New Solutions to the Firing Squad Synchronization Problem for Neural and Hyperdag P Systems

Michael J. Dinneen Yun-Bum Kim
Radu Nicolescu

Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
{mjd,radu}@cs.auckland.ac.nz tkim021@aucklanduni.ac.nz

We propose a unified solution to an open question: the Firing Squad Synchronization Problem (FSSP) for hyperdag P systems and for neural P systems with symmetric communication channels. Our solution takes $6e+7$ steps, where e is the eccentricity of the commander cell of the dag or digraph underlying these extended P systems. When restricted to tree-based P systems, unlike previously proposed FSSP solutions, we allow the commander node to be any node of the underlying tree. Also our solution works without membrane polarization techniques or conditional rules, but requires states, as used in hyperdag and neural P systems.

Keywords: P systems, neural P systems, hyperdag P systems, synchronization, cellular automata.

1 Introduction

The *Firing Squad Synchronization Problem* (FSSP) [5, 6, 9, 12, 14, 15] is one of the best studied problems for cellular automata. The problem involves finding a cellular automaton, such that, after a command is given, all the cells, after some finite time, enter a designated *firing* state *simultaneously* and *for the first time*. Several variants of FSSP [12, 14], have been proposed and studied. Studies of these variations mainly focus on finding a solution with as few states as possible and possibly running in optimum time.

There are several applications that require synchronization. We list just three 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 [4]. Once synchronized, monitoring the progression from one phase to another allows us to calculate the timing of specific cells' phases. A second example relates to operating systems [13], where process synchronization is the coordination of simultaneous threads or processes to complete a task without race conditions. Finally, in telecommunication networks [3], 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* [2] provided a non-deterministic 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* [1] 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*, algorithmic techniques with a high potential for parallelism.

This is a preliminary version of a paper
that will appear in Electronic Proceedings
in Theoretical Computer Science.

In this paper, we continue the study of FSSP in the framework of P systems, by providing solutions for hyperdag P systems [7] and for neural P systems [10] with symmetric communication channels. We propose deterministic solutions to a variant of FSSP [14], in which there is a single commander, at an arbitrary position. We further generalize this problem by synchronizing a subset of cells of the considered hyperdag or neural P system.

Our first and "trivial" solution uses simple rules, but requires structural extensions, that may or may not be allowed. However, the simplicity of this solution supports our hypothesis that basing P systems on dag, instead of tree, structures allows more natural expressions of some fundamental distributed algorithms [7, 8].

Our second solution is more traditional and does not require structural extensions, but is substantially more complex. It does not require polarizations or conditional rules, but requires *states*, as defined for hyperdag and neural P systems and also combines a *breadth-first-search*, a *broadcast* and a *convergecast*. This solution takes $6e+7$ steps, where e is the eccentricity of the commander cell of the underlying dag or digraph. When restricted to P systems, our algorithm takes more steps than Alhazov *et al* [1], if the commander is the root node, but comparable to this, when the commander is a central node of an unbalanced rooted tree.

Section 2 provides background definitions and introduces the families of P systems considered for synchronization. Next, in Section 3, we cite the communication models for hyperdag P systems and neural P systems, and the transition and rewrite rules available for solving the FSSP. Our two FSSP solutions are described in Sections 4 and 5. Finally, after illustrating the evolution of our FSSP algorithm in Section 5, we end with some concluding remarks.

2 Preliminary

A (binary) *relation* R over two sets X and Y is a subset of their Cartesian product, $R \subseteq X \times Y$. For $A \subseteq X$ and $B \subseteq Y$, we set $R(A) = \{y \in Y \mid \exists x \in A, (x,y) \in R\}$, $R^{-1}(B) = \{x \in X \mid \exists y \in B, (x,y) \in R\}$.

A *digraph* (directed graph) G is a pair (X,A), where X is a finite set of elements called *nodes* (or *vertices*), and A is a binary relation $A \subseteq X \times X$, of elements called *arcs*. A length $n-1$ *path* is a sequence of n distinct nodes $x_1,\dots,x_n$, such that $\{(x_1,x_2),\dots,(x_{n-1},x_n)\} \subseteq A$. A *cycle* is a path $x_1,\dots,x_n$, where $n \geq 1$ and $(x_n,x_1) \in A$. A digraph is *symmetric* if its relation A is symmetric, i.e., $(x_1,x_2) \in A \Leftrightarrow (x_2,x_1) \in A$. By default, all digraphs considered in this paper, and all structures from digraphs (dag, rooted tree, see below) will be *weakly connected*, i.e., each pair of nodes is connected via a chain of arcs, where the arc direction is not relevant.

A *dag* (directed acyclic graph) is a digraph (X,A) without cycles. For $x \in X$, $A^{-1}(x)$ are x's *parents*, $A(x)$ are x's *children*, and $A(A^{-1}(x)) \backslash \{x\}$ are x's *siblings*.

A *rooted tree* is a *special case of dag*, where each node has exactly one parent, except a distinguished node, called *root*, which has none.

Throughout this paper, we will use the term *graph* to denote a *symmetric digraph* and *tree* to denote a *rooted tree*.

For a given tree, dag or digraph, we define e_c, the *eccentricity* of a node c, as the maximum length of a shortest path between c and any other reachable node in the corresponding structure.

For a tree, the set of *neighbors* of a node x, $Neighbor(x)$, is the union of x's parent and x's children. For a dag δ and node x, we define $Neighbor(x) = \delta(x) \cup \delta^{-1}(x) \cup \delta(\delta^{-1}(x)) \backslash \{x\}$, if we want to include the siblings, or, $Neighbor(x) = \delta(x) \cup \delta^{-1}(x)$, otherwise. For a graph $G = (X,A)$, we set $Neighbor(x) = A(x) = \{y \mid (x,y) \in A\}$. Note that, as defined, $Neighbor$ is always a symmetric relation.

