Journal of Computational Science 26 (2018) 39–54

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

A cellular automaton that solves distributed spanning tree problem

Souvik Roy*, Arunima Ray, Sukanta Das

Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, Howrah, West Bengal 711103, India

ARTICLE INFO

Article history:
Received 14 August 2017
Received in revised form 27 January 2018
Accepted 6 March 2018
Available online 10 March 2018

Keywords:
Cellular automata (CAs)
Anonymous systems
Concurrent initiator distributed spanning tree problem
Grid graph

ABSTRACT

This work introduces the distributed spanning tree problem in the domain of cellular automata. We present a cellular automaton that computes a spanning tree of a given (grid) graph. The time required for this computation is $\mathcal{O}(n \log n)$, where n is the number of nodes of the graph.

1. Introduction

A cellular automaton (CA) is defined over a regular grid, each cell of which consists of a finite automaton that interacts with its neighbours to go to its next state [1]. One of the exciting aspects of CA is to solve computational problems, such as leader election problem [2–7], mutual exclusion problem [8], shortest path problem [9], generation of circles and parabola [10], density classification problem [11–13] and firing squad synchronization problem [14]. In this paper we attempt to solve another computational problem, named distributed spanning tree problem [15] using a CA.

1.1. Distributed spanning tree problem

A spanning of a graph is a tree that covers all the vertices of the graph. Finding of spanning tree of a given graph is a well known computational problem. Under distributed stetting of the same problem, it is considered that the nodes of the graph are themselves computing elements and they compute locally. The problem is to find a spanning tree of the graph by means of independent local computation [15].

In this present work, we consider the anonymous instance of the distributed spanning tree problem. In distributed systems, non-anonymous models assume that the system's components (i.e. nodes) are distinguishable, which requires a central entity to assign unique IDs to the components (i.e. nodes). It violates the very essence of a distributed control. However, a distributed system is anonymous if unique identities are not available to distinguish the nodes.

1.2. History of the problem and motivation

There has been a great interest among the distributed system researchers to solve this problem under different assumptions. Many of those contributions are concerned with non-anonymous instance of spanning trees, see for example [15–19], this problem is also addressed under parallel settings [20–27]. In anonymous instance of the problem, all the nodes can initiate the construction of spanning tree concurrently. For anonymous instance of spanning tree construction, Augluin [28] showed that it is impossible to deterministically construct a spanning tree in uniform networks. Therefore, the research effort shifted towards single initiator spanning tree algorithm, see for example [15–19]. In this case, the algorithm requires a central entity to assign a single initiator. Though a single initiator must register first, which again violates the very essence of a distributed control.

One of the properties of CA is anonymous cells, which do not have any pre-defined identification number. In fact, this property makes the problem interesting in the domain of cellular automata (CAs), because a CA based solution to this problem could imply the existence of solutions of the same problem in anonymous systems. This is one of the motivations of undertaking this research. In the literature of cellular automata, few researchers have explored dif-

* Corresponding author.
E-mail addresses: svkr89@gmail.com (S. Roy), rayarunima55@gmail.com (A. Ray), sukanta@it.iiests.ac.in (S. Das).

https://doi.org/10.1016/j.jocs.2018.03.005
1877-7503/

ferent new type of cellular automata-based, *Physarum*-inspired,[1] network design models which can also solve the distributed spanning tree problem [29–32].

In this context, it is worth mentioning that the problem is closely related to the *leader election problem*. Electing leader in the context of spanning tree problem indicates finding of a single initiator for the spanning tree problem. The literature of leader election problem as a cellular automaton computational task is remarkably rich; see for example [2–6]. Note that, tree computation and leader election is also widely used in algorithms of reconfiguration of array processors [33,34].

In this scenario, this paper addresses the distributed spanning tree problem as a *new* cellular automaton computational problem. In the proposed solution, all the nodes of input graph initiate the computation.

2. Overview of the computation

2.1. The CA model

Classically, cellular automata (CAs) are dynamical systems that are homogeneous and discrete in both time and space [1]. A CA can be defined as a two-dimensional grid $\mathcal{L} \subseteq \mathbb{Z}^2$ of identical automata (cells). Formally, CA is a triple $(\mathcal{S}, \mathcal{N}, f)$, where $\mathcal{S}$ is the finite set of states, $\mathcal{N}$ is the self and non-orthogonal four neighbouring cells which is the standard *von Neumann* neighbourhood,[2] $f : \mathcal{S}^5 \longrightarrow \mathcal{S}$ is a local rule or transition function of the automaton. Here, $\Sigma \subseteq \mathcal{S}$ denotes the set of *input alphabet*.

At any given time, the configuration of the automaton is a mapping $m : \mathcal{L} \longrightarrow \mathcal{S}$ that specifies the states of all cells. The set $\mathcal{S}^{\mathcal{L}}$ of all configurations is denoted by $\mathcal{C}$. After one time stamp, the next configuration of c becomes $e = G(c)$; $G : \mathcal{C} \longrightarrow \mathcal{C}$ is the *global transition function* of the CA. During the evolution of a CA, it generates a sequence of configurations, and sometime approaches to fixed point. A fixed point is a CA configuration, next configuration of which is the configuration itself.

CAs are the natural model of computation which can compute a function $\mathcal{T}$. We call that a CA computes a function $\mathcal{T}$ if an input to $\mathcal{T}$ is an initial configuration $\mathcal{I} \subset \mathcal{C}$ of the CA, and the final configuration $\mathcal{F} \subset \mathcal{C}$ is the evaluated value of $\mathcal{T}$. Let us consider two configurations x and z, where $x \in \mathcal{I}$, $z \in \mathcal{F}$, and $z = \mathcal{T}(x)$. For a CA and a finite time $T \in \mathbb{N}$, if $z = G^T(x)$, then the CA computes $\mathcal{T}$ in time T.

A CA solving spanning tree problem, transforms an initial configuration, representing a connected graph, to a final configuration, which represents the corresponding spanning tree of the graph. Hence, the rule for the CA represents the corresponding (distributed) algorithm for the problem. In this work, we consider that a fixed point is final configuration of any computation. The proposed CA, however, considers only *grid graphs*.

2.2. Grid graph and configurations

Definition 1. A graph $G_k(n) = (V, E)$ is a $k \times n$ $(k, n \in \mathbb{Z})$ grid graph if the vertex and edges are as follows

$$V \subseteq \{V_{i,j} | 1 \le i \le k, 1 \le j \le n\}$$
$$E \subseteq \{(V_{i,j}, V_{a,b}) \,|\, |i-a| + |j-b| = 1\}$$

where, any two vertex $V_{i,j}, V_{a,b} \in V$, there is a path from $V_{i,j}$ to $V_{a,b}$.

[1] *Physarum polycephalum* (*P. polycephalum*), which is a large single cell visible by an unaided eye, has been proven as a reliable living substrate for implementing biological computing devices for computational geometry, graph theoretical problems etc. [29–32].

[2] $\mathcal{N} = \{(0,0),(-1,0),(0,1),(1,0),(0,-1)\}$

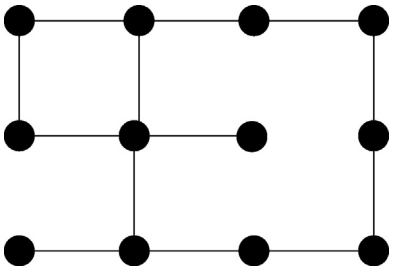

Fig. 1. A grid graph.

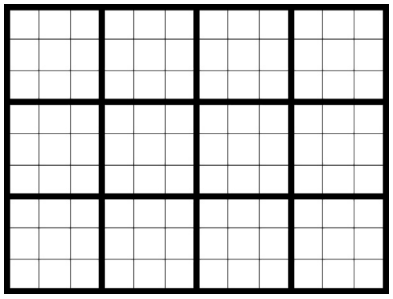

Fig. 2. 2-D grid structure: the blocks are surrounded by thicker lines.

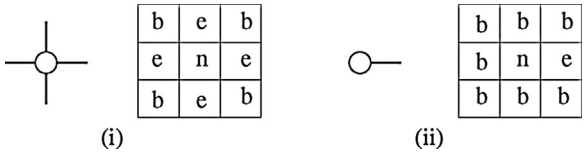

Fig. 3. (i) A node with 4 edges and corresponding grid representation; (ii) a node with 1 edge and corresponding grid representation.

Fig. 1 shows an example gird graph where $k = 4$, $n = 3$. Hence, the arrangement of CA cells $\mathcal{L}$ forms a grid graph, and any other grid graphs are subgraph of $\mathcal{L}$. It is, therefore, natural to emulate any grid graph by a CA grid. In this paper, we consider two-dimensional grid with five neighbourhood dependency to find the spanning tree of grid graphs.

To find out spanning tree of a graph, a CA takes the graph as its initial configuration. So, our first task is to represent a graph as a configuration. Let us divide the CA grid as blocks of 3×3 cells (see Fig. 2). These blocks also form a grid in meta-level. A block is used to represent a node, and the edges connected to it. We name of block as $B_{i,j}$ which represents a vertex $V_{i,j}$ of a grid graph, which $i \in \mathbb{Z}$; $j \in \mathbb{Z}$.

Let us consider 3 states (of cells) to represent a graph – n (node), e (edge) and b (blank). That is, $\Sigma = \{n, e, b\}$. To represent a node, the state of the center cell of a block is to be n. In this work, maximum degree of a node is 4. So, the states of corner cells are set as b, and the rest 4 cells can be in state e. Fig. 3 shows two example cases.

Let us consider the graph of Fig. 1. The configuration corresponding to this graph is shown in Fig. 4. Obviously, a configuration that represents an input graph is in Σ^*

2.3. The computation

Let us consider $G = (V, E)$ be the input grid graph, and $T = (V_T, E_T)$ be a tree where $V_T \subseteq V$ and $E_T \subseteq E$. T is spanning tree of G if $V_T = V$. In the proposed scheme, the CA develops a forest of trees, and the trees grow in the forest by joining the individuals; and finally the forest contains a single tree which is the spanning tree.

Let F be the set of all trees, that is, the forest. Initially, F is set of null trees formed by all the nodes of G. During the evolution of CA,

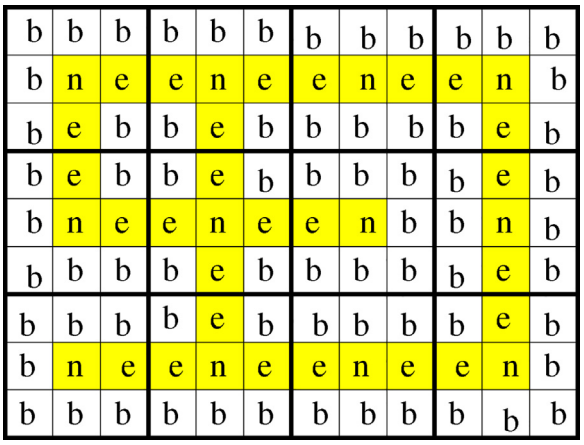

Fig. 4. The configuration corresponding to graph of Fig. 1.

|F| reduces and sizes of trees grow. However, in this computation, there is no central control, and each node computes locally and independently. Therefore, the challenge is to connect the individual trees in such a fashion that, in spite of distributed computation by all, no circuit is formed after connecting two or more trees in F.

For ease of presentation, we divide the computation in two parts: find an appropriate node of each $T \in F$ through which T grows, and connect T with another tree $T' \in F$ to get a larger tree using that node. We name these two parts as

- Finding of bridging node; and
- Growing of trees.

We discuss these two parts in detail in next sections.

To give an overview of the working procedure of our scheme, let us use the graph of Fig. 1. Initially, null trees are formed by all nodes of G, which are marked with "sky-blue" colour, see Fig. 5(a). In Fig. 5(b), each tree (i.e. each individual node) selects an edge following a *fixed rule* to grow. The *fixed rule* which is described in Section 3, determines which one of the available edges is to be used to bridge two trees. The arrows of Fig. 5(b) show the selected edges of trees. Fig. 5(c) shows two trees that are generated after joining the individual trees. Here we get two bigger trees. How the trees are joined and they grow are discussed in Section 3.

The next task is to find a node for each of two trees through which the trees grow. This node selection is a crucial issue, because arbitrary growth of trees may induce a cycle in the graph. In this particular example, both the trees have to choose the same edge through which they can grow, otherwise circuit will be formed. The details of these node selection are given in Section 4. Using that technique, two nodes of two trees of Fig. 5(d) are selected. Fig. 5(e) shows the growing of trees following the *fixed rule*, to be discussed in Section 4. Finally, we get the spanning tree of Fig. 1 (see Fig. 5(f)).

3. Growing of trees

Let us recall the symbols used in this work: $G = (V, E)$ is the input grid graph, $T = (V_T, E_T)$ is a tree where $V_T \subseteq V$ and $E_T \subseteq E$, and $F = \{T \mid T = (V_T, E_T)$ is a tree$\}$ is a forest. Initially F is the set of all nodes of G only. This Section shows the growth of trees of F by joining two or more trees.

