Calculating the Past images of CA Using the Search Path Count Algorithms on the Neighborhood Based Image Network

http://www.rattus.info/stran/al/

Iztok Jeras
email: iztok.jeras@rattus.info
web page: http://www.rattus.info/

University of Ljubljana
Faculty of Computer and Information Science
Trzaska cesta 25, SI-1001 Ljubljana, Slovenia
http://www.fri.uni-lj.si/

Abstract

This article proposes a new algorithm for calculating pre-images of 1D CA. The algorithm works on a 2D network where vertexes are defined by the CA space dimension index and the neighborhood value. The vertexes which neighborhood does not lead to the desired cell value are removed. Each valid pre-image is a path trough this constrained network. The calculation is divided into two passes. The first is a backward pass that marks which parts of the network are to be used in the forward pass. The forward past directly produces the pre-image list.

Keywords

cellular automata, preimages, pre-images, past values, in-degree

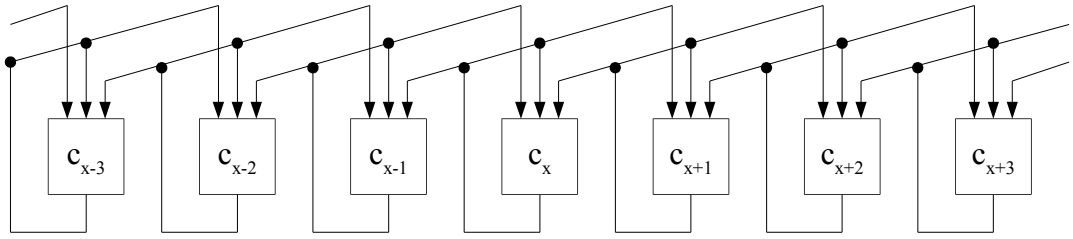
Table of Contents

1 Introduction to Cellular Automata

A CA (cellular automata) is a regular network of cells that can be arranged into different geometries, the most usual are 1D, 2D and 3D blocks with constant sizes in each dimension. This article will focus on 1D CA.

Each cell is a logical element with **K** inputs and a single output that represents the cell value. The inputs into the cell are past cell values. All cells are updated synchronously.

The **K** cells used for the input are chosen according to a fixed geometrical template. An important property of CA is its locality which means that only cells close to each other can be used as inputs to a single element. This input cells are neighbors of the output cell, so the template is called **neighborhood**.



1.1 CA cell element

The discrete cell value is described by a integer number **c** running from **0** to **C-1**. **C** is the number of available cell values but can also be used as a group of available cell values **C**.

$c=(0,1,..C-1)$ where $C=\|C\|$ or $c\in C$

1.2 CA image (configuration)

The CA image **q** or configuration is a list of cell values. The cell position inside the CA is described by the CA dimension index x running from **0** to **X-1**. (for 1D CA **x** is the only space dim. index). **X** is the number of cells inside **q** but can also be used as a group of available space dim. indexes **X**.

$x=(0,1,..X-1)$ where $X=\|X\|$ or $x\in X$

c_0	c_1	c_2			c_{x-1}	c_x	c_{x+1}			c_{X-3}	c_{X-2}	c_{X-1}
0	1	2	...		x-1	x	x+1		...	X-3	X-2	X-1

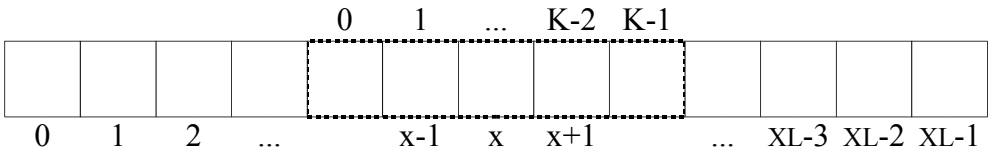
As a compact representation the CA can be described by an integer number **q** (**N** digit base **C**) running from **0** to **Q-1**. **Q** is the number of all possible CA images but can also be used as a group of available CA images **Q**.

$q=(0,1,..Q-1)$ where $Q=\|Q\|=C^X$ or $q\in Q$

$$q=c_{x=0}\cdot C^{X-1}+c_{x=1}\cdot C^{X-2}+...+c_x\cdot C^{X-1-x}+...+c_{x=X-2}\cdot C^1+c_{x=X-1}\cdot C^0$$

1.3 Neighborhood

The neighborhood is a group of cell values used as input to calculate a certain cell value. The position of the neighborhood is described by the same index **x** as the output cell.



A certain cell inside the neighborhood can be described by the index **k** running from **0** to **K-1**. **K** is the number of cell taking part of each neighborhood but can also be used as a group of available indexes **K**.

$k=(0,1,..K-1)$ where $K=\|K\|$ or $k\in K$

The neighborhood representing a certain combination of neighboring cell values $\mathbf{c_k}$ can be described by an integer number **n** (**K** digit base **C**) running from **0** to **N-1**. **N** is the number of available neighborhood combinations but can also be used as a group of available neighborhood combinations **N**.

