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.
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 [11], 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 \(x_1, \ldots, x_n\), 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 a *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, 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 \( \{1, \ldots, n\} \) 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.

```plaintext
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 in 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 \( \text{level}_c \), *predecessors*, *successors*, *peers* and \( \text{count}_c \), 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 \( \Pi_h = (O, \sigma_1, \ldots, \sigma_n, \delta, I_{out}) \), 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,
   - \( w_i, 0 \subseteq O^* \) is the initial multiset of objects,
   - \( P_i \) is a finite set of multiset rewrite rules of the form: \( sx \rightarrow s'x' \uparrow u_{j} \downarrow v_{j} \leftrightarrow w \wedge y \rightarrow z \), where \( s, s' \in Q_i \), \( x, x' \subseteq O^* \), \( u_j \subseteq O^*_v \), \( v_j \subseteq O^*_u \), \( w, y \subseteq O^*_w \), \( y \rightarrow z \subseteq O^*_y \), and \( z \rightarrow z \subseteq O^*_z \), with the restriction that \( z = \lambda \) for all \( i \in \{1, \ldots, n\} \backslash \mathcal{I}_{out} \).

3. \( \mathcal{D} \) is a set of dag parent-child arcs on \( \{1, \ldots, n\} \), i.e., \( \mathcal{D} \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\} \), representing duplex communication channels between cells;

4. \( \mathcal{I}_{out} \subseteq \{1, \ldots, n\} \) indicates the output cells, the only cells allowed to send objects to the “environment”.

**Definition 3 (Neural P systems)** A neural P system (of order \( n \geq 1 \), in short an \( nP \) system, is a system \( \Pi_n = (O, \sigma_1, \ldots, \sigma_n, \mathcal{S}_{syn}, \mathcal{I}_{out}) \), 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: \( s x \rightarrow s' x' y_{go} z_{out} \), where \( s, s' \in Q_i, x, x' \in O^*, y_{go} \in O_{go}^*, z_{out} \in O_{out}^* \), with the restriction that \( z_{out} = \lambda \) for all \( i \in \{1, \ldots, n\} \setminus \{i_{out}\} \).

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 \( \text{syn} \) 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 \( s_{go} \). 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

<table>
<thead>
<tr>
<th>Node</th>
<th>level ( e )</th>
<th>predecessors</th>
<th>successors</th>
<th>peers</th>
<th>count ( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>3,7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2,4,5</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3,7</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3,7</td>
<td>6</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4,5</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>4,5</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 3: Left: a graph with commander \( c = 1, e_1 = 3 \); Right: table with node levels, predecessors, successors, peers and \( c \)'s.
may be relayed to all cells, as necessary. Eventually, all cells in the squad set \( F \) will enter the designated firing state \( s_0 \), 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 \( \sigma_c \) and a set of squad cells \( F \) to be synchronized. We extend its underlying dag or digraph structure by adding a new sergeant cell \( \sigma_{n+1} \).

- For hP systems, we extend the original dag by an arc \((\sigma_{n+1}, \sigma_i)\), for each \( i \in F \cup \sigma_c \).
- For nP systems, we extend the original digraph by an arc \((\sigma_c, \sigma_{n+1})\) and by an arc \((\sigma_{n+1}, \sigma_i)\), 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 \( \sigma_c \) is marked by one object \( a \), each squad cell is marked by one object \( t \) (this can include the commander \( \sigma_c \) or the sergeant \( \sigma_{n+1} \), or both) and the sergeant \( \sigma_{n+1} \) 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 $\alpha = \text{min}$ and transfer mode $\beta = \text{repl}$.

0. For state $s_0$, the rules will run under these priorities:

1) $s_0a \rightarrow s_0b_{go}$
2) $s_0bct \rightarrow s_0f_{ff_{go}}$
3) $s_0bc \rightarrow s_0f_{go}$
4) $s_0bft \rightarrow s_1$
5) $s_0ft \rightarrow s_1$
6) $s_0bf \rightarrow s_0$
7) $s_0b \rightarrow s_0$

**Example 6.** Figure 4 illustrates this algorithm for a system which consists of seven cells $\{\sigma_1, \ldots, \sigma_7\}$. The commander cell is $\sigma_5$ 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 $\sigma_8$ 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.