To do this, we need to carefully search a node of T through which T is extended. We select such a node in *Finding of bridging node* part (Section 4). Let $V_{i,j} \in V_T$ be the selected node. Now, $V_{i,j}$ may have more than one adjacent node which are of other trees. To avoid circuit formation, only one node, say $V_{a,b} \notin V_T$ is chosen, and to choose $V_{a,b}$ a rule is followed. We use the following rule: select

Table 1
Rule table for 'Growing of trees'. Here, '*' means any valid state.

Rule index	t					$t+1$
	$V_{i,j}$	$V_{i-1,j}$	$V_{i,j+1}$	$V_{i+1,j}$	$V_{i,j-1}$	$V_{i,j}$
1	n	e	*	*	*	n^w
	n	$\neq e$	e	*	*	n^n
	n	$\neq e$	$\neq e$	e	*	n^e
	n	$\neq e$	$\neq e$	$\neq e$	e	n^s
2	e	n^e/e'	*	*	*	e'
	e	*	n^s/e'	*	*	e'
	e	*	*	n^w/e'	*	e'
	e	*	*	*	n^n/e'	e'
3	n^w	e'	*	*	*	n'
	n^n	*	e'	*	*	n'
	n^e	*	*	e'	*	n'
	n^s	*	*	*	e'	n'

an edge $e = (V_{i,j}, V_{a,b}) \in E_T$ where $V_{a,b} \notin V_T$ and $V_{a,b}$ is the first node identified in clockwise direction starting from the left side of $V_{i,j}$. That is,

$$\begin{aligned} \text{If } V_{i-1,j} &\in V \backslash V_T,\ V_{a,b} = V_{i-1,j} \\ \text{Otherwise if } V_{i,j+1} &\in V \backslash V_T,\ V_{a,b} = V_{i,j+1} \\ \text{Otherwise if } V_{i+1,j} &\in V \backslash V_T,\ V_{a,b} = V_{i+1,j} \\ \text{Otherwise } & V_{a,b} = V_{i,j-1} \end{aligned} \tag{1}$$

For more clarity, we present this rule by Fig. 6. However, it is guaranteed that we can always identify such an edge $e \in E_T$ along with a node $V_{i,j} \in V_T$. Note that if we choose $V_{a,b}$ arbitrarily, circuit may be formed. In this rule we have given highest priority to the left node. One may develop other rule like Statement (1), giving priority to right or other nodes, or consider anticlockwise direction.

To develop a CA, this scheme is to be emulated by its rule. Recall that, a block $B_{i,j}$ represents a node $V_{i,j}$. Within $B_{i,j}$ the center cell actually emulates the action of a node whereas the other cells note the connections between $V_{i,j}$ and its neighbouring nodes of the graph. The rule of the proposed CA updates the state of the center cell according to Statement (1) to assign a new state to the cell. The new state keeps the record about the particular edge which is used.

In the next step, the cell say x representing the e also changes its state to record the event. However, according to our construction, there is another cell say y which is adjacent to x but in another block, is also used to represent e. In the next step this cell is also updated.

To implement this concept in our CA, each cell is associated to a register M, which is initialized with the state of $\Sigma = \{n, e, b\}$. The center cell of $B_{i,j}$ moves to a new states following the rule of Statement (1), see Table 1: rule index 1. Let us use 4 more states to keep record which particular edge is used to join – n^w (west edge), n^n (north edge), n^e (east edge) and n^s (south edge). Further, for edge e, the cell x and its adjacent cell y move to another state (say e'), see Table 1: rule index 2. In our scheme, when two cells x and y move to state e', we declare that two trees are joined. When this task is done, the center cell further moves to a fresh state in next time step (see Table 1: rule index 3. Let this state be n'. Note that, initially all the nodes of input graph are individual trees, so after this task, all the center cells settle in state n'. Fig. 7 shows the computation for the initial configuration of Fig. 4 in this part.

It is obvious here that number of trees in F reduces after completion of this part. Following is a related result.

Proposition 1. *Let n be the number of trees in the forest. After single execution of the Growing of Trees, the number of trees reduces to m, where* $1 \leq m \leq \lfloor \frac{n}{2} \rfloor$.

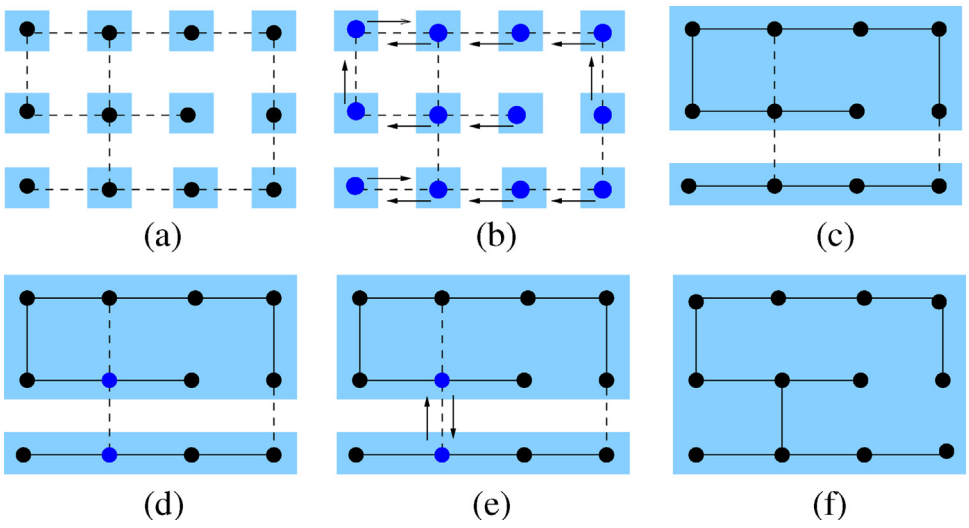

Fig. 5. Spanning tree computation according to proposed algorithm.

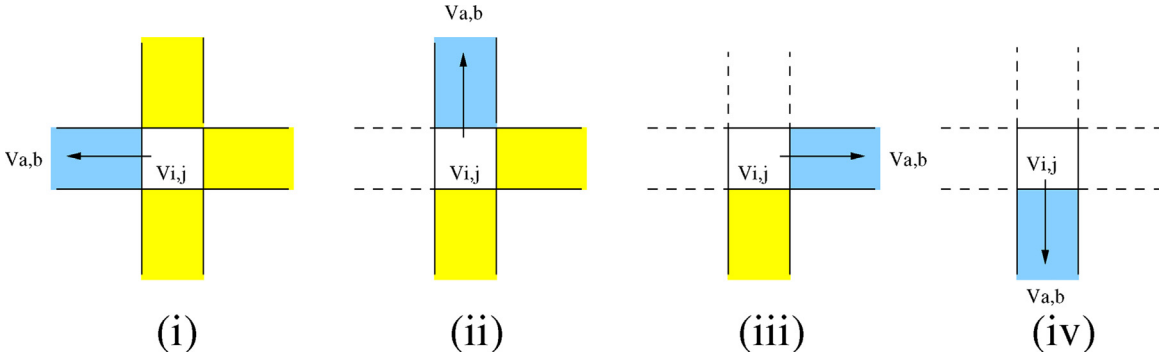

Fig. 6. The graphical view of Statement (1) according to order, where the selected edge $e = (V_{i,j}, V_{a,b}) \in E_T$ is marked with 'sky-blue' colour.

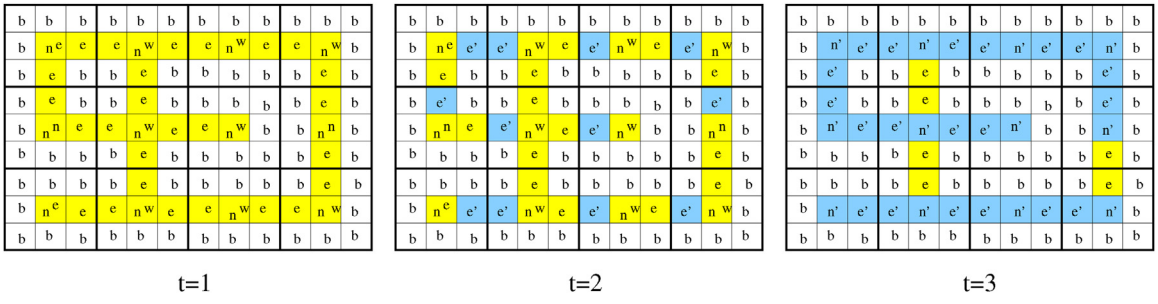

Fig. 7. Computation during 'growing of trees'.

Proof. The best case is, all the trees of the forest are joined in such a fashion that a single tree is formed. In that case, $m = 1$.

However, at least two trees are joined in this execution to get a larger tree. Therefore, in worst case, we observe the number of trees has become half. According to our construction, no individual tree resist itself from growing. So the maximum possible trees after single execution of this part is $\lfloor \frac{n}{2} \rfloor$. Hence the proof. □

4. Finding of bridging node

In the previous section, we have formed larger trees by joining two or more trees. Two trees are joined through a special node; let us call the node as *bridging node*. This section finds a node of T as *bridging node*, through which T grows. As said before, if we choose bridging node $V_{i,j} \in V_T$ arbitrarily, circuit may be formed.

Recall that some of edges of the input graph have already been used to join the trees (see Section 4). Those edges are marked, and in our CA, the cells representing those edges are in a special state (e'). Apart from those edges, one can identify two classes of edges in the graph. We name them *internal* and *external* edges.

Definition 2. An edge $e = (V_{i,j}, V_{a,b}) \in E_T$ of a tree $T = (V_T, E_T)$ is called internal edge if $V_{i,j}$, $V_{a,b} \in V_T$. The edge is external if $V_{i,j} \in V_T$ and $V_{a,b} \notin V_T$.

Fig. 8 shows one internal and two external edges. However, it is quite obvious that, a bridging node $V_{i,j} \in V_T$ should be associated with an external edge through which T can grow. Since the cells of CA perform local computation, $V_{i,j} \in V_T$ can not identify itself

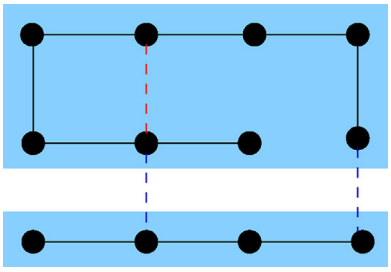

Fig. 8. Internal (red dotted line) and external (blue dotted line) edges.

as bridging node. And even if it somehow identifies, the $V_{i,j}$ can not distinguish external edges from a collection of internal and external edges. Therefore, to avoid circuit formulation, the internal edges should be deleted first. For this purpose, we divide the computation in two parts:

- Identify and delete internal edges; and
- Finding of bridging node associated with external edge.

4.1. Identify and delete internal edges

In this part, the task is to identify and delete the internal edges $e = (V_{i,j}, V_{a,b})$ where $V_{i,j}, V_{a,b} \in V_T$ of tree $T = (V_T, E_T)$. The graph G is a grid graph; so $|i - a| = 1$ or $|j - b| = 1$. As $V_{i,j}$ is emulated by block $B_{i,j}$, the block $B_{a,b}$ is one of its four nearest neighbours.

However, the block $B_{i,j}$ can easily decide whether there is an internal/external edge. Recall that the cells, representing the edges, are in a specific state (state e') when they are used for join. Note that there may be some nodes of T which are not associated to any internal/external edges. The challenge of this phase, therefore, is to decide whether $V_{a,b}$ is in V_T. Since the cells of a CA perform only local computation, we need to adopt some strategy which would enable the CA to collectively decide the same. Let us first describe the strategy in meta level.

Imagine that each edge $e \in E_T$ is a bi-directional channel and each node $V_{i,j} \in V_T$ follows a law to receive and send messages through the adjacent channels. As a result, we can get a unidirectional ring covering all the nodes and edges of T. Fig. 9 clarifies this idea. Observe that if $V_{i,j} \in V_T$ has degree 1, then it appears once, whereas if $V_{i,j}$ has degree 2, it appears twice in the ring.

Now, the node $V_{i,j} \in V_T$ that has external/internal edge *sends* a "token" to its next node in the ring. If there are k number of nodes like $V_{i,j}$, all the k nodes *send* token simultaneously. Note that, if $e = (V_{i,j}, V_{a,b}) \in E_T$, then both $V_{i,j}$ and $V_{a,b}$ send the tokens. In Fig. 9, the number of k is 3, i.e. node b, e and h send token simultaneously. Each token is composed of two counters – C_H and C_V which are initialized to 0. When the token moves in horizontal (resp. vertical) direction, C_H (resp. C_V) is incremented or decremented. If the next node of $V_{i,j}$ in the ring is at left (resp. right), then 1 is subtracted from (resp. added to) C_H. Similarly if the next node is above (resp. below) the $V_{i,j}$, then 1 is added to (resp. subtracted from) C_V. It can be easily shown that $V_{i,j}$ receives a token with $C_H = 0$ and $C_V = 0$ if and only if the token has previously sent by itself.

