CDMTCS
Research
Report
Series

An Adaptive Algorithm for P System Synchronization

Michael J. Dinneen
Yun-Bum Kim
Radu Nicolescu

Department of Computer Science,
University of Auckland,
Auckland, New Zealand

CDMTCS-404
May 2011

Centre for Discrete Mathematics and
Theoretical Computer Science

An Adaptive Algorithm for P System Synchronization

Michael J. Dinneen, Yun-Bum Kim and Radu Nicolescu

Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand

{mjd,yun,radu}@cs.auckland.ac.nz

Abstract

We present an improved solution for the Firing Squad Synchronization Problem (FSSP) for digraph-based P systems. We improve our previous FSSP algorithm by allowing the general to delegate a more central cell in the P system to send the final command to synchronize. With e being the eccentricity of the general and r denoting the radius of the underlying digraph, our new algorithm guarantees to synchronize all cells of the system, between $e + 2r + 3$ steps (for all trees structures and many digraphs) and up to $3e + 7$ steps, in the worst case for any digraph. Empirical results show our new algorithm for tree-based P systems yields at least 20% reduction in the number of steps needed to synchronize over the previous best-known algorithm.

Keywords: cellular automata, firing squad synchronization, P systems.

1 Introduction

The *Firing Squad Synchronization Problem* (FSSP) is one of the best studied problems for cellular automata, originally proposed by Myhill in 1957 [11]. The initial problem involves finding a cellular automaton, such that after the "firing" order is given by the general, after some finite time, all the cells in a line enter a designated *firing* state, *simultaneously* and *for the first time*. For an array of length n with the general at one end, minimal time $(2n - 2)$ solutions was presented by Goto [6], Waksman [18] and Balzer [2]. Several variations of the FSSP have been proposed and studied [12, 15]. The FSSP have been proposed and studied for variety of structures [10, 13, 7, 4].

In the field of membrane computing, deterministic solutions to the FSSP for a tree-based P system have been presented by Bernardini et al. [3] and Alhazov et al. [1]. For digraph-based P systems, we presented a deterministic solution in [5] for the generalized FSSP (in which the general is located at an arbitrary cell of the digraph), which runs in $3e + 11$ steps, where e is the eccentricity of the general.

In this paper, we present an improved FSSP solution for tree-based P systems, where the key improvement comes in having the general delegate a more central cell, as an alternative to itself, to broadcast the final "firing" order, to enter the firing state. We also give details on how to use this approach to improve the synchronization time of digraph-based P systems.

It is well known in cellular automata [17], where "signals" with propagating speeds $1/1$ and $1/3$ are used to find a half point of one-dimensional arrays; the signal with speed $1/1$ is reflected and meets the signal with speed $1/3$ at half point. We generalize the idea used in cellular automata to find the center of a tree that defines the membrane structure of a P system.

Let r denote the radius of the underlying graph of a digraph, where $e/2 \leq r \leq e$. Our new algorithm is guaranteed to synchronize in t steps, where $e + 2r + 3 \leq t \leq 3e + 7$. In fact, the lower bound is achieved, for all digraphs that are trees. In addition to our FSSP solution, determining a center cell has many potential real world applications, such as *facility location problems* and *broadcasting.*

The rest of the paper is organized as follows. In Section 2, we give some basic preliminary definitions including our P system model and formally introduce the synchronization problem that we solve. In Section 3, we provide a detailed P system specification for solving the FSSP for tree-based P systems. In Section 4, we provide a detailed P system specification for solving the FSSP for digraph-based P systems. Finally, in Section 5, we summarize our results and conclude with some open problems.

2 Preliminary

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

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

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

Recall that a (free or unrooted) tree has either one or two *center nodes*—any node with minimum eccentricity. We denote a tree $T = (X, A)$, rooted at node $g \in X$ by T_g. The height of a node x in T_g is denoted by $\texttt{height}_g(x)$. For a tree T_g, we define the *middle node* to be the center node closest to g of the underlying tree T of T_g. Let $T_g(x)$ denote the subtree rooted at node x in T_g.

Given nodes x and y, if $y \in \texttt{Neighbor}(x)$ and $\texttt{depth}_g(y) = \texttt{depth}_g(x) + 1$, then x is a *predecessor* of y and y is a *successor* of x. Similarly, a node z is a *peer* of x, if $z \in \texttt{Neighbor}(x)$ and $\texttt{depth}_g(z) = \texttt{depth}_g(x)$. Note that, for node x, the set of peers and the set of successors are disjoint with respect to g. For node x, $\texttt{Pred}_g(x) = \{y \mid y \text{ is a predecessor of } x\}$, $\texttt{Peer}_g(x) = \{y \mid y \text{ is a peer of } x\}$ and $\texttt{Succ}_g(x) = \{y \mid y \text{ is a successor of } x\}$.

Definition 1. A *P system* of order n with *duplex* channels and *cell states* is a system $\Pi = (O, K, \delta)$, where:

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

2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ is a finite set of *cells*;

3. δ is an *irreflexive* binary relation on K, which represents a set of structural arcs between cells, with duplex communication capabilities.

Each cell, $\sigma_i \in K$, has the initial configuration $\sigma_i = (Q_i, s_{i0}, w_{i0}, R_i)$, and the current configuration $\sigma_i = (Q_i, s_i, w_i, R_i)$, where:

- Q_i is a finite set of *states*;

- $s_{i0} \in Q_i$ is the *initial state*; $s_i \in Q_i$ is the *current state*;

- $w_{i0} \in O^*$ is the *initial content*; $w_i \in O^*$ is the *current content*; note that, for $o \in O$, $|w_i|_o$ denotes the *multiplicity* of object o in the multiset w_i;

- R_i is a finite *ordered* set of multiset rewriting *rules* (with *promoters*) of the form: $s\ x \rightarrow_\alpha s'\ x'\ (u)_\beta \mid z$, where $s, s' \in Q$, $x, x', u \in O^*$, $z \in O^*$ is the promoter [9], $\alpha \in \{\texttt{min}, \texttt{max}\}$ and $\beta \in \{\uparrow, \downarrow, \updownarrow\}$. For convenience, we also allow a rule to contain zero or more instances of $(u)_\beta$. For example, if $u = \lambda$, i.e. the *empty* multiset of objects, this rule can be abbreviated as $s\ x \rightarrow_\alpha s'\ x'$.

A cell *evolves* by applying one or more rules, which can change its content and state and can send objects to its neighbors. For a cell $\sigma_i = (Q_i, s_i, w_i, R_i)$, a rule $s\ x \rightarrow_\alpha s'\ x'\ (u)_\beta \mid z \in R_i$ is *applicable*, if $s = s_i$, $x \subseteq w_i$, $z \subseteq w_i$, $\delta(i) \neq \emptyset$ for $\beta = \downarrow$, $\delta^{-1}(i) \neq \emptyset$ for $\beta = \uparrow$ and $\delta(i) \cup \delta^{-1}(i) \neq \emptyset$ for $\beta = \updownarrow$.

The application of a rule transforms the current state s to the *target state* s' transforms multiset x to x' and sends multiset u as specified by the transfer operator β (as further described below). Note that, multisets x' and u will not be visible to other applicable rules in this same step, but they will be visible after all the applicable rules have been applied.

The rules are applied in the *weak priority* order [14], i.e. (1) higher priority applicable rules are applied before lower priority applicable rules, and (2) a lower priority applicable rule is applied only if it indicates the same target state as the previously applied rules.

The rewriting operator $\alpha = \texttt{max}$ indicates that an applicable rewriting rule of R_i is applied as many times as possible. The rewriting operator $\alpha = \texttt{min}$ indicates that an applicable rewriting rule of R_i is applied once. If the right-hand side of a rule contains $(u)_\beta$, $\beta \in \{\uparrow, \downarrow, \updownarrow\}$, then for each application of this rule, a copy of multiset u is replicated and sent to each cell $\sigma_j \in \delta^{-1}(i)$ if $\beta = \uparrow$, $\sigma_j \in \delta(i)$ if $\beta = \downarrow$ and $\sigma_j \in \delta(i) \cup \delta^{-1}(i)$ if $\beta = \updownarrow$.

All applicable rules are applied in one *step*. An *execution* of a P system is a sequence of steps, that starts from the initial configuration. An execution *halts* if no further rules are applicable for all cells.

Problem 2. We formulate the FSSP to P systems as follows:
Input: An integer $n \geq 2$ and an integer g, $1 \leq g \leq n$.
Output: A class $\mathcal{C}$ of P systems that satisfies the following two conditions for any weakly connected digraph (X, A), isomorphic to the structure of a member of $\mathcal{C}$ with $n = |X|$ cells.

1. Cell σ_g is the only cell with an applicable rule (i.e. σ_g can evolve) from its initial configuration.

2. There exists state $s_f \in Q_i$, for all $\sigma_i \in K$, such that during the last step of the system's execution, all cells enter state s_f, simultaneously and for the first time.

We want to find a general-purpose solution to the FSSP that synchronizes in the fewest number of steps, as a function of some of the natural structural properties of a weakly-connected digraph (X, A), such as the eccentricity of node $g \in X$ in the communication graph defined by A.

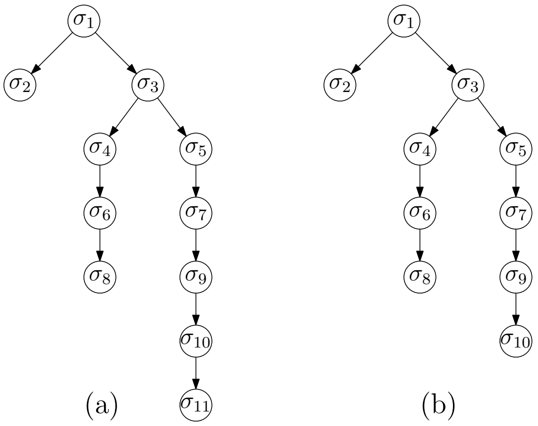


Figure 1: (a) a tree with the center σ_5; (b) a tree with two centers σ_3 and σ_5, σ_3 being the middle cell.

3 Deterministic FSSP solution for rooted trees

We first solve Problem 2 for the subclass of weakly-connected digraphs (X, A), where the underlying graph of (X, A) is a tree. This section is organized as follow. In Section 3.1, we present the P system for solving the FSSP for trees rooted at the general. In order to help the comprehension of our FSSP algorithm, we provide a trace of the FSSP algorithm in Table 1. Phase I of our FSSP algorithm is described in Section 3.2, which finds the *middle* cell (i.e. a center of a tree, closest to the root) and determines the height of the middle cell. Phase II of our FSSP algorithm is described in Section 3.3, which broadcasts the "command" that prompts all cells to enter the firing state. Finally, in Section 3.4, we present some empirical results that show improvements of our algorithm over the previously best-known FSSP algorithms for tree-based P systems [1, 5].