A special node c of a structure will be designated as the *commander*. For a given commander c, we define the *level* of a node x, $level_c(x) \in \mathbb{N}$, as the length of a shortest path between the c and x, over the $Neighbor$ relation.

For a given tree, dag or digraph and commander c, for nodes x and y, if $y \in Neighbor(x)$ and $level_c(y) = level_c(x) + 1$, then x is a *predecessor* of y and y is *successor* of x. Similarly, a node z is a *peer* of a node x, if $z \in Neighbor(x)$ and $level_c(z) = level_c(x)$. Note that, for a given 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*. We define $maxlevel_c = max\{level_c(x) \mid x \in X\}$ and we note $e_c = maxlevel_c$. A *level-preserving path* from c to a node y is a sequence $x_0, \dots, x_k$, such that $x_0 = c, x_k = y, x_i \in Neighbor(x_{i-1}), level_c(x_i) = i, 1 \le i \le k$. We further define $count_c(y)$ as the number of distinct level-preserving paths from c to y.

The level of a node and number of level-preserving paths to it can be determined by a standard breadth-first-search, as shown in Algorithm 1. Intuitively, if the original structure is a tree, this algorithm will "reset" the root at another node in the tree.

Algorithm 1 (Determine levels and count level-preserving paths)

- INPUT: A tree, dag or digraph, with nodes $\{1, \dots, n\}$ and a commander $c \in \{1, \dots, n\}$.
- OUTPUT: The arrays $level_c[]$ and $count_c[]$ of shortest distances and number of level-preserving paths from c to each node in the structure, over the $Neighbor$ relation.

```
array $level_c[1,\dots,n] = [-1,\dots,-1]$; $count_c[1,\dots,n] = [0,\dots,0]$
queue $Q = ()$
$Q \Leftarrow c$
$level_c[c] = 0$; $count_c[c] = 1$
while $Q \neq ()$ do
    $x \Leftarrow Q$
    for each $y \in Neighbor(x)$ do
        if $level_c[y] = -1$ then
            $Q \Leftarrow y$
            $level_c[y] = level_c[x] + 1$
        if $level_c[y] = level_c[x] + 1$ then
            $count_c[y] = count_c[y] + count_c[x]$
return $level_c$
```

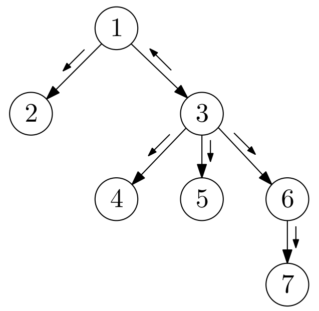
Example 1. Figures 1, 2 and 3 show $level_c$, *predecessors*, *successors*, *peers* and $count_c$, for a tree, a dag and a digraph structure, respectively. Small side-arrows indicate the arcs traversed while computing the levels, over the induced $Neighbor$ relation, as described in Algorithm 1.

3 P Systems and the Firing Squad Synchronization Problem

In this section, we briefly recall several fundamental definitions for P systems and describe a P systems version of the Firing Squad Synchronization Problem (FSSP).

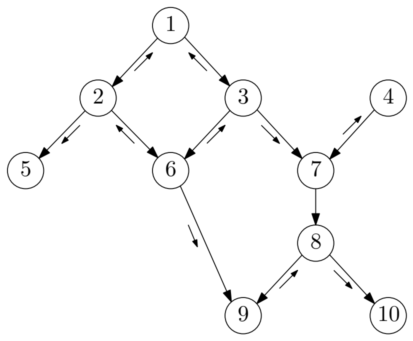
For the definitions of tree-based P systems, see Păun [10]. Here we reproduce the basic definitions of dag-based hyperdag P systems, from our previous work [7] and digraph-based neural P systems, from Păun [10].

Definition 2 (Hyperdag P systems [7]) A *hyperdag P system* (of order n), in short an *hP system*, is a system $\Pi_h = (O, \sigma_1, \dots, \sigma_n, \delta, I_{out})$, where:



Node	$level_c$	*predecessors*	*successors*	*peers*	$count_c$
1	1	3	2	–	1
2	2	1	–	–	1
3	0	–	1,4,5,6	–	1
4	1	3	–	–	1
5	1	3	–	–	1
6	1	3	7	–	1
7	2	6	–	–	1

Figure 1: Left: a tree (taken from Bernardini *et al* [2]), with commander $c = 3$, $e_3 = 2$; Right: table with node levels, predecessors, successors, peers and $count_c$'s.



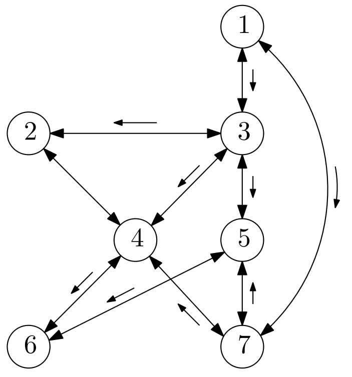
Node	$level_c$	*predecessors*	*successors*	*peers*	$count_c$
1	2	2,3	–	–	2
2	1	6	1,5	–	1
3	1	6	1,7	–	1
4	3	7	–	–	1
5	2	2	–	–	1
6	0	–	2,3,9	–	1
7	2	3	4	8	1
8	2	9	10	7	1
9	1	6	8	–	1
10	3	8	–	–	1

Figure 2: Left: a dag with commander $c = 6$, $e_6 = 3$ (siblings excluded); Right: table with node levels, predecessors, successors, peers and $count_c$'s.

