Faster synchronization in P systems

Michael J. Dinneen · Yun-Bum Kim ·
Radu Nicolescu

Abstract In the field of molecular computing, in particular P systems, synchronization is an important requirement for composing or sequentially linking together congenial P system activities. We provide a deterministic algorithm to the *Firing Squad Synchronization Problem*, for digraph-based P systems, which runs in $3e + 11$ steps, where e is the eccentricity of the general. Our algorithm uses a convenient framework, called simple P modules, which embraces the essential features of several popular types of P systems.

Keywords Cellular automata · P systems · Simple P modules · Firing squad synchronization

1 Introduction

The *Firing Squad Synchronization Problem* (FSSP) (Goto 1962; Moore 1964) is one of the best studied problems for cellular automata. The initial problem, involves finding a cellular automaton, such that, some time after a command is given, all the cells in a line enter a designated *firing* state *simultaneously* and *for the first time*. Several variants of FSSP have been proposed and studied, for variety of structures (Nishitani and Honda 1981; Szwerinski 1982; Grefenstette 1983; Schmid and Worsch 2004). Studies of these variations mainly focus on finding a solution with as few states as possible and possibly running in optimum

M. J. Dinneen (✉) · Y.-B. Kim · R. Nicolescu
Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
e-mail: mjd@cs.auckland.ac.nz

Y.-B. Kim
e-mail: yun@cs.auckland.ac.nz

R. Nicolescu
e-mail: radu@cs.auckland.ac.nz

time (Waksman 1966; Balzer 1967; Mazoyer 1987; Imai et al. 2002; Kobayashi and Goldstein 2005; Umeo et al. 2005).

There are several applications that require synchronization. We list just a few here. At the biological level, cell synchronization is a process by which cells at different stages of the cell cycle (division, duplication, replication) in a culture are brought to the same phase. There are several biological methods used to synchronize cells at specific cell phases (Humphrey 2005). Once synchronized, monitoring the progression from one phase to another allows us to calculate the timing of specific cells' phases. Another example relates to computer networks (Freeman 2005), where we often want to synchronize computers to the same time, i.e. primary reference clocks should be used to avoid clock offsets.

The synchronization problem has recently been studied in the framework of P systems. Using tree-based P systems, Bernardini et al. (2008) provided a non-deterministic solution with time complexity $3h$ and a deterministic solution with time complexity $4n + 2h$, where h is the height of the tree structure underlying the P system and n is the number of membranes of the P system. The deterministic solution requires membrane *polarization* techniques and uses a *depth-first-search*.

More recently, Alhazov et al. (2008) described an improved deterministic algorithm for tree-based P systems, that runs in $3h + 3$ steps. This solution requires conditional rules (promoters and inhibitors) and combines a *breadth-first-search*, a *broadcast* and a *convergecast*.

We continue our study of FSSP for digraph-based P systems (Dinneen et al. 2009, 2010c), where we proposed uniform deterministic solutions to a variant of FSSP (Szwerinski 1982), in which there is a single general, at an arbitrary position, and we synchronize a subset of cells (or

membranes) of the considered P system. In contrast to the previous FSSP solutions, our solutions require states and priority rules, instead of membrane polarizations or conditional rules. We have earlier presented (Dinneen et al. 2010c) two FSSP algorithms with the running times of $4e + 13$ and $3e + 13$, where e is the eccentricity of the initiator; the former does not use cell IDs and latter uses cell IDs. All cells of our solutions start with the same state and rules, and have no priori knowledge of the network topology.

In this paper, we present an improved deterministic FSSP algorithm, for P systems with digraph membrane structure, which runs in $3e + 11$ steps, without the support of cell IDs. The rest of the paper is organized as follows. In Sect. 2, we define a virtual communication structure for a given P system structure, based on the recursive construction of the transitive closure of the neighboring relation. We also establish a convenient P system framework, called simple P modules (Dinneen et al. 2010a). In Sect. 3, we provide an overview and the P module specification for our FSSP algorithm. Finally, in Sect. 4, we summarize our results and conclude with some open problems.

2 Preliminaries

We assume that the reader is familiar with the basic terminology and notations, such as relations, graphs, nodes (vertices), arcs, edges, directed graphs (digraphs), directed acyclic graphs (dags), alphabets, strings and multisets.

For a digraph (X, δ), recall that $\mathsf{Neighbor}(x) = \delta(x) \cup \delta^{-1}(x)$. The relation $\mathsf{Neighbor}$ is always symmetric and defines a graph structure, which will be here called the virtual *communication graph* defined by δ.

A special node g of X is designated as the (fixed) *general*. For a given general g, we define the *depth* of a node x, $\mathsf{depth}_g(x) \in \mathbb{N}$, as the length of any shortest path between the c and x, over the $\mathsf{Neighbor}$ relation. Recall that the *eccentricity* of a node $x \in X$, e_x, as the maximum length of a shortest path between x and any other node. We note $\mathsf{e}_g = \max\{\mathsf{depth}_g(x) \mid x \in X\}$.

Given nodes x, y, if $y \in \mathsf{Neighbor}(x)$ and $\mathsf{depth}_g(y) = \mathsf{depth}_g(x) + 1$, then (x, y) is a *depth-increasing arc*, x is a *predecessor* of y and y is a *successor* of x. Similarly, a node z is a *peer* of x, if $z \in \mathsf{Neighbor}(x)$ and $\mathsf{depth}_g(z) = \mathsf{depth}_g(x)$. Note that, for node x, the set of peers and the set of successors are disjoint. A node without a *successor* will be referred to as a *terminal*. For node x, $\mathsf{Pred}_g(x) = \{y \mid y \text{ is a predecessor of } x\}$, $\mathsf{Peer}_g(x) = \{y \mid y \text{ is a peer of } x\}$, $\mathsf{Succ}_g(x) = \{y \mid y \text{ is a successor } of x\}$.

The depth-increasing arcs form a virtual *shortest-paths dag*, where each path from general c to a node y is a shortest path, over the $\mathsf{Neighbor}$ relation. We further define $\mathsf{height}_g(y)$ as the height of y in the shortest-paths dag and $\mathsf{paths}_g(y)$ as the number of shortest paths from g to y.

If, as we further assume, the original digraph, δ, is *weakly connected*, then the shortest paths dag has a single *source*, general c.