Table 1: The traces of the FSSP algorithm on a P system with the membrane structure defined by the tree shown in Figure 1 (a), where the general is σ_1 and the middle cell is σ_5. The step in which the Phase I ends (or the Phase II begins) is indicated by the shaded table cells.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}
0	$s_0\ o$	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ ahou$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
2	$s_2\ ace^2h^2ou$	$s_4\ a$	$s_1\ ah$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0	s_0	s_0
3	$s_2\ aeh^3o^2$	$s_4\ a$	$s_2\ ae^2h^2$	$s_1\ ah$	$s_1\ ah$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0
4	$s_2\ aeh^4o^3$	$s_4\ a$	$s_2\ ae^2h^3$	$s_2\ aeh^2$	$s_2\ aeh^2$	$s_1\ ah$	$s_1\ ah$	$s_0\ b$	$s_0\ b$	s_0	s_0
5	$s_2\ a^2h^4$	$s_4\ ao$	$s_2\ ae^2h^4o$	$s_2\ aeh^3$	$s_2\ aeh^3$	$s_2\ aceh^2$	$s_2\ aeh^2$	$s_4\ a$	$s_1\ ah$	$s_0\ b$	s_0
6	$s_3\ ah^4$	$s_4\ a$	$s_2\ ae^2h^5o$	$s_2\ aceh^4$	$s_2\ aeh^4$	$s_2\ a^2h^2$	$s_2\ aeh^3$	$s_4\ a$	$s_2\ aeh^2$	$s_1\ ah$	$s_0\ b$
7	$s_4\ a$	$s_4\ a$	$s_2\ ace^2h^6o$	$s_2\ a^2h^4$	$s_2\ aeh^5$	$s_3\ ah^2$	$s_2\ aeh^4$	$s_4\ a$	$s_2\ aeh^3$	$s_2\ aceh^2$	$s_4\ a$
8	$s_4\ a$	$s_4\ a$	$s_2\ aeh^7o^2$	$s_3\ ah^4$	$s_2\ aeh^6$	$s_4\ a$	$s_2\ aeh^5$	$s_4\ a$	$s_2\ aceh^4$	$s_2\ a^2h^2$	$s_4\ a$
9	$s_4\ a$	$s_4\ a$	$s_2\ aeh^8o^3$	$s_4\ a$	$s_2\ aeh^7$	$s_4\ a$	$s_2\ aceh^6$	$s_4\ a$	$s_2\ a^2h^4$	$s_3\ ah^2$	$s_4\ a$
10	$s_4\ a$	$s_4\ a$	$s_2\ a^2h^8$	$s_4\ ao$	$s_2\ aceh^8o$	$s_4\ a$	$s_2\ a^2h^6$	$s_4\ a$	$s_3\ ah^4$	$s_4\ a$	$s_4\ a$
11	$s_4\ a$	$s_4\ a$	$s_3\ ah^8$	$s_4\ a$	$s_2\ a^3h^8$	$s_4\ a$	$s_3\ ah^6$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$
12	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ ah^8$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$
13	$s_4\ a$	$s_4\ a$	$s_4\ av^4$	$s_4\ a$	$s_5\ aw^4$	$s_4\ a$	$s_4\ av^4$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$
14	$s_4\ av^3$	$s_4\ a$	$s_5\ aw^3$	$s_4\ av^3$	$s_5\ av^6w^3$	$s_4\ a$	$s_5\ aw^3$	$s_4\ a$	$s_4\ av^3$	$s_4\ a$	$s_4\ a$
15	$s_5\ aw^2$	$s_4\ av^2$	$s_5\ av^4w^2$	$s_5\ aw^2$	$s_5\ av^6w^2$	$s_4\ av^2$	$s_5\ av^2w^2$	$s_4\ a$	$s_5\ aw^2$	$s_4\ av^2$	$s_4\ a$
16	$s_5\ avw$	$s_5\ aw$	$s_5\ av^4w$	$s_5\ avw$	$s_5\ av^6w$	$s_5\ aw$	$s_5\ av^2w$	$s_4\ av$	$s_5\ avw$	$s_5\ aw$	$s_4\ av$
17	s_6	s_6	s_6	s_6	s_6	s_6	s_6	s_6	s_6	s_6	s_6

3.1 P systems for solving the FSSP for rooted trees

Given a tree (X, A) and $g \in X$, our FSSP algorithm is implemented using the P system $\Pi = (O, K, \delta)$ of order $n = |X|$, where:

1. $O = \{a, b, c, e, h, o, v, w\}$.
2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.
3. δ is a rooted tree, with an underlying graph isomorphic to (X, A), where the general $\sigma_g \in K$ (the root of δ) corresponds to $g \in X$.

All cells have the same set of states, the same set of rules and start at the same initial *quiescent* state s_0, but with different initial contents. The first output condition of Problem 2 will be satisfied by our chosen set of rules.

For each cell $\sigma_i \in K$, its initial configuration is $\sigma_i = (Q, s_0, w_{i0}, R)$ and its final configuration at the end of the execution is $\sigma_i = (Q, s_6, \emptyset, R)$, where:

- $Q = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$, where s_0 is the initial quiescent state and s_6 is the *firing* state.
- $w_{i0} = \begin{cases} \{o\} & \text{if } \sigma_i = \sigma_g, \\ \emptyset & \text{if } \sigma_i \neq \sigma_g. \end{cases}$
- R is defined by the following rulesets.

  Rules used in Phase I: all the rules in states s_0, s_1, s_2, s_3 and Rule 4.6 in state s_4.

  Rules used in Phase II: all the rules in states s_4 and s_5, except Rule 4.6.

0. Rules in state s_0:
   1. $s_0\ o \rightarrow_{\texttt{max}} s_1\ ahou\ (b)_{\downarrow}$
   2. $s_0\ b \rightarrow_{\texttt{max}} s_1\ ah\ (e)_{\uparrow}\ (b)_{\downarrow}$
   3. $s_0\ b \rightarrow_{\texttt{max}} s_4\ a\ (ce)_{\uparrow}$
1. Rules in state s_1:
   1. $s_1\ a \rightarrow_{\texttt{max}} s_2\ ah$
2. Rules in state s_2:
   1. $s_2\ aaa \rightarrow_{\texttt{max}} s_4\ a$
   2. $s_2\ aa \rightarrow_{\texttt{max}} s_3\ a$
   3. $s_2\ ceu \rightarrow_{\texttt{max}} s_2$
   4. $s_2\ ce \rightarrow_{\texttt{max}} s_2$
   5. $s_2\ aee \rightarrow_{\texttt{max}} s_2\ aeeh$
   6. $s_2\ aeooo \rightarrow_{\texttt{max}} s_2\ aa\ (o)_{\downarrow}$
   7. $s_2\ aeou \rightarrow_{\texttt{max}} s_2\ aa\ (o)_{\downarrow}$
   8. $s_2\ aeo \rightarrow_{\texttt{max}} s_2\ aehoo$
   9. $s_2\ ao \rightarrow_{\texttt{max}} s_2\ aaa$
   10. $s_2\ ae \rightarrow_{\texttt{max}} s_2\ aeh$
   11. $s_2\ a \rightarrow_{\texttt{max}} s_2\ aa\ (c)_{\uparrow}$
   12. $s_2\ u \rightarrow_{\texttt{max}} s_2$
3. Rules in state s_3:
   1. $s_3\ a \rightarrow_{\texttt{max}} s_4\ a$
   2. $s_3\ h \rightarrow_{\texttt{max}} s_4$
4. Rules in state s_4:
   1. $s_4\ hh \rightarrow_{\texttt{max}} s_5\ w\ (v)_{\updownarrow}$
   2. $s_4\ avv \rightarrow_{\texttt{max}} s_5\ aw\ (v)_{\updownarrow}$
   3. $s_4\ avv \rightarrow_{\texttt{max}} s_5\ aw$
   4. $s_4\ av \rightarrow_{\texttt{max}} s_6$
   5. $s_4\ v \rightarrow_{\texttt{max}} s_5\ w\ (v)_{\updownarrow}$
   6. $s_4\ o \rightarrow_{\texttt{max}} s_4$
5. Rules in state s_5:
   1. $s_5\ aww \rightarrow_{\texttt{max}} s_5\ aw$
   2. $s_5\ aw \rightarrow_{\texttt{max}} s_6$
   3. $s_5\ v \rightarrow_{\texttt{max}} s_6$
   4. $s_5\ o \rightarrow_{\texttt{max}} s_6$

