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 two uniform solutions to an open question: the Firing Squad Synchronization Problem (FSSP), for hyperdag and symmetric neural P systems, with anonymous cells. Our solutions take $e_c + 5$ and $6e_c + 7$ steps, respectively, where $e_c$ is the eccentricity of the commander cell of the dag or digraph underlying these P systems. The first and fast solution is based on a novel proposal, which dynamically extends P systems with mobile channels. The second solution is substantially longer, but is solely based on classical rules and static channels. In contrast to the previous solutions, which work for tree-based P systems, our solutions synchronize to any subset of the underlying digraph; and do not require membrane polarizations or conditional rules, but require states, as typically 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, 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.

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.

These more complex structures pose additional challenges, not considered in the previous FSSP papers on tree-based P systems, such as multiple network sources (no single root) and multiple paths between cells. Additionally, by allowing an arbitrary position for the commander, we cannot anymore take advantage of the sense of direction between adjacent cells; practically, our structures need to be treated as undirected graphs.

Our first solution uses simple rules, but requires *dynamical structures*. In this paper we propose a novel extension, which supports the creation of dynamical structures, by allowing mobile channels. This solution works for hP systems and symmetric nP systems; it will also work for tree-based P systems, but only if we reconsider them as dag-based P systems, because the resulting structures will be dags, not trees. This solution takes $e_c + 7$ steps, where $e_c$ is the eccentricity of the commander cell of the underlying dag or digraph. The relative simplicity and the speed 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 dynamical structures, but is substantially more complex, combining a *breadth-first-search*, a *broadcast* and a *convergecast*. This solution works for tree-based P systems, hP systems and symmetric nP systems and takes $6e_c + 7$ steps. When restricted to P systems, our algorithm takes more steps than Alhazov et al [11], if the commander is the root node, but comparable to this, when the commander is a central node of an unbalanced rooted tree.

Our two solutions do not require polarizations or conditional rules, but require states, as defined for hyperdag and neural P systems.

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, where we also illustrate the evolution of our FSSP algorithms. Finally, 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 $x_1, \ldots, x_n$, such that $\{ (x_1, x_2), \ldots, (x_{n-1}, x_n) \} \subseteq A$. A cycle is a path $x_1, \ldots, x_n$, where $n \geq 1$ and $(x_n, x_1) \in A$. A digraph is symmetric if its relation $A$ is symmetric, i.e., $(x_1, x_2) \in A \iff (x_2, x_1) \in A$. 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 \(A(A^{-1}(x)) \setminus \{x\}\) 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 \(x\), \(\text{Neighbor}(x)\), is the union of \(x\)'s parent and \(x\)'s children. For a dag \(\delta\) and node \(x\), we define \(\text{Neighbor}(x) = \delta(x) \cup \delta^{-1}(x) \cup \delta(\delta^{-1}(x)) \setminus \{x\}\), if we want to include the siblings, or, \(\text{Neighbor}(x) = \delta(x) \cup \delta^{-1}(x)\), otherwise. For a graph \(G = (X,A)\), we set \(\text{Neighbor}(x) = A(x) = \{y \mid (x,y) \in A\}\). Note that, as defined, \(\text{Neighbor}\) 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 \(x\), \(\text{level}_c(x) \in \mathbb{N}\), 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 \(y \in \text{Neighbor}(x)\) and \(\text{level}_c(y) = \text{level}_c(x) + 1\), 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 \(z \in \text{Neighbor}(x)\) and \(\text{level}_c(z) = \text{level}_c(x)\). 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 \(\text{maxlevel}_c = \max\{\text{level}_c(x) \mid x \in X\}\) and we note \(e_c = \text{maxlevel}_c\). A level-preserving path from \(c\) to a node \(y\) is a sequence \(x_0, \ldots, x_k\), such that \(x_0 = c, x_k = y, x_i \in \text{Neighbor}(x_{i-1})\), \(\text{level}_c(x_i) = i, 1 \leq i \leq k\). We further define \(\text{count}_c(y)\) 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, this algorithm defines a virtual dag based on successor relation and, if the original structure is a tree, this algorithm will “reset” the root at another node in that tree.