Computation in this part ends when $V_{i,j}$ receives a token with $C_H = C_V = 0$. Since all the nodes having internal/external edges send token simultaneously, all of them get back their tokens with $C_H = C_V = 0$ simultaneously. Before the end of computation, however, when a node $V_{a,b} \in V_T$ receives the token, it forwards it to the next node. Following additional action is taken by $V_{a,b}$ when it has internal/external edge and $|C_H| = 1$ or $|C_V| = 1$.

- If $V_{a,b}$, which has internal/external edge at left (resp. right), receives a token with $C_H = 1$ (resp. -1), then $V_{a,b}$ can identify and delete the left (resp. right) edge as internal edge (i.e. $V_{i,j}, V_{a,b} \in V_T$).
- If $V_{a,b}$, which has internal/external edge at below (resp. above), receives a token with $C_V = 1$ (resp. -1), then $V_{a,b}$ can identify and delete the below (resp. above) edge as internal edge (i.e. $V_{i,j}$, $V_{a,b} \in V_T$).

As an example, in Fig. 9 node e, which has internal/external edge at below, receives a token with $C_V = 1$. Therefore, node e can identify that the downward node (here, node b) is actually in the same tree. Hereafter, node e can delete the downward edge, because it is an

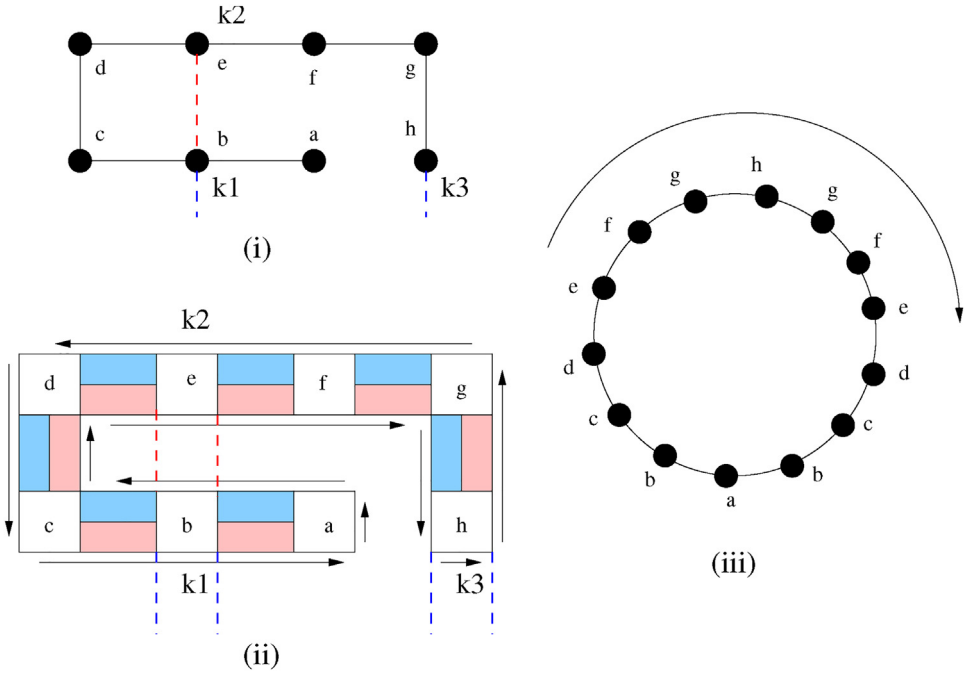

Fig. 9. (i) A tree; (ii) an edge is a bi-directional channel; (iii) the ring representation.

internal edge. Now the challenge is to develop a CA to implement the meta level strategy.

Here, a reader may worry about deadlock as a good number of 'tokens' may parallely participate in this process. However, the proposed CA implements the concept of 'tokens' by changing the local states only. Therefore, there is no chance of deadlock.

4.1.1. Implementation detail

Recall that, according to our consideration, each edge $e \in E_T$ is a bi-directional channel and each node $V_{i,j} \in V_T$ receives and sends messages through the adjacent channels. To implement our imagination, we need some registers, and states, which are described below:

R_w, R_n, R_e, R_s : Each center cell of $B_{i,j}$ is associated with four registers – R_w, R_n, R_e and R_s, which are responsible to send messages through the west, north, east and south channels respectively (see Fig. 10(a)). These registers are initialized with null state, i.e. '$\emptyset$'.
R_1, R_2 : The rest of cells of block $B_{i,j}$ are associated with two registers R_1 and R_2 for dealing with bi-directional channel. Fig. 10(b) depicts the idea. Here, registers R_1 and R_2 are initialized with state '$\longleftarrow$' (left), '$\rightarrow$' (right), '$\uparrow$' (up), '$\downarrow$' (down) indicating the direction of channel (see Fig. 10(c)). It can be easily initialized with a simple set of rules at the time of joining an edge in 'growing of trees' part.

Let us name these registers as set R; that is, $R = \{R_w, R_n, R_e, R_s, R_1, R_2\}$. Now, to develop a CA, a tree $T=(V_T, E_T)$ should be represented as a ring covering all the nodes and edges of T through which the 'token' will move. Therefore, a rule indicating the direction of 'token' movement should be initialized first. In this context, we divide the computation in two subcomponents:

(i) Rule for path tracing of tree T as a ring; and
(ii) 'Token' implementation.

(i) Rule for path tracing: Once the edges are viewed as a bi-directional channel with direction state (i.e. $\longleftarrow$, $\rightarrow$, $\uparrow$, $\downarrow$), the path direction for token movement can be easily traced locally with a simple set of rules.

Note that, traditionally in *Graph theory*, nodes are viewed as active computation elements, whereas edges are viewed as passive elements. However, in the proposed CA, both nodes and edges are represented by *cell*. So both of them are active computational elements. Therefore, the journey of a 'token' starting from node $V_{i,j} \in V_T$ to its next node $V_{a,b} \in V_T$ in the ring is visualized as following three actions. (a) The cell representing node $V_{i,j} \in V_T$ sends the 'token' to its next cell x representing edge in the ring. (b) According to our construction, an edge is represented by two cells x and y of two adjacent block $B_{i,j}$ and $B_{a,b}$. Therefore, a cell x representing edge sends the 'token' to its next cell y representing edge in the ring. (c) Then, the cell y representing edge sends the 'token' to its next cell representing node $V_{a,b} \in V_T$ in the ring.

Fig. 12 clarifies the above rule for path tracing of the proposed CA for the following situations:

Situation (i): According to our construction, a cell representing node $V_{i,j} \in V_T$ that has external/internal edge(s) generate(s) the 'token'. The 'token' is sent through the cell which represent an edge. However, which edge, hence cell, is to be selected is determined by a rule (which is similar to Statement (1), page 8) – select left edge, if it exists; otherwise search in clockwise direction to get the edge. Fig. 12(a) clarifies the rule. Recall that, each cell representing edge is associated with two registers R_1 and R_2 for indicating direction. The 'token' is passed through one of these two registers according to state of those registers. The 'token' is marked with 'black' colour at the starting point. The rule of Fig. 12(a) is summarized below. Here, V_{i,j_R} indicates the R register of $V_{i,j}$; $V_{i,j}[M]$ indicates the state of M register for $V_{i,j}$; and 'Dir' indicates the next direction for path tracing

```
if V_{i,j}[M] = n^i then
    if V_{i-1,j}[M] = e' then
        Dir[V_{i,j R_w}] = V_{i-1,j R_1};
    end
    else if V_{i-1,j}[M] ≠ e' & V_{i,j+1}[M] = e' then
        Dir[V_{i,j R_n}] = V_{i,j+1 R_2};
    end
    else if V_{i-1,j}[M] ≠ e' & V_{i,j+1}[M] ≠ e' & V_{i+1,j}[M] = e' then
        Dir[V_{i,j R_e}] = V_{i+1,j R_2};
    end
    else
        Dir[V_{i,j R_s}] = V_{i,j-1 R_1};
    end
else
    Dir[V_{i,j}] = Dir[V_{i,j}];
end
```

Situation (ii): Recall that, an edge is represented by two cells of two adjacent blocks. Therefore, a cell x representing edge $e \in E_T$ forward the 'token' through its adjacent cell y representing the same edge $e \in E_T$ depending on the state of the corresponding R registers. Fig. 12(b) depicts the rule. 'Token' with 'black' (resp. 'blue') colour is marked for t (resp. $t+1$) time step. Let us summarize this rule as following. Here, 'Coming' indicates the previous position of the token.

```
if V_{i,j}[M] = e' then
    if Coming[V_{i,j}[M]] = n'/n^i then
        if V_{i-1,j}[M] = e' then
            Dir[V_{i,j R_1}] = V_{i-1,j R_1};
        end
        else if V_{i,j+1}[M] = e' then
            Dir[V_{i,j R_2}] = V_{i,j+1 R_2};
        end
        else if V_{i+1,j}[M] = e' then
            Dir[V_{i,j R_2}] = V_{i+1,j R_2};
        end
        else
            Dir[V_{i,j R_1}] = V_{i,j-1 R_1};
        end
    else
        if V_{i-1,j}[M] = n'/n^i then
            Dir[V_{i,j R_1}] = V_{i-1,j};
        end
        else if V_{i,j+1}[M] = n'/n^i then
            Dir[V_{i,j R_2}] = V_{i,j+1};
        end
        else if V_{i+1,j}[M] = n'/n^i then
            Dir[V_{i,j R_2}] = V_{i+1,j};
        end
        else
            Dir[V_{i,j R_1}] = V_{i,j-1};
        end
    end
else
    Dir[V_{i,j}] = Dir[V_{i,j}];
end
```

Situation (iii): A cell representing node $V_{i,j} \in V_T$ that has 'token' coming from a cell representing edge $e_1 \in E_T$, sends the 'token' through its adjacent cell representing edge $e_2 \in E_T$ according to the direction of Fig. 12(*c-f*). The registers of R in a cell representing

node $V_{i,j} \in V_T$ keeps the record to implement it. In a cell representing node $V_{i,j} \in V_T$, the 'token' moves through one of the R registers depending on the destination cell representing edge $e_2 \in E_T$ of the 'token'. In Fig. 12(c) (resp. d, e, f), the 'token' is coming from right (resp. above, left, below) side cell representing edge $e_1 \in E_T$. The 'token' with 'black' (resp. 'blue', 'red') colour is marked for t (resp. $t+1$, $t+2$) time step. Following are the summary of the rule of Fig. 12(c). Here, 'Pos' indicates the R register position of the token.

```
if V_{i,j}[M] = n'/n'' then
    if Coming[V_{i,j}] = V_{i+1,j} then
        if V_{i,j+1}[M] = e' then
            Pos[V_{i,j}] = R_n, Dir[V_{i,j}R_n] = V_{i,j+1}R_2;
        else if V_{i,j+1}[M] ≠ e' & V_{i-1,j}[M] = e' then
            Pos[V_{i,j}] = R_w, Dir[V_{i,j}R_w] = V_{i-1,j}R_1;
        else if V_{i,j+1}[M] ≠ e' & V_{i-1,j}[M] ≠ e' & V_{i,j-1}[M] = e' then
            Pos[V_{i,j}] = R_s, Dir[V_{i,j}R_w] = V_{i,j-1}R_1;
        else
            Pos[V_{i,j}] = R_e, Dir[V_{i,j}R_e] = V_{i+1,j}R_2;
    else if Coming[V_{i,j}] = V_{i,j+1} then
        if V_{i-1,j}[M] = e' then
            Pos[V_{i,j}] = R_w, Dir[V_{i,j}R_w] = V_{i-1,j}R_1;
        else if V_{i-1,j}[M] ≠ e' & V_{i,j-1}[M] = e' then
            Pos[V_{i,j}] = R_s, Dir[V_{i,j}R_s] = V_{i,j-1}R_1;
        else if V_{i-1,j}[M] ≠ e' & V_{i,j-1}[M] ≠ e' & V_{i+1,j}[M] = e' then
            Pos[V_{i,j}] = R_e, Dir[V_{i,j}R_e] = V_{i+1,j}R_2;
        else
            Pos[V_{i,j}] = R_n, Dir[V_{i,j}R_n] = V_{i,j+1}R_2;
    else if Coming[V_{i,j}] = V_{i-1,j} then
        if V_{i,j-1}[M] = e' then
            Pos[V_{i,j}] = R_s, Dir[V_{i,j}R_s] = V_{i,j-1}R_1;
        else if V_{i,j-1}[M] ≠ e' & V_{i+1,j}[M] = e' then
            Pos[V_{i,j}] = R_e, Dir[V_{i,j}R_e] = V_{i+1,j}R_2;
        else if V_{i,j-1}[M] ≠ e' & V_{i+1,j}[M] ≠ e' & V_{i,j+1}[M] = e' then
            Pos[V_{i,j}] = R_n, Dir[V_{i,j}R_n] = V_{i,j+1}R_2;
        else
            Pos[V_{i,j}] = R_w, Dir[V_{i,j}R_w] = V_{i-1,j}R_1;
    else
        if V_{i+1,j}[M] = e' then
            Pos[V_{i,j}] = R_e, Dir[V_{i,j}R_e] = V_{i+1,j}R_2;
        else if V_{i+1,j}[M] ≠ e' & V_{i,j+1}[M] = e' then
            Pos[V_{i,j}] = R_n, Dir[V_{i,j}R_n] = V_{i,j+1}R_2;
        else if V_{i+1,j}[M] ≠ e' & V_{i,j+1}[M] ≠ e' & V_{i-1,j}[M] = e' then
            Pos[V_{i,j}] = R_w, Dir[V_{i,j}R_w] = V_{i,j-1}R_1;
        else
            Pos[V_{i,j}] = R_s, Dir[V_{i,j}R_s] = V_{i,j-1}R_1;
else
    Dir[V_{i,j}] = Dir[V_{i,j}];
```

Therefore, we can get a ring covering all the nodes and edges of T, which can be viewed as a one-dimensional arrangement of cells. According to our construction, the 'token' moves through the registers of R actually, that is, the ring is constructed with R registers. To illustrate the proposed CA, Table 2 shows the one-dimensional ring arrangement for a tree of Fig. 7. Note that, in Table 2, the cell numbers are used for ease of reference. Fig. 11 depicts the two-dimensional abstract arrangement of Table 2. Here, a node $V_{i,j} \in V_T$ that has external/internal edge is marked with a special state (say state n^i). Hereafter, it is essentially a "one-dimensional algorithm".