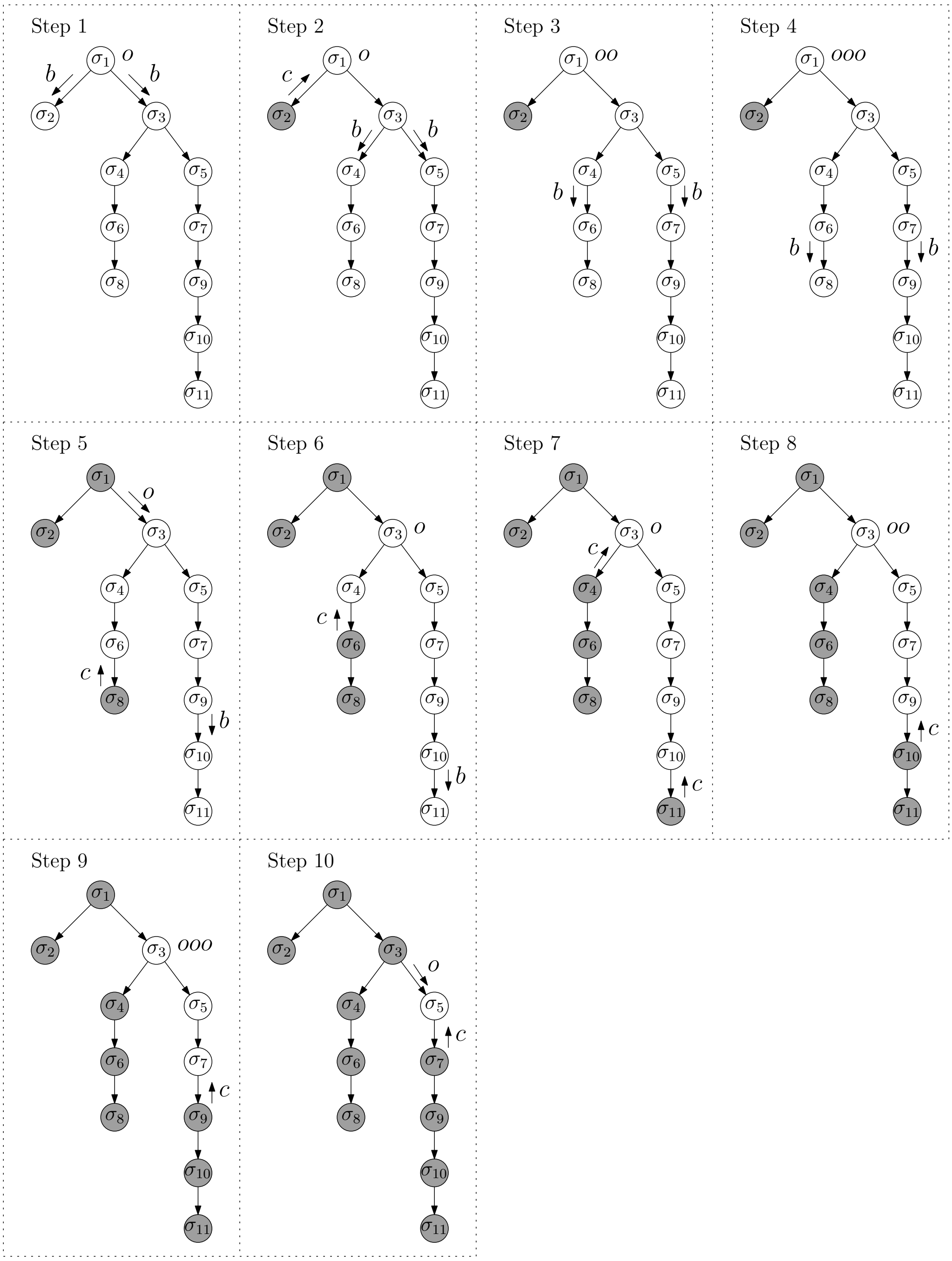

Step 1
σ1 o
b b
σ2 σ3 σ4 σ5 σ6 σ7 σ8 σ9 σ10 σ11
Step 2
σ1 o
c
b b
σ2 σ3 σ4 σ5 σ6 σ7 σ8 σ9 σ10 σ11
Step 3
σ1 oo
b b
σ2 σ3 σ4 σ5 σ6 σ7 σ8 σ9 σ10 σ11
Step 4
σ1 ooo
b b
σ2 σ3 σ4 σ5 σ6 σ7 σ8 σ9 σ10 σ11
Step 5
σ1
o
c
b
σ2 σ3 σ4 σ5 σ6 σ7 σ8 σ9 σ10 σ11
Step 6
σ1
σ3 o
c
b
σ2 σ4 σ5 σ6 σ7 σ8 σ9 σ10 σ11
Step 7
σ1
σ3 o
c
c
σ2 σ4 σ5 σ6 σ7 σ8 σ9 σ10 σ11
Step 8
σ1
σ3 oo
c
σ2 σ4 σ5 σ6 σ7 σ8 σ9 σ10 σ11
Step 9
σ1
σ3 ooo
c
σ2 σ4 σ5 σ6 σ7 σ8 σ9 σ10 σ11
Step 10
σ1
σ3
o
c
σ2 σ4 σ5 σ6 σ7 σ8 σ9 σ10 σ11

Figure 2: Propagations of symbols b, c and o, in a tree with one center. The symbols c and o meet at the middle cell σ_5. Cells that have sent symbol c or o are shaded. The propagation of symbol o to a shaded cell is omitted. In cell σ_j, $j \in \{1, 3\}$, $|w_j|_o - 1$ represents the number of steps since σ_j received symbol c from all of its children but one.

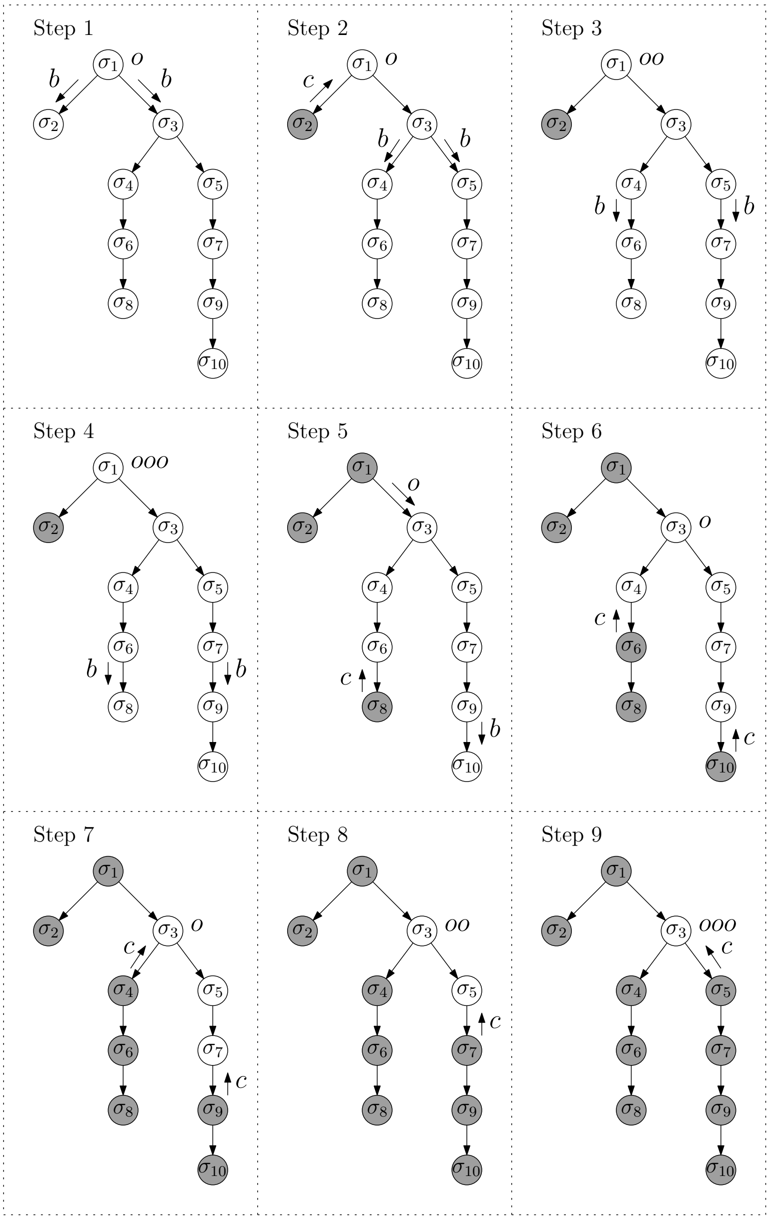


Figure 3: Propagations of symbols b, c and o, in a tree with two centers. The symbols c and o meet at the middle cell σ_3. Cells that have sent symbol c or o are shaded. The propagation of symbol o to a shaded cell is omitted. In cell σ_j, $j \in \{1, 3\}$, $|w_j|_o - 1$ represents the number of steps since σ_j received symbol c from all of its children but one.

3.2 Phase I: Find the middle cell of rooted trees

In this phase, a breadth-first search (BFS) is performed from the root, which propagates symbol b from the root to all other cells. When the symbol b from the BFS reaches a leaf cell, symbol c is *reflected* back up the tree. Starting from the root, the search for the middle cell is performed as described below, where symbol o represents the current *search pivot.* Note that symbol o's propagation speed is 1/3 of the propagation speed of symbols b and c; intuitively, this ensures that o and c meet in the middle cell.

We provide a visual description of the propagations of symbols b, c and o in Figure 2 (for a tree with one center) and Figure 3 (for a tree with two centers).

Details of Phase I

Objective: The objective of Phase I is to find the middle cell, σ_m, and its height, $\texttt{height}_g(m)$.

Precondition: Phase I starts with the initial configuration of P system Π, described in Section 3.1.

Postcondition: Phase I ends when σ_m enters state s_4. At the end of Phase I, the configuration of cell $\sigma_i \in K$ is (Q, s_4, w_i, R), where $|w_i|_a = 1$; $|w_i|_h = 2 \cdot \texttt{height}_g(i)$, if $\sigma_i = \sigma_m$.

Description: In Phase I, each cell starts in state s_0, transits through states s_1, s_2, s_3, and ends in state s_4; a cell in state s_4 will ignore any symbol o that it may receive.

The behaviors of cells in this phase are described below.

- **Propagation of symbol b:** The root cell sends symbol b to all its children (Rule 0.1). An internal cell forwards the received symbol b to all its children (Rule 0.2) After applying Rule 0.1 or 0.2, each of these non-leaf cells produces a copy of symbol h in each step, until it receives symbol c from all its children (Rules 1.1, 2.5 and 2.10).

- **Propagation of symbol c:** If a leaf cell receives symbol b, then it sends symbol c to its parent (Rule 0.3) and enters state s_4 (the end state of Phase I). If a non-leaf cell receives symbol c from all its children, then it sends symbol c to its parent (Rule 2.11), consumes all copies of symbol h and enters state s_4 (Rule 3.2).

Note, when a cell applies Rule 0.2 or 0.3, it sends one copy of symbol e up to its parent. A copy of symbol e is consumed with a copy of symbol c by Rule 2.4. Hence, $|w_i|_e = k$ indicates the number of σ_i's children that have not sent symbol c to σ_i.

- **Propagation of symbol o:** The root cell initially contains the symbol o. We denote σ_j as the current cell that contains symbol o and has not entered state s_4.

  Assume, at step t, σ_j received symbol c from all but one subtree rooted at σ_v. Starting from step $t+1$, σ_j produces a copy of symbol o in each step, until it receives symbol c from σ_v (Rule 2.8), That is, $|w_j|_o - 1$ indicates the number of steps since σ_j received symbol c from all of its children except σ_v.

If σ_j receives symbol c from σ_v by step $t+2$, i.e. $|w_j|_o \leq 3$, then σ_j is the middle cell; σ_j keeps all copies of symbol h and enters state s_4 (Rule 2.1). Otherwise, σ_j sends a copy of symbol o to σ_v at step $t+3$ (Rule 2.6 or 2.7); in the subsequent steps, σ_j consumes all copies of symbol h and enters state s_4 (Rules 2.2 and 3.2). Note, using current setup, σ_j cannot send a symbol to a specific child; σ_j has to send a copy of symbol o to all its children. However, all σ_j's children, except σ_v, would have entered state s_4.

Proposition 3 indicates the step in which σ_m receives symbol c from all its children and Proposition 4 indicates the number of steps needed to propagate symbol o from σ_g to σ_m.

Proposition 3. *Cell σ_m receives the symbol c from all its children by step* $\mathtt{height}_g(g) + \mathtt{height}_g(m)$.

Proof. Cell σ_m is at distance $\mathtt{height}_g(g) - \mathtt{height}_g(m)$ from σ_g, hence σ_m receives symbol b in step $\mathtt{height}_g(g) - \mathtt{height}_g(m)$. In the subtree rooted at σ_m, the propagations of the symbol b from σ_m to its farthest leaf and the symbol c reflected from the leaf to σ_m take $2 \cdot \mathtt{height}_g(m)$ steps. Thus, σ_m receives symbol c from all its children by step $\mathtt{height}_g(g) - \mathtt{height}_g(m) + 2 \cdot \mathtt{height}_g(m) = \mathtt{height}_g(g) + \mathtt{height}_g(m)$. □

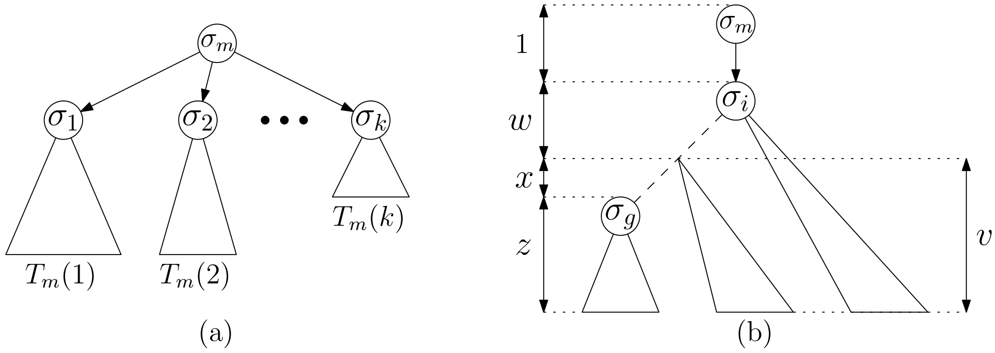


Figure 4: (a) k subtrees of σ_m, $T_m(1), T_m(2), \ldots, T_m(k)$. (b) The structure of subtree $T_m(j)$, which contains σ_g.