1. O is an ordered finite non-empty alphabet of *objects*;
2. $\sigma_1, \ldots, \sigma_n$ are *cells*, of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i)$, $1 \le i \le n$, where:
   - Q_i is a finite set (of *states*),
   - $s_{i,0} \in Q_i$ is the *initial state*,
   - $w_{i,0} \in O^*$ is the *initial multiset* of objects,
   - P_i is a finite set of multiset *rewrite rules* of the form: $sx \rightarrow s'x'u_{\uparrow}v_{\downarrow}w_{\leftrightarrow}y_{go}z_{out}$, where $s, s' \in Q_i$, $x, x' \in O^*$, $u_{\uparrow} \in O^*_{\uparrow}$, $v_{\downarrow} \in O^*_{\downarrow}$, $w_{\leftrightarrow} \in O^*_{\leftrightarrow}$, $y_{go} \in O^*_{go}$ and $z_{out} \in O^*_{out}$, with the restriction that $z_{out} = \lambda$ for all $i \in \{1, \ldots, n\} \backslash I_{out}$.
3. δ is a set of dag parent-child arcs on $\{1, \ldots, n\}$, i.e., $\delta \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\}$, representing *duplex* communication channels between cells;
4. $I_{out} \subseteq \{1, \ldots, n\}$ indicates the *output cells*, the only cells allowed to send objects to the "environment".

Definition 3 (Neural P systems [10]) A *neural P system* (of order $n \ge 1$), in short an *nP system*, is a system $\Pi_n = (O, \sigma_1, \ldots, \sigma_n, syn, i_{out})$, where:

1. O is an ordered finite non-empty alphabet of *objects*;



Node	$level_c$	*predecessors*	*successors*	*peers*	$count_c$
1	0	–	3,7	–	1
2	2	3	–	4	1
3	1	1	2,4,5	–	1
4	2	3,7	6	2	2
5	2	3,7	6	–	2
6	3	4,5	–	–	4
7	1	1	4,5	–	1

Figure 3: Left: a graph with commander $c = 1$, $e_1 = 3$; Right: table with node levels, predecessors, successors, peers and $count_c$'s.

2. $\sigma_1, \ldots, \sigma_n$ are *cells*, of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i)$, $1 \le i \le n$, where:
   - Q_i is a finite set (of *states*),
   - $s_{i,0} \in Q_i$ is the *initial state*,
   - $w_{i,0} \in O^*$ is the *initial multiset* of objects,
   - P_i is a finite set of multiset *rewrite rules* of the form: $sx \rightarrow s'x'y_{go}z_{out}$, where $s, s' \in Q_i$, $x, x' \in O^*$, $y_{go} \in O^*_{go}$ and $z_{out} \in O^*_{out}$, with the restriction that $z_{out} = \lambda$ for all $i \in \{1, \ldots, n\} \backslash \{i_{out}\}$.
3. *syn* is a set of *digraph* arcs on $\{1, \ldots, n\}$, i.e., $syn \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\}$, representing *unidirectional* communication channels between cells, known as *synapses*;
4. $i_{out} \in \{1, \ldots, n\}$ indicates the *output cell*, the only cell allowed to send objects to the "environment".

A *symmetric* nP system, (here) in short, a *snP system*, is an nP system where the underlying digraph *syn* is symmetric (i.e., a graph). For further definitions describing the evolution of hP and nP systems, such as *configuration*, *rewrite modes*, *transfer modes*, *transition steps*, *halting* and *results*, see our previous work [7]. For all structures, we also utilize the *weak policy* for applying *priorities* to rules, as defined by Păun [11].

Remark 4. Most of the P systems considered here (i.e., nP systems, snP systems, hP systems with siblings and hP systems without siblings) define a tag *go* that sends a multiset of objects along the previously defined *Neighbor* relation. Traditional tree-based P systems do not directly provide this facility, however, it can be easily provided by the union of *out* and *in*! target indications, that represent sending "to parent" and "to all children", respectively. That is, $(w, go) \equiv (w, out)(w, in!)$.

Definition 5 (FSSP for P systems with states—informal definition) We are given a P, hP, snP or nP system with n cells, $\{\sigma_1, \ldots, \sigma_n\}$, where all cells have the *same states set* and *same rules set*. Two states are distinguished: an *initial* state s_0 and a *firing* state s_ω. We select an arbitrary *commander* cell σ_c and an arbitrary subset of *squad* cells, $F \subseteq \{\sigma_1, \ldots, \sigma_n\}$ (possibly the whole set), that we wish to synchronize; the commander itself may or may not be part of the firing squad. At startup, all cells start in the initial state s_0; the commander and the squad cells may contain specific objects, but all other cells are empty. Initially, all cells, except the commander, are idle, and will remain idle until they receive a message. The commander sends one or more orders, to one or more of its neighbors, to start and control the synchronization process. Idle cells may become active upon receiving a first message. Notifications may be relayed to all cells, as necessary. Eventually, all cells in the squad set F will enter the designated firing state s_ω, *simultaneously* and for the *first time*. At that time, all the other cells have returned to the initial state s_0, but without passing through the firing state. Optionally, at that time, all cells should be empty.

In this paper, we propose two new deterministic FSSP solutions, that are described in the next two sections:

- The first solution is "trivially" straightforward, and assumes that we are allowed to extend the original structure. This solution works for hP systems and (not necessarily symmetric) nP systems, but fails for tree-based P systems.
- The second solution is more sophisticated and assumes that we are not allowed to extend the structure. This solution works for tree-based P systems, hP systems and snP systems.