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

- **INPUT**: A tree, dag or digraph, with nodes \(\{1, \ldots, n\}\) and a commander \(c \in \{1, \ldots, n\}\).
- **OUTPUT**: The arrays \(\text{level}_c[]\) and \(\text{count}_c[]\) 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,\ldots,n] = [-1,\ldots,-1]; count_c[1,\ldots,n] = [0,\ldots,0]
queue Q = ()
Q ← c
level_c[c] = 0; count_c[c] = 1
while Q ≠ () do
  x ← Q
  for each y ∈ Neighbor(x) do
    if level_c[y] = −1 then
      Q ← 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, predecessors, successors, peers and count, 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].

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

| Node | level | predecessors | successors | peers | count |
|------|-------|--------------|------------|-------|-------|
| 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 2: Left: a dag with commander c = 6, e = 3 (siblings excluded); Right: table with node levels, predecessors, successors, peers and count’s.

| Node | level | predecessors | successors | peers | count |
|------|-------|--------------|------------|-------|-------|
| 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 3: Left: a graph with commander c = 1, e = 3; Right: table with node levels, predecessors, successors, peers and count’s.

| Node | level | predecessors | successors | peers | count |
|------|-------|--------------|------------|-------|-------|
| 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     |
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 \( \Pi_h = (O, \sigma_1, \ldots, \sigma_n, \delta, I_{out}) \), where:

1. \( O \) is an ordered finite non-empty alphabet of objects;
2. \( \sigma_1, \ldots, \sigma_n \) are cells, of the form \( \sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i) \), \( 1 \leq i \leq n \), 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: \( sx \rightarrow s'x'u_i|v_i|w_{i,0}|y_{go}z_{out} \), where \( s, s' \in Q_i \), \( x, x' \in O^* \), \( u_i \in O_i^* \), \( v_i \in O_i^* \), \( w_{i,0} \in O_i^{* \Delta} \), \( y_{go} \in O_{go}^* \), and \( z_{out} \in O_{out}^* \), with the restriction that \( z_{out} = \lambda \) for all \( i \in \{1, \ldots, n\} \\backslash I_{out} \).
3. \( \delta \) 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 \( n \geq 1 \)), in short an nP system, is a system \( \Pi_n = (O, \sigma_1, \ldots, \sigma_n, \text{syn}, I_{out}) \), where:

1. \( O \) is an ordered finite non-empty alphabet of objects;
2. \( \sigma_1, \ldots, \sigma_n \) are cells, of the form \( \sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i) \), \( 1 \leq i \leq n \), 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: \( sx \rightarrow s'y_{go}z_{out} \), 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. \( \text{syn} \) is a set of digraph arcs on \( \{1, \ldots, n\} \), i.e., \( \text{syn} \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\} \), 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 \( \text{go} \) that sends a multiset of objects along the previously defined \( \text{Neighbor} \) relation. Traditional tree-based P systems do not directly provide this facility, however, it can be easily provided by the union of \( \text{out} \) and \( \text{in!} \) target indications, that represent sending “to parent” and “to all children”, respectively. That is, \((w, \text{go}) \equiv (w, \text{out}) (w, \text{in!})\).

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 \( s_\phi \). We select an arbitrary commander cell \( \sigma_c \) 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 \( s_\phi \), simultaneously and for the first time. At that time, all the other cells have reached a different state, typically \( s_0 \) or \( s_1 \), without ever passing through the firing state \( s_\phi \). 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. 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—Dynamic Structures via Mobile Channels

In this section, we further refine our solution given in an earlier paper [8]. A natural solution is possible when we are allowed to extend the cell structure of the given hP or snP system. We achieve this by supporting mobile channels. The endpoints of our mobile channels appear in the rules like all other objects and are subject to usual rewriting and transfer rules. The end result is a channel that grows step-by-step, not unlike a nerve which extends in a growing or regrowing tissue.