Figure 1a illustrates a digraph with the general $g = 1$. Figure 1b illustrates its communication graph, where the nodes at even distance from g are shaded and the depth-increasing arcs of the shortest-paths dag are marked by additional arrows. For each node x, Fig. 1c indicates $\mathsf{Neighbor}(x)$, $\mathsf{Pred}_g(x)$, $\mathsf{Peer}_g(x)$, $\mathsf{Succ}_g(x)$, $\mathsf{depth}_g(x)$, $\mathsf{height}_g(x)$ and $\mathsf{paths}_g(x)$.

Definition 1 (*simple P module*, Dinneen et al. 2010a). A *simple P module with duplex channels* is a system $\Pi = (O, K, \delta)$, where:

1. O is a finite non-empty alphabet of *objects*;
2. K is a finite set of *cells*;
3. δ is an *irreflexive* binary relation on K, which represents a set of structural arcs between cells, with *duplex* communication capabilities.
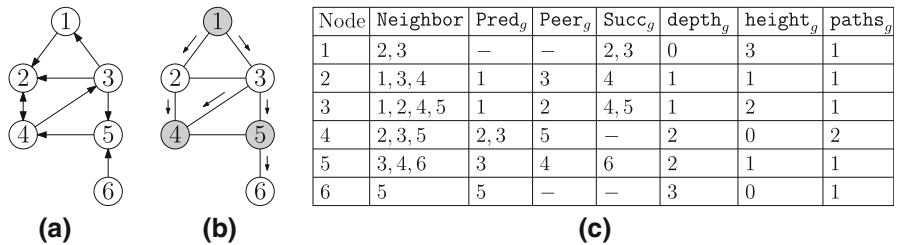
Each cell, $\sigma_i \in K$, has the initial configuration $\sigma_i = (Q_i, s_{i0}, w_{i0}, R_i)$, and the current configuration $\sigma_i = (Q_i, s_i, w_i, R_i)$, where:

- Q_i is a finite set of *states*;
- $s_{i0} \in Q_i$ is the *initial state*; $s_i \in Q_i$ is the *current state*;
- $w_{i0} \in O^*$ is the *initial content*; $w_i \in O^*$ is the *current content*; note that, $|w_i|_o, o \in O$, denotes the *multiplicity* of object o in the multiset w_i;
- R_i is a finite *ordered* set of multiset rewriting *rules* of the form: $sx \rightarrow_\alpha s'x'(u)_{\beta_\gamma}$, where $s, s' \in Q, x, x' \in O^*$, $u \in O^*$, $\alpha \in \{\mathsf{min}, \mathsf{max}\}$, $\beta \in \{\uparrow, \downarrow, \updownarrow\}$, $\gamma \in \{\mathsf{one}, \mathsf{spread}, \mathsf{repl}\} \cup K$. If $u = \lambda$, i.e. the *empty* multiset of objects, this rule can be abbreviated as $sx \rightarrow_\alpha s'x'$.

Cells *evolve* by applying rules. An evolving cell applies one or more rules, which can change its content and state and send objects to its neighbors. For a cell $\sigma_i = (Q_i, s_i, w_i, R_i)$, a rule $sx \rightarrow_\alpha s'x'(u)_{\beta_\gamma} \in R_i$ is *applicable* if $s = s_i$ and $x \subseteq w_i$. The application of a rule takes two sub-steps, after which the cell's current state s is replaced by *target state* $s´$, the current content x is replaced by $x´$ and multiset u is sent as specified by the transfer operator β_γ (as further described below).

The rules are applied in the *weak priority* order (Păun 2006), i.e. (1) higher priority applicable rules are applied before lower priority applicable rules, and (2) a lower priority applicable rule is applied only if it indicates the same target state as the previously applied rules. We use the notation $s \Rightarrow s'$ to indicate a state transition from current state s to target state s'.
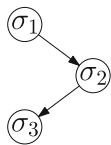
Fig. 1 a A sample digraph with the general]] $g = 1$. **b** Its virtual communication graph and shortest-paths dag. **c** A table with node attributes introduced in this section

Node	Neighbor	$Pred_g$	$Peer_g$	$Succ_g$	$depth_g$	$height_g$	$paths_g$
1	2, 3	–	–	2, 3	0	3	1
2	1, 3, 4	1	3	4	1	1	1
3	1, 2, 4, 5	1	2	4, 5	1	2	1
4	2, 3, 5	2, 3	5	–	2	0	2
5	3, 4, 6	3	4	6	2	1	1
6	5	5	–	–	3	0	1

In this paper, we use the rewriting operator $\alpha = \mathsf{max}$ and the transfer operator $\updownarrow_{\mathsf{repl}}$. The rewriting operator $\alpha = \mathsf{max}$ indicates that an applicable rewriting rule of R_i is applied as many times as possible. If the right-hand side of the rule contains $(u)_{\updownarrow_{\mathsf{repl}}}$, then, for each application of this rule, a copy of multiset u is sent to each of the neighboring cells (i.e. cells in $\delta(i) \cup \delta^{-1}(i)$). Other rewriting and transfer operators, not used in this paper, are described in Dinneen et al. (2010b). The following example illustrates the behavior of operators that are used in this paper.

Example 2 Consider a simple P module $\Pi = (\{a, b, c, d, e, f, g\}, \{\sigma_1, \sigma_2, \sigma_3\}, \{(\sigma_1, \sigma_2), (\sigma_2, \sigma_3)\})$, where each cell $\sigma_i \in K$ has the initial form (Q, s_{i0}, w_{i0}, R), where:

- $Q = \{s_0, s_1\}$.
- $s_{i0} = s_0$.
- $w_{i0} = \begin{cases} aabbc & \text{if } \sigma_i = \sigma_2, \\ \lambda & \text{if } \sigma_i \neq \sigma_2. \end{cases}$
- R is the following sequence of rules:

1. $s_0 a \rightarrow_{\mathsf{max}} s_0 d(d)_{\updownarrow_{\mathsf{repl}}}$
2. $s_0 b \rightarrow_{\mathsf{max}} s_0 e$
3. $s_0 c \rightarrow_{\mathsf{max}} s_1 f$
4. $s_0 c \rightarrow_{\mathsf{max}} s_0 g$

In this scenario, all rules are applicable for cell σ_2. First, rule 1 is applied twice and sets the target state to s_0. Next, rule 2 is applied twice, then rule 3 is not applied (because it indicates a different target state, s_1) and, finally, rule 4 is applied once. In the final configuration of the system, after one step, cell σ_1 contains dd, cell σ_2 contains $ddeeg$ and cell σ_3 contains dd.