$n=(0,1,..N-1)$ where $N=\|N\|=C^K$ or $n\in N$

$$n=c_{k=0}\cdot C^{K-1}+c_{k=1}\cdot C^{K-2}+...+c_k\cdot C^{(K-1-k)}+...+c_{k=K-2}\cdot C^1+c_{k=K-1}\cdot C^0$$

1.4 Neighborhood based CA image

In certain literature it is called transition table based CA image. The neighborhood based CA image $\mathbf{q_n}$ is a CA described by the neighborhoods, used to calculate each future cell value, instead of the current cell values. It is usually used as input into filters that emphasize less frequent transition events inside the CA.

n_0	n_1	n_2			n_{x-1}	n_x	n_{x+1}			n_{X-3}	n_{X-2}	n_{X-1}
0	1	2	...		x-1	x	x+1		...	X-3	X-2	X-1

1.5 Transition table

The CA is described by a logical rule **r** that gives an output value **c** for an input neighborhood value **n**.

$c_{out}=f_{rule}(n_{in.})$

A common way to describe this function is a transition table. In this table each input value **n** corresponds to an output value **c**.

n	N-1	N-2	...	n	...	2	1	0
c(n)	c(N-1)	c(n-2)	...	c(n)	...	c(2)	c(1)	c(0)

The shortest way to write a transition table is an integer number **r** (**N** digit base **C**) running from **0** to **R-1**. **R** is the number of available transition table combinations (rules) but can also be used as a group of available rules **R**.

$r=(0,1,..R-1)$ where $R=\|R\|=C^N$ or $r\in R$

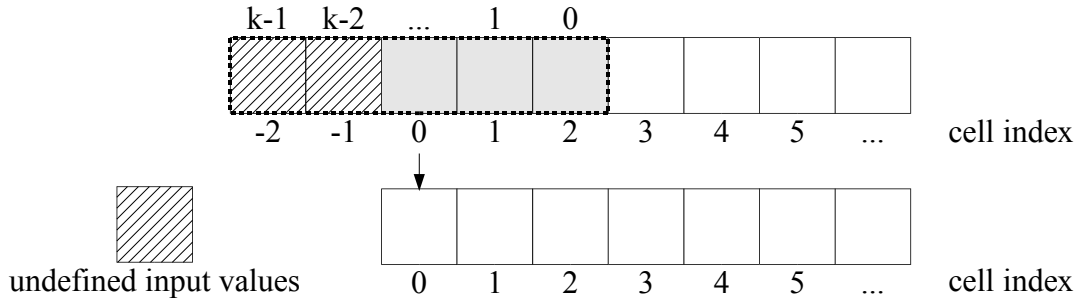
$$r = c_{n=N-1} \cdot C^{N-1} + c_{n=N-2} \cdot C^{N-2} + ... + c_n \cdot C^n + ... + c_{n=1} \cdot C^1 + c_{n=0} \cdot C^0$$

Example: the commonly referred rule 110 (C=2, K=3, R=1, N=8, rule=110)

neighborhood	111	110	101	100	011	010	001	000
n	7	6	5	4	3	2	1	0
c_n	0	1	1	0	1	1	1	0

1.6 Boundaries of the CA space

On the boundaries of the CA space the neighborhood overlaps the CA space.



The value of the overlapping part of the neighborhood can be defined in different ways:

- *constant boundaries*, the undefined cells are constant
- *time variable border,* the undefined cells change their value in time on a known or random basis
- *periodic boundaries*, for each dimension the beginning and the ending boundary are connected, the unknown cells form the beginning are taken from the end and vice versa

- *time delayed periodic boundaries,* for each dimension the beginning border is connected to the ending border, but there is a time delay in the connection, the method can be used for glider analysis
- *infinite boundaries*, periodic infinity, random infinity

Most of the calculations in this article refer to periodic boundaries. These is are commonly used for theoretical CA studies, because such CA do not need additional description of the boundary value.

1.7 CA future

The transition of the CA from its present to its future image is the implementation of the transition rules for each cell's transition from it's input to it's output. The transitions are performed synchronously for all the cells. The group of all inputs (CA_{in}) is equal to

the current CA image. The group of all outputs (CA_{out}) is equal to the CA image after the transition, called the future CA image.