(ii) 'Token' implementation: According to meta level description, a "token" with two counters C_H and C_V is used to identify internal edge. A straightforward implementation of this strategy can be possible with a sufficiently large number of state. Though, that will not be affordable. Therefore, the challenge is to propose a CA[3] with minimum (obviously, finite) number of states.

In the proposed CA, the 'token' counters are implemented by states of R registers, i.e. $\longleftarrow$, $\rightarrow$, $\uparrow$, $\downarrow$ and $\emptyset$. According to the construction, a node $V_{a,b} \in V_T$, which has internal/external edge, may identify internal edge when it receives a token with $|C_H|$=1 or $|C_V|$=1. Here, the counter C_H is implemented by counting number of '$\longleftarrow$' and '$\rightarrow$', whereas C_V is implemented by counting number of '$\uparrow$' and '$\downarrow$'. In detail, C_H = 1 (resp. -1) when number of '$\rightarrow$' (resp. '$\longleftarrow$') state is one more than the number of '$\longleftarrow$' (resp. '$\rightarrow$') state; analogously C_V is also implemented. However, $C_H = C_V = 0$ is implemented by state '$\emptyset$'. Now, to implement C_H and C_V, following two registers are introduced.

S : *Static* register S is associated with every register (of R) of a cell, which is initialized with same state according to the corresponding R register for R_1 and R_2, and being empty for rest of the R registers.
W : *Working* register W is also associated with every register (of R) of a cell, which is initialized with being empty.

In the proposed CA, the 'token' moves through the working register. Whereas, the role of static register is to build the 'token'. Here, we mark starting of a 'token' with symbol 's' along with the state of 'token', i.e. $\overset{s}{\longleftarrow}$, $\overset{s}{\rightarrow}$, $\uparrow_s$, $\downarrow_s$, $\emptyset_s$. Whereas, we can identify the ending of a 'token' with symbol 'e' along with the state of 'token', i.e. $\underset{e}{\longleftarrow}$, $\underset{e}{\rightarrow}$, $e\uparrow$, $e\downarrow$, $e\emptyset$. Therefore, both of starting and ending of a 'token' is marked with both, i.e. $\overset{s}{\underset{e}{\longleftarrow}}$, $\overset{s}{\underset{e}{\rightarrow}}$, $e\uparrow s$, $e\downarrow s$, $e\emptyset s$.

Recall that, the ring covering all the nodes and edges of T can be viewed as a one-dimensional arrangement. Therefore, the 'token' implementation is described in one-dimensional fashion. Here, the cell representing node $V_{i,j} \in V_T$ which has external/internal edge are marked with a special state (state n^i) to initiate the token. Therefore, the 'token' is initiated through the working register according to *path tracing rule*. Note that, more than one 'token' may be simultaneously initiated. Initially, the 'token' at the working register is associated with corresponding state of static register with starting and ending symbol. The proposed rule for the journey of a 'token' can be summarized as following situations:

Situation 1: The states in the working register (i.e. 'token') are shifted to the right one cell per step. Therefore, at each time step, one *new* cell is encountered by the 'token'. In this procedure, the token is created by copying the content of the *new* static register (cell) at the end of the token, whereas the static register becomes empty. However, more than one 'token' may be simultaneously initiated. Therefore, the empty static register will move to its previous state at the immediate time step to create other 'tokens'.

Situation 2: Here, state $\longleftarrow$ (resp. $\uparrow$) is the opposite state of $\rightarrow$ (resp. $\downarrow$) and vice-versa. In this journey, whenever two opposite states at static and working register of a cell are encountered, the state is deleted (i.e. moves to state $\emptyset$) from the 'token'. Hence, at any point of this journey, only one kind of state among states $\longleftarrow$ and $\rightarrow$ will be left, which has the majority. Those in the minority will all be deleted. The states $\uparrow$ and $\downarrow$ are also processed analogously. During this deletion process, the state of static register is marked with delete symbol 'd' (i.e. $\overset{d}{\longleftarrow}$, $\overset{d}{\rightarrow}$, $\uparrow_d$, $\downarrow_d$), which will move to its previous sate at the *end* of the 'token' to create other 'tokens'. Note that, a static register state with delete symbol neither participate

[3] The proposed strategy is similar with the work by [4], which elects leader on the boundary of a 2-D pattern. However, here, the purpose and implementation both are different.

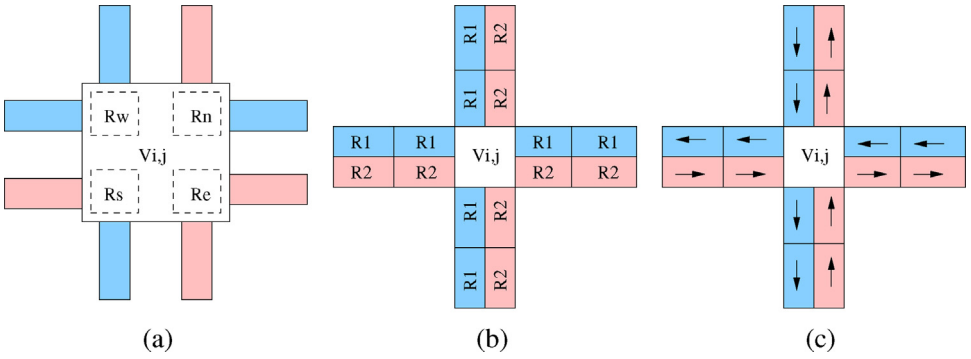

Fig. 10. (a) Registers associated with center cell of $B_{i,j}$; (b) Registers associated with rest of cells of $B_{i,j}$; (c) States associated with rest of cells.

Table 2
One dimensional arrangement corresponding to a tree T of Fig. 7. Line 1 (resp. 2, 3, 4) is for cell number (state of register M, corresponding register R following the path, state of corresponding register R).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
n^i	e'	e'	n'	e'	e'	n'	e'	e'	n^i	e'	e'	n'	e'	e'	n'	e'	e'	n^i	e'	e'
R_n	R_2	R_2	R_w	R_1	R_1	R_w	R_1	R_1	R_w	R_1	R_1	R_s	R_1	R_1	R_e	R_2	R_2	R_e	R_2	R_2
$\emptyset$	↑	↑	$\emptyset$	⟵	⟵	$\emptyset$	⟵	⟵	$\emptyset$	⟵	⟵	$\emptyset$	↓	↓	$\emptyset$	→	→	$\emptyset$	→	→

21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
n'	e'	e'	n^i	e'	e'	n'	e'	e'	n'	e'	e'	n^i	e'	e'	n'	e'	e'	n^i	e'	e'
R_w	R_1	R_1	R_w	R_1	R_1	R_n	R_2	R_2	R_e	R_2	R_2	R_e	R_2	R_2	R_e	R_2	R_2	R_s	R_1	R_1
$\emptyset$	⟵	⟵	$\emptyset$	⟵	⟵	$\emptyset$	↑	↑	$\emptyset$	→	→	$\emptyset$	→	→	$\emptyset$	→	→	$\emptyset$	↓	↓

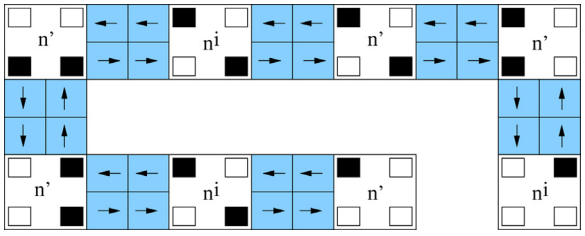

Fig. 11. The two-dimensional abstract arrangement of Table 2. Here, state '$\emptyset$' is marked by 'black' box and the edges (state e') are marked by 'blue'.

to encounter opposite state nor participate to create the 'token' at the end.

Situation 3: Recall that, some states of a 'token' are in state $\emptyset$ due to opposite state encounter. However, these states are not useful within a 'token'. Therefore, the state of working register are not always shifted towards right to avoid unnecessary state $\emptyset$ within a 'token'.

Situation 4: Now, if a cell representing $V_{a,b} \in V_T$, which has external/internal edge (i.e. marked by special state n^i), receives a 'token' with only state ⟵ or → or ↑ or ↓, then the cell can identify the presence of internal edge in the following way:

- The cell in state n^i moves to state n^i_r (resp. n^i_l) when it receives a 'token' with only state ⟵ (resp. →), i.e. $C_H = -1$ (resp. $C_H = 1$). Here, state n^i_r (resp. n^i_l) indicates that internal edge can exist at right (resp. left) side.
- The cell in state n^i moves to state n^i_d (resp. n^i_u) when it receives a 'token' with only state ↑ (resp. ↓), i.e. $C_V = 1$ (resp. $C_V = -1$). Here, state n^i_d (resp. n^i_u) indicates that internal edge can exist at down (resp. up) side.

Therefore, the internal edges are deleted using states n^i_r, n^i_l, n^i_d and n^i_u at the next time step. In the proposed CA, a cell representing internal edge moves to a special state (state b). Note that, a cell representing $V_{a,b} \in V_T$ may have more than one internal edges. Hence, these states moves to previous state n^i to delete other internal edges.

Situation 5: At the end of the computation, the 'token' will stop after visiting the whole path which is identified by state '$\emptyset$' with both the starting and ending symbol (i.e. $C_H = C_V = 0$).

Table 3 shows the rule indices for the journey of a 'token'. Here, x and y indicate any valid states (i.e. ⟵, →, ↑, ↓). In Table 3, x^c indicates the opposite state of x [i.e. state ⟵ (resp. ↑) is the opposite state of → (resp. ↓) and vice-versa]. Note that, the rule indices in Table 3 are written in a combined fashion, where two neighbouring cells act together, to make it more readable.

To sum up, the essential components of 'identify and delete internal edges' step are noted in Algorithm 1.

Algorithm 1. Identify and delete internal edges

Step 1: **If** $V_{i,j}[M]{=}n'$ and $V_{i,j}$ is associated with an edge in state e
then $V_{i,j}[M]{=}n^i$;
else $V_{i,j}[M]{=}n'$;

Step 2: **If** $V_{i,j}[M]{=}n^i$
then $V_{i,j}$ initiates the 'token' by using the path tracing rule of situation (i).

Step 3: Repeat Step 4 and 5 until the terminating condition as stated in Situation 5 is satisfied.

Step 4: Use Table 3 following the path tracing rule of situation (ii) and (iii) to update the state of $V_{i,j}$.