3 Deterministic FSSP solution

In FSSP, all cells start in a *quiescent* state, i.e. in a state where no rules are applicable if the cell is empty. Also, all cells are *empty*, except the *general* cell. In principle, the general sends a "firing order" to all cells, which will prompt them to synchronize, by entering a designated *firing* state (different from the initial quiescent state), simultaneously and for the first time. However, in general, the general does not have direct communication channels to all cells, thus, the firing order has to be relayed through intermediate cells. Relaying the order through intermediate cells results in some cells receiving the order before other cells. To ensure that all cells enter the firing state simultaneously, each cell needs to wait until all other cells receive the order.

Our FSSP algorithm works in four phases. In Phases I and II, prior to sending the firing order, the general determines its *eccentricity*, using all shortest paths available. In Phase III, the general sends the firing order, paired with a *hop-count*, initially set to its eccentricity. The order is further broadcasted to all cells, again via shortest paths. Each cell decrements the hop-count by one, before forwarding the order. In Phase IV, each cell keeps decrementing the hop-count by one, until the hop-count becomes zero, and then enters the firing state; this ensures that cells enter the firing state simultaneously.

Our FSSP algorithm is implemented using the simple P module $\Pi = (O, K, \delta)$, where

1. $O = \{a, b, c, d, e, h, o, r, v, x\}$.
2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.
3. δ is a weakly connected digraph.

The general is an arbitrary cell $\sigma_g \in K$. All cells have the same set of states, the same set of rules and start at the same initial quiescent state; however, they have different initial contents. Thus, each cell $\sigma_i \in K$ has the initial form $\sigma_i = (Q, s_0, w_{i0}, R)$, where:

- $Q = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$, where s_0 is the initial quiescent state and s_9 is the firing state.
- $w_{i0} = \begin{cases} \{a\} & \text{if } \sigma_i = \sigma_g, \\ \emptyset & \text{if } \sigma_i \neq \sigma_g. \end{cases}$
- R is defined by the following rulesets, grouped by the conceptual four phases.
  - ○ Rules used in Phase I:
    - 0. Rules for state s_0:
      - (1) $s_0 a \rightarrow_{\mathsf{max}} s_1 abbbde(o)_{\updownarrow_{\mathsf{repl}}}$
      - (2) $s_0 o \rightarrow_{\mathsf{max}} s_1 a(x)_{\updownarrow_{\mathsf{repl}}}$
      - (3) $s_0 x \rightarrow_{\mathsf{max}} s_1 ae(o)_{\updownarrow_{\mathsf{repl}}}$

1. Rules for state s_1:
   (1) $s_1 a \rightarrow_{\max} s_2 a$
   (2) $s_1 o \rightarrow_{\max} s_2 r$
   (3) $s_1 x \rightarrow_{\max} s_2 r$

2. Rules for state s_2:
   (1) $s_2 e \rightarrow_{\max} s_3$
   (2) $s_2 a \rightarrow_{\max} s_4 a$
   (3) $s_2 x \rightarrow_{\max} s_3 v$
   (4) $s_2 o \rightarrow_{\max} s_4 v$

○ Rules used in Phase II:

3. Rules for state s_3:
   (1) $s_3 h \rightarrow_{\max} s_5 r$
   (2) $s_3 xv \rightarrow_{\max} s_3$
   (3) $s_3 av \rightarrow_{\max} s_3 av$
   (4) $s_3 a \rightarrow_{\max} s_3 ah (o)_{\downarrow_{\text{repl}}}$
   (5) $s_3 bd \rightarrow_{\max} s_3 bbd$

4. Rules for state s_4:
   (1) $s_4 h \rightarrow_{\max} s_5 r$
   (2) $s_4 ov \rightarrow_{\max} s_4$
   (3) $s_4 av \rightarrow_{\max} s_4 av$
   (4) $s_4 a \rightarrow_{\max} s_4 ah (x)_{\downarrow_{\text{repl}}}$

5. Rules for state s_5:
   (1) $s_5 dr \rightarrow_{\max} s_6$
   (2) $s_5 bb \rightarrow_{\max} s_6 b$
   (3) $s_5 rx \rightarrow_{\max} s_5$
   (4) $s_5 ro \rightarrow_{\max} s_5$
   (5) $s_5 r \rightarrow_{\max} s_5 r$
   (6) $s_5 a \rightarrow_{\max} s_6 a$

○ Rules used in Phase III:

6. Rules for state s_6:
   (1) $s_6 ab \rightarrow_{\max} s_7 ah$
   (2) $s_6 b \rightarrow_{\max} s_7 c (b)_{\downarrow_{\text{repl}}}$

7. Rules for state s_7:
   (1) $s_7 h \rightarrow_{\max} s_7$
   (2) $s_7 a \rightarrow_{\max} s_8 a$
   (3) $s_7 b \rightarrow_{\max} s_8$

○ Rules used in Phase IV:

8. Rules for state s_8:
   (1) $s_8 ac \rightarrow_{\max} s_8 a$
   (2) $s_8 a \rightarrow_{\max} s_9$

To simplify our arguments, for each cell $\sigma_i \in K$, we define $\mathsf{paths}_g(i) = \mathsf{paths}_{\sigma_g}(\sigma_i), \mathsf{depth}_g(i) = \mathsf{depth}_{\sigma_g}(\sigma_i),$ $\mathsf{height}_g(i) = \mathsf{height}_{\sigma_g}(\sigma_i), \mathsf{Pred}_g(i) = \mathsf{Pred}_{\sigma_g}(\sigma_i), \mathsf{Succ}_g(i)$ $= \mathsf{Succ}_{\sigma_g}(\sigma_i), \mathsf{Peer}_g(i) = \mathsf{Peer}_{\sigma_g}(\sigma_i)$. Additionally, we define the following "variable" objects, which depend on the depth of the cell σ_i: $\mu_i = x, \overline{\mu}_i = o$, if $\mathsf{depth}_g(i)$ is even and $\mu_i = o, \overline{\mu}_i = x$, if $\mathsf{depth}_g(i)$ is odd. In Phases I and II, this alternation between μ_i and $\overline{\mu}_i$ enables cell σ_i to distinguish between "useful" objects, μ_i, received from predecessors and successors, and "noise" objects, $\overline{\mu}_i$,

received from peers; cell σ_i will itself send out $\overline{\mu}_i$ objects, to all its neighbors.