$$q_{out.} = F(q_{in.})$$

The future transition formula is often as a interactive formula with a time variable **t**

$$q_{t+1} = F(q_t)$$

1.8 CA past

2 Constrained Image network

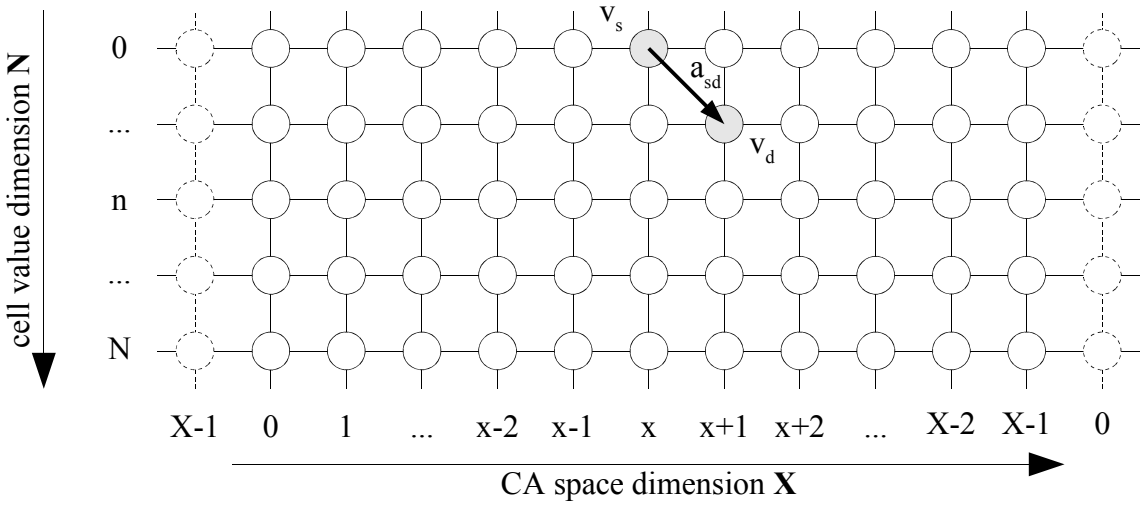
The image network based on neighborhood values $\mathbf{G_n}$ has one additional dimension to the CA space, in this dimension neighborhood values are listed.

$$G_n = X \times N$$

The size of the network is

$$\|G_n\| = \|X\| \cdot \|N\|$$

The elements of the network are vertexes and arcs connecting them.



Vertex

For each cell index **x** there are **N** vertexes in the additional dimension representing all possible neighborhood values. Each vertex is an element of the network $\mathbf{G_n}$.

$$v \in G_n$$

The position of the vertex **v** is defined by its dimension index **x** and by its neighborhood value **n**.

$$v_{\substack{x \\ n}} = v(x, n) \quad \text{where} \quad \begin{matrix} x \in X \\ n \in N \end{matrix}$$

Arc

Arcs are connecting pairs of **compatible** vertexes in the image network.

$v_s = v_s(x_s, n_s)$ source vertex

$v_d = v_d(x_d, n_d)$ drain vertex

For the vertexes to be compatible two requests must be satisfied, the vertexes must be related to a cell pair by their distance in the **x** dimension and by their neighborhood value **n**.

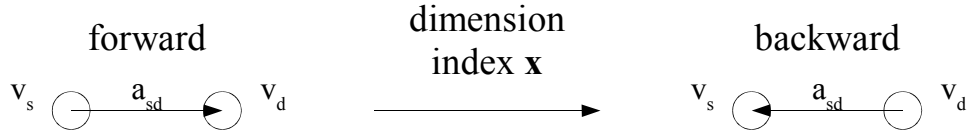
The distance $\Delta\mathbf{x}$ between the connected vertexes must be exactly **1** for the relative cells to be neighboring.

$$|\Delta x| = |x_d(v_d) - x_s(v_s)| = 1$$

If the arc is directed, it is traced from the source $\mathbf{v_s}$ to the drain $\mathbf{v_d}$. The arc can be directed forward or backward.

If $\Delta x = x_d(v_d) - x_s(v_s) = 1$ then the arc is oriented **forward**.

If $\Delta x = x_d(v_d) - x_s(v_s) = -1$ then the arc is oriented **backward**.

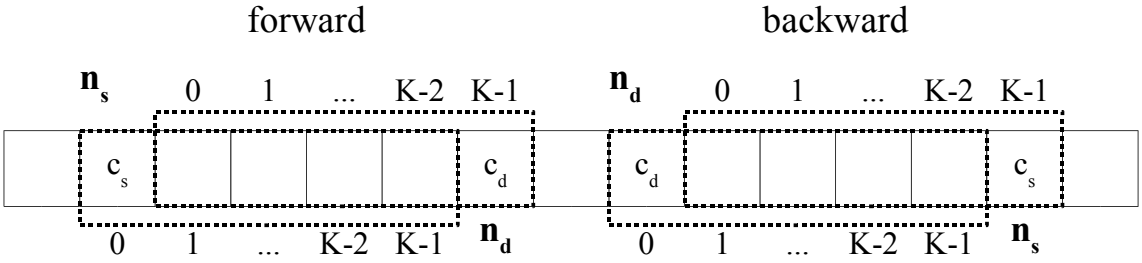


If neighborhood values are to be a part of a CA image they must follow some rules. The overlapping part of the source neighborhood $\mathbf{n_s}$ and the drain neighborhood $\mathbf{n_d}$ must be the equal. For forward oriented arcs this means