![Figure 4: An hP system extended for Algorithm 2. Right: Traces for the hP system of Figure 1.](image-url)

**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, \(\sigma_c\) 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 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 \(\{s_{8}, s_{9}\}\).
- \(s_{0} = s_{9}\) 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;
- \(w_{l,0} = \{f\}\) for all \(\sigma_l \in F \setminus \{c\}\);
- \(w_{l,0} = \emptyset\), for all \(\sigma_l \in \Gamma \setminus (F \cup \{c\})\);
• Priority rules, in the weak interpretation:

0. For state $s_0$:
   1) $s_0a \rightarrow s_1ad_{go}$
   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) $s_4aeее \rightarrow s_6aее$
   5) $s_4ееее \rightarrow s_6e$
   6) $s_4a \rightarrow s_5ak$
   7) $s_4l \rightarrow s_5lh_{go}$

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) $s_5l \rightarrow s_5lh_{go}$
   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 $\sigma_i$ 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 $\sigma_i$’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 $\sigma_i = \sigma_i$;
  ○ 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 $\sigma_i$. By induction:
• At step $level_c(i)$, $\sigma_i$ (except the commander) receives a total of $count_c(i)$ 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 $f$. 

Lemma 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 $level_c(i) + 2$. 

\[ \text{Algorithm 4 (FSSP-II: Convergecasts from terminal nodes)} \]

**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 $a$;
  - $e_c + 2$ 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 $e$’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 $\sigma$ indicates its necessary synchronization delay.

A terminal 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 $e$ 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 $\sigma_i$, 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.

\[ \text{Algorithm 5 (FSSP-III: Second broadcast from the commander)} \]
Precondition: As described in the postcondition of Algorithm 4.

Postcondition:

- The end state is $s_8$.
- A cell $\sigma_i$ has
  - $\text{count}_c(i)$ copies of $a$;
  - $(e_c + 1 - \text{level}_c(i))\text{count}_c(i)$ 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 $\sigma_i$ receives a total of $(e_c + 2 - \text{level}_c(i))\text{count}_c(i)$ copies of $e$’s from its predecessors, reduces this count by $\text{count}_c(i)$ (i.e., the count of $a$’s), forwards the remaining $(e_c + 1 - \text{level}_c(i))\text{count}_c(i)$ copies of $e$’s to all its successors and creates for itself $(e_c + 1 - \text{level}_c(i))\text{count}_c(i)$ 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. \hfill \Box

Lemma 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. \hfill \Box

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 $\text{count}_c(i)$ (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, $(e_c + 1 - \text{level}_c(i))\text{count}_c(i)$, and the number of $a$’s, $\text{count}_c(i)$. 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 $\sigma_i$ enters $s_9$, if $\sigma_i$ has one $f$, or $s_0$, otherwise. At the same time, all existing objects are removed. \hfill \Box

Lemma 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 $(e_c + 1 - \text{level}_c(i)) + 1 = e_c + 2 - \text{level}_c(i)$. \hfill \Box

Theorem 11. For each cell $\sigma_i$, 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 $\sigma_c$.
Proof: The result is obtained by summing the individual running times of the three phases, as given by Lemmas 7, 8, 9 and 10 \((\text{level}_c(i) + 2) + (5e_c - \text{level}_c(i)) + (\text{level}_c(i) + 3) + (e_c + 2 - \text{level}_c(i)) = 6e_c + 7\)

\[\square\]

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 \textit{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 \(c = 6, e_6 = 3, F = \{\sigma_1, \sigma_4, \sigma_5, \sigma_7, \sigma_9, \sigma_{10}\}\).
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