Proposition 4. *The propagation of the symbol o from σ_g to σ_m takes at most* $\mathtt{height}_g(g) + \mathtt{height}_g(m)$ *steps.*

Proof. For a given tree T_g, rooted at σ_g, we construct a tree T_m, which re-roots T_g at σ_m. Recall, $T_m(i)$ denotes a subtree rooted at σ_i in T_m. Assume that σ_m has $k \geq 2$ subtrees, $T_m(1), T_m(2), \ldots, T_m(k)$, such that $\mathtt{height}_m(1) \geq \mathtt{height}_m(2) \geq \cdots \geq \mathtt{height}_m(k)$ and $\mathtt{height}_m(1) - \mathtt{height}_m(2) \leq 1$. Figure 4 (a) illustrates the subtrees of σ_m.

Assume $T_m(i)$ is a subtree of σ_m, which contains σ_g. In $T_m(i)$, let z be the height of σ_g and $x + w \geq 0$ be the distance between σ_g and σ_i. Figure 4 (b) illustrates the z, x and w in $T_m(i)$.

To prove Proposition 4, we determine the number of steps needed to propagate symbol o from σ_g to σ_m. In $T_m(i)$, let p be a path from σ_i to its farthest leaf and t be the number of steps needed to propagate symbol o from σ_g to σ_m. Note, $\mathtt{height}_m(m) = \mathtt{height}_g(m)$ and $x + w + 1 = \mathtt{height}_g(g) - \mathtt{height}_g(m)$.

- If σ_g is a part of path p, then $z + x + w + 1 = \texttt{height}_m(i) + 1 = \texttt{height}_m(m) - j$, $j \geq 0$, and $t = 2z + 3(x + w + 1)$. Hence,

$$\begin{aligned} t &= 2(z + x + w + 1) + (x + w + 1) \\ &= 2(\texttt{height}_m(m) - j) + (\texttt{height}_g(g) - \texttt{height}_g(m)) \\ &= \texttt{height}_g(g) + \texttt{height}_g(m) - 2j \end{aligned}$$

- If σ_g is not a part of p, then $z + x + w + 1 < v + w + 1 = \texttt{height}_m(i) + 1 = \texttt{height}_m(m) - j$, $j \geq 0$, and $t = x + 2v + 3(w + 1)$. Hence,

$$\begin{aligned} t &= 2(v + w + 1) + (x + w + 1) \\ &= 2(\texttt{height}_m(m) - j) + (\texttt{height}_g(g) - \texttt{height}_g(m)) \\ &= \texttt{height}_g(g) + \texttt{height}_g(m) - 2j \end{aligned}$$

□

Proposition 5. *Phase I takes* $\texttt{height}_g(g) + \texttt{height}_g(m) + 2$ *steps.*

Proof. From Propositions 3 and 4, symbols o and c meets in σ_m at step $\texttt{height}_g(g) + \texttt{height}_g(m)$. Cell σ_m enters state s_4 by applying Rule 2.9 and 2.1, which takes two steps. Thus, Phase I takes $\texttt{height}_g(g) + \texttt{height}_g(m) + 2$ steps. □

3.3 Phase II: Determine the step to enter the firing state

Phase II begins immediately after Phase I. In Phase II, the middle cell broadcasts the "firing" order, which prompts receiving cells to enter the firing state. In general, the middle cell does not have direct communication channels to all cells. Thus, the firing order has to be relayed through intermediate cells, which results in some cells receiving the order before other cells. To ensure that all cells enter the firing state simultaneously, each cell needs to determine the number of steps it needs to wait, until all other cells receive the order.

The firing order is paired with a *counter*, which is initially set to the eccentricity of the middle cell. Propagating an order from one cell to another decrements its current counter by one. The current counter of the received order equals the number of remaining steps before all other cells receive the order. Hence, each cell waits according to the current counter, before it enters the firing state. Figure 5 illustrates the propagation of the firing order.

Details of Phase II

Objective: The objective of Phase II is to determine the step to enter the firing state, such that during the last step of Phase II, i.e. the system's execution, all cells enter the firing state, simultaneously and for the first time.

Precondition: Phase II starts with the postcondition of Phase I, described in Section 3.2.

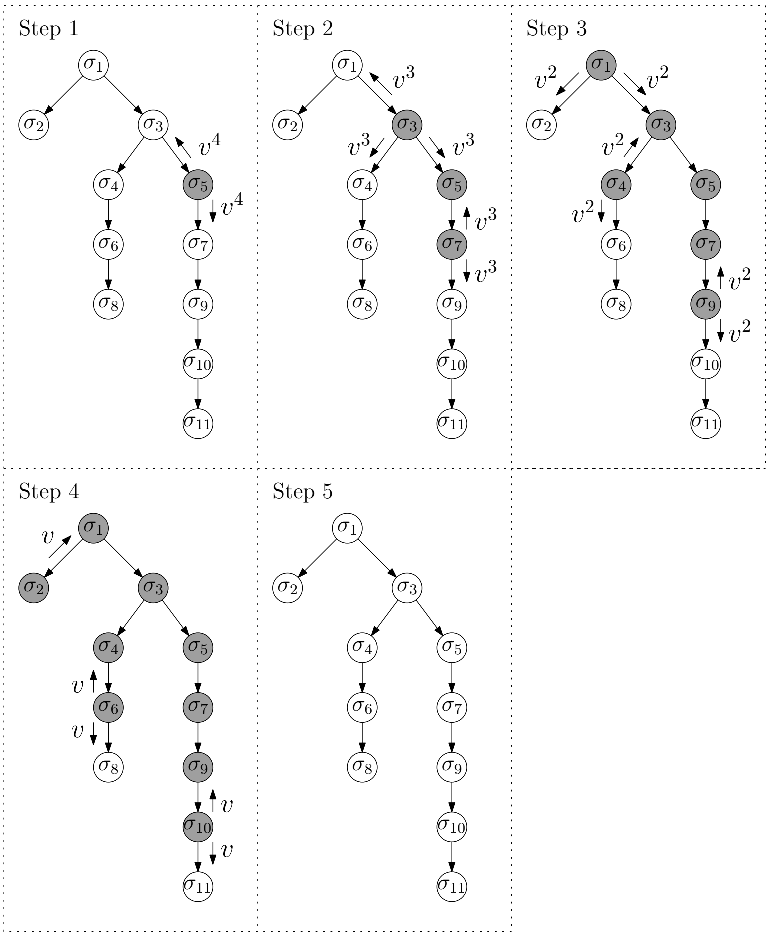


Figure 5: Propagations of the firing order from the middle cell, σ_5, where the counter is represented by the multiplicity of symbol v. Cells that have propagated the order are shaded.

Postcondition: Phase II ends when all cells enter the firing state s_6. At the end of Phase II, the configuration of cell $\sigma_i \in K$ is $(Q, s_6, \emptyset, R)$.

Description: The behaviors of the middle cell σ_m and a non-middle cell, $\sigma_i \neq \sigma_m$, in this phase are as follow. We also indicate which rules accomplish the described behaviors.

- We first describe the behavior of σ_m. For every two copies of symbol h, σ_m produces one copy of symbol w and sends one copy of symbol v to all its neighbors (Rule 4.1). In the next sequence of steps, σ_m consumes one copy of symbol w (Rule 5.1). If σ_m consumes all copies of symbol w, then σ_m enters the firing state (Rule 5.2).

- Next, we describe the behavior of $\sigma_i \neq \sigma_m$. Let $k_i \geq 1$ denote the multiplicity of symbol v that σ_i receives for the first time. If $k_i = 1$, then σ_i enters the firing state (Rule 4.4). If $k_i \geq 2$, then σ_i consumes k_i copies of symbol v, produces $k_i - 1$ copies of symbol w and sends $k_i - 1$ copies of symbol v to all its neighbors (Rules 4.2, 4.3 and 4.5); in each subsequent step, σ_i consumes one copy of symbol w (Rule 5.1) and σ_i enters the firing state (Rule 5.2), after all copies of symbol w is consumed.

Proposition 6. *Cell σ_m produces $\mathtt{height}_g(m)$ copies of symbol w and sends $\mathtt{height}_g(m)$ copies of symbol v to all is neighbors.*

Proof. At the beginning of Phase II, σ_m contains $2 \cdot \mathtt{height}_g(m)$ copies of symbol h. As described earlier, for every two copies of the symbol h that σ_m consumes, σ_m produces one copy of symbol w and sends one copy of symbol v to all its neighbors. □

Proposition 7. *Cell σ_i receives k copies of symbol v at step t and sends $k-1$ copies of symbol v to all its neighbors at step $t+1$, where $k = \mathtt{height}_g(m) - \mathtt{depth}_m(i) + 1$ and $t = \mathtt{height}_g(g) + \mathtt{height}_g(m) + \mathtt{depth}_m(i) + 2$.*

Proof. Proof by induction on $\mathtt{depth}_m(i) \geq 1$. First, σ_m sends $\mathtt{height}_g(m)$ copies of symbol v to all its neighbors. Thus, each cell σ_i, at distance 1 from σ_m, receives $\mathtt{height}_g(m)$ copies of symbol v. By Rules 4.3, 4.4, 4.7 and 4.8, σ_i consumes $\mathtt{height}_g(m)$ copies of symbol v, produces $\mathtt{height}_g(m)-1$ copies of symbol w and sends $\mathtt{height}_g(m)-1$ copies of symbol v to all its neighbors.

Assume that the induction hypothesis holds for each cell σ_j at distance $\mathtt{depth}_m(j)$. Consider cell σ_i, where $\mathtt{depth}_m(i) = \mathtt{depth}_m(j) + 1$. By the induction hypothesis, cell $\sigma_j \in \mathtt{Neighbor}(i)$, sends $\mathtt{height}_g(m) - \mathtt{depth}_m(j) = \mathtt{height}_g(m) - \mathtt{depth}_m(i) + 1$ copies of symbol v, such that σ_i receives $\mathtt{height}_g(m) - \mathtt{depth}_m(i) + 1$ copies of symbol v. By Rules 4.3, 4.4, 4.7 and 4.8, σ_i consumes $\mathtt{height}_g(m) - \mathtt{depth}_m(i) + 1$ copies of symbol v, produces $\mathtt{height}_g(m) - \mathtt{depth}_m(i)$ copies of symbol w and sends $\mathtt{height}_g(m) - \mathtt{depth}_m(i)$ copies of symbol v to all its neighbors. □

Proposition 8. *Phase II takes $\mathtt{height}_g(m) + 1$ steps.*

Proof. Each cell σ_i receives $\mathtt{height}_g(m) - \mathtt{depth}_m(i) + 1$ copies of symbol v at step $\mathtt{height}_g(g) + \mathtt{height}_g(m) + \mathtt{depth}_m(i) + 2$.