$$c_{\substack{n_s \\ k=1}} \cdot C^{K-2} + \ldots + c_{\substack{n_s \\ k=K-1}} \cdot C^0 = c_{\substack{n_d \\ k=0}} \cdot C^{K-1} + \ldots + c_{\substack{n_d \\ k=K-2}} \cdot C^1$$

and for backward oriented arcs

$$c_{\substack{n_d \\ k=1}} \cdot C^{K-2} + \ldots + c_{\substack{n_d \\ k=K-1}} \cdot C^0 = c_{\substack{n_s \\ k=0}} \cdot C^{K-1} + \ldots + c_{\substack{n_s \\ k=K-2}} \cdot C^1$$



The fan-in and fan-out of vertexes are limited to the independent part of the neighborhood pair if one of the neighborhood is fixed.

$$fan_{in.} = fan_{out} = C$$

The position of the directed arc $\mathbf{a_{sd}}$ in the network is defined by the dimension index **x** from the source neighborhood vertex $\mathbf{v_s}$, by its value $\mathbf{n_s}$ and by the non overlapping part $\mathbf{c_d}$ of the drain neighborhood vertex $\mathbf{v_d}$. The index **np** (neighborhood pair) is defined and is calculated from $\mathbf{n_s}$ and $\mathbf{c_d}$ or from $\mathbf{c_s}$ and $\mathbf{n_d}$.

$$a_{v_s, v_d} = a_{\substack{x \\ np}} = a(x, np) \quad \text{where} \quad x = x_s(v_s)$$

The neighborhood pair index is an integer number **np** (**K+1** digit base **C**) running from **0** to **NP-1**. **NP** is the number of available arcs or compatible vertexes at a single index **x** but can also be used as a group of available arcs **NP**.

$$np = (0, 1, .. NP-1) \quad \text{where} \quad NP = \|NP\| = C^{K+1} = N \cdot C \quad \text{or} \quad np \in NP$$

for forward oriented arcs the neighborhood dimension index is

$$np = n_s(v_s) \cdot C + c_d(v_d) = c_s(v_s) \cdot N + n_d(v_d)$$

and for backward oriented arcs the source and drain vertexes are exchanged

$$np = n_d(v_d) \cdot C + c_s(v_s) = c_d(v_d) \cdot N + n_s(v_s)$$

2.1 CA image as a part of the image network

A certain neighborhood based CA image is a path through this network. There are restrictions to the possible paths that derive from the nature of the CA space:

1. all paths must have the same size as the CA (the number of used arcs and vertexes is the same as the number of CA cells)
2. the vertex indexes of a path must monotonically increase or decrease

The second rule suggests the usage of *directed arcs* as part of a *citation network*. The default direction of arcs is the increasing direction of index **x**.

For periodic boundaries paths must be cyclic, so ending at the same vertex as they begin. This article will concentrate on periodic boundaries but it is plain enough to apply the procedures to other boundary types.