Our two solutions do not require *polarities* or *conditional* rules, but require *priorities* and *states*. Both hP systems and snP systems already have states, by definition. However, it seems that traditional tree-based P systems have not used states so far, or not much.

4 FSSP—Solution via Structural Extensions

A straightforward solution is possible when we are allowed to extend the cell structure of the given hP or nP system. In this scenario we can also consider (not necessarily symmetric) general digraphs. In this section, we further refine our solution given in an earlier paper [8].

Intuitively, we extend the original hP or nP system by an external cell, which will be called the *sergeant*. The commander initiates the synchronization process by sending an order to the sergeant, who "shouts" to all squad cells, prompting these cells to enter the firing state, all at the same time. Because of their structural constraints, trees have only limited expansion possibilities (essentially adding new roots or leaves) and this approach will *fail* for tree-based P systems.

In the final version we will indicate a natural way to achieve this structural extension (which will add $e_c + 2$ steps), that we believe is compatible with the existing P systems rules.

Algorithm 2 (FSSP—Structural Extensions)

Precondition: An hP or nP system, with n cells $\sigma_1, \ldots, \sigma_n$, a commander cell σ_c and a set of squad cells F to be synchronized. We extend its underlying dag or digraph structure by adding a new sergeant cell σ_{n+1}.

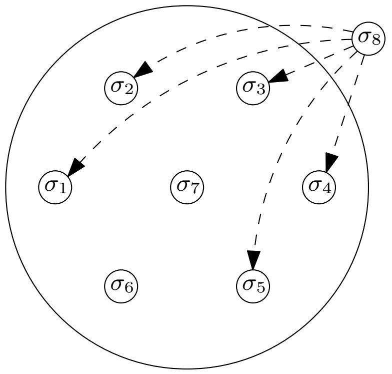
- For hP systems, we extend the original dag by an arc (σ_{n+1}, σ_i), for each $i \in F \cup \sigma_c$.
- For nP systems, we extend the original digraph by an arc (σ_c, σ_{n+1}) and by an arc (σ_{n+1}, σ_i), for each $i \in F \cup \sigma_c$.

All cells start in the state s_0 and have the same rules. The state s_1 is the firing state. Initially, the commander σ_c is marked by one object a, each squad cell is marked by one object t (this can include the commander σ_c or the sergeant σ_{n+1}, or both) and the sergeant σ_{n+1} is marked by one object c; all other cells have no objects.
Postcondition: All cells in the set F enter the state s_1, simultaneously and for the first time, after 3 steps.

Rules: All rules and states use rewrite mode $\alpha = min$ and transfer mode $\beta = repl$.

0. For state s_0, the rules will run under these priorities:
   1) $s_0 a \rightarrow s_0 b_{go}$
   2) $s_0 bct \rightarrow s_0 f f_{go}$
   3) $s_0 bc \rightarrow s_0 f_{go}$
   4) $s_0 bft \rightarrow s_1$
   5) $s_0 ft \rightarrow s_1$
   6) $s_0 bf \rightarrow s_0$
   7) $s_0 b \rightarrow s_0$

Example 6. Figure 4 illustrates this algorithm for a system which consists of seven cells $\{\sigma_1, \ldots, \sigma_7\}$. The commander cell is σ_5 and the squad set is $F = \{\sigma_1, \ldots, \sigma_5\}$. In the diagram on the left, this system's structure has already been extended by the sergeant cell σ_8 and the required communication channels, represented by dotted arrows. The actual system structure is not actually used by this algorithm and was here replaced by a blob that circumscribes the original cells. The table on the right shows specific traces for a specific hP system with seven cells—an hP system based on the tree shown in Figure 1, with $c = \sigma_3$. The extended structure is a dag, which is not a tree anymore.



Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
0	s_0t	s_0t	s_0at	s_0t	s_0t	s_0	s_0	s_0c
1	s_0bt	s_0t	s_0t	s_0bt	s_0bt	s_0b	s_0	s_0bc
2	s_0ft	s_0ft	s_0ft	s_0ft	s_0ft	s_0	s_0	s_0
3	s_1	s_1	s_1	s_1	s_1	s_0	s_0	s_0

Figure 4: Left: An hP system extended for Algorithm 2. Right: Traces for the hP system of Figure 1.

5 FSSP—Solution via Rules

Here we consider a more complex scenario, where we are only allowed to modify the rules of the given hP or nP system, but not its original structure. A brief description of this solution follows. The commander intends to send an order to all cells in the set F, which will prompt them to synchronize by entering the designated firing state. However, in general, the commander does not have direct communication channels with all the cells. In this case, the process of sending a command to the destination cell will cause delays (some steps), as the command is relayed through intermediate cells. Hence, to ensure all firing squad cells enter the firing state simultaneously, each firing squad cell determines the number of steps it needs to wait before entering the firing state.

As in our earlier paper [8], cells have no built-in knowledge of the network topology. The cells are initially empty, except the commander, which is initially marked by one a, and the squad cells, which are initially marked by one f each. All cells start with the same set of rules, which are applied in the *max* rewrite mode, using *weak* priorities, and the *repl* transfer mode. In the proofs, all rules that are concurrently applied will be grouped together within parentheses; e.g., $(x,y),z$ indicates two steps, first rules x and y, concurrently executed, followed by rule z.

Each cell *independently* progresses through *three phases*, called FSSP-I, FSSP-II and FSSP-III, which are detailed in Algorithms 3, 4 and 5, respectively. Phase FSSP-I is a broadcast from the commander, starts in s_0 and ends in s_2. Phase FSSP-II is a convergecast from terminal cells, followed by a second broadcast from the commander; this phase starts in s_2 and ends in s_8. Phase FSSP-III starts in s_8 and continues with a countdown until squad cells end in the firing state s_9, and all other cells end in s_0. A sample run of our algorithm will follow at the end of this section, in Example 12.