3.1 FSSP Phase I

Phase I is a broadcast initiated by the general, relayed from *predecessors* to *successors*, using $\mu_i, \overline{\mu}_i$ as *broadcast objects*. Intuitively, cell σ_i expects to receive "useful" objects μ_i, first from its predecessors, then from its successors, "noise" objects $\overline{\mu}_i$, from its peers, and sends $\overline{\mu}_i$ objects, to all its neighbors. Additionally, the general starts a *counter*, which is incremented by one in each step.

Phase I (First broadcast from the general)

Precondition: Phase I starts with P module Π in its initial configuration.

Postcondition: At the end of Phase I, the configuration of cell $\sigma_i \in K$ is (Q, s_i, w_i, R), where

- $s_i = \begin{cases} s_3 & \text{if } \mathsf{depth}_g(i) \text{ is even,} \\ s_4 & \text{if } \mathsf{depth}_g(i) \text{ is odd}s. \end{cases}$

- $|w_i|_a = \mathsf{paths}_g(i)$, is the number of shortest paths from σ_g to σ_i.

- $|w_i|_b = 3$ and $|w_i|_d = 1$, if $\sigma_i = \sigma_g$, is used to implement the general's counter.

- $|w_i|_r = \sum_{\sigma_j \in \mathsf{Peer}_g(i)} \mathsf{paths}_g(j)$, used in Phase II, as the expected number of convergecast objects from peers.

- $|w_i|_v = \sum_{\sigma_k \in \mathsf{Succ}_g(i)} \mathsf{paths}_g(k)$, used in Phase II, as the expected number of convergecast objects from successors.

Description: In Phase I, each cell σ_i makes three state transitions: if $\mathsf{depth}_g(i)$ is even (which includes the general), then $s_0 \Rightarrow s_1 \Rightarrow s_2 \Rightarrow s_3$; otherwise $s_0 \Rightarrow s_1 \Rightarrow s_2 \Rightarrow s_4$. The general σ_g, identified by its initial content a, sends the first broadcast object, one copy of $\overline{\mu}_g = o$, to each of its neighbors. Each other cell $\sigma_i \neq \sigma_g$ receives its first broadcast objects μ_i from its predecessors, in state s_0. Rules applied in transition $s_0 \Rightarrow s_1$ rewrite received μ_i's into a's and send $\overline{\mu}_i$'s to all σ_i's neighbors. Additionally, for general σ_g, these rules produce three copies of b and one copy of d. Rules applied in transition $s_1 \Rightarrow s_2$ rewrite $\overline{\mu}_i$'s received from peers into r's. Rules applied in transitions $s_2 \Rightarrow s_3$ or $s_2 \Rightarrow s_4$ rewrite μ_i's received from successors into v's.

Propositions 3, 4 and 5, indicate the number of broadcast objects respectively received from predecessors, peers and successors. Cells complete this phase in the number of steps indicated by Proposition 6. Proposition 7 characterizes the terminals.

Proposition 3 *Cell $\sigma_i \neq \sigma_g$ receives k copies of μ_i from its predecessors, in step $i\mathsf{depth}_g(i)$ and sends k copies of $\overline{\mu}_i$ to*

each of its successors, in step $\mathsf{depth}_g(i) + 1$, *where* $k = \mathsf{paths}_g(i)$.

Proof Proof by induction, on $m = \mathsf{depth}_g(i) \geq 1$. In step 1, the general sends o to all its neighbors. Hence, in step 1, each cell σ_i in depth 1 receives $o = \mu_i$. Then, in step 2, by state transition $s_0 \Rightarrow s_1$, σ_i sends $x = \overline{\mu}_i$ to each of its successors.

Assume that the induction hypothesis holds for each cell σ_j at depth m. Consider cell σ_i at $\mathsf{depth}_g(i) = m + 1 = \mathsf{depth}_g(j) + 1$. By induction hypothesis, in step $\mathsf{depth}_g(j) + 1$, each $\sigma_j \in \mathsf{Pred}_g(i)$ sends $\mathsf{paths}_g(j)$ copies of $\overline{\mu}_j$ to all its neighbors. Thus, in step $\mathsf{depth}_g(j) + 1 = \mathsf{depth}_g(i), \sigma_i$ receives $\sum_{\sigma_j \in \mathsf{Pred}_g(i)} \mathsf{paths}_g(j) = \mathsf{paths}_g(i)$ copies of $\overline{\mu}_j = \mu_i$. In step $\mathsf{depth}_g(i) + 1$, by state transition $s_0 \Rightarrow s_1$, σ_i sends $\mathsf{paths}_g(i)$ copies of $\overline{\mu}_i$ to all its neighbors. □

Proposition 4 *Cell σ receives i k copies of $\overline{\mu}_i$ from its peers, in step* $\mathsf{depth}_g(i) + 1$, *where* $k = \sum_{\sigma_j \in \mathsf{Peer}_g(i)} \mathsf{paths}_g(j)$.

Proof From Proposition 3, each cell $\sigma_j \in \mathsf{Peer}_g(i)$ sends $\mathsf{paths}_g(j)$ copies of $\overline{\mu}_j$ to all its neighbors in step $\mathsf{depth}_g(j) + 1$. Hence, σ_i receives $\mathsf{paths}_g(i)$ copies of $\overline{\mu}_j = \overline{\mu}_i$ from σ_j in step $\mathsf{depth}_g(j) + 1 = \mathsf{depth}_g(i) + 1$. Thus, in step $\mathsf{depth}_g(i) + 1, \sigma_i$ receives $\overline{\mu}_i^k$, where $k = \sum_{\sigma_j \in \mathsf{Peer}_g(i)} \mathsf{paths}_g(j)$. □

Proposition 5 *Cell σ_i receives k copies of μ_i from its successors, in step* $\mathsf{depth}_g(i) + 2$, *where* $k = \sum_{\sigma_j \in \mathsf{Succ}_g(i)} \mathsf{paths}_g(j)$.