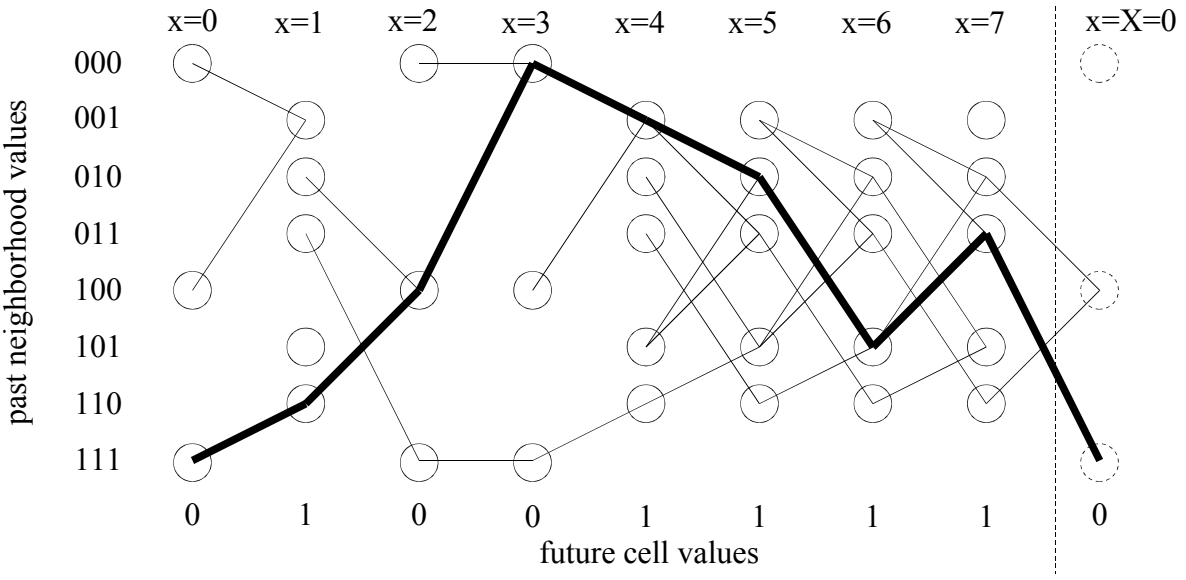
2.2 Applying constraints

Constrained image networks based on neighborhood values are used to calculate the past images of a current CA image. Constraints are based on the cell values of the current CA image and are applied to the neighborhood image network that represents all the possible pasts of the CA. If the past neighborhood value **n** translates into the expected present cell value **c** than it is *locally valid* else it is *locally invalid* and can not take part of a valid past and thus can not be *globally valid*.

$c = f_{rule}(n)$ locally valid neighborhoods

$c \neq f_{rule}(n)$ locally and globally invalid neighborhoods

Example: The next network shows a rule 110 automata (1D, C=2, K=3) with 8 cells (X=8). The last cell (column of input neighborhoods related to the cell) on the right after the dashed line is the repeated image of the first cell (x=X=0). The constraints are applied (invalid vertexes are hidden) and a single globally valid past as a bold path is shown.



2.3 Tracing paths trough the constrained network

The simplest method for finding a path trough the network would be the same used to solve labyrinths. As for the image network such labyrinth must have marked for each road the direction that may lead to the end.

1. The path starts at the BEGIN point and is traced in the forward direction.
2. Arriving to a crossroad one of the proceeding paths is chosen. The same is done for all of the following crossroads.
3. If the chosen path is blind ended than the path must be backtracked to the last crossroad and another path must be chosen. If all the paths at the crossroad have already been tried than backtracking must continue to the previous crossroad.
4. If the END point is reached the traced path is marked as valid.
5. To find all the other possible paths the step 3. must be performed till all the possibilities at all the crossroads are exhausted.

The drawback of this method is the need for backtracking. To avoid it blind choices at crossroads should be marked and avoided. The described algorithm is not meant for a person but can be easily performed by a computer. Here the paths are traced in the forward direction.

1. Start the path at the END point and is traced in the backward direction.
2. At each crossroad duplicate the path tracing agents.
3. At a blind end destroy the corresponding agent.
4. At each crossroad write the number of agents reached it.
5. At the BEGIN point write the number of agents reached it and destroy them.

The usage of multiple agents gives the number of all paths trough the labyrinth, when they are counted at the end point. Because the agents do not have to backtrack their paths they do not need any memory but in that case they do not hold the data on the path they have taken. So the first algorithm must be performed again in the forward direction to collect the data on the paths, with the difference that at all crossroads the choices to be used are described by a number that tells in how many ways they let you proceed to the end.

3 Search Path Count (SPC) algorithm

The SPC algorithm is used to calculate arc weights and vertex weights that represent the number of paths that can be traced between two selected vertexes in a citation network (network with directed arcs).

The search path algorithm has three steps:

1. forward count
2. backward count
3. counts multiplication

The third step produces redundant information for the calculation of pre-images, but it is useful for a graphic representation of the image network.

3.1 The forward and backward count

This are progressive procedures for counting the number of paths connecting a

vertex pair (beginning vertex $\mathbf{v_B}$ and ending vertex $\mathbf{v_E}$). Progressive because they also produce counts for pairs of the beginning vertex and each intermediate vertex $\mathbf{v_i}$. Two counts have to be performed in two directions. For the forward count arcs are oriented in the increasing direction of the dimension index **x** and the count goes from $\mathbf{v_B}$ to $\mathbf{v_E}$, for the backward count the arcs are oriented in the decreasing direction of the dimension index **x** and the count goes from $\mathbf{v_E}$ to $\mathbf{v_B}$ (the opposite direction).

A weight applied to each vertex $\mathbf{v_i}$ represents the forward count from $\mathbf{v_B}$ to $\mathbf{v_x}$

$$wf_{\substack{v_B(x_B,n_B)\\ v_i(x_i,n)}}$$

and another weight applied to the same vertex $\mathbf{v_i}$ represents the backward count $\mathbf{v_E}$ to $\mathbf{v_x}$.

$$wb_{\substack{v_E(x_E,n_E)\\ v_i(x_i,n)}}$$

The counts are performed by progressively stepping trough the CA space dimension index, each step depends on the previous step results. At a single step there are no dependencies based on neighborhood value.

3.1.1 Selecting the beginning and ending vertexes

The procedure must be performed for a group of locally valid vertex pairs.