Step 5:

if $V_{i,j}[M] = n^i_r$ **then**
  $V_{i+1,j}[M] = b$, $V_{i,j}[M] = n^i$;
else if $V_{i,j}[M] = n^i_l$ **then**
  $V_{i-1,j}[M] = b$, $V_{i,j}[M] = n^i$;
else if $V_{i,j}[M] = n^i_d$ **then**
  $V_{i,j-1}[M] = b$, $V_{i,j}[M] = n^i$;
else if $V_{i,j}[M] = n^i_u$ **then**
  $V_{i,j+1}[M] = b$, $V_{i,j}[M] = n^i$;
else
  $V_{i,j}[M] = V_{i,j}[M]$;

Step 6: **If** $V_{i,j}[M]{=}n^i$ and $V_{i,j}$ is associated with an edge in state e
then $V_{i,j}[M]{=}n^f$;
else $V_{i,j}[M]{=}n'$;

Let us take one-dimensional arrangement of Table 2 to illustrate the 'token' implementation (see Tables 4 and 5). In Table 2, two consecutive cells in state e' will always be having the same register states as they are treated as a single edge. Therefore, we consider (in Table 4) one out the two consecutive cells to make the representation shorter. Again, the aim is to delete internal edges which are performed at cell in state n^i. Therefore, we also omit the presence of the cells in state n' due to space limitations. In Tables 4 and 5, the first configuration consists of five lines [line 1 (resp. 2, 3, 4, 5) is for M register state (resp. corresponding R register, R register state, S register state, W register state)]. The rest of the configurations consist only of S register state and W register state. Since, three registers of R in cell with state n^i are responsible for generating 'token', their presence are marked by double vertical line. As an example of internal edge deletion, working register (i.e. 'token') of 8th cell, which is in state n^i, is in state '$_e\downarrow_s$' at sixth time step. Therefore, 8*th* cell moves to state n^i_u to delete internal edge at the next time step.

4.2. Finding of bridging node associated with external edge

Recall that, a *bridging node* $V_{i,j} \in V_T$ should be associated with an external edge through which $T = (V_T, E_T)$ can grow. Now, the block $B_{i,j}$ can easily decide whether there is an external edge after deletion of internal edges (state n^f). The cells that represent the internal edges, are in a specific state (state b) after internal edge deletion. Therefore, the cells that represent the external edge, are in a specific state (state e). Note that there may be more than one node of T which are associated with external edges. Now, we have to choose one of those nodes as bridging node.

Table 3
Rule table for 'identify and delete internal edges' step. Here, '*' means any valid state and 'u' means 'no change'.

Time	Reg	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$
t	M	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	R	y	*	y	*	x^c	*	y	*	y	*	x	*	y	*	x^c	*	y	*	x^c	*	y	*	y^c	*
	S		*	y	*	x^c	*	y_d	*	y	*		*	y	*	x^c	*	y	*	x^c	*	y_d	*	y^c	*
	W	$_ex_s$	*	$_ex_s$	*	$_ex_s$	*	$_ex_s$	*	$_e\emptyset_s$	*	*	*	x	*	x	*	$_ex$	*	$_ex$	*	$_ex$	*	$_ey$	x_s
$t+1$	M	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u
	R	y	u	y	u	x^c	u	y	u	y	u	x	u	y	u	x^c	u	y	u	x^c	*	y	u	y^c	u
	S	y	u		u	x^c_d	u		u		u	x	*	y	*	x^c_d	*		*		*		*		u
	W		$_ex_s$	$_ey$	x_s	$_e\emptyset_s$	u	$_ex_s$	u	$_ey_s$	u	*	*	*	x	*	$\emptyset$	$_ey$	x	$_e\emptyset$	$\emptyset$	$_e\emptyset$	x	$_e\emptyset$	x_s

Time	Reg	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$	i	$i+1$
t	M	*	*	*	*	*	*	*	*	*	n^i	*	n^i	*	n^i	*	n^i	*	n^i	*	n^i	*	n^i	*	n^i
	R	y	*	*	*	y	*	*	x^c	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	S	y_d	*	*	*	y	*	*	x^c	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	W	$_e\emptyset$	x_s	$\emptyset$	x_s	x_s	*	y	x_s	*	$_e\uparrow_s$	*	$_e\downarrow_s$	*	$_e\rightarrow_s$	*	$_e\longleftarrow_s$	$_e\emptyset$	$\uparrow_s$	$_e\emptyset$	$\downarrow_s$	$_e\emptyset$	$\rightarrow_s$	$_e\emptyset$	$\longleftarrow_s$
$t+1$	M	u	u	u	u	u	u	u	u	u	n^i_d	u	n^i_u	u	n^i_l	u	n^i_r	u	n^i_d	u	n^i_u	u	n^i_l	u	n^i_r
	R	y	u	u	u	y	u	u	x^c	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u
	S		u	u	u	y	u	u	x^c_d	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u
	W	$_e\emptyset$	x_s	*	x_s	*	x_s	*	y_s	*	*	*	*	*	*	*	*	*	$_e\emptyset$	*	$_e\emptyset$	*	$_e\emptyset$	*	$_e\emptyset$

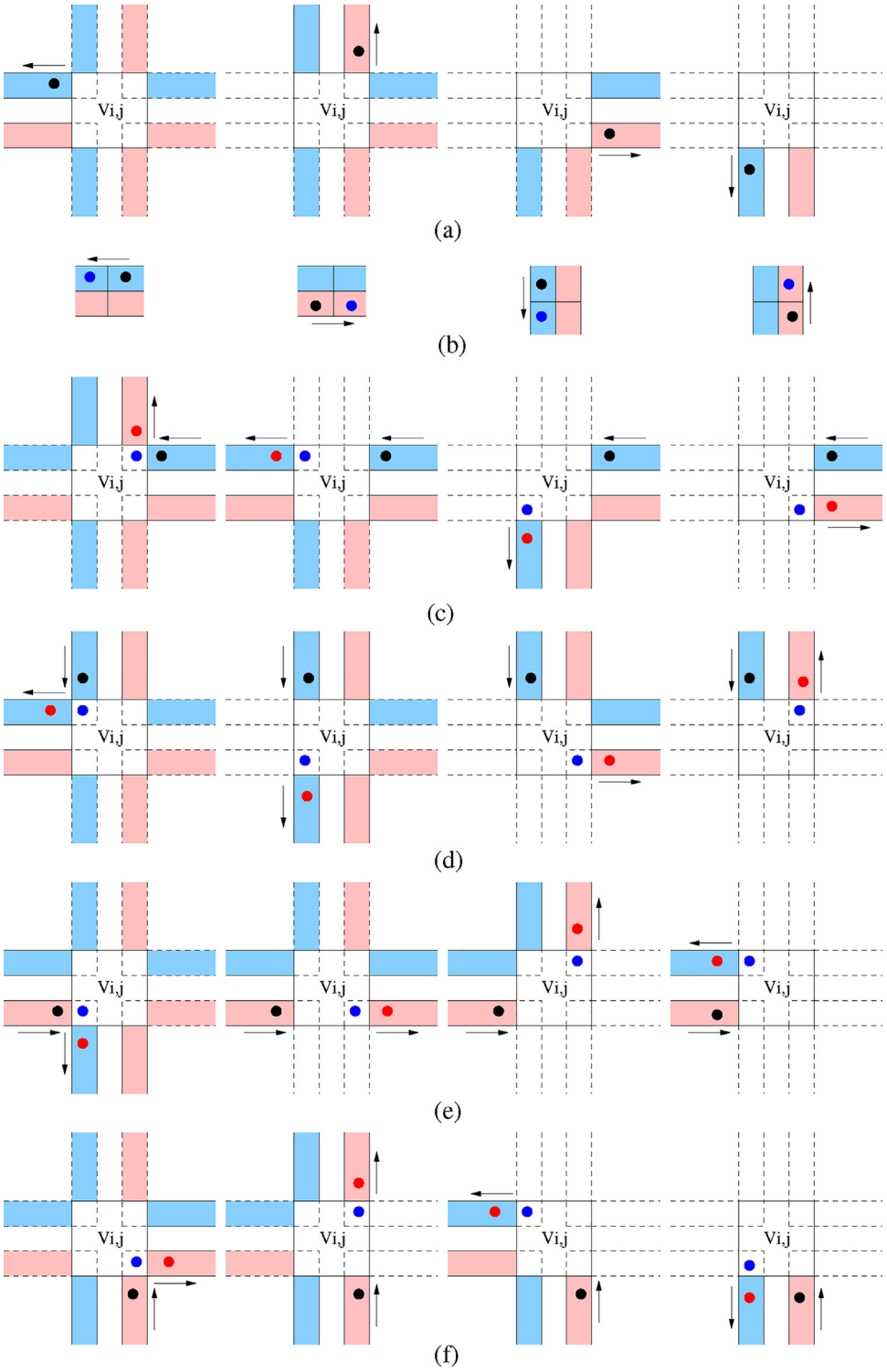

Fig. 12. Rule for path direction tracing. Here, the channel formed by register R_1 (resp. R_2) is marked by 'sky blue' (resp. 'pink') colour. Now, a coloured channel with dotted (resp. straight) boundary line says about *don't care* (resp. *necessary*) channel situation. An absence of channel is marked by 'white' colour with dotted boundary line.

Definition 3. We define *bridging node* of a tree as the uppermost node amongst its leftmost nodes which is associated with external edge.

It is quite obvious that a node $V_{i,j} \in V_T$ that has external edge can not identify itself as bridging node under local computation environment. We follow here the same strategy which we have just followed to identify and delete internal edges. Here, the node $V_{i,j} \in V_T$ that has external edge sends a 'token' following the ring covering all the nodes and edges of T. Similarly, all the nodes having external edge send 'token' simultaneously, i.e. if two nodes $V_{i,j} \in V_T$ and $V_{a,b} \in V_T$ both are associated with external edges, then both $V_{i,j}$ and $V_{a,b}$ send the 'token'. The 'token' counters (C_H and C_V) operate following the old fashion. Here, a node $V_{a,b} \in V_T$ which is not associated with any external edge, receives a token, it forwards it to the next node. Following additional action taken by $V_{a,b} \in V_T$ when it has external edge.

Table 4
'Token' implementation: step 1–20.