The statechart in Figure 5 illustrates the combined flow of these three phases. The nodes represent the states of the hP or nP system and the arcs are labelled with numbers of the rules that match the corresponding transitions.

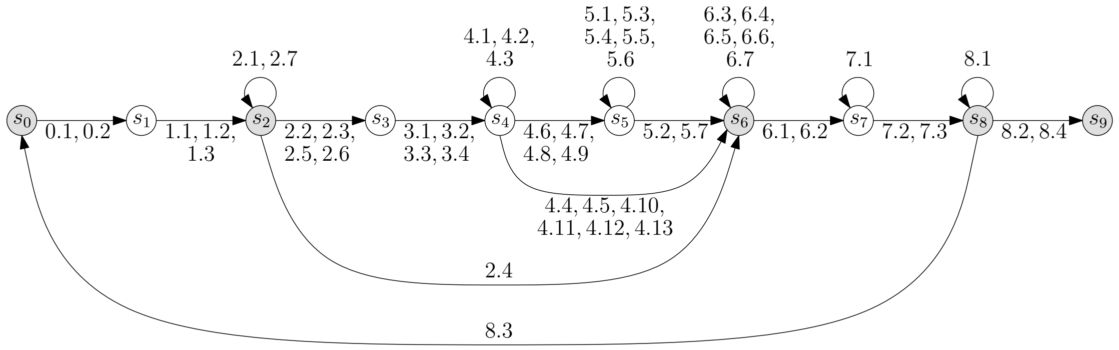


Figure 5: Statechart view of the combined FSSP algorithm phases.

The initial configuration is defined below for our FSSP algorithm.

FSSP: The initial configuration

- $\Gamma = \{\sigma_1, \ldots, \sigma_n\}$, $n > 1$, is the set of all cells, σ_c is the commander, and the firing squad is $F \subseteq \Gamma$;
- $O = \{a, b, c, d, e, f, g, h, k, l, p, q\}$;
- $Q_i = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$, for $i \in \{1, \ldots, n\}$, which is "allocated" to three phases as follows:
  - FSSP-I contains rules for states $\{s_0, s_1\}$;
  - FSSP-II contains rules for states $\{s_2, s_3, s_4, s_5, s_6\}$;
  - FSSP-III contains rules for states $\{s_6, s_7\}$;
  - FSSP-IV contains rules for states $\{s_8, s_9\}$.
- $s_\omega = s_9$ is the firing state;
- $s_{i,0} = s_0$, for $i \in \{1, \ldots, n\}$;
- $w_{c,0} = \{a\}$, if $\sigma_c \notin F$, or $\{a, f\}$, otherwise;
- $w_{i,0} = \{f\}$ for all $\sigma_i \in F \setminus \sigma_c$;
- $w_{i,0} = \emptyset$, for all $\sigma_i \in \Gamma \setminus (F \cup \{\sigma_c\})$;

- Priority rules, in the weak interpretation:

0. For state s_0:
   1) $s_0a \rightarrow s_1aed_{go}$
   2) $s_0d \rightarrow s_1ad_{go}$
1. For state s_1:
   1) $s_1ae \rightarrow s_2aeek$
   2) $s_1a \rightarrow s_2ak$
   3) $s_1d \rightarrow s_2l$
2. For state s_2:
   1) $s_2k \rightarrow s_2$
   2) $s_2ae \rightarrow s_3aee$
   3) $s_2d \rightarrow s_3d$
   4) $s_2a \rightarrow s_6ac_{go}$
   5) $s_2l \rightarrow s_3lg_{go}$
   6) $s_2g \rightarrow s_3$
   7) $s_2ae \rightarrow s_2aee$
3. For state s_3:
   1) $s_3ae \rightarrow s_4aee$
   2) $s_3a \rightarrow s_4a$
   3) $s_3g \rightarrow s_4p$
   4) $s_3c \rightarrow s_4$
4. For state s_4:
   1) $s_4cd \rightarrow s_4$
   2) $s_4ade \rightarrow s_4adee$
   3) $s_4d \rightarrow s_4d$
   4) $s_4aeeeee \rightarrow s_6aeee$
   5) $s_4eeeee \rightarrow s_6e$
   6) $s_4a \rightarrow s_5ak$
   7) $s_4l \rightarrow s_5lh_{go}$
   8) $s_4h \rightarrow s_5$
   9) $s_4q \rightarrow s_5$
   10) $s_4c \rightarrow s_6$
   11) $s_4g \rightarrow s_6$
   12) $s_4h \rightarrow s_6$
   13) $s_4q \rightarrow s_6$
5. For state s_5:
   1) $s_5k \rightarrow s_5$
   2) $s_5a \rightarrow s_6ac_{go}$
   3) $s_5hp \rightarrow s_5p$
   4) $s_5pq \rightarrow s_5$
   5) $s_5p \rightarrow s_5kp$
   6) $s_5l \rightarrow s_5lh_{go}$
   7) $s_5l \rightarrow s_6q_{go}$
6. For state s_6:
   1) $s_6ae \rightarrow s_7ak$
   2) $s_6e \rightarrow s_7be_{go}$
   3) $s_6c \rightarrow s_6$
   4) $s_6g \rightarrow s_6$
   5) $s_6h \rightarrow s_6$
   6) $s_6p \rightarrow s_6$
   7) $s_6q \rightarrow s_6$
7. For state s_7:
   1) $s_7k \rightarrow s_7$
   2) $s_7a \rightarrow s_8a$
   3) $s_7e \rightarrow s_8$
8. For state s_8:
   1) $s_8ab \rightarrow s_8a$
   2) $s_8af \rightarrow s_9$
   3) $s_8a \rightarrow s_0$
   4) $s_8a \rightarrow s_9$