For **fixed boundaries** the selected vertexes must be from each boundary

$$v_B(x=0,n_B) \quad \text{for} \quad \forall n_B \quad \text{where} \quad c_{x=0}=f_{rule}(n_B)$$

pairs with

$$v_E(x=X-1,n_E) \quad \text{for} \quad \forall n_E \quad \text{where} \quad c_{x=X-1}=f_{rule}(n_E)$$

because the neighborhoods at the boundaries are partly fixed the number of possible pairs is limited to (and further by the local validity zahteva)

$$(R\cdot C)\cdot(R\cdot C)=N\cdot C$$

For **periodic boundaries** the beginning and the ending vertexes are the same and the whole group must have the same CA space dimension index

$$v_B=v_E=v_0(x=x_{0,}n_0)$$

in the group there are all locally valid neighborhoods at that dimension index

$$\forall n_0 \quad \text{where} \quad c_{x=x_0}=f_{rule}(n_0)$$

the number of possible pairs is limited to N (and further by the local validity zahteva)

3.1.2 Initialization

First the count has to be initialized by applying the weight 1 to the beginning vertex $\mathbf{v_B}$, all the other vertexes $\mathbf{v_i}$ at the same dimension index must be 0 for the current count

$$wf_{\substack{v_B(x_B,n_B)\\ v_i(x_i=x_B,n)}}=\begin{Bmatrix}1 & if & n=n_B\\ 0 & if & n\neq n_B\end{Bmatrix}$$ or for the backward count $$wf_{\substack{v_E(x_E,n_E)\\ v_i(x_i=x_E,n)}}=\begin{Bmatrix}1 & if & n=n_E\\ 0 & if & n\neq n_E\end{Bmatrix}$$

3.1.3 Stepping forward or backward

In the first step the index $\mathbf{x_i}$ is incremented (or decremented for backward count)

$x_{i+1}=x_i+1$,for the first step $x_1=x_B+1$

The weight for all locally valid drain vertexes $\mathbf{v_d}$ at the position $\mathbf{x_{i+1}}$ is calculated as the sum of its source vertexes $\mathbf{v_s}$ weights. For locally invalid vertexes $\mathbf{v_d}$ the weight is 0.

$$wf_{\substack{v_B(x_B,n_B)\\ v_d(x_{i+1},n_d)}}=\begin{Bmatrix}\sum\limits_{n_s} wf_{\substack{v_B(x_B,n_B)\\ v_s(x_i,n_s)}} & if & c_{i+1}=f(n_d)\\ 0 & if & c_{i+1}\neq f(n_d)\end{Bmatrix}$$

for the forward count the neighborhood $\mathbf{n_s}$ is

$n_s=floor(c_s(v_s)\cdot N+n_d(v_d),N)$ where the fan-in is $c_s\in C$

and for backward oriented arcs the source and drain vertexes are exchanged

$n_s=mod(c_d(v_d)\cdot N+n_s(v_s),N)$ where the fan-in is $c_s\in C$

Proof: The number of available paths between two vertexes ($\mathbf{v_B}$ and $\mathbf{v_i}$) is equal to the sum of incoming paths passing through a group of intermediate vertexes ($\mathbf{v_{i-1}}$), that do not share the same paths (see the parallel principle). The incoming path count for each intermediate vertex ($\mathbf{v_{i-1}}$) is the product of the path count between $\mathbf{v_B}$ and $\mathbf{v_{i-1}}$ (calculated in the previous step) and the path count between $\mathbf{v_{i-1}}$ and $\mathbf{v_i}$ which is 1 because there is only a single arc connecting them (see the serial principle).

3.1.4 The final step

In the final step the ending vertex must be reached (the beginning vertex for the backward count) and a weight representing the number of paths (valid pasts) passing between the beginning and ending vertex is produced

$$w_{v_B,v_E}=wf_{\substack{v_B,v_E\\ v=v_E(x_E,n_E)}}=wb_{\substack{v_B,v_E\\ v=v_B(x_B,n_B)}}$$

3.1.5 Calculating the number of valid pasts

In the previous steps the path counts $w_{B,E}$ have been calculated for each needed pair beginning vertex $\mathbf{v_B}$ ending vertex $\mathbf{v_E}$ (passing vertex $\mathbf{v_B}=\mathbf{v_E}=\mathbf{v_0}$ for periodic boundaries). The sum of all such path counts gives the total number of valid pasts **P**.

$$P=\sum_{v_B,v_E} w_{v_B,v_E}=\sum_{v_B,v_E} wf_{\substack{v_B,v_E\\ v=v_E(x_E,n_E)}}=\sum_{v_B,v_E} wb_{\substack{v_B,v_E\\ v=v_B(x_B,n_B)}}$$

3.2 Count multiplication

For each vertex and each arc (not only the beginning and ending vertex) the number of valid pasts passing through it can be calculated. First the weight for a pair beginning vertex $\mathbf{v_B}$ ending vertex $\mathbf{v_E}$ (passing vertex $\mathbf{v_B}=\mathbf{v_E}=\mathbf{v_0}$ for periodic

boundaries) must be calculated (see the serial principle).