Proof From Proposition 3, each cell $\sigma_j \in \mathsf{Succ}_g(i)$ sends $\mathsf{paths}_g(j)$ copies of $\overline{\mu}_j$ to all its neighbors in step $\mathsf{depth}_g(j) + 1$. Hence, σ_i receives $\mathsf{paths}_g(i)$ copies of $\overline{\mu}_j = \mu_i$ from σ_j in step $\mathsf{depth}_g(j) + 1 = \mathsf{depth}_g(i) + 2$. Thus, in step $\mathsf{depth}_g(i) + 2, \sigma_i$ receives μ_i^k, where $k = \sum_{\sigma_j \in \mathsf{Succ}_g(i)} \mathsf{paths}_g(j)$. □

Proposition 6 *Cell σ_i takes* $\mathsf{depth}_g(i) + 3$ *steps in Phase I.*

Proof From Proposition 3, cell σ_i receives $\mathsf{paths}_g(i)$ copies of μ_i from its predecessors in step $\mathsf{depth}_g(i)$. Cell σ_i takes three state transitions, where each state transition takes one step. Hence, σ_i takes $\mathsf{depth}_g(i) + 3$ steps. □

Proposition 7 *A cell σ_i, which does not receive any broadcast objects in step* $\mathsf{depth}_g(i) + 2$ *is terminal.*

Proof Follows from Proposition 5. □

3.2 FSSP Phase II

Phase II is a convergecast initiated by the terminals, relayed from *successors* to *predecessors*. using $\mu_i, \overline{\mu}_i$ as *convergecast objects* (identical to the broadcast objects

sent in Phase I). Intuitively, cell σ_i expects to receive "useful" objects μ_i, first from its successors, then from its predecessors, "noise" objects $\overline{\mu}_i$, from its peers, and sends $\overline{\mu}_i$ objects, to all its neighbors. At the end of this phase, the general stops its counter (used to compute its eccentricity).

Phase II (Convergecast from terminals)

Precondition: Phase II starts with the postcondition of Phase I.

Postcondition: Phase II ends when the general σ_g enters state s_6. At the end of Phase II, the configuration of cell $\sigma_i \in K$ is (Q, s_i, w_i, R), where

- $s_i = s_6$.
- $|w_i|_a = \mathsf{paths}_g(i)$, is the number of shortest paths from σ_g to σ_i.
- $|w_i|_b = \mathsf{e}_g + 2$, if $\sigma_i = \sigma_g$.

Description: Immediately after the first broadcast (Phase I), each terminal cell σ_i initiates a convergecast, by sending convergecast objects $\overline{\mu}_i$, to all its predecessors. A non-terminal cell σ_i expects $|w_i|_v$ copies of μ_i from its successors. After receiving this expected number, cell σ_i sends $|w_i|_a$ copies of $\overline{\mu}_i$, to each of its neighbors.

Additionally, when $i = c$, the general cell σ_g stops its counter (which was started in Phase I). The resulting general counter is $2\mathsf{e}_g + 6$, i.e. the round-trip time from σ_g to one of its farthest terminal (plus some overheads). Thus, at the end of Phase II, using this counter, σ_g can determine its eccentricity.

Proposition 8 *A non-terminal cell σ_i receives u copies of μ_i from its successors and sends k copies of $\overline{\mu}_i$ to each of its predecessors, where* $k = \mathsf{paths}_g(i)$ *and* $u = \sum_{\sigma_j \in \mathsf{Succ}_g(i)} \mathsf{paths}_g(j)$.

Proof Proof by induction, on cell σ_i's height, $m = \mathsf{height}_g(i)$.

In the base case, when $m = 0$, cell σ_i is a terminal cell. Clearly, cell σ_i has zero successors and therefore receives zero copies μ_i. By rule 3.4 or 4.4, cell σ_i sends $\mathsf{paths}_g(i)$ copies of $\overline{\mu}_i$ to all its neighbors (one copy of $\overline{\mu}_i$ for each copy of a).

Assume that the induction hypothesis holds for each cell σ_k with height $\mathsf{height}_g(k) \leq m$, and consider cell σ_i at height $\mathsf{height}_g(i) = m + 1$. In this case case, cell σ_i is non-terminal. Each cell $\sigma_j \in \mathsf{Succ}_g(i)$ has height $\mathsf{height}_g(j) \leq m$, thus it satisfies the induction hypothesis and sends out, to σ_i, $\mathsf{paths}_g(j)$ copies of $\overline{\mu}_j$. In total, cell σ_i receives, from all its successors, $u = \sum_{\sigma_j \in \mathsf{Succ}_g(i)} \mathsf{paths}_g(j)$ copies of μ_i. Next, by rule 3.2 or 4.2, σ_i consumes its u copies of μ_i and v (one copy of μ_i for each copy of v). After consuming all its v's, by rule 3.4 or 4.4, cell σ_i sends $k = \mathsf{paths}_g(i)$ copies of $\overline{\mu}_i$ to each of its neighbors (one copy of $\overline{\mu}_i$ for each copy of a). □

Proposition 9 *Cell σ_i takes* $2\mathsf{e}_g - \mathsf{depth}_g(i) + 3$ *steps in Phase II.*

Proof The general σ_g starts Phase II after completing its Phase I transitions ($s_0 \Rightarrow s_1 \Rightarrow s_2 \Rightarrow s_3$) i.e. three steps after sending its Phase I broadcast object o. The broadcast needs e_g steps to reach σ_t, one of the farthest terminal (with respect to σ_g). Cell σ_t needs three more steps to decide that it is terminal. Further e_g steps are needed for the convergecast to reach back σ_g. Finally, σ_g needs three more overhead steps to reach state s_6. Thus, σ_g needs a total of $2e_g + 3$ steps to reach Phase II's final state. Each cell $\sigma_j \neq \sigma_g$ starts Phase II, with a delay of $\mathsf{depth}_g(j)$ steps, after σ_g starts Phase II. Therefore, σ_i takes $2e_g - \mathsf{depth}_g(i) + 3$ steps in Phase II. $\square$

3.3 FSSP Phase III

Phase III is a second broadcast initiated by the general, relayed from *predecessors* to *successors*. Initially, the general sends $e_g + 1$ copies of object b to each of its successors. After receiving a number of b's (which depends on its depth and on its number of shortest paths from the general), each cell σ_i sends $e_g + 1 - \mathsf{depth}_g(i)$ copies of b to each of its successors.

Phase III (Second broadcast from the general)
 Precondition: Phase III starts with the postcondition of Phase II.
 Postcondition: At the end of Phase III, the configuration of cell $\sigma_i \in K$ is (Q, s_i, w_i, R), where

- $s_i = s_8$.
- $|w_i|_a = \mathsf{paths}_g(i)$, is the number of shortest paths from σ_g to σ_i.
- $|w_i|_g = (e_g + 1 - \mathsf{depth}_g(i))\mathsf{paths}_g(i)$, where $e_g + 1 - \mathsf{depth}_g(i)$ is the countdown counter used in Phase IV.