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	n^i	e'	e'	e'	n^i	e'	e'	e'	n^{i*}	e'	e'	n^i	e'	e'	e'	n^{i*}	e'	e'	e'
	R_n	R_2	R_1	R_1	R_w	R_1	R_1	R_2	R_e	R_2	R_1	R_w	R_1	R_2	R_2	R_e	R_2	R_2	R_1
	$\emptyset$	$\uparrow$	$\leftarrow$	$\leftarrow$	$\emptyset$	$\leftarrow$	$\downarrow$	$\rightarrow$	$\emptyset$	$\rightarrow$	$\leftarrow$	$\emptyset$	$\leftarrow$	$\uparrow$	$\rightarrow$	$\emptyset$	$\rightarrow$	$\rightarrow$	$\downarrow$
		$\leftarrow$	$\leftarrow$				$\downarrow$	$\rightarrow$		$\rightarrow$	$\leftarrow$			$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
		$e\uparrow s$				$\overset{s}{\underset{e}{\leftarrow}}$							$\overset{s}{\underset{e}{\leftarrow}}$						
2		$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
			$e\uparrow s$				$\overset{s}{\underset{e}{\leftarrow}}$							$\overset{s}{\underset{e}{\leftarrow}}$					
3		$\uparrow$		$\leftarrow$		$\leftarrow$		$\rightarrow$		$\rightarrow$	$\leftarrow$		$\leftarrow$		$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
			$\overset{}{\underset{e}{\leftarrow}}$	$\uparrow s$			$e\downarrow$	$\overset{s}{\leftarrow}$						$e\uparrow$	$\overset{s}{\leftarrow}$				
4		$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\overset{d}{\rightarrow}$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\overset{d}{\rightarrow}$		$\rightarrow$	$\rightarrow$	$\downarrow$
				$\overset{}{\underset{e}{\leftarrow}}$	$\uparrow s$			$e\downarrow s$							$e\uparrow s$				
5		$\uparrow$	$\leftarrow$			$\leftarrow$	$\downarrow$			$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$			$\rightarrow$	$\rightarrow$	$\downarrow$
				$\overset{}{\underset{e}{\leftarrow}}$	$\leftarrow$	$\uparrow s$		$e\downarrow s$							$e\uparrow s$				
6		$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
					$\overset{}{\underset{e}{\leftarrow}}$	$\leftarrow$	$\uparrow s$		$e\downarrow s$							$e\uparrow s$			
7	n^i	e'	e'	e'	$n^i{}_d*$	e'	e'	e'	$n^i{}_u$	e'	e'	$n^i{}_u*$	e'	e'	e'	$n^i{}_d$	e'	e'	e'
		$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\downarrow_d$	$\rightarrow$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
						$\overset{}{\underset{e}{\leftarrow}}$	$\overset{s}{\leftarrow}$			$e\downarrow s$							$e\uparrow s$		
8		$\uparrow$	$\leftarrow$	$\leftarrow$			$\downarrow_d$	$\rightarrow$			$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$			$\rightarrow$	$\downarrow$
						$\overset{}{\underset{e}{\leftarrow}}$	$\leftarrow$	$\overset{s}{\leftarrow}$		$\overset{}{\underset{e}{\rightarrow}}$	$\downarrow_s$						$\overset{}{\underset{e}{\rightarrow}}$	$\uparrow_s$	
9		$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\downarrow_d$	$\overset{d}{\rightarrow}$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
							$\overset{}{\underset{e}{\leftarrow}}$	$\overset{s}{\leftarrow}$			$\overset{}{\underset{e}{\rightarrow}}$	$\downarrow_s$						$\overset{}{\underset{e}{\rightarrow}}$	$\uparrow_s$
10		$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$		$\overset{d}{\rightarrow}$		$\rightarrow$			$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$		$\downarrow_d$
							$_e\emptyset$	$\leftarrow$	$\overset{s}{\leftarrow}$		$_e\emptyset$	$\downarrow_s$						$\overset{}{\underset{e}{\rightarrow}}$	$\overset{s}{\rightarrow}$
11	n^i	e'	e'	e'	n^i	e'	e'	e'	$n^i{}_u*$	e'	e'	$n^i{}_u$	e'	e'	e'	n^i	e'	e'	e'
		$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\overset{d}{\rightarrow}$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow_d$
	$\overset{s}{\rightarrow}$							$_e\emptyset$	$\leftarrow$	$\overset{s}{\leftarrow}$		$_e\emptyset$	$\downarrow_s$						$\overset{}{\underset{e}{\rightarrow}}$
12		$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\downarrow$			$\overset{d}{\rightarrow}$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	
	$\rightarrow$	$\overset{s}{\rightarrow}$						$_e\emptyset$	$\emptyset$	$\overset{s}{\leftarrow}$			$_e\emptyset$	$\downarrow_s$					$_e\emptyset$
13		$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\overset{d}{\rightarrow}$	$\leftarrow$			$\uparrow_d$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
	$_e\emptyset$	$\rightarrow$	$\overset{s}{\rightarrow}$						$_e\emptyset$	$\overset{s}{\leftarrow}$			$\overset{}{\underset{e}{\leftarrow}}$	$\emptyset_s$					
14		$\uparrow$	$\overset{d}{\leftarrow}$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\overset{d}{\rightarrow}$	$\leftarrow$		$\leftarrow$	$\uparrow_d$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
		$_e\emptyset$	$\overset{s}{\rightarrow}$							$_e\emptyset$	$\overset{s}{\leftarrow}$			$\overset{s}{\underset{e}{\leftarrow}}$					
15			$\overset{d}{\leftarrow}$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\rightarrow$			$\leftarrow$		$\leftarrow$		$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
		$_e\uparrow$	$\emptyset$	$\overset{s}{\rightarrow}$						$_e\emptyset$	$\emptyset$	$\overset{s}{\leftarrow}$		$\overset{s}{\underset{e}{\leftarrow}}$					
16		$\uparrow$	$\overset{d}{\leftarrow}$	$\overset{d}{\leftarrow}$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
		$_e\uparrow$	$\emptyset_s$								$_e\emptyset$	$\overset{s}{\leftarrow}$			$\overset{s}{\underset{e}{\leftarrow}}$				
17		$\uparrow$		$\overset{d}{\leftarrow}$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$			$\leftarrow$	$\uparrow$			$\rightarrow$	$\rightarrow$	$\downarrow$
			$_e\emptyset$	$\uparrow_s$							$\overset{}{\underset{e}{\leftarrow}}$	$\emptyset$	$\overset{s}{\leftarrow}$		$_e\emptyset_s$				
18		$\uparrow$	$\leftarrow$	$\overset{d}{\leftarrow}$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
				$_e\emptyset$	$\uparrow_s$							$\overset{}{\underset{e}{\leftarrow}}$	$\overset{s}{\leftarrow}$			$_e\emptyset_s$			
19		$\uparrow$	$\leftarrow$			$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
				$_e\emptyset$	$\uparrow_s$								$\overset{}{\underset{e}{\leftarrow}}$	$\overset{s}{\leftarrow}$			$_e\emptyset_s$		
20	n^i	e'	e'	e'	$n^i{}_d$	e'	e'	e'	n^i	e'	e'	n^i	e'	e'	e'	$n^i{}_d*$	e'	e'	e'
		$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$	$\leftarrow$			$\uparrow$	$\rightarrow$			$\rightarrow$	$\downarrow$
					$_e\emptyset$	$\uparrow_s$							$\overset{}{\underset{e}{\leftarrow}}$	$\leftarrow$	$\overset{s}{\leftarrow}$		$\rightarrow\overset{s}{\underset{e}{}}$		

- If $V_{a,b}$ which has external edge, receives a token with $C_V < 0$, then $V_{a,b}$ marks itself as a *lost* node (state n^l). Note that, $C_V < 0$ indicates that $\exists V_{i,j} \in V_T$, which has external edge, is the upper node. Here, a node $V_{a,b}$ which has external edge, receives a token with $C_V > 0$, it forwards it to the next node following the ring.
- If $V_{a,b}$ which has external edge, receives a token with $C_V = 0$ and $C_H > 0$, then $V_{a,b}$ marks itself as a *lost* node (state n^l). Note that, $C_H > 0$ indicates that $\exists V_{i,j} \in V_T$, which has external edge, is the left node. Here, a node $V_{a,b}$ which has external edge, receives a token with $C_H < 0$, it forwards it to the next node following the ring.
- Computation in this part ends when $V_{a,b} \in V_T$ which has external edge, receives a token with $C_H = C_V = 0$. Therefore, if $V_{a,b}$ with external edge which is not marked as *lost*, receives a token with $C_H = C_V = 0$, then $V_{a,b}$ marks itself as a *bridging* node (state n). However, if $V_{a,b}$ with external edge which is marked as *lost* (state n^l), receives a token with $C_H = C_V = 0$, then $V_{a,b}$ marks itself with state n'.

As an example (see Fig. 9), node h which has an external edge, receives a token with $C_V = 0$ and $C_H > 0$ and marks itself as *lost* node.

Table 5
'Token' implementation: step 21–33.

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
21	n^i R_n $\emptyset$	e' R_2 $\uparrow$ $\uparrow$	e' R_1 $\leftarrow$ $\leftarrow$	e' R_1 $\leftarrow$ $\leftarrow$	n^i R_w $\emptyset$	e' R_1 $\leftarrow$ $\leftarrow$ ${}_e\emptyset$	e' R_1 $\downarrow$ $\downarrow$ $\uparrow_s$	e' R_2 $\rightarrow$ $\rightarrow$	n^{i*} R_e $\emptyset$	e' R_2 $\rightarrow$ $\rightarrow$	e' R_1 $\leftarrow$ $\leftarrow$	n^i R_w $\emptyset$	e' R_1 $\leftarrow$ $\leftarrow$	e' R_2 $\uparrow$ $\uparrow$ $\overset{}{\underset{e}{\leftarrow}}$	e' R_2 $\overset{d}{\rightarrow}$ $\overset{s}{\underset{}{\leftarrow}}$	n^{i*} R_e $\emptyset$	e' R_2 $\rightarrow$ $\rightarrow$	e' R_2 $\rightarrow$ $\rightarrow{}^s_e$	e' R_1 $\downarrow$ $\downarrow$
22		$\uparrow$	$\leftarrow$	$\leftarrow$		$\overset{}{\underset{e}{\leftarrow}}$	$\downarrow_d$ $\emptyset_s$	$\rightarrow$		$\rightarrow$	$\leftarrow$		$\leftarrow$	${}_e\uparrow$	$\overset{d}{\rightarrow}$ $\leftarrow$	$\overset{s}{\underset{}{\leftarrow}}$	$\rightarrow$	$\overset{}{\underset{e}{\rightarrow}}$	$\downarrow$ $\overset{s}{\rightarrow}$
23	$\overset{s}{\rightarrow}$	$\uparrow$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\downarrow_d$ $\overset{s}{\underset{e}{\leftarrow}}$	$\rightarrow$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\overset{d}{\rightarrow}$ ${}_e\uparrow$	$\leftarrow$	$\rightarrow$ $\overset{s}{\underset{}{\leftarrow}}$	$\rightarrow$	$\downarrow$ $\overset{}{\underset{e}{\rightarrow}}$
24	$\rightarrow$	$\uparrow$ $\overset{s}{\rightarrow}$	$\leftarrow$	$\leftarrow$		$\leftarrow$	$\overset{s}{\underset{e}{\leftarrow}}$	$\rightarrow$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	${}_e\emptyset$	$\uparrow$	$\overset{d}{\rightarrow}$ $\overset{s}{\underset{}{\leftarrow}}$	$\rightarrow$	${}_e\downarrow$
25	${}_e\downarrow$	$\uparrow$ $\rightarrow$	$\leftarrow$ $\overset{s}{\rightarrow}$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\rightarrow$ $\overset{s}{\underset{e}{\leftarrow}}$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$	${}_e\emptyset$	$\overset{d}{\rightarrow}$ $\uparrow$	$\rightarrow$ $\overset{s}{\underset{}{\leftarrow}}$	$\downarrow$
26		$\uparrow$ ${}_e\downarrow$	$\overset{d}{\leftarrow}$ $\overset{s}{\rightarrow}$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\rightarrow$ ${}_e\emptyset_s$		$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\overset{d}{\rightarrow}$ ${}_e\emptyset$	$\overset{d}{\rightarrow}$ $\uparrow_s$	$\downarrow$
27		${}_e\emptyset$	$\overset{d}{\leftarrow}$ $\overset{s}{\rightarrow}$	$\leftarrow$		$\leftarrow$	$\downarrow$	$\rightarrow$	${}_e\emptyset_s$	$\rightarrow$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		${}_e\emptyset$	$\overset{d}{\rightarrow}$ $\uparrow_s$	$\downarrow$
28		$\uparrow$	$\overset{d}{\leftarrow}$ ${}_e\emptyset$	$\leftarrow$ $\overset{s}{\rightarrow}$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$ ${}_e\emptyset_s$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\overset{d}{\rightarrow}$ ${}_e\emptyset$	$\downarrow$ $\uparrow_s$
29		$\uparrow$	${}_e\emptyset$	$\overset{d}{\leftarrow}$ $\emptyset_s$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow{}^s_e$	$\leftarrow$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	${}_e\emptyset$	$\downarrow_s$ $\emptyset_s$
30		$\uparrow$	$\leftarrow$	$\overset{d}{\leftarrow}$ ${}_e\emptyset_s$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$	$\leftarrow$ $\rightarrow{}^s_e$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow_d$ ${}_e\emptyset_s$
31		$\uparrow$	$\leftarrow$	${}_e\emptyset_s$		$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$	${}_e\emptyset_s$		$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	${}_e\emptyset_s$
32	${}_e\emptyset_s$	$\uparrow$	$\leftarrow$	$\leftarrow$	${}_e\emptyset_s$	$\leftarrow$	$\downarrow$	$\rightarrow$		$\rightarrow$	$\leftarrow$	${}_e\emptyset_s$	$\leftarrow$	$\uparrow$	$\rightarrow$		$\rightarrow$	$\rightarrow$	$\downarrow$
33	n^f	e' $\uparrow$	e' $\leftarrow$	e' $\leftarrow$	n^f	e' $\leftarrow$	e' $\downarrow$	e' $\rightarrow$	n^{f*}	e' $\rightarrow$	e' $\leftarrow$	n^f	e' $\leftarrow$	e' $\uparrow$	e' $\rightarrow$	n^{f*}	e' $\rightarrow$	e' $\rightarrow$	e' $\downarrow$

At the end of this part of computation, node b marks itself as a *bridging* node.

The proposed meta level description can be implemented following the same description semantics as *identify and delete internal edges* with some modification on the constraints. Therefore, we will describe the implementation detail again. Here, a cell representing bridging node moves to special state (state $n \in \Sigma$) to grow (see Section 3). Whereas, the cells representing lost nodes settle in special state (state n^l, and then state n').

Algorithm 2. Distributed spanning tree algorithm

Step 1: Form a forest F with all nodes.
Step 2: For each tree T in F
(**2.1:**) Find the 'bridging nodes' (state n) having an edge in a special state e.
(using the scheme of 'Finding the bridging node associated with external edges' step).
(**2.2:**) Identify one edge in state e, connected with the bridging node following a predef
rule according to the 'Growing of trees' step (using rule index 1 of Table 1).
(**2.3:**) Use this edge to connect (i.e. mark with state e') T with another tree $T' \in$ F (using
rule index 2 and 3 of Table 1).
Step 3: For each tree T in F
(**3.1:**) Find an internal edge (in state e) which connects two nodes of T (using the scher
of Algorithm 1).
(**3.2:**) Mark the edge with state b (using the scheme of Algorithm 1).
Step 4: If there exists an edge with state e in F
then go to *Step 2*;
otherwise F contains a single tree which is the spanning tree.

Let us now point out the ending of computation. In the proposed CA, each $T \in F$ connects another $T' \in F$ to get a larger tree using bridging node. Hence, the number of trees of F reduces. This process stops when there is a single tree in the forest, i.e. $|F| = 1$. That is, the CA has reached a fixed point when none of cell representing $V_{i,j} \in V_T$ of a tree $T = (V_T, E_T)$ is not associated with external edge i.e. none of the cells, representing edges, are not in a special state (state $e \in \Sigma$). Finally, the essential components of the proposed scheme are noted in Algorithm 2. We follow here a sketchy way to state the algorithm. To make it more clear, we further provide a flow chart of this scheme in Fig. 13.