Algorithm 3 (FSSP-I: First broadcast from the commander)

Precondition: The initial configuration as specified earlier.
Postcondition:

- The end state is s_2.
- A cell σ_i has
  - $count_c(i)$ copies of a and $count_c(i)$ copies of k;
  - u copies of l, where u is the total number of a's in σ_i's *peers*;
  - v copies of d, where v is the total number of a's in σ_i's *successors*;
  - two copies of e, if $\sigma_i = \sigma_c$;
  - one copy of f, if $\sigma_i \in F$.

Proof. This phase of the algorithm is a *broadcast* that follows the virtual dag created by the *levels* determined by Algorithm 1.

Consider a cell σ_i. By induction:

- At step $level_c(i)$, σ_i (except the commander) receives a total of $count_c(i)$ copies of d from its *predecessors*.
- At step $level_c(i)+1$, σ_i broadcasts $count_c(i)$ copies of d to each of its *neighbors* and transits to state s_1. At the same time, σ_i accumulates one local copy of a for each sent d, for a total count of $count_c(i)$ of a's. Also, σ_i receives u copies of d, similarly sent by its *peers*, where u is equal to the total number of a's similarly accumulated, at the same time step, by σ_i's peers.
- At step $level_c(i)+2$, σ_i receives v copies of d, sent back by its *successors*; and transits to state s_2, where v is equal to the total number of a's created, at the same time step, by σ_i's successors;

The commander, by initially having one a, creates two copies of e. Finally, the rules associated with this phase do not change the number of f's, thus, each cell in the firing squad still ends with one f. □

Lemma 7 (FSSP-I: Number of steps)**.** *For each cell σ_i, the phase FSSP-I takes $level_c(i)+2$ steps.*

Proof. As indicated in the proof of the Algorithm 3, the total number of steps is $level_c(i)+2$. □

Algorithm 4 (FSSP-II: Convergecasts from terminal nodes)

Precondition: As described in the postcondition of Algorithm 3.
Postcondition:

- This phase ends when the commander enters state s_6.
- A cell σ_i has
  - $count_c(i)$ copies of a;
  - e_c+2 copies of e, if $\sigma_i=\sigma_c$;
  - one copy of f, if $\sigma_i \in F$.

Proof. Briefly, this phase of the algorithm starts with a *convergecast* of c's from the terminal cells; followed by a second broadcast of e's from the commander, where the number of e's received by a cell σ_i indicates its necessary synchronization delay.

A terminal cell σ_i enters this phase $level_c(i)$ steps after the commander, idles one step in state s_2, then starts its role in the convergecast, by broadcasting $count_c(i)$ copies of c to its predecessors and peers (it does not have successors) and transits to state s_6. This cell further idles in state s_6 until it receives e's from its predecessors. The convergecast takes four steps at each level. The total run-time is dominated by e_c, the length of the longest level-preserving path from commander. Therefore, the convergecast wave will complete at commander after $e_c+4e_c-2=5e_c-2$ steps after the commander starts this phase. When the commander receives the convergecast from all its successors, it takes two steps to transit to state s_6. Therefore, the commander enters state s_6, $5e_c$ steps after it starts this phase. □

Lemma 8 (FSSP-II: Number of steps)**.** *For each cell σ_i, the phase FSSP-II takes $5e_c-level_c(i)$ steps.*

Proof. As indicated in the proof of Algorithm 4, this phase takes $5e_c$ steps. □

Algorithm 5 (FSSP-III: Second broadcast from the commander)

Precondition: As described in the postcondition of Algorithm 4.
Postcondition:

- The end state is s_8.
- A cell σ_i has
  - $count_c(i)$ copies of a;
  - $(e_c + 1 - level_c(i))count_c(i)$ copies of b;
  - one copy of f, if $\sigma_i \in F$.

Proof. In this phase, commander starts its second *broadcast*, by sending $e_c + 1$ copies of e's to all its successors. By induction on level, a cell σ_i receives a total of $(e_c + 2 - level_c(i))count_c(i)$ copies of e's from its predecessors, reduces this count by $count_c(i)$ (i.e., the count of a's), forwards the remaining $(e_c + 1 - level_c(i))count_c(i)$ copies of e's to all its successors and creates for itself $(e_c + 1 - level_c(i))count_c(i)$ copies of b's. A more detailed description will be given in the final version.

All rules of this phase do not change the number of a's or the number of f's; therefore, the corresponding postcondition holds. □

Lemma 9 (FSSP-III: Number of steps)**.** *For each cell σ_i, the phase FSSP-III takes $level_c(i) + 3$ steps.*

Proof. As indicated in the proof of Algorithm 5, this phase takes $level_c(i) + 3$ steps. □

Algorithm 6 (FSSP-IV: Timing for entering the firing state)

Precondition: As described in the postcondition of Algorithm 5.
Postcondition:

- The end state is s_9 for cells in the firing squad, or s_0, otherwise.
- Each cell is empty.

Proof. As long as b's are present, a cell σ_i performs a transition step that decreases the number of b's by $count_c(i)$ (i.e., the number of a's). This step will be repeated $(e_c + 1 - level_c(i))$ times, as given by the initial ratio between the number of b's, $(e_c + 1 - level_c(i))count_c(i)$, and the number of a's, $count_c(i)$. This is the delay every cell needs to wait, before entering either the firing state s_9 or the initial state s_0.