We first extend the original hP or nP system by an external cell, which will be called the sergeant. Next, this sergeant will send a self-replicating mobile endpoint, that will be repeatedly broadcasted, until all cells are reached. A mobile endpoint will leave a fixed endpoint in a squad cell and disappear without trace from the other cells. In the end, the structure will be extended with new channels which will link the sergeant with all squad cells. Finally, when there are no more structural changes, the sergeant, will send a firing command to all squad cells, prompting these cells to enter the firing state, all at the same time.

Our algorithm uses the following special objects, which can in principle be rewritten and transferred as all other traditional objects, but, at the same time, are also endpoints for dynamically created channels:

- \( \alpha \) is here the fixed endpoint of all dynamically created channels (here we use only one \( \alpha \))
- \( \theta \) is a mobile endpoint of a dynamically created channel (here this symbol is further processed by rewriting and transfer rules)
- \( \omega \) is a new fixed endpoint of a dynamically created channel (here this symbol will remain fixed)
Briefly, the initial structure is dynamically extended by all arcs \((\sigma_4, \sigma_5)\), where \(\sigma_4\) is the cell that contains \(\alpha\) and \(\sigma_5\) is a cell that contains \(\theta\) or \(\omega\).

The following algorithm assumes that the first step has already been completed, i.e., the sergeant was already created by one of the existing cell creation or division rules, already available for P systems.

\begin{algorithm}[FSSP—Dynamic structures via mobile channels]

**Precondition:** An hP or nP system, with \(n\) cells \(\sigma_1, \ldots, \sigma_n\), a commander cell \(\sigma_c\) and a set of squad cells \(F\) to be synchronized. Additionally, we already have a sergeant cell, \(\sigma_{n+1}\), linked to the commander by one arc, \((\sigma_{n+1}, \sigma_c)\), for hP systems, or by two arcs, \((\sigma_{n+1}, \sigma_c), (\sigma_c, \sigma_{n+1})\), for snP systems.

All cells start in the state \(s_0\) and have the same rules. The state \(s_\phi\) is the firing state. Initially, the sergeant \(\sigma_{n+1}\) is marked by one object \(\alpha\), and each squad cell is marked by one object \(f\) (this can include the commander \(\sigma_c\), or the sergeant \(\sigma_{n+1}\), or both); all other cells have no objects.

**Postcondition:** All cells in the set \(F\) enter state \(s_\phi\), simultaneously and for the first time, after \(e_c + 5\) steps, where \(e_c\) is the commander’s eccentricity in the underlying graph. All other cells enter state \(s_1\), without ever passing through state \(s_\phi\).

**Rules** (rules are applied under the weak interpretation of priorities, in the rewrite mode \(\alpha = \text{min}\) and transfer mode \(\beta = \text{repl}\)):

1. \(s_0\alpha \rightarrow s_2\alpha\theta_g\)
2. \(s_0f\theta \rightarrow s_4\omega\theta_g\)
3. \(s_0\theta \rightarrow s_1\theta_g\)
4. \(s_1\theta \rightarrow s_1\)
5. \(s_2\alpha \rightarrow s_3\alpha\)
6. \(s_3\theta \rightarrow s_3\)
7. \(s_3f\alpha \rightarrow s_4f\alpha\phi\phi_g\)
8. \(s_3\alpha \rightarrow s_1\alpha\phi_g\)
9. \(s_4f\phi \rightarrow s_\phi\)
10. \(s_4\theta \rightarrow s_4\)

**Example 6.** Figure 4 and Table 1 illustrate this algorithm for an hP system based on the dag of Figure 1. Here, the commander cell is \(\sigma_3\), the squad set is \(F = \{\sigma_1, \ldots, \sigma_5\}\) and this system’s structure has already been extended by the sergeant cell \(\sigma_8\) and the arc \((\sigma_8, \sigma_3)\). The mobile channels are represented by dotted arrows.

![Figure 4: Running Algorithm 2 on the hP system of Example 6](image_url)

\[\text{Figure 4: Running Algorithm 2 on the hP system of Example 6}\]
5 FSSP—Static Structures and Rules

Here we consider a second scenario, where we are 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. Additionally, cells are anonymous, i.e., not identified by cell IDs, and not implicitly named by membrane polarization techniques. 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 \( \text{max} \) rewrite mode, using weak priorities, and the \( \text{repl} \) transfer mode. In the proofs, all rules that are concurrently applied will be grouped together within parentheses; e.g., \((x, y), \omega \) indicates two steps, first rules \( x \) and \( y \), concurrently executed, followed by rule \( \omega \).