The formula for vertex path count weights

$$wv_{\substack{v_B,v_E \\ v(x,n)}} = wfv_{\substack{v_B,v_E \\ v(x,n)}} \cdot wbv_{\substack{v_B,v_E \\ v(x,n)}}$$

and the formula for arc weights (the CA space dimension index for arc weights is the same as for forward oriented arcs)

$$wa_{\substack{v_B,v_E \\ a(x,np_{s,d})}} = wf_{\substack{v_B,v_E \\ v_s(x,n_s)}} \cdot wb_{\substack{v_B,v_E \\ v_d(x+1,n_d)}} \quad \text{where} \quad np_{s,d} = n_s(v_s)\cdot C + c_d(v_d) = c_s(v_s)\cdot N + n_d(v_d)$$

The number of pasts passing through each vertex or arc is the sum of multiplied weights for all pairs beginning vertex $\mathbf{v_B}$ ending vertex $\mathbf{v_E}$ (passing vertex $\mathbf{v_B}=\mathbf{v_E}=\mathbf{v_0}$ for periodic boundaries).

for vertexes $wv_{v(x,n)} = \sum_{v_B,v_E} wv_{\substack{v_B,v_E \\ v(x,n)}}$ and for arcs $wa_{a(x,np)} = \sum_{v_B,v_E} wa_{\substack{v_B,v_E \\ a(x,np)}}$

The number of paths that can be traced through the network (valid pasts) is equal at any CA space dimension index

$$P = P_x = \sum_n wv_{v(x,n)} = \sum_{np} wa_{a(x,np)}$$

The multiplied counts are not needed for the calculation of the list of past, but are useful to recognize **globally valid vertexes** that are defined as

$$pv_{v(x,n)} > 0$$

The arc counts can be used for a graphical representation of the image network where arc width is defined by its weight.

4 Creating the list of pasts

In this document the pasts will be created in the forward direction, but it can be done in the reversed direction depending on the direction in which the pasts should be sorted. If the forward direction is chosen, only the backward count is needed.

4.1 Backward count

To perform the backward count first beginning and ending vertexes must be chosen, as described. After the backward count the number of all valid pasts P is already known. Memory for all the past images can be reserved.

4.2 Forward constructing the past images

The forward construction must be performed for the same beginning and ending vertex pairs as the backward count. The next steps must be performed for each pair.

4.2.1 Initialization of past images

The backward count weight at the chosen beginning vertex $\mathbf{v_B}$

$$P_{v_B,v_E} = wb_{\substack{v_B,v_E \\ v=v_B(x_B,n_B)}}$$

represents the number of valid pasts $\mathbf{P_{BE}}$ that have the beginning vertex's neighborhood $\mathbf{n_B}$ at the beginning vertex's dimension index $\mathbf{x_B}$.

To the images of all such pasts neighborhood values $\mathbf{n_B}$ or K cell values must be applied at the dimension index $\mathbf{x_B}$.

4.2.2 Stepping through the CA space dimension

At each step the dimension index is increased

$x_{i+1}=x_i+1$,for the first step $x_1=x_B+1$

For each past the neighborhood value at $\mathbf{x_{i+1}}$ is depending on the neighborhood value at $\mathbf{x_i}$. and the current past index. Staying at the same CA space dimension index $\mathbf{x_{i+1}}$ the cell or neighborhood value must be applied to each of the $\mathbf{P_{BE}}$ pasts.

4.2.2.1 Stepping through the past index

First the current past index $\mathbf{p_{BE}}$ must be initialized to 0.

$p_{v_B,v_E}=0$

For the first step the neighborhood $\mathbf{n_s}$ at $\mathbf{x_i}$ is retrieved from the current past.

For each neighborhood $\mathbf{n_s}$ at index $\mathbf{x_i}$ there are **C** possible neighborhoods $\mathbf{n_d}$ at index $\mathbf{x_{i+1}}$.

$n_d=mod(n_s\cdot C+c_d,N)$ where $c_d\in C$

The fan-out index $\mathbf{c_d}$ is initialized to 0.

$c_d=0$

The backward count weights define how many pasts $\mathbf{P_{Ed}}$ are there with the combination of $\mathbf{n_s}$ and $\mathbf{n_d}$.

$$P_{\substack{v_E \\ v_d}}=wb_{\substack{v_E \\ v=v_d(x_d,n_d)}}$$

The neighborhood values $\mathbf{n_d}$ or the cell value $\mathbf{c_d}$ is applied to $\mathbf{P_{Ed}}$ and for each past the past index $\mathbf{p_{BE}}$ is incremented

$p_{v_B,v_E}=p_{v_B,v_E}+1$

For the next step the neighborhood $\mathbf{n_s}$ at $\mathbf{x_i}$ is again retrieved from the current past.

If its value is the same as the previously retrieved value, than the fan-out index $\mathbf{c_d}$ is incremented.