Consider σ_j, where $\mathtt{depth}_m(j) = \mathtt{height}_g(m)$. Cell σ_j receives one copy of symbol v. As described earlier, if a cell receives one copy of symbol v, then it enters the firing state at the next step. Hence, σ_j enters the firing state at step $\mathtt{height}_g(g) + 2 \cdot \mathtt{height}_g(m) + 3$.

Consider σ_k, where $\mathtt{depth}_m(k) < \mathtt{height}_g(m)$. Cell σ_k contains $\mathtt{height}_g(m) - \mathtt{depth}_m(i)$ copies of symbol w at step $\mathtt{height}_g(g) + \mathtt{height}_g(m) + \mathtt{depth}_m(i) + 3$. Since σ_k consumes one copy of symbol w in each step, σ_k will take $\mathtt{height}_g(m) - \mathtt{depth}_m(i)$ steps to consume all copies of symbol w. Hence, σ_j enters the firing state at step $(\mathtt{height}_g(g) + \mathtt{height}_g(m) + \mathtt{depth}_m(i) + 3) + (\mathtt{height}_g(m) - \mathtt{depth}_m(i)) = \mathtt{height}_g(g) + 2 \cdot \mathtt{height}_g(m) + 3$.

Phase I ends at step $\mathtt{height}_g(g) + \mathtt{height}_g(m) + 2$ and all cells enter the firing state at step $\mathtt{height}_g(g) + 2 \cdot \mathtt{height}_g(m) + 3$. Thus, Phase II takes $\mathtt{height}_g(m) + 1$ steps. □

Theorem 9. *The synchronization time of our FSSP solution, for a P system with underlying structure of a tree, is $\mathtt{height}_g(g) + 2 \cdot \mathtt{height}_g(m) + 3$.*

Proof. The result is obtained by summing the individual running times of Phases I and II, as given by Propositions 5 and 8: $(\mathtt{height}_g(g) + \mathtt{height}_g(m) + 2) + (\mathtt{height}_g(m) + 1) = \mathtt{height}_g(g) + 2 \cdot \mathtt{height}_g(m) + 3$. □

Table 2: Statistics for improvement on many random trees of various (smaller) orders.

n	average height	average radius	average 3·height	average height+2·radius	average % gain
100	22.12	14.49	66.36	51.1	23.00
200	31.91	21.35	95.73	74.61	22.06
300	41.13	26.79	123.39	94.71	23.24
400	47.86	31.3	143.58	110.46	23.07
500	51.52	33.77	154.56	119.06	22.97
600	57.16	37.76	171.48	132.68	22.63
700	63.43	42.19	190.29	147.81	22.32
800	68.12	45.37	204.36	158.86	22.26
900	72.46	47.83	217.38	168.12	22.66
1000	79.94	52.21	239.82	184.36	23.13

3.4 Empirical results

We tested the improvement in running times over the previously best-known FSSP algorithms that synchronize tree-based P systems [1, 5]. We wanted to see how our new running time, that is proportional to $e + 2r$, compares with the earlier value of $3e$, where e is the eccentricity of the general (which is also the height of the tree, rooted at the general) and r is the radius of a tree. We did two tests suites; one for relatively small trees and one for larger trees as shown in Tables 2 and 3, respectively. In both cases, our empirical results show at least 20% reduction in the number of steps needed to synchronize, which we believe is significant.

For the statistics given in Table 2, we generated random (free) trees by starting from a single node and repeatedly add new leaf nodes to the partially generated tree. We then averaged over all possible locations for the general node. The "average gain" is the average difference $3e-(e+2r)$ and the "average % gain" is improvement as a percentage speedup over $3e$.

For the statistics given in Table 3, we generated random labeled trees using the well-known Prüfer correspondence [19] (using the implementation given in Sage [16]). In these sets of trees, the first indexed vertex is randomly placed, unlike the random trees generated in our first test suite. Hence, for this test suite, we did not need to average over all possible general node locations per tree. Due to the uniform randomness of the labeled tree generator, we assumed the general is placed at the node labeled by 1. Each row in Table 3 is based on 100 random trees of that given order.

We have run both test suites several times and the results are consistent with these two tables. Hence, we are pretty confident in the practical speedup that our new synchronization algorithm provides.

Table 3: Statistics for improvement on random trees of various (larger) orders.

n	diameter	radius	average eccentricity	average gain	average % gain
1000	21	11	16.27	10.54	21.59
2000	32	16	23.45	14.90	21.18
3000	26	13	19.97	13.95	23.27
4000	30	15	22.65	15.30	22.51
5000	35	18	26.51	17.01	21.40
6000	32	16	23.82	15.64	21.89
7000	34	17	25.29	16.58	21.85
8000	34	17	25.01	16.03	21.36
9000	40	20	28.37	16.74	19.67
10000	37	19	27.16	16.32	20.03
10000	38	19	27.36	16.72	20.37
20000	37	19	28.23	18.47	21.80
30000	43	22	31.74	19.49	20.46
40000	43	22	31.55	19.09	20.18
50000	42	21	30.81	19.63	21.23
60000	44	22	32.50	21.00	21.54
70000	48	24	34.55	21.09	20.35
80000	45	23	33.08	20.17	20.32
90000	50	25	36.00	22.01	20.37
100000	47	24	34.15	20.29	19.81

4 FSSP solution for digraphs

The key idea of FSSP solution for digraphs is as follows. For a given digraph, perform a BFS from the general on the communication graph and construct a *virtual* spanning tree, implemented via pointer symbols, not by changing existing arcs. If a node finds multiple parents in the BFS, then one of the parents is chosen as its spanning tree parent. In Figure 6, (a) illustrates a digraph G, (b) illustrates the underlying graph of G and (c) illustrates a spanning tree of the underlying graph of G, rooted at σ_1.

Using the spanning tree constructed from the BFS, the FSSP algorithm described in Section 3, is applied to achieve the synchronization.

We present the details of P system for solving the FSSP (Problem 2) for digraphs in Section 4.1. A trace of the FSSP algorithm for digraphs is given in Table 4. The details Phases I and II of this FSSP algorithm are described in Sections 4.2 and 4.3, respectively. Finally, in Section 4.4, we present some empirical results that illustrates expected improvements of our new algorithm over our previous FSSP algorithm for digraphs [5].

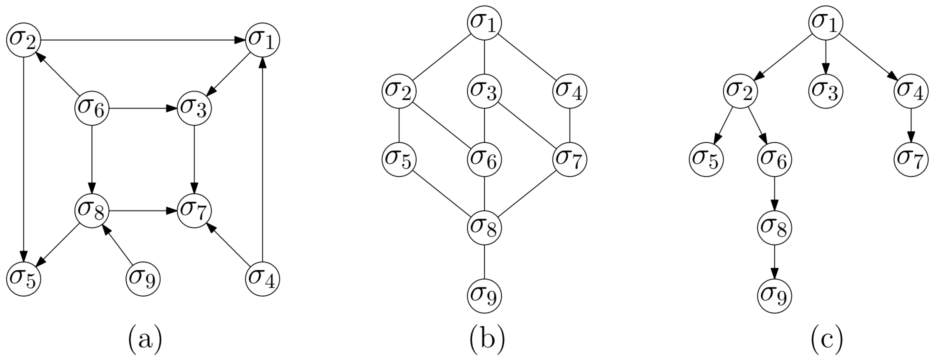


Figure 6: (a) A digraph G. (b) The underlying graph of G. (c) A spanning tree of the underlying graph of G, rooted at σ_1.

4.1 P systems for solving the FSSP for digraphs

Given a digraph (X, A) and $g \in X$, our FSSP algorithm is implemented using the P system $\Pi' = (O, K, \delta)$ of order $n = |X|$, where:

1. $O = \{a, h, o, v, w, x, z\} \cup \{\iota_k, b_k, c_k, e_k, p_k \mid 1 \leq k \leq n\}$.
2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.
3. δ is a digraph, isomorphic to (X, A), where the general $\sigma_g \in K$ corresponds to $g \in X$.

All cells have the same set of states and start at the same initial quiescent state s_0, but with different initial contents and set of rules. The first output condition of Problem 2 will be satisfied by our chosen set of rules.

In this FSSP solution, we extend the basic P system framework, described Section 2. Specifically, we assume that each cell $\sigma_i \in K$ has a unique *cell ID* symbol ι_i, which will be used as an *immutable promoter* and we allow rules with a simple form of *complex symbols*.

To explain these additional features, consider rules 3.10 and 3.11 from the ruleset R, listed below. In this ruleset, symbols i and j are *free variables* (which in our case happen to match cell IDs). Symbols e_i and e_j are complex symbols. Rule 3.11 deletes all existing e_j symbols, regardless of the actual values matched by the free variable j. However, the preceding Rule 3.10 fires only for symbols e_i, with indices i matching the local cell ID, as required by the right-hand side promoter ι_i. Together, Rules 3.10 and 3.11, applied in a weak priority scheme, keep all symbols e_i, with indices i matching the local cell ID, and delete all other symbols e_j.

For each cell $\sigma_i \in K$, its initial configuration is $\sigma_i = (Q, s_0, w_{i0}, R)$ and its final configuration at the end of the execution is $\sigma_i = (Q, s_7, \{\iota_i\}, R)$, where:

- $Q = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$, where s_0 is the initial quiescent state and s_7 is the firing state.
- $w_{i0} = \begin{cases} \{\iota_g o\} & \text{if } \sigma_i = \sigma_g, \\ \{\iota_i\} & \text{if } \sigma_i \neq \sigma_g. \end{cases}$

- R is defined by the following rulesets.

  Rules used in Phase I: all the rules in states s_0, s_1, s_2, s_3, s_4 and Rules 5.5 and 5.6 in state s_5.

  Rules used in Phase II: all the rules in states s_5 and s_6, except Rules 5.5 and 5.6.