Finally, in the last step, cell σ_i enters s_9, if σ_i has one f, or s_0, otherwise. At the same time, all existing objects are removed. □

Lemma 10 (FSSP-IV: Number of steps)**.** *For each cell σ_i, the phase FSSP-IV takes $e_c + 2 - level_c(i)$ steps.*

Proof. As indicated in the proof of Algorithm 6, this phase takes $(e_c + 1 - level_c(i)) + 1 = e_c + 2 - level_c(i)$. □

Theorem 11. *For each cell σ_i, the combined running time of the three phases Algorithm 3, 4, 5 and 6 is $6e_c + 7$, where e_c is the eccentricity of the commander σ_c.*

Proof. The result is obtained by summing the individual running times of the three phases, as given by Lemmas 7, 8, 9 and 10: $(level_c(i)+2)+(5e_c-level_c(i))+(level_c(i)+3)+(e_c+2-level_c(i))=6e_c+7$. □

Example 12. We present traces of the FSSP algorithm for the hP system given in Figure 2 in Table 1, where the cells are ordered according to their *levels* and the starting states of phases FSSP-II, FSSP-III and FSSP-IV are highlighted.

Table 1: The FSSP trace on the dag of Figure 2, where $c=6$, $e_6=3$, $F=\{\sigma_1,\sigma_4,\sigma_5,\sigma_7,\sigma_9,\sigma_{10}\}$.

	σ_6	σ_2	σ_3	σ_9	σ_1	σ_5	σ_7	σ_8	σ_4	σ_{10}
0	s_0af	s_0	s_0	s_0f	s_0	s_0f	s_0f	s_0	s_0f	s_0f
1	s_1aef	s_0d	s_0d	s_0df	s_0	s_0f	s_0f	s_0	s_0f	s_0f
2	$s_2ad^3e^2fk$	s_1a	s_1a	s_1af	s_0d^2	s_0df	s_0df	s_0d	s_0f	s_0f
3	$s_2ad^3e^3f$	s_2ad^3k	s_2ad^3k	s_2adfk	s_1a^2	s_1af	s_1adf	s_1ad	s_0df	s_0df
4	$s_3ad^3e^4f$	s_2ad^3	s_2ad^3	s_2adf	$s_2a^2k^2$	s_2afk	s_2adfkl	s_2adkl	s_1af	s_1af
5	$s_4ad^3e^5f$	s_3ad^3	s_3ad^3	s_3adf	s_2a^2	s_2af	s_2adfl	s_2adl	s_2afk	s_2afk
6	$s_4ad^3e^6f$	$s_4ac^3d^3$	$s_4ac^2d^3g$	s_4adfg	s_6a^2	s_6af	s_3adfgl	s_3adgl	s_2afg	s_2afg
7	$s_4ad^3e^7f$	s_4a	s_4adg	s_4adfg	s_6a^2	s_6af	$s_4acdflp$	s_4acdlp	s_6afg	s_6afg
8	$s_4ad^3e^8f$	s_5ak	s_4adg	s_4adfg	s_6a^2	s_6af	s_4aflp	s_4alp	s_6af	s_6af
9	$s_4ad^3e^9f$	s_5a	s_4adgh	s_4adfgh	s_6a^2	s_6af	$s_5afhklp$	s_5ahklp	s_6afh	s_6afh
10	$s_4acd^3e^{10}f$	s_6a	s_4adgh^2	s_4adfgh^2	s_6a^2c	s_6acf	s_5afhlp	s_5ahlp	s_6afh	s_6afh
11	$s_4ad^2e^{11}f$	s_6a	s_4acdgh^2q	$s_4acdfgh^2q$	s_6a^2	s_6af	$s_6acfhpq$	s_6achpq	s_6acfq	s_6acfq
12	$s_4ad^2e^{12}f$	s_6a	s_4agh^2q	s_4afgh^2q	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
13	$s_4ad^2e^{13}f$	s_6a	s_5agk	s_5afgk	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
14	$s_4ad^2e^{14}f$	s_6a	s_5ag	s_5afg	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
15	$s_4ac^2d^2e^{15}f$	s_6a	s_6ag	s_6afg	s_6a^2c	s_6af	s_6acf	s_6ac	s_6af	s_6af
16	$s_4ae^{15}f$	s_6a	s_6a	s_6af	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
17	s_6ae^5f	s_6a	s_6a	s_6af	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
18	s_7ab^4fk	s_6ae^4	s_6ae^4	s_6ae^4f	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
19	$s_7ab^4e^9f$	s_7ab^3k	s_7ab^3k	s_7ab^3fk	$s_6a^2e^6$	s_6ae^3f	s_6ae^3f	s_6ae^3	s_6af	s_6af
20	s_8ab^4f	$s_7ab^3e^6$	$s_7ab^3e^6$	$s_7ab^3e^2f$	$s_7a^2b^4k^2$	s_7ab^2fk	$s_7ab^2e^2fk$	$s_7ab^2e^2k$	s_6ae^2f	s_6ae^2f
21	s_8ab^3f	s_8ab^3	s_8ab^3	s_8ab^3f	$s_7a^2b^4$	s_7ab^2f	$s_7ab^2e^3f$	$s_7ab^2e^3$	s_7abfk	s_7abfk
22	s_8ab^2f	s_8ab^2	s_8ab^2	s_8ab^2f	$s_8a^2b^4$	s_8ab^2f	s_8ab^2f	s_8ab^2	s_7abf	s_7abf
23	s_8abf	s_8ab	s_8ab	s_8abf	$s_8a^2b^2$	s_8abf	s_8abf	s_8ab	s_8abf	s_8abf
24	s_8af	s_8a	s_8a	s_8af	s_8a^2	s_8af	s_8af	s_8a	s_8af	s_8af
25	s_9	s_0	s_0	s_9	s_0	s_9	s_9	s_0	s_9	s_9

6 Conclusion

We have presented two new algorithms for the Firing Squad Synchronization Problem that operate on several families of P systems. Out of the box, both algorithms work for hyperdag P systems and symmetric neural P systems. The first "trivial" algorithm, based on structural extensions, highlights the merits of dags as underlying structures for P systems. The second algorithm, which is more complex, is applicable to P systems with fixed membrane topologies and is uniformly defined in terms of a structural *Neighbor* relation. Both our FSSP algorithms handle a generalized version of the FSSP, where the commander can assume an arbitrary position and only a specified subset of the cells need to be synchronized.