Each cell independently progresses through four phases, called FSSP-I, FSSP-II, FSSP-III and FSSP-IV, which are detailed in Algorithms 3, 4, 5 and 6, respectively. An overview of these four phases is as follows:

- Phase FSSP-I is a broadcast from the commander, that follows the virtual dag defined by \( \text{level}_c \). This phase starts in state \( s_0 \) and ends in state \( s_2 \). Also, the commander starts a counter, which, at the end of Phase FSSP-II, will determine its eccentricity.
- Phase FSSP-II is a subsequent convergecast from terminal cells, that follows the same virtual dag. This phase starts in state \( s_2 \) and ends when the commander enters state \( s_6 \). At the end of this phase, the commander’s counter determines its eccentricity.
- Phase FSSP-III is a second broadcast, initiated from the commander, that follows the same virtual dag. This phase starts in state \( s_6 \) and ends in state \( s_8 \). The commander sends out its eccentricity, which is successively decremented at each level.
- Phase FSSP-IV is a timing (countdown) for entering the firing state. This phase starts in state \( s_8 \) and continues with a countdown, until squad cells simultaneously enter the firing state \( s_9 \), and all other cells enter state \( s_0 \).
The statechart in Figure 5 illustrates the combined flow of these four 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. The rest of this section describes these four phases, proving their correctness and time complexities. A sample run of our algorithm will follow at the end of this section, in Example 12.

**FSSP: The initial configuration**

- \( \Gamma = \{\sigma_1, \ldots, \sigma_n\}, n > 1\), is the set of all cells, \( \sigma_c \) 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 four 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 \( \{s_8, s_9\}\);

- \( s_\phi = s_9 \) 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 \( \sigma_i \in \Gamma \setminus (F \cup \{\sigma_c\}) \);

- The following rules are applied under the weak interpretation of priorities, in the rewrite mode \( \alpha = \min \) and transfer mode \( \beta = \text{repl} \):
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) $s_1ae \rightarrow s_2aek$
   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) $s_2l \rightarrow s_3l_{go}$
   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) $s_4de \rightarrow s_4adee$
   3) $s_4d \rightarrow s_4d$
   4) $s_4eee \rightarrow s_6ae$
   5) $s_4eeee \rightarrow s_6ae$
   6) $s_4a \rightarrow s_5ak$
   7) $s_4l \rightarrow s_5l_{go}$

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) $s_5l \rightarrow s_5l_{go}$
   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 $\sigma_i$ has
  - $\text{count}_c(i)$ copies of $a$ and $\text{count}_c(i)$ copies of $k$;
  - $u$ copies of $l$, where $u$ is the total number of $a$'s in $\sigma_i$'s peers;
  - $v$ copies of $d$, where $v$ is the total number of $a$'s in $\sigma_i$'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 I. Consider a cell $\sigma_i$. By induction:
- At step $\text{level}_c(i)$, $\sigma_i$ (except the commander) receives a total of $\text{count}_c(i)$ copies of $d$ from its predecessors.
At step $level_c(i) + 1$, $\sigma_i$ broadcasts $count_c(i)$ copies of $d$ to each of its neighbors and transits to state $s_1$. At the same time, $\sigma_i$ accumulates one local copy of $a$ for each sent $d$, for a total count of $count_c(i)$ of $a$’s. Also, $\sigma_i$ 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 $\sigma_i$’s peers.

At step $level_c(i) + 2$, $\sigma_i$ 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 $\sigma_i$’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 $f$.

**Corollary 7 (FSSP-I: Number of steps).** For each cell $\sigma_i$, 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 $\sigma_i$ has
  - $count_c(i)$ copies of $a$;
  - $e + 2$ copies of $e$, if $\sigma_i = \sigma_c$;
  - one copy of $f$, if $\sigma_i \in F$.

