ ]^\mathrm{T} \quad c_0 = 0$$

There are 5 preimages for the bounded configuration.

$$A_{o_0} = [\underbrace{00_2}_{\substack{p_{o_{\mathrm{L}}=00_2 \leftarrow b_{\mathrm{R}}}=2 \\ o_{\mathrm{L}}=00_2}}, \underbrace{00_2}, \underbrace{10_2}_{\substack{p_{o_{\mathrm{L}}=10_2 \leftarrow b_{\mathrm{R}}}=2 \\ o_{\mathrm{L}}=10_2}}, \underbrace{10_2}, \underbrace{11_2}_{\substack{p_{o_{\mathrm{L}}=11_2 \leftarrow b_{\mathrm{R}}}=1 \\ o_{\mathrm{L}}=11_2}}]^{\mathrm{T}}$$

$$A_{o_1} = [\underbrace{00_2}_{\substack{p_{o_1=00_2 \leftarrow b_{\mathrm{R}}}=2 \\ c_1=0}}, \underbrace{00_2}, \underbrace{00_2}_{\substack{p_{o_1=00_2 \leftarrow b_{\mathrm{R}}}=2 \\ c_1=0}}, \underbrace{00_2}, \underbrace{11_2}_{\substack{p_{o_1=11_2 \leftarrow b_{\mathrm{R}}}=1 \\ c_1=1}}]^{\mathrm{T}}$$

$$A_{o_2} = [\underbrace{00_2}_{\substack{p_{o_2=00_2 \leftarrow b_{\mathrm{R}}}=2 \\ c_2=0}}, \underbrace{00_2}, \underbrace{00_2}_{\substack{p_{o_2=00_2 \leftarrow b_{\mathrm{R}}}=2 \\ c_2=0}}, \underbrace{00_2}, \underbrace{11_2}_{\substack{p_{o_2=11_2 \leftarrow b_{\mathrm{R}}}=1 \\ c_2=1}}]^{\mathrm{T}}$$

$$A_{o_3} = [\underbrace{01_2}_{\substack{p_{o_3=01_2 \leftarrow b_{\mathrm{R}}}=2 \\ c_3=1}}, \underbrace{01_2}, \underbrace{01_2}_{\substack{p_{o_3=01_2 \leftarrow b_{\mathrm{R}}}=2 \\ c_3=1}}, \underbrace{01_2}, \underbrace{10_2}_{\substack{p_{o_3=10_2 \leftarrow b_{\mathrm{R}}}=1 \\ c_3=0}}]^{\mathrm{T}}$$

$$A_{o_4} = [\underbrace{10_2}_{\substack{p_{10_2 \leftarrow b_{\mathrm{R}}}=1 \\ c_4=0}}, \underbrace{11_2}_{\substack{p_{11_2 \leftarrow b_{\mathrm{R}}}=1 \\ c_4=1}}, \underbrace{10_2}_{\substack{p_{10_2 \leftarrow b_{\mathrm{R}}}=1 \\ c_4=0}}, \underbrace{11_2}_{\substack{p_{11_2 \leftarrow b_{\mathrm{R}}}=1 \\ c_4=1}}, \underbrace{01_2}_{\substack{p_{o_4=01_2 \leftarrow b_{\mathrm{R}}}=1 \\ c_4=1}}]^{\mathrm{T}}$$

References

[1] Erica Jen, (1989): Enumeration of Preimages in Cellular Automata, *Complex Systems* **3** (5): 421-456.

[2] Burton Voorhees, (1993): Predecessors of cellular automata states II. Pre-images of finite sequences, *Physica D* **73** (1-2): 136-151.

[3a] Andrew Wuensche, Mike Lesser, (1992): *The Global Dynamics of Cellular Automata*, Addison-Wesley.
`http://www.cogs.susx.ac.uk/users/andywu/gdca.html`

[3b] Andrew Wuensche, (1997): *Attractor Basins of Discrete Networks*, Cognitive Science Research Paper 461, Univ. of Sussex, D.Phil thesis.
`ftp://ftp.cogs.susx.ac.uk/pub/users/andywu/papers/aw_thesis.pdf`

[3c] Andrew Wuensche, (1999): Classifying Cellular Automata Automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins, and the Z parameter, *COMPLEXITY*, **4** (3), 47-66. `ftp://ftp.cogs.susx.ac.uk/pub/users/andywu/papers/cplex.pdf`

[4] J. C. S. T. Mora, G. Juárez, H. V. McIntosh, (2004): Calculating ancestors in one-dimensional cellular automata, *International Journal of Modern Physics C*, **15** (8), 1151-1169

[5a] Harold V. McIntosh, (1994): *Linear Cellular Automata Via de Bruijn Diagrams*. `http://delta.cs.cinvestav.mx/~mcintosh/newweb/marcodebruijn.html`

[5b] Harold V. McIntosh, (1993): *Ancestors: Commentaries on The Global Dynamics of Cellular Automata by Andrew Wuensche and Mike Lesser*. `http://delta.cs.cinvestav.mx/~mcintosh/oldweb/wandl/wandl.html`

[6] Iztok Jeras, Andrej Dobnikar, (2006): Cellular Automata Preimages: Count and List Algorithm. *International Conference on Computational Science (3) 2006*: 345-352.

[7] Vladimir Batagelj, (2003): *Efficient Algorithms for Citation Network Analysis*. `http://arxiv.org/abs/cs/0309023`

[8] Iztok Jeras, (2005): *Artificial Life & Cellular Automata*, Source code for listing algorithms. `http://www.rattus.info/al/al.html`