  0. Rules for cells in state s_0:
     1. $s_0\ o \rightarrow_{\texttt{min}} s_1\ ao\ (xb_i)_{\updownarrow} \mid \iota_i$
     2. $s_0\ x \rightarrow_{\texttt{min}} s_1\ a\ (xb_i)_{\updownarrow} \mid \iota_i$
     3. $s_0\ b_j \rightarrow_{\texttt{max}} s_1\ p_j$
  1. Rules for cells in state s_1:
     1. $s_1\ ap_j \rightarrow_{\texttt{max}} s_2\ ap_j\ (e_j)_{\updownarrow}$
     2. $s_1\ a \rightarrow_{\texttt{max}} s_2\ a$
     3. $s_1\ p_j \rightarrow_{\texttt{max}} s_2$
  2. Rules for cells in state s_2:
     1. $s_2\ a \rightarrow_{\texttt{max}} s_3\ a$
     2. $s_2\ b_j \rightarrow_{\texttt{max}} s_3$
     3. $s_2\ x \rightarrow_{\texttt{max}} s_3$
  3. Rules for cells in state s_3:
     1. $s_3\ aaa \rightarrow_{\texttt{max}} s_5\ a$
     2. $s_3\ aa \rightarrow_{\texttt{max}} s_4\ a$
     3. $s_3\ c_ie_i \rightarrow_{\texttt{max}} s_3 \mid \iota_i$
     4. $s_3\ aoooe_i \rightarrow_{\texttt{max}} s_3\ aa\ (o)_{\updownarrow} \mid \iota_i$
     5. $s_3\ aoe_ie_i \rightarrow_{\texttt{max}} s_3\ ahoe_ie_i \mid \iota_i$
     6. $s_3\ aoe_i \rightarrow_{\texttt{max}} s_3\ ahooe_i \mid \iota_i$
     7. $s_3\ ao \rightarrow_{\texttt{max}} s_3\ aaa$
     8. $s_3\ ae_i \rightarrow_{\texttt{max}} s_3\ ae_ih \mid \iota_i$
     9. $s_3\ ap_j \rightarrow_{\texttt{max}} s_3\ aa\ (c_j)_{\updownarrow}$
     10. $s_3\ e_i \rightarrow_{\texttt{max}} s_3\ e_i \mid \iota_i$
     11. $s_3\ e_j \rightarrow_{\texttt{max}} s_3$
     12. $s_3\ p_j \rightarrow_{\texttt{max}} s_4$
     13. $s_3\ p_j \rightarrow_{\texttt{max}} s_5$
  4. Rules for cells in state s_4:
     1. $s_4\ a \rightarrow_{\texttt{max}} s_5$
     2. $s_4\ h \rightarrow_{\texttt{max}} s_5$
     3. $s_4\ c_j \rightarrow_{\texttt{max}} s_5$
  5. Rules for cells in state s_5:
     1. $s_5\ a \rightarrow_{\texttt{max}} s_6\ a\ (z)_{\updownarrow}$
     2. $s_5\ hh \rightarrow_{\texttt{max}} s_6\ w\ (v)_{\updownarrow}$
     3. $s_5\ zv \rightarrow_{\texttt{max}} s_6\ a\ (z)_{\updownarrow}$
     4. $s_5\ v \rightarrow_{\texttt{max}} s_6\ w\ (v)_{\updownarrow}$
     5. $s_5\ o \rightarrow_{\texttt{max}} s_5$
     6. $s_5\ c_j \rightarrow_{\texttt{max}} s_5$
  6. Rules for cells in state s_6:
     1. $s_6\ aw \rightarrow_{\texttt{max}} s_6\ a$
     2. $s_6\ a \rightarrow_{\texttt{max}} s_7$
     3. $s_6\ z \rightarrow_{\texttt{max}} s_7$
     4. $s_6\ v \rightarrow_{\texttt{max}} s_7$

Table 4: The traces of the FSSP algorithm on the digraph of Figure 6 (a), where the general is σ_1 and the middle cell is σ_2. The step in which the Phase I ends (or the Phase II begins) is indicated by the shaded table cells.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	$s_0\ \iota_1 o$	$s_0\ \iota_2$	$s_0\ \iota_3$	$s_0\ \iota_4$	$s_0\ \iota_5$	$s_0\ \iota_6$	$s_0\ \iota_7$	$s_0\ \iota_8$	$s_0\ \iota_9$
1	$s_1\ \iota_1 ao$	$s_0\ \iota_2 b_1 x$	$s_0\ \iota_3 b_1 x$	$s_0\ \iota_4 b_1 x$	$s_0\ \iota_5$	$s_0\ \iota_6$	$s_0\ \iota_7$	$s_0\ \iota_8$	$s_0\ \iota_9$
2	$s_2\ \iota_1 ab_2 b_3 b_4 ox^3$	$s_1\ \iota_2 ap_1$	$s_1\ \iota_3 ap_1$	$s_1\ \iota_4 ap_1$	$s_0\ \iota_5 b_2 x$	$s_0\ \iota_6 b_2 b_3 x^2$	$s_0\ \iota_7 b_3 b_4 x^2$	$s_0\ \iota_8$	$s_0\ \iota_9$
3	$s_3\ \iota_1 ae_1^3 o$	$s_2\ \iota_2 ab_5 b_6 p_1 x^2$	$s_2\ \iota_3 ab_6 b_7 p_1 x^2$	$s_2\ \iota_4 ab_7 p_1 x$	$s_1\ \iota_5 ae_1 p_2$	$s_1\ \iota_6 ae_1^2 p_2 p_3 x$	$s_1\ \iota_7 ae_1^2 p_3 p_4 x$	$s_0\ \iota_8 b_5 b_6 b_7 x^3$	$s_0\ \iota_9$
4	$s_3\ \iota_1 ae_1^3 ho$	$s_3\ \iota_2 ae_2^2 p_1$	$s_3\ \iota_3 ae_2 e_4 p_1$	$s_3\ \iota_4 ae_4 p_1$	$s_2\ \iota_5 ab_8 e_1 p_2 x$	$s_2\ \iota_6 ab_8 e_1^2 p_2 x^2$	$s_2\ \iota_7 ab_8 e_1^2 p_4 x^2$	$s_1\ \iota_8 ae_2^2 e_4 p_5 p_6 p_7 x^2$	$s_0\ \iota_9 b_8 x$
5	$s_3\ \iota_1 ac_1 e_1^3 h^2 o$	$s_3\ \iota_2 ae_2^2 hp_1$	$s_3\ \iota_3 a^2$	$s_3\ \iota_4 ae_4 hp_1$	$s_3\ \iota_5 ae_1 e_6 p_2$	$s_3\ \iota_6 ac_1 e_1^2 e_6 p_2$	$s_3\ \iota_7 ac_1 e_1^2 e_6 p_4$	$s_2\ \iota_8 ab_9 e_2^2 e_4 p_6 x^3$	$s_1\ \iota_9 ae_6 p_8$
6	$s_3\ \iota_1 ae_1^2 h^3 o$	$s_3\ \iota_2 ac_2 e_2^2 h^2 p_1$	$s_4\ \iota_3 ac_4$	$s_3\ \iota_4 ac_4 e_4 h^2 p_1$	$s_3\ \iota_5 a^2$	$s_3\ \iota_6 ac_1 e_6 hp_2$	$s_3\ \iota_7 a^2 c_1$	$s_3\ \iota_8 ac_2 c_4 e_2^2 e_4 e_8 p_6$	$s_2\ \iota_9 ae_6 p_8$
7	$s_3\ \iota_1 ac_1 e_1^2 h^4 o$	$s_3\ \iota_2 ae_2 h^3 p_1$	$s_5\ \iota_3$	$s_3\ \iota_4 a^2 h^2$	$s_4\ \iota_5 a$	$s_3\ \iota_6 ac_1 e_6 h^2 p_2$	$s_4\ \iota_7 ac_1^2$	$s_3\ \iota_8 ac_2 c_4 e_8 hp_6$	$s_3\ \iota_9 ae_6 p_8$
8	$s_3\ \iota_1 ae_1 h^5 o^2$	$s_3\ \iota_2 ae_2 h^4 p_1$	$s_5\ \iota_3$	$s_4\ \iota_4 ah^2$	$s_5\ \iota_5$	$s_3\ \iota_6 ac_1 e_6 h^3 p_2$	$s_5\ \iota_7$	$s_3\ \iota_8 ac_2 c_4 c_8 e_8 h^2 p_6$	$s_3\ \iota_9 a^2$
9	$s_3\ \iota_1 ae_1 h^6 o^3$	$s_3\ \iota_2 ae_2 h^5 p_1$	$s_5\ \iota_3$	$s_5\ \iota_4$	$s_5\ \iota_5 c_6$	$s_3\ \iota_6 ac_1 c_6 e_6 h^4 p_2$	$s_5\ \iota_7 c_6$	$s_3\ \iota_8 a^2 c_2 c_4 h^2$	$s_4\ \iota_9 ac_6$
10	$s_3\ \iota_1 a^2 h^6$	$s_3\ \iota_2 ac_2 e_2 h^6 op_1$	$s_5\ \iota_3 c_2 o$	$s_5\ \iota_4 o$	$s_5\ \iota_5$	$s_3\ \iota_6 a^2 c_1 h^4$	$s_5\ \iota_7$	$s_4\ \iota_8 ac_2^2 c_4 h^2$	$s_5\ \iota_9$
11	$s_4\ \iota_1 ah^6$	$s_3\ \iota_2 a^3 h^6 p_1$	$s_5\ \iota_3$	$s_5\ \iota_4$	$s_5\ \iota_5$	$s_4\ \iota_6 ac_1 h^4$	$s_5\ \iota_7$	$s_5\ \iota_8$	$s_5\ \iota_9$
12	$s_5\ \iota_1$	$s_5\ \iota_2 ah^6$	$s_5\ \iota_3$	$s_5\ \iota_4$	$s_5\ \iota_5$	$s_5\ \iota_6$	$s_5\ \iota_7$	$s_5\ \iota_8$	$s_5\ \iota_9$
13	$s_5\ \iota_1 v^3 z$	$s_6\ \iota_2 aw^3$	$s_5\ \iota_3$	$s_5\ \iota_4$	$s_5\ \iota_5 v^3 z$	$s_5\ \iota_6 v^3 z$	$s_5\ \iota_7$	$s_5\ \iota_8$	$s_5\ \iota_9$
14	$s_6\ \iota_1 aw^2$	$s_6\ \iota_2 av^6 w^2 z^3$	$s_5\ \iota_3 v^4 z^2$	$s_5\ \iota_4 v^2 z$	$s_6\ \iota_5 aw^2$	$s_6\ \iota_6 aw^2$	$s_5\ \iota_7$	$s_5\ \iota_8 v^4 z^2$	$s_5\ \iota_9$
15	$s_6\ \iota_1 av^3 wz^3$	$s_6\ \iota_2 av^6 wz^3$	$s_6\ \iota_3 a^2 w^2$	$s_6\ \iota_4 aw$	$s_6\ \iota_5 av^2 wz^2$	$s_6\ \iota_6 av^4 wz^4$	$s_5\ \iota_7 v^5 z^5$	$s_6\ \iota_8 a^2 w^2$	$s_5\ \iota_9 v^2 z^2$
16	$s_6\ \iota_1 av^3 z^3$	$s_6\ \iota_2 av^6 z^3$	$s_6\ \iota_3 a^2 z^5$	$s_6\ \iota_4 az^5$	$s_6\ \iota_5 av^2 z^2$	$s_6\ \iota_6 av^4 z^4$	$s_6\ \iota_7 a^5$	$s_6\ \iota_8 a^2 z^7$	$s_6\ \iota_9 a^2$
17	$s_7\ \iota_1$	$s_7\ \iota_2$	$s_7\ \iota_3$	$s_7\ \iota_4$	$s_7\ \iota_5$	$s_7\ \iota_6$	$s_7\ \iota_7$	$s_7\ \iota_8$	$s_7\ \iota_9$

4.2 Phase I: Find the middle cell of a BFS spanning tree

For a given digraph-based P system, a (virtual) spanning tree is constructed by a standard BFS originated from the general, where the tree parent of each cell is one of its BFS parents (randomly chosen). Each cell keeps the track of its spanning tree parent and this is achieved by the use of cell IDs (unique identifier ID), e.g., i is the cell ID of σ_i.