**Proof.** Briefly, this phase of the algorithm is a convergecast of $c$’s, starting from terminal cells, and further relayed up, on the virtual dag, until the commander is reached.

For the purpose of this phase, the non-commander cells can be organized in the following three groups: TC cells = terminal cells; NTC-NTP cells = non-terminal cells without non-terminal peers (i.e., cells without peers or cells with terminal peers only); NTC+NTP cells = non-terminal cells with non-terminal peers (these cells may also have terminal peers).

During this phase, these cells will make transitions between the following three conceptual stages: WCS = waiting for convergecasts from successors (state $s_4$); RTC = ready to convergecast (state $s_5$); HC = have convergecasted (state $s_6$). Specifically, the following transitions will be made: the TC cells will transit immediately from the WCS stage to the HC stage; the NTC-NTP cells will linger in the WCS stage until they receive convergecasts from all their successors, after which they will transit directly to the HC stage; the NTC+NTP cells will linger in the WCS stage until they receive convergecasts from all their successors, subsequently they will linger in the RTC stage until all their non-terminal peers reach the RTC stage as well, after which they will transit to the HC stage.

During this process, cells will exchange $c$-notifications, which are messages consisting of number of $c$’s and $h$-notifications, which are messages consisting of number of $h$’s. The actual numbers depend on network topology and take into account the multiple paths that appear in the virtual dag.

The $c$-notification broadcasted by cell $\sigma_i$ consists of $count_c(i)$ copies of $c$ and is only sent once when $\sigma_i$ transits into the HC stage.

The $h$-notification broadcasted by cell $\sigma_i$ consists of $u$ copies of $h$, where $u$ is the number defined in the precondition. This notification is sent repeatedly, while $\sigma_i$ remains in the RTC stage, until $\sigma_i$
transits into the HC stage. The $h$-notifications synchronize the non-terminal peers that cannot transit to the HC stage until all of them have reached the RTC stage. This avoids the potential confusion that could otherwise arise when a non-terminal cell receives an “ambiguous” $c$-notification, i.e., a $c$-notification that could come both from a successor or from a non-terminal peer.

Without loss of generality, we illustrate our solution on the dag from Figure 6. This figure shows a typical sub-dag of the virtual dag created by Algorithm 1, where cells $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ and $\sigma_5$ are at the same level, and the horizontal lines indicate peer relations.

![Figure 6: A typical sub-dag of the virtual dag.](image)

Cell $\sigma_1$ is a TC cell and will transit from stage WCS to stage HC, immediately after detecting that it has no successors (there is no need for any synchronization with its peer $\sigma_2$).

Cell $\sigma_5$ is a NTC-NTP cell and will transit from stage WCS to stage HC, after receiving $c$-notifications from all its successors from all its successors.

Cells $\sigma_2, \sigma_3$ and $\sigma_4$ are NTC+NTP cells. Each of these cells will linger in stage WCS until it receives $c$-notifications from all its successors, when it will enter stage RTC. Cells $\sigma_2$ and $\sigma_3$ are peers, therefore none of them will be allowed to transit to stage HC, until both of them have reached the RTC stage. Similarly, cells $\sigma_3$ and $\sigma_4$ are peers, therefore none of them will be allowed to transit to stage HC, until both of them have reached the RTC stage. Assume that cells $\sigma_2, \sigma_3$ and $\sigma_4$ will reach stage RTC in this order. Then, cell $\sigma_2$ will wait in stage RTC until $\sigma_3$ also reaches the same stage. When this eventuates, $\sigma_2$ will transit to stage HC, while $\sigma_3$ will still linger in stage RTC until $\sigma_4$ reaches the same stage. When this eventuates, both $\sigma_3$ and $\sigma_4$ will transit at the same time to stage HC.

A TC cell $\sigma_i$ 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.

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

**Proof.** As indicated in the proof of Algorithm 1, 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 $\sigma_i$ has
  - $\text{count}_e(i)$ copies of $a$;
  - $(e_c + 1 - \text{level}_c(i)) \text{count}_e(i)$ 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 $\sigma_i$ receives a total of $(e_c + 2 - \text{level}_c(i)) \text{count}_e(i)$ copies of $e$’s from its predecessors, reduces this count by $\text{count}_e(i)$ (i.e., the count of $a$’s), forwards the remaining $(e_c + 1 - \text{level}_c(i)) \text{count}_e(i)$ copies of $e$’s to all its successors and creates for itself $(e_c + 1 - \text{level}_c(i)) \text{count}_e(i)$ 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.