Description: In Phase III, each cell σ_i makes three transitions, $s_6 \Rightarrow s_7 \Rightarrow s_7 \Rightarrow s_8$, where each transition takes one step. The general σ_g initiates a second broadcast by sending $e_g + 1$ copies of b to all its successors. hop count is represented by the multiplicity of the second broadcast object. A cell $\sigma_i \neq \sigma_g$ receives b's simultaneously via all shortest paths from σ_g to σ_i. Specifically, σ_i receives $(e_g + 2 - \mathsf{depth}_g(i))\mathsf{paths}_g(i)$ copies of b, then transition $s_6 \Rightarrow s_7$ removes $\mathsf{paths}_g(i)$ copies of b and sends remaining $(e_g + 1 - \mathsf{depth}_g(i))\mathsf{paths}_g(i)$ copies of b to all its neighbors. During transitions $s_7 \Rightarrow s_7 \Rightarrow s_8$, σ_i accumulates and removes superfluous b's from its peers and successors.

Proposition 10 *Cell σ_i receives p copies of b from its predecessors, where $p = (e_g + 2 - \mathsf{depth}_g(i))\mathsf{paths}_g(i)$, and sends q copies of b to each of its successor, where $q = (e_g + 1 - \mathsf{depth}_g(i))\mathsf{paths}_g(i)$.*

Proof Proof by induction, on $m = \mathsf{depth}_g(i) \geq 1$. First, the general sends b^y to all its neighbors, where $y = e_g + 1$. Thus, each cell σ_i at depth 1 receives b^y, where $y = e_g + 1 = (e_g + 2 - \mathsf{depth}_g(i))\mathsf{paths}_g(i)$ (since $\mathsf{paths}_g(i) = 1$). Also, by state transition $s_6 \Rightarrow s_7$, sends b^z to each of its successors, where $z = e_g = (e_g + 1 - \mathsf{depth}_g(i))\mathsf{paths}_g(i)$.

Assume that the induction hypothesis holds for each cell σ_j at depth m. Consider cell σ_i at $\mathsf{depth}_g(i) = m + 1 = \mathsf{depth}_g(j) + 1$. By induction hypothesis, each $\sigma_j \in \mathsf{Pred}_g(i)$ sends $(e_g + 1 - \mathsf{depth}_g(j))\mathsf{paths}_g(j)$ copies of b. In total, σ_i receives $u = \sum_{\sigma_j \in \mathsf{Pred}_g(i)}(e_g + 1 - m)\mathsf{paths}_g(j)$ copies of b. Unrolling this expression, we obtain $u = (e_g + 1 - m)\sum_{\sigma_j \in \mathsf{Pred}_g(i)}\mathsf{paths}_g(j) = (e_g + 1 - m)\mathsf{paths}_g(i) = (e_g + 1 - (\mathsf{depth}_g(i) - 1))\mathsf{paths}_g(i) = (e_g + 2 - \mathsf{depth}_g(i))\mathsf{paths}_g(i)$. By state transition $s_6 \Rightarrow s_7$, σ_i removes $\mathsf{paths}_g(i)$ copies of b and sends the remaining copies of b, i.e. $(e_g + 1 - \mathsf{depth}_g(i))\mathsf{paths}_g(i)$ copies, to all its successors. $\square$

Proposition 11 *Cell σ_i takes $\mathsf{depth}_g(i) + 3$ steps in Phase III.*

Proof Phase III starts when the general sends the second broadcast object(s). Similar to Phase I, each cell σ_i receives its first broadcast object(s) $\mathsf{depth}_g(i)$ steps after the general sends the second broadcast object(s). Cell σ_i then takes three state transitions as described above. Thus, σ_i takes $\mathsf{depth}_g(i) + 3$ steps in Phase III. $\square$

3.4 FSSP Phase IV

Phase IV is the countdown towards firing state. This phase uses the countdown counters $e_g + 1 - \mathsf{depth}_g(i)$ determined in Phase III.

Phase IV (Countdown towards synchronization)
 Precondition: Phase IV starts with the postcondition of Phase III.
 Postcondition: At the end of Phase IV, the configuration of cell $\sigma_i \in K$ is (Q, s_i, w_i, R), where $s_i = s_9$ (the firing state) and $w_i = \lambda$.
 Description: Immediately after receiving the second broadcast (Phase III), each cell initiates a countdown (Phase IV). To achieve synchronization, each cell σ_i needs to idle for $e_g - \mathsf{depth}_g(i) + 1$ steps, before entering the firing state. Specifically, during state transition $s_8 \Rightarrow s_8$, σ_i removes $\mathsf{paths}_g(i)$ copies of c in each step, such that after $e_g - \mathsf{depth}_g(i) + 1$ steps, all c's disappear. When there are no c's, by state transition $s_8 \Rightarrow s_9$, σ_i enters the firing state s_9.

Proposition 12 *Cell σ_i takes $e_g - \mathsf{depth}_g(i) + 2$ steps in Phase IV.*

Proof Each cell σ_i removes $\mathsf{paths}_g(i)$ copies of the second broadcast objects. Hence, σ_i takes $\mathsf{e}_g - \mathsf{depth}_g(i) + 1$ steps to remove all the broadcast objects. Cell σ_i takes one step to enter the firing state. Thus, σ_i takes $\mathsf{e}_g - \mathsf{depth}_g(i) + 2$ steps in Phase IV. □

Theorem 13 *The running time of our FSSP solution is $3\mathsf{e}_g + 11$ steps, where e_g is the eccentricity of the general σ_g.*

Proof The result is obtained by summing the individual running times of the four phases, as given by Propositions 6, 9, 11 and 12: $(\mathsf{depth}_g(i) + 3) + (2\mathsf{e}_g - \mathsf{depth}_g(i) + 3) + (\mathsf{depth}_g(i) + 3) + (\mathsf{e}_g - \mathsf{depth}_g(i) + 2) = 3\mathsf{e}_g + 11$. □

3.5 FSSP complete example

In Table 1, we present the traces of the FSSP algorithm for the virtual structure of the simple P module shown in Fig. 1. Note, for convenience, the phase boundaries are indicated in bold in Table 1.

3.6 FSSP optimality

Finally, we show that our algorithm is optimal, up to an additive constant, if we assume that all cells, except the general, start *empty* and in a *quiescent* state s_0 (no rules are applicable until some objects appear).