Figs. 14–16 show the three example cases of the computation, where Fig. 14 (resp. Figs. 15 and 16) deals with a 3×3 fully connected grid (resp. 3×3 spanning tree; and 4×4 not fully connected grid with cycles).

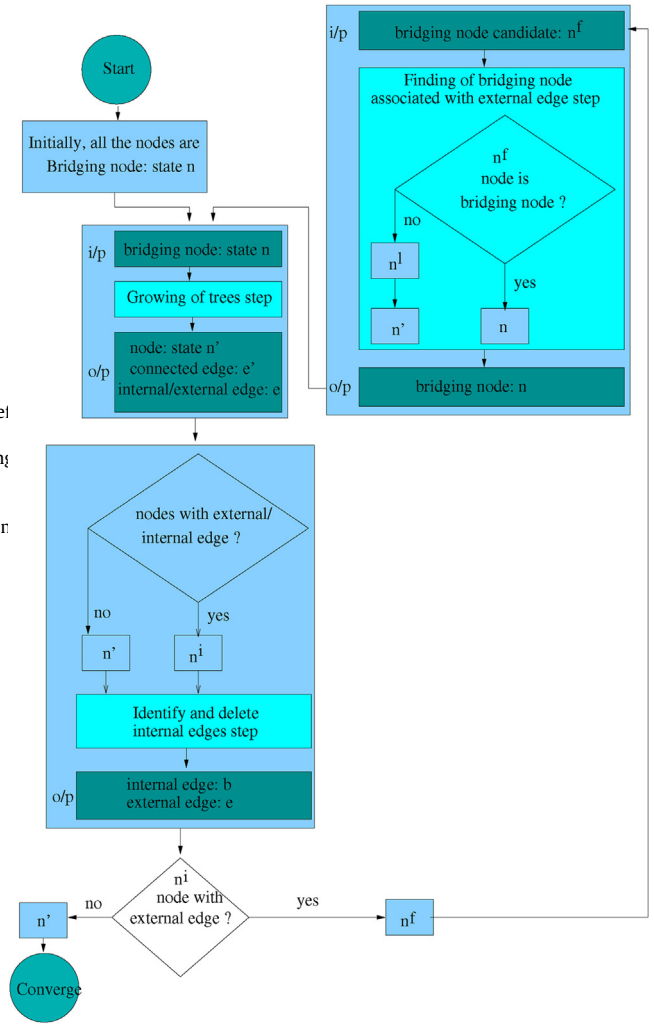

Fig. 13. The essential components and state changes of the proposed computation.

Remark 1. In the proposed CA, the set of input alphabet is $\Sigma = \{n, e, b\}$. Here, the 'growing of trees' part needs six states – $\{n^w, n^n, n^e, n^s, n', e'\}$. The 'finding of bridging node' part needs 27 states – $\{\emptyset, \longleftarrow, \rightarrow, \uparrow, \downarrow, \overset{s}{\leftarrow}, \overset{s}{\rightarrow}, \uparrow_s, \downarrow_s, \emptyset_s, \underset{e}{\longleftarrow}, \underset{e}{\rightarrow}, e\uparrow, e\downarrow, e\emptyset, \overset{s}{\underset{e}{\longleftarrow}}, \overset{s}{\underset{e}{\rightarrow}}, e\uparrow_s, e\downarrow_s, e\emptyset_s, n^i, n^i_r, n^i_l, n^i_d, n^i_u, n^f, n^l\}$. Hence, total number of states of the proposed CA is 36.

5. Discussion

Proposition 2. *No circuit is formed when trees are connected (following Growing of Trees rules) by the proposed CA. As a consequence, the new graph is also a tree.*

Proof. In the proposed CA, all trees are initially null trees. That is, all the nodes act as bridging nodes. If F is the set of all trees, then there is no internal edge in any tree of F.

Now to get a circuit by connecting some trees of F (through external edges), there has to exist a set $F' \subseteq F$, so that each $T \in F'$ is inclined to at least two external edges (Fig. 17 is an example of F'). According to the rule of the proposed CA, only one external edge having highest priority is used to connect two trees. So, as a matter of fact, no circuit is formed by executing the CA on null trees (otherwise, an external edge with lower priority is to be used when an edge with higher priority exists; see Fig. 17 for clarification).

While the CA work with forest of null trees, we get a new F which does not have any null tree. In this situation, there exists no left side external edge of any node after the 'growing of trees' on null trees. This implies that there also exists no right side external edges of nodes. Hence, the only possible situation for circuit formation after the 'growing of trees' on null trees is that: a circuit is formed by two trees T and T', where a bridging node $V_{i,j} \in V_T$ of the tree $T = (V_T, E_T)$ joins through edge $e = (V_{i,j}, V_{a,b}) \in E_T$ (north or south edge), $V_{a,b} \in V_{T'}$, whereas a bridging node $V_{a',b'} \in V_{T'}$ of the tree $T' = (V_{T'}, E_{T'})$ joins through edge $e' = (V_{a',b'}, V_{i',j'}) \in E_{T'}$ (south or north edge), $V_{i',j'} \in V_T$.

Now, $V_{i,j}$ must be uppermost amongst leftmost nodes of the tree T which have external edge(s) (according to Definition 3). Node $V_{a,b}$ is also associated with external edge(s), where $e = (V_{i,j}, V_{a,b})$. Therefore, either $V_{a,b}$ is the uppermost amongst leftmost nodes of tree T' or $V_{i,j}$ is not the bridging node of tree T. The same analogy is also true for $V_{a',b'}$ which is the bridging node of tree T'. Hence, if $V_{i,j}$ (resp. $V_{a',b'}$) is the bridging node of the tree T (resp. T'), then $V_{a,b}$ (resp. $V_{i',j'}$) must be the bridging node of tree T' (resp T). This implies, $e = e'$. Hence, the proof. □

Proposition 3. *The worst case convergence time for the proposed CA is $\mathcal{O}(n \log n)$, where n is the number of nodes in the initial configuration.*

Proof. In the proposed CA computation is divided into two parts. Firstly, the 'growing of trees' part takes constant (here, 3) time steps. Secondly, the 'finding of bridging node' is also subdivided into two computational part. Here, both of the computational time depends on the perimeter of the tree, which is clearly dependent on the

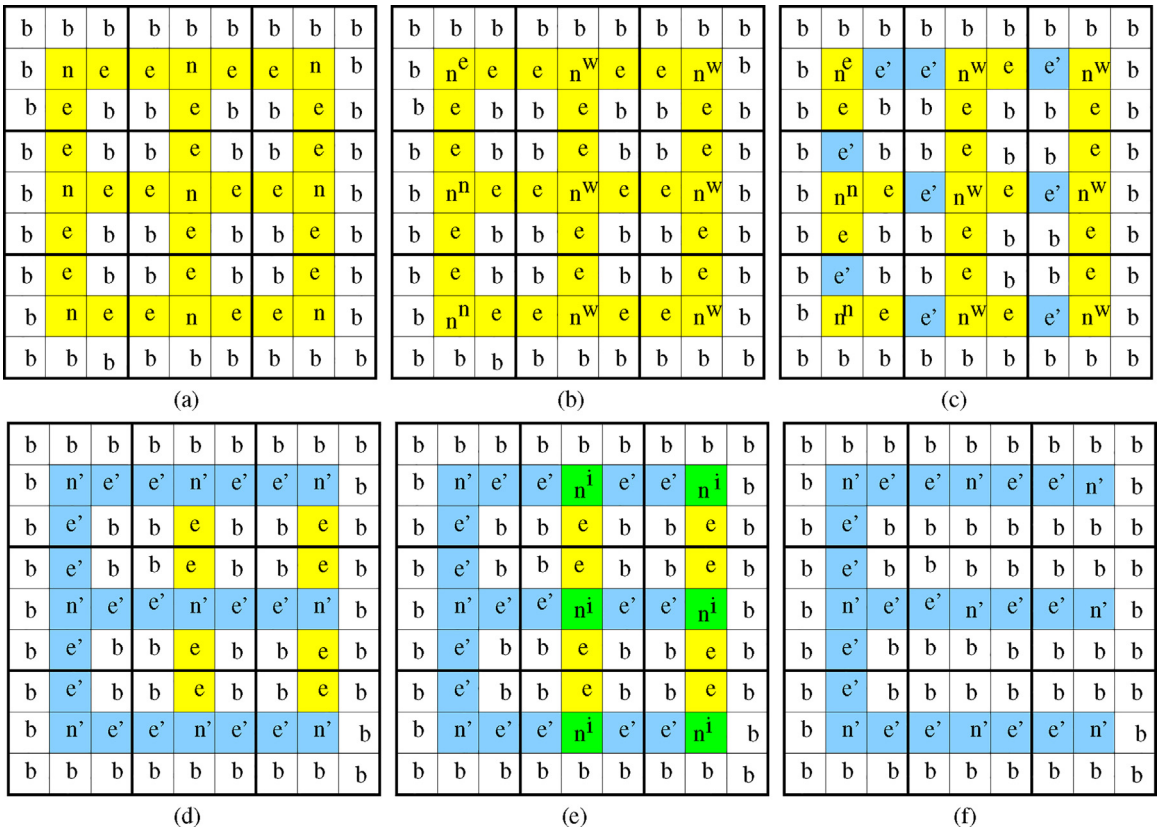

Fig. 14. (a) The initial configuration corresponding to a 3×3 fully connected grid; (b–d) computation during 'growing of trees'; (e) computation during 'identify and delete internal edges'; and (f) the final spanning tree.

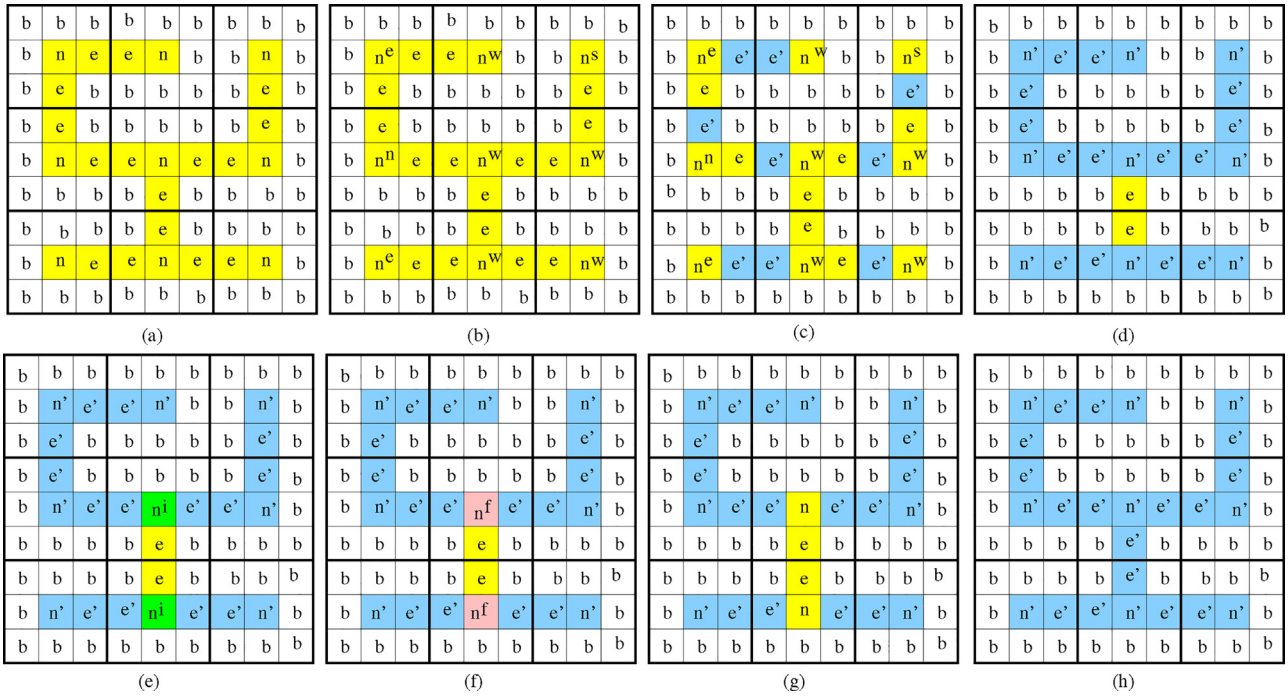

Fig. 15. (a) The initial configuration corresponding to a 3×3 spanning tree; (b–d) computation during 'growing of trees'; (e) computation during 'identify and delete internal edges'; (f–g) computation during 'finding of bridging node associated with external edge'; and (h) the final spanning tree.

number of nodes in the tree T. Therefore, 'finding of bridging node' takes $\mathcal{O}(n)$ time steps, where n is the number of nodes in the initial configuration in worst case. Let the steps be repeated k times. Hence, the worst case convergence time is $\mathcal{O}(kn)$ time steps.