Corollary 9 (FSSP-III: Number of steps). For each cell $\sigma_i$, the phase FSSP-III takes $\text{level}_c(i) + 3$ steps.

Proof. As indicated in the proof of Algorithm 5, this phase takes $\text{level}_c(i) + 3$ 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 $\sigma_i$ performs a transition step that decreases the number of $b$’s by $\text{count}_b(i)$ (i.e., the number of $a$’s). This step will be repeated $(e_c + 1 - \text{level}_c(i))$ times, as given by the initial ratio between the number of $b$’s, $(e_c + 1 - \text{level}_c(i)) \text{count}_b(i)$, and the number of $a$’s, $\text{count}_a(i)$. 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 $\sigma_i$ enters $s_9$, if $\sigma_i$ has one $f$, or $s_0$, otherwise. At the same time, all existing objects are removed.

Corollary 10 (FSSP-IV: Number of steps). For each cell $\sigma_i$, the phase FSSP-IV takes $e_c + 2 - \text{level}_c(i)$ steps.

Proof. As indicated in the proof of Algorithm 6, this phase takes $(e_c + 1 - \text{level}_c(i)) + 1 = e_c + 2 - \text{level}_c(i)$.
Theorem 11. For each cell \( \sigma_i \), the combined running time of the four phases Algorithm 2, 4, 5, and 6 is 
\[ 6e_c + 7, \text{ where } e_c \text{ is the eccentricity of the commander } \sigma_c. \]

Proof. The result is obtained by summing the individual running times of the four phases, as given by Corollaries 7, 8, 9, and 10. 
\[ (\text{level}_c(i) + 2) + (5e_c - \text{level}_c(i)) + (\text{level}_c(i) + 3) + (e_c + 2 - \text{level}_c(i)) = 6e_c + 7. \]

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

| \( \sigma_0 \) | \( \sigma_1 \) | \( \sigma_2 \) | \( \sigma_3 \) | \( \sigma_4 \) | \( \sigma_5 \) | \( \sigma_6 \) | \( \sigma_7 \) | \( \sigma_8 \) | \( \sigma_9 \) | \( \sigma_{10} \) |
|---|---|---|---|---|---|---|---|---|---|---|
| \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) |
| \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) |
| \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) |
| \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) |
| \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) |
| \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) |
| \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) |
| \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) |
| \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) |
| \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) | \( s_0 \) |

Table 2: The FSSP trace on the dag of Figure 2, where \( c = 6, e_6 = 3, F = \{ \sigma_1, \sigma_4, \sigma_5, \sigma_7, \sigma_9, \sigma_{10} \} \).
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 algorithm is based on dynamic structures and highlights the merits of dags as underlying structures for P systems. To support the required dynamic structures, we propose an extended interpretation of P systems which allows mobile channels, a solution which we believe is fully compatible with the existing P systems rules.

The second algorithm, which is more complex, is applicable to P systems with static membrane topologies and is uniformly defined in terms of a structural Neighbor relation. These two algorithms do not require naming facilities, such as cell IDs or cell polarization and handle a generalized version of the FSSP, where the commander can assume an arbitrary position and only a specified subset of the cells needs to be synchronized.

The work started in this paper leaves open several interesting problems. Can we find simpler and more efficient solutions for hP systems based on single-sourced dags? Can we find simpler and more efficient solutions for hP or snP systems using named cells (unique cell IDs)? Can we find a solution for arbitrary strongly-connected (non necessarily symmetric) nP systems? What is relation between the mobile channels, which we have here proposed for P systems, and the support for mobile channels in the $\pi$-calculus?

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 (WMC10)*. Curtea de Argeș, Romania, August 24–27, 2009, pp. 426–451.

[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](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](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](http://link.springer.de/link/service/series/0558/bibs/2493/24930069.htm).