Theorem 14 *Any algorithm that solves the FSSP problem must use at least $3\mathsf{e}_g + 3$ steps, on an infinity of systems defined as simple P modules (where e_g is the eccentricity of the general σ_g).*

Proof The proof is by contradiction. Consider an algorithm Φ, that solves the FSSP problem and the following two simple P modules (see Fig. 2):

1. Π_k, with an underlying structure of a simple path $\sigma_0, \sigma_1, \ldots, \sigma_{2k}$, with the general located at cell σ_k (i.e. the middle). Clearly, this general's eccentricity is $e_k = k$.
2. Π'_k, with an underlying structure of a simple path $\sigma'_0, \sigma'_1, \ldots, \sigma'_{4k+2}$, with the general located at cell σ'_k (Π'_k can be thought as a clone of Π_k, extended on its right, with $2k + 2$ more cells).

First, a straightforward induction shows that cells σ_0 and σ'_0 need at least $3k + 3$ steps to reach different states. Specifically, we show the converse, that, after $t \leq 3k + 2$ steps, cells σ_0 and σ'_0 still have identical states.

- Initially, all Π_k and Π'_k cells are in their initial quiescent state, say s_0, and all cells are empty, except the two generals, σ_k and σ'_k.
- At step 1, only these two generals evolve. Other quiescent cells cannot evolve, until they receive objects from their evolving neighbors.

Table 1 The FSSP trace of the simple P module shown in Fig. 1, where $\sigma_g = \sigma_1$

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
0	$s_0\, a$	s_0	s_0	s_0	s_0	s_0
1	$s_1\, ab^3de$	$s_0\, o$	$s_0\, o$	s_0	s_0	s_0
2	$s_2\, ab^3dex^2$	$s_1\, ax$	$s_1\, ax$	$s_0\, x^2$	$s_0\, x$	s_0
3	$\boldsymbol{s_3\, ab^3dv^2}$	$s_2\, ao^2r$	$s_2\, ao^3r$	$s_1\, a^2e^2o$	$s_1\, aeo^2$	$s_0\, o$
4	$s_3\, ab^4dv^2$	$\boldsymbol{s_4\, arv^2}$	$\boldsymbol{s_4\, arv^3}$	$s_2\, a^2e^2r$	$s_2\, aer^2x$	$s_1\, a$
5	$s_3\, ab^5dv^2$	$s_4\, arv^2$	$s_4\, arv^3$	$\boldsymbol{s_3\, a^2r}$	$\boldsymbol{s_3\, ar^2v}$	$s_2\, a$
6	$s_3\, ab^6dv^2$	$s_4\, ao^2rv^2$	$s_4\, ao^2rv^3$	$s_3\, a^2h^2r$	$s_3\, ao^2r^2v$	$\boldsymbol{s_4\, a}$
7	$s_3\, ab^7dv^2x$	$s_4\, ahr$	$s_4\, arvx$	$s_5\, a^2r^3x$	$s_3\, ao^2r^2vx$	$s_4\, ah$
8	$s_3\, ab^8dv$	$s_5\, ar^2$	$s_4\, aorvx$	$s_5\, a^2or^2$	$s_3\, aho^2r^2$	$s_5\, aor$
9	$s_3\, ab^9dvx$	$s_5\, ar^2x$	$s_4\, ahrx$	$s_5\, a^2rx$	$s_5\, ao^2r^3x$	$s_5\, a$
10	$s_3\, ab^{10}dh$	$s_5\, aor$	$s_5\, aor^2x$	$s_5\, a^2$	$s_5\, a$	$s_6\, a$
11	$s_5\, ab^{10}dr$	$s_5\, a$	$s_5\, a$	$s_6\, a^2$	$s_6\, a$	$s_6\, a$
12	$\boldsymbol{s_6\, ab^5}$	$\boldsymbol{s_6\, a}$	$\boldsymbol{s_6\, a}$	$\boldsymbol{s_6\, a^2}$	$\boldsymbol{s_6\, a}$	$\boldsymbol{s_6\, a}$
13	$s_7\, ac^4h$	$s_6\, ab^4$	$s_6\, ab^4$	$s_6\, a^2$	$s_6\, a$	$s_6\, a$
14	$s_7\, ab^6c^4$	$s_7\, ab^3c^3h$	$s_7\, ab^3c^3h$	$s_6\, a^2b^6$	$s_6\, ab^3$	$s_6\, a$
15	$\boldsymbol{s_8\, ac^4}$	$s_7\, ab^7c^3$	$s_7\, ab^9c^3$	$s_7\, a^2b^2c^4h^2$	$s_7\, ab^4c^2h$	$s_6\, ab^2$
16	$s_8\, ac^3$	$\boldsymbol{s_8\, ac^3}$	$\boldsymbol{s_8\, ac^3}$	$s_7\, a^2b^2c^4$	$s_7\, ab^5c^2$	$s_7\, ach$
17	$s_8\, ac^2$	$s_8\, ac^2$	$s_8\, ac^2$	$\boldsymbol{s_8\, a^2c^4}$	$\boldsymbol{s_8\, ac^2}$	$s_7\, ac$
18	$s_8\, ac$	$s_8\, ac$	$s_8\, ac$	$s_8\, a^2c^2$	$s_8\, ac$	$\boldsymbol{s_8\, ac}$
19	$s_8\, a$	$s_8\, a$	$s_8\, a$	$s_8\, a^2$	$s_8\, a$	$s_8\, a$
20	s_9	s_9	s_9	s_9	s_9	s_9

Fig. 2 The two P modules used in Theorem 14
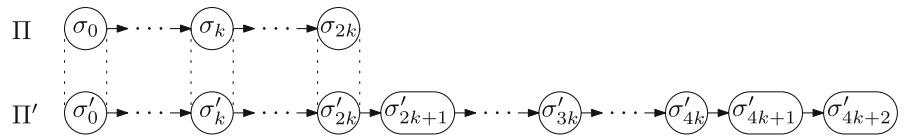
- Thus, after u steps, where $0 \le u \le k$, cells $\sigma_{k-u}, \ldots, \sigma_{k+u}$ have, respectively, identical contents and states as their pair cells $\sigma'_{k-u}, \ldots, \sigma'_{k+u}$; all remaining Π_k and Π'_k cells are still empty and quiescent.
- Then, after a total of $k+1$ steps, cells $\sigma_0, \sigma_1, \ldots, \sigma_{2k}$ have, respectively, identical contents and states as their pair cells $\sigma'_0, \sigma'_1, \ldots, \sigma'_{2k}$; while cells $\sigma'_{2k+2}, \sigma'_{2k+3}, \ldots, \sigma'_{4k+2}$ are still empty and quiescent. However, at this stage, cell σ'_{2k+1} can be non-empty and send, in the next step, objects to cell σ'_{2k}.
- Next, after a total of v steps, where $k + 1 \le v \le 3k + 1$, cells $\sigma_0, \sigma_1, \ldots, \sigma_{3k+1-v}$ have, respectively, identical contents and states as their pair cells $\sigma'_0, \sigma'_1, \ldots, \sigma'_{3k+1-v}$.
- Thus, after a total of up to $3k + 1$ steps (inclusive), cells σ_0 and σ'_0 still have identical contents and states.
- Finally, after a total of up to $3k + 2$ steps (inclusive), cells σ_0 and σ'_0 still have identical states (contents could now be different, but this is not relevant here).