An open problem is to find an efficient solution for transition P systems without polarizations and without states. Another natural problem, left open by the work started in this paper, is to find an efficient synchronization algorithm for arbitrary strongly-connected (non necessarily symmetric) nP systems. It would also be interesting to find an FSSP algorithm for hP or snP systems that runs in fewer than $6e+7$ steps, where the multiplier is less than 6.

Acknowledgements

The authors wish to thank John Morris and the three anonymous reviewers for detailed comments and feedback that helped us improve the paper.

References

[1] Artiom Alhazov, Maurice Margenstern & Sergey Verlan (2009): *Fast Synchronization in P Systems*. In: David W. Corne, Pierluigi Frisco, Gheorghe Păun, Grzegorz Rozenberg & Arto Salomaa, editors: *Workshop on Membrane Computing*, *Lecture Notes in Computer Science* 5391. Springer, pp. 118–128. Available at `http://dx.doi.org/10.1007/978-3-540-95885-7_9`.

[2] Francesco Bernardini, Marian Gheorghe, Maurice Margenstern & Sergey Verlan (2008): *How to Synchronize the Activity of All Components of a P System? Int. J. Found. Comput. Sci.* 19(5), pp. 1183–1198. Available at `http://dx.doi.org/10.1142/S0129054108006224`.

[3] Roger L. Freeman (2005): *Fundamentals of Telecommunications, 2nd Edition*. Wiley-IEEE Press.

[4] Tim Carter Humphrey (2005): *Cell Cycle Control: Mechanisms and Protocols*. Humana Press.

[5] Kojiro Kobayashi & Darin Goldstein (2005): *On Formulations of Firing Squad Synchronization Problems*. In: Cristian Calude, Michael J. Dinneen, Gheorghe Păun, Mario J. Pérez-Jiménez & Grzegorz Rozenberg, editors: *UC*, *Lecture Notes in Computer Science* 3699. Springer, pp. 157–168. Available at `http://dx.doi.org/10.1007/11560319_15`.

[6] Jacques Mazoyer (1987): *A Six-State Minimal Time Solution to the Firing Squad Synchronization Problem. Theor. Comput. Sci.* 50, pp. 183–238.

[7] Radu Nicolescu, Michael J. Dinneen & Yun-Bum Kim (2008): *Structured Modelling with Hyperdag P Systems: Part A*. Report CDMTCS-342, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand. Available at `http://www.cs.auckland.ac.nz/CDMTCS/researchreports/342hyperdagA.pdf`. Also in: Rosa Gutiérrez-Escudero, Miguel Angel, Gutiérrez-Naranjo, Gheorghe Păun & Ignacio Pérez-Hurtado, editors: *Proceedings of Seventh Brainstorming Week on Membrane Computing (BWMC2009)*. Universidad de Sevilla, pp. 85–108.

[8] Radu Nicolescu, Michael J. Dinneen & Yun-Bum Kim (2009): *Discovering the Membrane Topology of Hyperdag P Systems*. In: *Proceedings of the 10th Workshop on Membrane Computing*. pp. 1–27. To appear.

[9] Kenichiro Noguchi (2004): *Simple 8-state minimal time solution to the firing squad synchronization problem*. *Theor. Comput. Sci.* 314(3), pp. 303–334. Available at `http://dx.doi.org/10.1016/S0304-3975(03)00425-0`.

[10] Gheorghe Păun (2002): *Membrane Computing-An Introduction*. Springer-Verlag.

[11] Gheorghe Păun (2006): *Introduction to Membrane Computing*. In: Gabriel Ciobanu, Mario J. Pérez-Jiménez & Gheorghe Păun, editors: *Applications of Membrane Computing*, Natural Computing Series. Springer, pp. 1–42. Available at `http://dx.doi.org/10.1007/3-540-29937-8_1`.

[12] Hubert Schmid & Thomas Worsch (2004): *The Firing Squad Synchronization Problem with Many Generals For One-Dimensional CA*. In: Jean-Jacques Lévy, Ernst W. Mayr & John C. Mitchell, editors: *IFIP TCS*. Kluwer, pp. 111–124.

[13] Abraham Silberschatz, Peter Baer Galvin & Greg Gagne (2004): *Operating System Concepts, 7th Edition*. Wiley.

[14] Helge Szwerinski (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, pp. 305–320.

[15] Hiroshi Umeo, Masashi Maeda & Norio Fujiwara (2002): *An Efficient Mapping Scheme for Embedding Any One-Dimensional Firing Squad Synchronization Algorithm onto Two-Dimensional Arrays*. In: Stefania Bandini, Bastien Chopard & Marco Tomassini, editors: *ACRI*, *Lecture Notes in Computer Science* 2493. Springer, pp. 69–81. Available at `http://link.springer.de/link/service/series/0558/bibs/2493/24930069.htm`.