Details of Phase I

Objective: The objective of Phase I is to find the middle cell, σ_m, and its height, $\mathtt{height}_g(m)$.

Precondition: Phase I starts with the initial configuration of P system Π, described in Section 4.1.

Postcondition: Phase I ends when σ_m enters state s_5. At the end of Phase I, the configuration of cell $\sigma_i \in K$ is (Q, s_5, w_i, R), where $|w_i|_{\iota_i} = 1$; $|w_i|_a = 1$ and $|w_i|_h = 2 \cdot \mathtt{height}_g(i)$, if $\sigma_i = \sigma_m$.

Description: We describe below the details of the BFS spanning tree construction and the propagation of the reflected symbol in the BFS tree. The symbol o, starting from the general, propagates from a tree parent to one of its children, as described in the FSSP solution for tree-based P systems (Section 3.2). Hence, the details of symbol o propagation are not given here.

- **The details of the BFS spanning tree construction:**

  A BFS starts from the general. When the search reaches cell σ_i, σ_i will send a copy of symbol b_i to all its neighbors (Rule 0.1 or 0.2).

  From the BFS, cell σ_i receives a copy of symbol b_j from each $\sigma_j \in \mathtt{Pred}_g(i)$, where σ_j is a BFS dag parent of σ_i. Cell σ_i temporarily stores all of its BFS dag parents by transforming each received symbol b_j to symbol p_j (Rule 0.3). Note, σ_i will also receive a copy of symbol b_k from each $\sigma_k \in \mathtt{Peer}_g(i) \cup \mathtt{Succ}_g(i)$; however, σ_i will discard each received symbol b_k.

  Each cell selects one of its BFS dag parents as its tree parent. If cell σ_i has chosen σ_j as its tree parent, then σ_i will discards each p_k, where $\sigma_k \in \mathtt{Pred}_g(i) \setminus \{\sigma_j\}$ (Rule 1.3). Additionally, σ_i will send a copy of symbol e_j to all its neighbors, which will be discarded by all σ_i's neighbors, except σ_j (Rule 1.1).

  Hence, in each cell σ_i, the multiplicity of symbol e_i will indicate the number of σ_i's tree children and symbol p_j will indicate that σ_j is the tree parent of σ_i; also, symbol p_j will later be used to propagate the reflected symbol back up the tree.

- **The details of reflected symbol propagation:**

  To replicate the propagation of a reflected symbol up the BFS tree, each internal cell of the BFS tree needs to check if the received a reflected symbol came from one of its BFS tree children.

  Let σ_i be a BFS tree child of σ_j, where $|w_i|_{e_i} = 0$. Recall that, in such case, cell σ_i contains symbol p_j, where the subscript j is the ID of its BFS tree parent, and σ_j contains symbol e_j, such that $|w_j|_{e_j}$ is the number of σ_j's BFS tree children.

Guided by symbol p_j, σ_i sends symbol c_j to all its neighbors (Rule 3.9). Cell σ_j consumes a copy of symbol e_j with a copy of symbol c_j by Rule 3.3; σ_j cannot consume symbol e_j with symbol c_k, where $j \neq k$. If σ_j receives symbol c_j from all its BFS tree children, then all copies of symbol e_j will be consumed, i.e. $|w_j|_{e_j} = 0$.

Proposition 10 indicates the step in which the BFS reaches cell σ_i and σ_i receives symbol b_j from each $\sigma_j \in \mathtt{Pred}_g(i)$. Proposition 11 indicates the step in which σ_i receives symbol e_i from its tree child.

Proposition 10. *Cell σ_i receives symbol b_j from each $\sigma_j \in \mathtt{Pred}_g(i)$ at step $\mathtt{depth}_g(i)$ and sends symbol b_i to all its neighbors at step $\mathtt{depth}_g(i) + 1$.*

Proof. Proof by induction, on $d = \mathtt{depth}_g(i) \geq 1$. At step 1, the general σ_g sends symbol b_g to all its neighbors by Rule 0.1. Hence, at step 1, each cell σ_k at depth 1 receives symbol b_g. Then, at step 2, by Rule 0.2, σ_k sends symbol b_k to each of its neighbors.

Assume that the induction hypothesis holds for each cell σ_j at depth d. Consider cell σ_i at $\mathtt{depth}_g(i) = m + 1 = \mathtt{depth}_g(j) + 1$. By induction hypothesis, at step $\mathtt{depth}_g(j) + 1$, each $\sigma_j \in \mathtt{Pred}_g(i)$ sends symbol b_j to all its neighbors. Thus, at step $\mathtt{depth}_g(j) + 1 = \mathtt{depth}_g(i)$, σ_i receives symbol b_j. At step $\mathtt{depth}_g(i) + 1$, by Rule 0.2, σ_i sends symbol b_i to all its neighbors. □

Proposition 11. *Cell σ_i receives a copy of symbol e_i from each of its tree children at step $\mathtt{depth}_g(i) + 3$.*

Proof. Assume that cell $\sigma_j \in \mathtt{Succ}_g(i)$ has chosen σ_i as its tree parent. From Proposition 10, cell σ_j receives symbol b_i at step $\mathtt{depth}_g(j) = \mathtt{depth}_g(i) + 1$. According to the description, σ_j will send symbol e_i at step $\mathtt{depth}_g(j) + 2$. Thus, σ_i will receive symbol e_i at step $\mathtt{depth}_g(i) + 3$. □

Remark 12. *From Proposition 11, σ_i receives symbol e_i from its tree child at step $\mathtt{depth}_g(i) + 3$. If σ_i does not receive symbol e_i at step $\mathtt{depth}_g(i) + 3$, then σ_i can recognize itself as a tree leaf and send a reflected symbol to its tree parent at step $\mathtt{depth}_g(i) + 4$. That is, once a leaf cell is reached by the BFS, it will take three additional steps to send reflected symbol to its tree parent. Recall, in the FSSP algorithm for tree-based P systems, a leaf cell sends reflected symbol to its parent, one step after reached by the BFS. Thus, this FSSP algorithm for digraph-based P systems takes three additional steps to send the reflected symbol than the FSSP algorithm for tree-based P systems.*

4.3 Phase II: Determine the step to enter the firing state

Similar to the Phase II described in Section 3.3, the firing order is broadcasted from the middle cell σ_m. The order is paired with a counter, which is initially set to the eccentricity of σ_m and decrements by one in each step of this broadcast operation.

Details of Phase II

Objective: The objective of Phase II is to determine the step to enter the firing state, such that during the last step of Phase II, i.e. the system's execution, all cells enter the firing state, simultaneously and for the first time.

Precondition: Phase II starts with the postcondition of Phase I, described in Section 4.2.

Postcondition: Phase II ends when all cells enter the firing state s_7. At the end of Phase II, the configuration of cell $\sigma_i \in K$ is $(Q, s_7, \{\iota_i\}, R)$.

Description: The order arrives in σ_i, along every shortest paths from σ_m to σ_i. Hence, to compute the correct step to enter the firing state, cell σ_i decrements, in each step, the sum of all received counter by the number of shortest paths from σ_m to σ_i and σ_i enters the firing state if the sum of all received counter becomes 0. The number of shortest paths from σ_m to σ_i is determined as follows. Cell σ_m sends a copy of symbol z. Each cell σ_i forwards symbol z, received from each $\sigma_j \in \texttt{Pred}_m(i)$. The number of shortest paths from σ_m to σ_i is the sum of all copies of symbol z that σ_i receives from each $\sigma_j \in \texttt{Pred}_m(i)$.

Let t be the current counter and k be the number of shortest paths from σ_m to the current cell. In the FSSP solution for tree-based P systems, the condition for entering the firing state in the next step is when $t = 1$ (note $k = 1$). However, the FSSP solution, as implemented in this section, cannot directly detect if $t = k$, since $k \geq 1$ Instead, a cell enters the firing state after $t = 0$ is detected. Thus, the FSSP algorithm for digraph-based P systems requires one additional step in Phase II.

Theorem 13. *The synchronization time of the FSSP solution for digraph-based P systems is* $\texttt{ecc}(g) + 2 \cdot \texttt{ecc}(m) + 7$.

Proof. This FSSP algorithm for digraph-based P systems requires four additional overhead steps than the FSSP algorithm for tree-based P systems. Three of these four overhead steps are described in Remark 12 and the remaining overhead step is mentioned in Section 4.3. □

We end this section with a comment regarding improving the communication requirements of our FSSP solution. Currently, there may be an exponential number of broadcast objects generated since a given cell currently receives a copy of the counter from every possible shortest path from the middle cell. We can reduce number of broadcasted counters from an exponential to a polynomial as follows. Assume that, a counter, sent or forwarded from a cell, is annotated with the cell's ID. In Phase II, if a cell receives counter from its BFS tree neighbor (from a BFS tree child for cells on the path from the general to the middle cell, otherwise from its original BFS tree parent), then it broadcasts the reduced-by-one counter, now annotated with its own ID, to all its neighbors. The total number of steps of this revised algorithm would still be the same as given in Theorem 13.

4.4 Empirical results

We also tested the improvement in running times over our previous FSSP algorithm on digraph-based P systems. The rate of improvement drops off as the number of edges increase over $n-1$, the size of trees of order n. But for several sparse digraph structured P systems the improvement is still worthwhile.

We did two tests suites; one for relatively small digraphs (illustrated in Figure 7) and one for larger digraphs as shown in Table 5. The graphs used in our empirical tests were generated using NetworkX [8].

For the statistics given in Table 5, we first generated connected random graphs of order n and size m. We then averaged over all possible locations for the general node. To model the parallel nature of P systems, we needed to generate a random BFS tree originating at the general. This was created by first performing a BFS from the general to constructing the BFS dag then randomly picking (for each non-general node) one parent within the dag structure as the parent for the BFS tree. The code for constructing a BFS tree from a dag structure is displayed in Figure 8.

For this BFS tree, with e denoting the eccentricity of the general and r denoting the radius of the BFS tree, the "average gain" is the average difference of $3e - (e + 2r)$ and the "average % gain" is the average of the $(3e-(e+2r))/(3e)$ values. From our empirical results, we can observe that the radius of the BFS spanning trees seems to be close to the actual radius of the given virtual communication graphs.