We now show that algorithm Φ needs at least $3k + 3$ steps on Π_k. Assume, by contradiction, that Π_k can be synchronized in $t \le 3k + 2$ steps. In this case, after t steps, cells σ_0 and σ'_0 are in the same state, say s_t, which is also the firing state. Therefore, if algorithm Φ is correct, then Π'_k must also synchronize in t steps, and its rightmost cell σ'_{4k+2} must also reach the same firing state, s_t.

However, a similar induction shows that, after $t \le 3k + 2$ steps, Π'_k cell σ'_{4k+2} is still in its initial quiescent state, s_0, which is different from the firing state s_t. This contradiction completes the proof. □

Remark 15 The lower bound $3e_g + 3$ indicated by Theorem 14 is for our current definition of simple P modules. With respect to its proof, at least two steps are required for cells σ_{2k} and σ'_{2k} to differentiate between their right context, i.e. to determine if they are terminal (σ_{2k}) or not (σ'_{2k}). Thus, the lower bound can be improved (e.g., to $3e_g + 1$), if we enable cells to detect faster (in fewer steps) if they are terminals or not; this could be achieved if each cell would be equipped with a predefined counter, indicating the number of its local structural connections.

4 Conclusion

We have proposed an improved deterministic FSSP solution in the framework of P systems, expressed using simple P modules, with the running time of $3e_g + 11$, where e_g is

the eccentricity of the general cell. We have also shown that the multiplier is optimal, up to an additive constant.

We end this paper with two open problems. One problem asks if there are adaptive FSSP algorithms which can run faster in many scenarios, by exploiting specific structural layouts, without otherwise incurring any runtime penalty. The other problem is related to the type of communication channels: are there FSSP solutions for a P system with strongly connected underlying membrane structure and *simplex* communication capability?

References

Alhazov A, Margenstern M, Verlan S (2008) Fast synchronization in P systems. In: Corne DW, Frisco P, Păun G, Rozenberg G, Salomaa A (eds) Workshop on membrane computing, volume 5391 of Lecture notes in computer science. Springer, New York, pp 118–128

Balzer R (1967) An 8-state minimal time solution to the firing squad synchronization problem. Inf Control 10(1):22–42

Bernardini F, Gheorghe M, Margenstern M, Verlan S (2008) How to synchronize the activity of all components of a P system? Int J Found Comput Sci 19(5):1183–1198

Dinneen MJ, Kim Y-B, Nicolescu R (2009) New solutions to the firing squad synchronization problems for neural and hyperdag P systems. Electron Proc Theor Comput Sci 11:107–122

Dinneen MJ, Kim Y-B, Nicolescu R (2010a) Edge- and node-disjoint paths in P systems. Electron Proc Theor Comput Sci 40:121–141

Dinneen MJ, Kim Y-B, Nicolescu R (2010b) P systems and the Byzantine agreement. J Log Algebr Program 79(6):334–349

Dinneen MJ, Kim Y-B, Nicolescu R (2010c) Synchronization in P modules. In: Calude CS, Hagiya M, Morita K, Rozenberg G, Timmis J (eds) Unconventional computation, volume 6079 of Lecture notes in computer science. Springer, Berlin, pp 32–44

Freeman RL (2005) Fundamentals of telecommunications, 2nd edn. Wiley-IEEE Press, New York

Goto E (1962) A minimal time solution of the firing squad problem. Course notes for applied mathematics, vol 298. Harvard University, Cambridge, pp 52–59

Grefenstette JJ (1983) Network structure and the firing squad synchronization problem. J Comput Syst Sci 26(1):139–152

Humphrey TC (2005) Cell cycle control: mechanisms and protocols. Humana Press, Totowa

Imai K, Morita K, Sako K (2002) Firing squad synchronization problem in number-conserving cellular automata. Fundam Inform 52(1–3):133–141

Kobayashi K and Goldstein D (2005) On formulations of firing squad synchronization problems. In: Calude CS, Dinneen MJ, Păun G, Pérez-Jiménez MJ, Rozenberg G (eds) Unconventional computation, 4th international conference, UC 2005, Sevilla, Spain, October 3–7, 2005, proceedings, volume 3699 of Lecture notes in computer science. Springer, New York, pp 157–168

Mazoyer J (1987) A six-state minimal time solution to the firing squad synchronization problem. Theor Comput Sci 50:183–238

Moore EF (1964) The firing squad synchronization problem. In: Moore EF (ed) Sequential machines, selected papers. Addison-Wesley, Reading, pp 213–214

Nishitani Y, Honda N (1981) The firing squad synchronization problem for graphs. Theor Comput Sci 14:39–61

Păun G (2006) Introduction to membrane computing. In: Ciobanu G, Pérez-Jiménez MJ, Păun G (eds) Applications of membrane computing. Natural computing series. Springer, New York, pp 1–42

Schmid H, Worsch T (2004) The firing squad synchronization problem with many generals for one-dimensional CA. In: Lévy J-J, Mayr EW, Mitchell JC (eds) IFIP TCS. Kluwer, Dordrecht, pp 111–124

Szwerinski H (1982) Time-optimal solution of the firing-squad-synchronization-problem for n-dimensional rectangles with the general at an arbitrary position. Theor Comput Sci 19(3):305–320

Umeo H, Hisaoka M, Akiguchi S (2005) A twelve-state optimum-time synchronization algorithm for two-dimensional rectangular cellular arrays. In: Calude CS, Dinneen MJ, Păun G, Pérez-Jiménez MJ, Rozenberg G (eds) Unconventional computation, 4th international conference, UC 2005, Sevilla, Spain, October 3–7, 2005, proceedings, volume 3699 of Lecture notes in computer science. Springer, New York, pp 214–223

Waksman A (1966) An optimum solution to the firing squad synchronization problem. Inf Control 9(1):66–78