Now, according to Proposition 1, the number of trees after single execution of these parts reduces to half. Therefore, the worst number of execution is $\mathcal{O}(\log n)$, where n is the number of nodes in

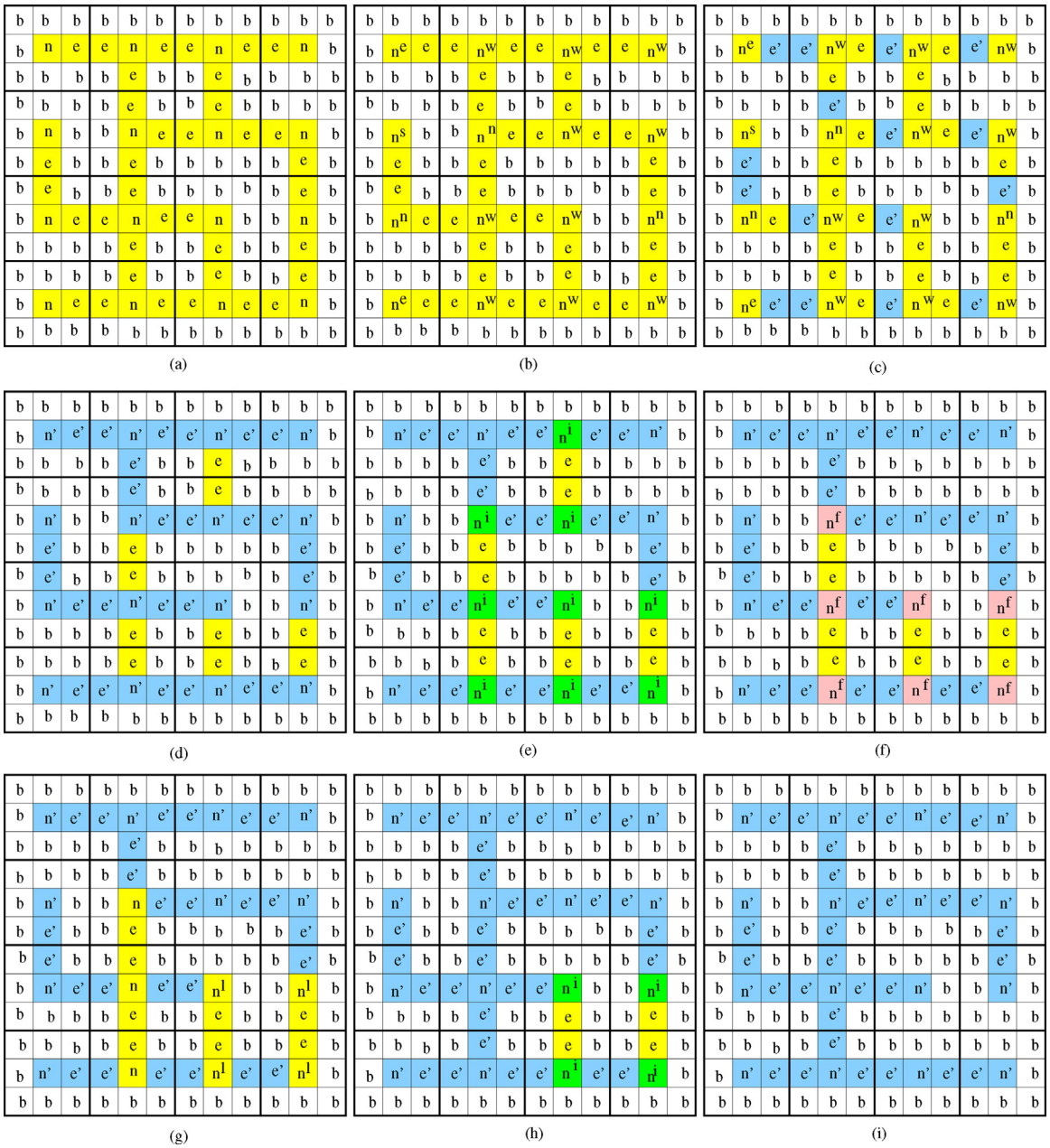

Fig. 16. (a) The initial configuration corresponding to a 4×4 not fully connected grid with cycles; (b–d) computation during 'growing of trees' in the first iteration; (e) computation during 'identify and delete internal edges' in the first iteration; (f–g) computation during 'finding of bridging node associated with external edge' in the first iteration; (h) computation during 'identify and delete internal edges' in the second iteration; and (i) the final spanning tree.

the initial configuration. Hence, the worst case convergence time for the proposed CA is $\mathcal{O}(n \log n)$. □

The work has reported a CA that computes spanning tree of given (grid) graph. Now, the natural extensions of this work on the sensibility to cellular automata include:

- What can be said for a neighbourhood with more neighbourhood dependency?
- What can be said for d-dimension case? Note that, a natural extension under more neighbourhood dependency and d-dimensional CA can address a larger class of graphs.
- What can be said for minimum number of states required to solve this problem?
- what can be said for a solution CA under asynchronous (see [7,35]) update scheme?

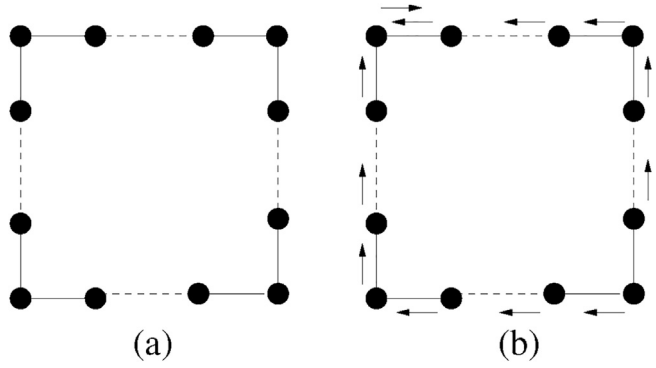

Fig. 17. (a) A situation to get a circuit by null trees; (b) 'growing of trees' on that situation.

- Is it possible to get a CA that solves the problem with lesser amount of time?

Acknowledgements

The authors acknowledge the anonymous reviewers for their comments and suggestions, which have helped to improve the quality and readability of the paper.

References

[1] J. von Neumann, in: A.W. Burks (Ed.), The Theory of Self-reproducing Automata, Univ. of Illinois Press, Urbana and London, 1966.
[2] A. Smith, Introduction to and Survey of Polyautomata Theory, Automata, Languages, Development, North Holland Publishing Co., 1976.
[3] M. Stratmann, T. Worsch, Leader election in d-dimensional ca in time diam log(diam), Future Gener. Comput. Syst. 18 (7) (2002) 939–950.
[4] A. Beckers, T. Worsch, A perimeter-time ca for the queen bee problem, Parallel Comput. 27 (5) (2001) 555–569.
[5] J. Mazoyer, C. Nichitiu, E. Rémila, Compass permits leader election, in: Proc. of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '99, Society for Industrial and Applied Mathematics, 1999, pp. 947–948.
[6] C. Nichitiu, E. Rémila, Leader election by *d* dimensional cellular automata, in: Automata, Languages and Programming, Springer, 1999, pp. 565–574.
[7] K. Bhattacharjee, N. Naskar, S. Roy, S. Das, A survey of cellular automata: types, dynamics, non-uniformity and applications, CoRR abs/1607.02291 (2016).
[8] S. Roy, S. Das, Distributed mutual exclusion problem in cellular automata, J. Cell. Autom. 12 (6) (2017) 493–512.
[9] A.I. Adamatzky, Computation of shortest path in cellular automata, Math. Comput. Model. 23 (4) (1996) 105–113.
[10] M. Delorme, J. Mazoyer, L. Tougne, Discrete parabolas and circles on 2d cellular automata, Theor. Comput. Sci. 218 (2) (1999) 347–417.
[11] P. de Oliveira, Conceptual connections around density determination in cellular automata, in: Cellular Automata and Discrete Complex Systems, vol. 8155, Springer, Berlin, Heidelberg, 2013, pp. 1–14.
[12] H. Fukś, Solution of the density classification problem with two cellular automata rules, Phys. Rev. E 55 (1997) 2081R–2084R.
[13] N. Fatès, Stochastic cellular automata solutions to the density classification problem – when randomness helps computing, Theory Comput. Syst. 53 (2) (2013) 223–242.
[14] J. Mazoyer, A six-state minimal time solution to the firing squad synchronization problem, Theor. Comput. Sci. 50 (2) (1987) 183–238.
[15] H. Attiya, J.L. Welch, Distributed Computing: Fundamentals, Simulations and Advanced Topics, Mc Graw-Hill, London-UK, 1998.
[16] B. Awerbuch, I. Cidon, S. Kutten, Optimal maintenance of a spanning tree, J. ACM 55 (4) (2008), 18:1-18:45.
[17] N.-S. Chen, H.-P. Yu, S.-T. Huang, A self-stabilizing algorithm for constructing spanning trees, Inf. Process. Lett. 39 (3) (1991) 147–151.
[18] S. Dolev, A. Israeli, S. Moran, Self-stabilization of dynamic systems assuming only read/write atomicity, Distrib. Comput. 7 (1) (1993) 3–16.
[19] C. Johnen, Memory efficient, self-stabilizing algorithm to construct BFS spanning trees, in: Proc. of the 16th Annual ACM Symposium on Principles of Distributed Computing, PODC '97, ACM, 1997, 288.
[20] J.-S. Yang, S.-M. Tang, J.-M. Chang, Y.-L. Wang, Parallel construction of optimal independent spanning trees on hypercubes, Parallel Comput. 33 (1) (2007) 73–79.
[21] C. Savage, J. Ja'Ja', Fast, efficient parallel algorithms for some graph problems, SIAM J. Comput. 10 (4) (1981) 682–691.
[22] D. Johnson, P. Metaxas, A parallel algorithm for computing minimum spanning trees, J. Algorithms 19 (3) (1995) 383–401.
[23] R. Miller, V.K. Prasanna-Kumar, D.I. Reisis, Q.F. Stout, Parallel computations on reconfigurable meshes, IEEE Trans. Comput. 42 (6) (1993) 678–692.
[24] D.A. Bader, G. Cong, Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs, J. Parallel Distrib. Comput. 66 (11) (2006) 1366–1378.
[25] M.J. Atallah, S.R. Kosaraju, Graph problems on a mesh-connected processor array, J. ACM 31 (3) (1984) 649–667.
[26] S.G. Akl, An adaptive and cost-optimal parallel algorithm for minimum spanning trees, Computing 36 (3) (1986) 271–277.
[27] K.N. Levitt, W.H. Kautz, Cellular arrays for the solution of graph problems, Commun. ACM 15 (9) (1972) 789–801.
[28] D. Angluin, Local and global properties in networks of processors (extended abstract), in: Proc. of the 12th Annual ACM Symposium on Theory of Computing, STOC '80, ACM, 1980, pp. 82–93.
[29] M.I. Tsompanas, G.C. Sirakoulis, A. Adamatzky, Evolving transport networks with cellular automata models inspired by slime mould, IEEE Trans. Cybern. 45 (9) (2015) 1887–1899.
[30] V. Evangelidis, M.-A. Tsompanas, G.C. Sirakoulis, A. Adamatzky, Slime mould imitates development of roman roads in the Balkans, J. Archaeol. Sci.: Rep. 2 (Suppl. C) (2015) 264–281.
[31] M.-A.I. Tsompanas, R. Mayne, G.C. Sirakoulis, A.I. Adamatzky, A cellular automata bioinspired algorithm designing data trees in wireless sensor networks, Int. J. Distrib. Sens. Netw. 11 (6) (2015) 471045.
[32] M.-A.I. Tsompanas, G.C. Sirakoulis, Modeling and hardware implementation of an amoeba-like cellular automaton, Bioinspir. Biomimet. 7 (3) (2012) 036013.
[33] M. Chean, J.A.B. Fortes, A taxonomy of reconfiguration techniques for fault-tolerant processor arrays, Computer 23 (1) (1990) 55–69.
[34] J.W. Greene, A. El Gamal, Configuration of VLSI arrays in the presence of defects, J. ACM 31 (4) (1984) 694–717.
[35] B. Sethi, S. Roy, S. Das, Asynchronous cellular automata and pattern classification, Complexity 21 (S1) (2016) 370–386.

Souvik Roy is a Ph.D. candidate at the Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, India. He received his M.E. degree from the Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, India in 2014. His research interests are Complex systems, Cellular automata models for distributed systems and Asynchronous cellular automata.

Arunima Ray received her M.E. degree from the Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, India in 2016. Her research interests include Complex systems and Cellular automata.

Sukanta Das is a professor of Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, India. He received his M.E. and Ph.D. degrees from Indian Institute of Engineering Science and Technology, Shibpur, India. His research interests are Cellular automata: classical, non-uniform, asynchronous and Distributed computing.