$c_d=c_d+1$

Else if its value is has changed, than the fan-out index $\mathbf{c_d}$ is again initialized to 0.

$c_d=0$

New nd values are calculated and applied to the past images till $\mathbf{P_{BE}}$ is reached.

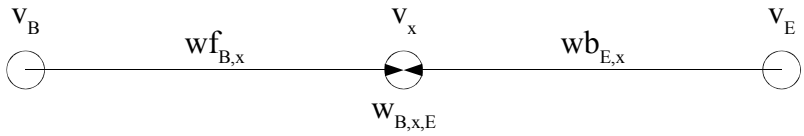
5 Principles used for profs

The serial and parallel principles are the basis for path count analysis of citation networks and are used to prove the other algorithms.

5.1 The serial principle

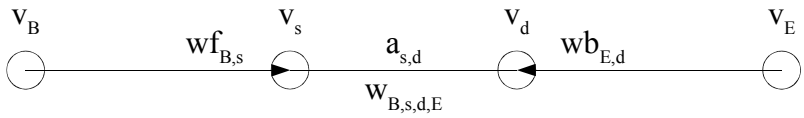
If there are $wf_{B,x}$ possible paths connecting v_x and v_x and $wb_{E,x}$ possible paths connecting v_E and v_x, than there are $w_{B,x,E}$ possible paths passing through v_x connecting v_B and v_E.

$$w_{B,x,E} = wf_{B,x} \cdot wb_{E,x}$$



If there are $wf_{B,s}$ possible paths connecting v_B and v_s and there are $wb_{E,d}$ possible paths connecting v_E and v_d, and there is only one arc $a_{s,d}$ connecting v_s and v_d than there are $w_{D,s,d,E}$ possible paths passing by it.

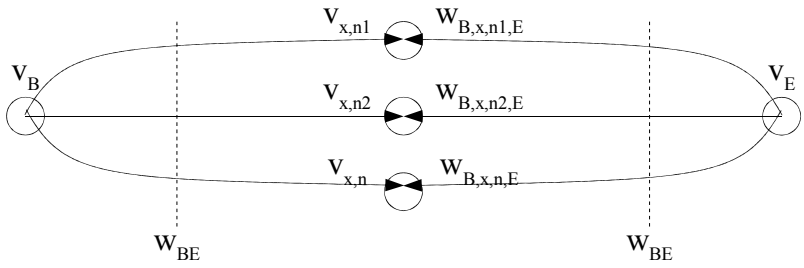
$$w_{B,s,d,E} = wf_{B,s} \cdot wb_{E,d}$$



5.2 The parallel principle

If there are $w_{B,x,n,E}$ paths passing through each $v_{x,n}$ connecting v_B and v_E a and this are all the paths connecting v_B and v_E, than there are a total of w_{BE} paths connecting v_B and v_E.

$$w_{BE} = \sum_n w_{B,x,n,E}$$



6 Implementation considerations

6.1 Periodic space dim. index

6.2 Logical vs. arithmetical operators

The transformation of groups of cell values into integers and back can be done in two ways with some differences:

1. Arithmetical operators, multiplication, division, modulo
2. Logical operators, shift left, shift right, AND

6.3 Memory consumption

There are some big memory consuming variable constructs:

CA image, CA image (neighborhood)

$$mem = X \cdot 8\,MB$$

Backward count parameters

$$mem = X \cdot N \cdot N \cdot 8\,MB$$

Past list

$$mem = P \cdot X \cdot 8\,MB$$

6.4 Indirect vs. lookup table addressing

Indirect addressing means that each variable construct is defined by a single pointer and each element is addressed with a single constructed index.

Advantages: fast access to construct elements, each element has it's exact position and is consequently easy to find, there is no redundancy

Drawbacks: the memory consumption of the construct may grow too big for the OS to find it in one pace, empty elements consume

Multi pointer Indirect addressing means that parts of variable constructs are accessed following many pointers, variable constructs are divided into smaller parts.

Advantages: Smaller parts of memory are needed, so larger constructs may be created without exceeding the OS limit for memory bloc

Drawbacks: the access time to each element increases by a constant depending on the number of pointer levels

Lookup table addressing means that elements of variable constructs must be searched for inside lookup tables.

Advantages: For sparse matrices there is no need to reserve memory for empty elements

Drawbacks: look up tables represent a large overhead, depending of the algorithm

that requests elements the searching of it can be slow

6.5 Integer number limits

The limit value of an unsigned integer is defined by it's Bit length and is different for 32 and 64 Bit processors.

Neighborhood

7 Future work

7.1 Grouping of vertexes into blocks

7.2 2D cellular automata

References

Andrew Wuensche, () "Classifying Cellular Automata Automatically"

Vladimir Batagelj, (2003) "Efficient Algorithms for Citation Network Analysis", http://arxiv.org/abs/cs.DL/0309023