For the statistics given in the three dimensional plots of Figure 7 (generated using Gnuplot [20]), we generated 100 random connected (n, m)-graphs, for each order n, $20 \leq n \leq 40$, and size $m = (n - 1) + 2k$, where $0 \leq k \leq 20$. Note, the integer value of $2k$ represents the number of edges added to a tree. We then averaged over all possible general starting positions. The vertical axis is the average percentage speedup of our new algorithm over our previous synchronization algorithm. One can also observe from this plot, at least 20% improvements (i.e. reduction in number of steps needed to synchronize), is maintained for $k = 0$ (i.e. the graph is a tree). However, as the graphs become less sparse, the expected improvement drops to near zero, when as few as 40 edges are added to the trees. In general, for fixed k, the expected improvement in performance, for $(n, n + k)$ digraphs slightly increases as n increases. However, for fixed n, the expected improvement in performance drops drastically as k increases.

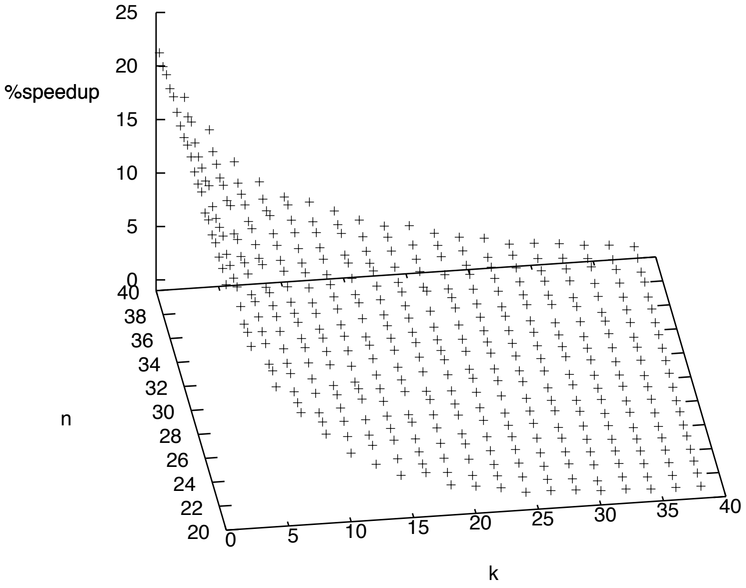
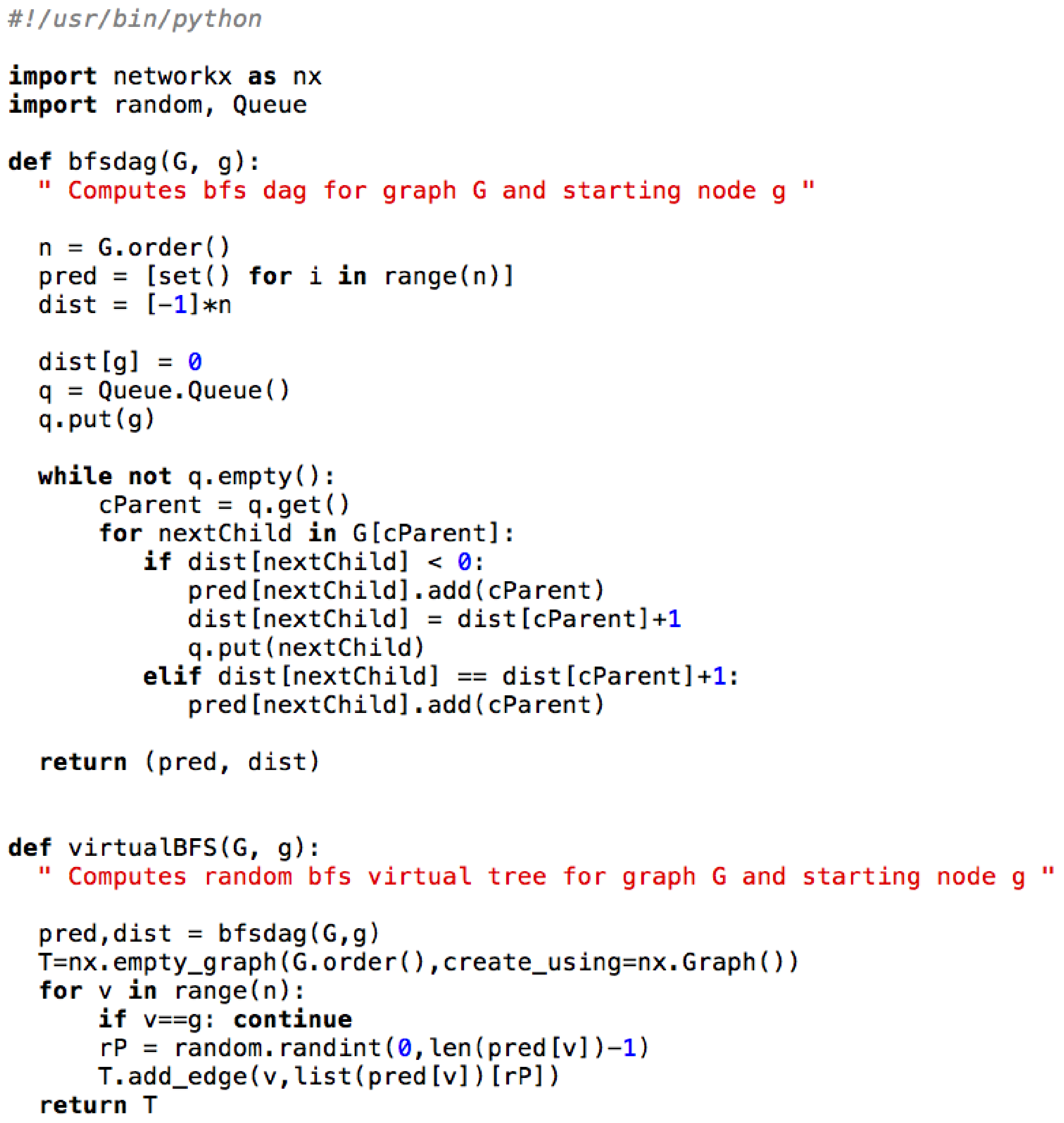


Figure 7: Discrete 3-dimensional plot of expected synchronization improvements for a small range of random connected (n, m)-graph structures, with $m = (n - 1) + k$ edges.

Table 5: Statistics for reduction in number of steps needed to synchronize on a few random (n, m)-graphs.

n	m	graph radius	avg tree radius	average gain	average gain %
100	100	15	15.68	16.7	23.16
100	110	9	11.47	3.14	8.02
100	120	7	8.97	1.6	5.45
100	130	7	8.13	1.0	3.86
100	140	6	7.33	0.72	3.12
200	200	20	20.73	17.91	20.10
200	210	16	19.12	5.08	7.81
200	220	13	15.74	3.9	7.34
200	230	9	11.24	2.24	6.04
200	240	9	11.41	2.13	5.68
300	300	25	25.00	22.32	20.57
300	310	17	18.95	7.95	11.56
300	320	16	18.61	8.29	12.14
300	330	12	15.0	3.37	6.73
300	340	12	14.03	2.46	5.37
400	400	24	24.56	24.10	21.94
400	410	22	24.79	7.73	8.99
400	420	19	21.91	7.12	9.31
400	430	15	17.85	2.78	4.81
400	440	13	15.86	2.29	4.48
500	500	28	29.14	23.30	19.04
500	510	24	27.28	9.68	10.04
500	520	19	23.17	8.72	10.56
500	530	16	19.87	5.68	8.34
500	540	16	19.25	5.70	8.60
600	600	28	30.99	22.35	17.66
600	610	25	28.78	14.63	13.51
600	620	22	24.965	5.39	6.49
600	630	19	22.065	5.72	7.64
600	640	17	20.32	4.15	6.18
700	700	35	38.68	25.58	16.56
700	710	23	29.55	10.09	9.72
700	720	23	26.59	8.39	9.08
700	730	21	24.69	7.70	9.00
700	740	20	25.11	7.50	8.66
800	800	40	42.66	26.93	15.99
800	810	28	32.50	13.08	11.16
800	820	29	33.91	9.13	7.91
800	830	23	26.36	8.06	8.84
800	840	20	25.19	7.80	8.93
900	900	53	60.73	25.92	11.72
900	910	35	39.23	12.94	9.44
900	920	24	30.37	7.44	7.27
900	930	25	29.23	7.42	7.50
900	940	21	24.90	5.74	6.88
1000	1000	60	66.96	26.72	11.09
1000	1010	33	37.43	20.27	14.20
1000	1020	26	31.19	8.64	8.11
1000	1030	25	29.63	7.87	7.81
1000	1040	26	30.32	11.41	10.55
1000	1000	46	48.45	26.58	14.35
1000	1010	31	34.77	20.07	14.93
1000	1020	28	32.98	11.91	10.19
1000	1030	24	29.30	9.23	9.07
1000	1040	23	27.62	6.66	7.17
2000	2000	76	76.07	85.98	24.07
2000	2010	55	61.33	30.50	13.27
2000	2020	39	44.73	18.55	11.45
2000	2030	33	42.11	11.21	7.83
2000	2040	32	39.78	13.68	9.78



```
#!/usr/bin/python

import networkx as nx
import random, Queue

def bfsdag(G, g):
  " Computes bfs dag for graph G and starting node g "

  n = G.order()
  pred = [set() for i in range(n)]
  dist = [-1]*n

  dist[g] = 0
  q = Queue.Queue()
  q.put(g)

  while not q.empty():
      cParent = q.get()
      for nextChild in G[cParent]:
         if dist[nextChild] < 0:
            pred[nextChild].add(cParent)
            dist[nextChild] = dist[cParent]+1
            q.put(nextChild)
         elif dist[nextChild] == dist[cParent]+1:
            pred[nextChild].add(cParent)

  return (pred, dist)


def virtualBFS(G, g):
  " Computes random bfs virtual tree for graph G and starting node g "

  pred,dist = bfsdag(G,g)
  T=nx.empty_graph(G.order(),create_using=nx.Graph())
  for v in range(n):
      if v==g: continue
      rP = random.randint(0,len(pred[v])-1)
      T.add_edge(v,list(pred[v])[rP])
  return T
```

Figure 8: Python code for generating a BFS dag from a given graph and constructing a BFS tree from the BFS dag.

5 Conclusions and future works

In this paper, we explicitly presented an improved solution to the FSSP for tree-based P systems. We improved our previous FSP algorithm [5] by allowing the general to delegate a more central cell in the tree structure, as an alternative to itself, to send the final "firing" command. This procedure for trees-based P systems was extended to digraph-based P systems. Here we use a virtual spanning BFS tree (rooted at the general) in the digraph and use our tree-based middle-cell algorithm for that tree to improve the synchronization time. Alternatively, we would like to develop a way to compute a center of an arbitrary graph since the radius of the graph may be less than the radius of a particular BFS spanning tree. Thus this future work may possibly provide even more guaranteed improvements in synchronization time.

We summarize our work as follows. With e being the eccentricity of the general and r denoting the radius of the graph, where $e/2 \leq r \leq e$, we note the radius r' of the spanning BFS tree satisfies $e/2 \leq r \leq r' \leq e$. Thus, we have the following results:

- If the membrane structure of a considered P system is a tree, then synchronization time is $e + 2r + 3$.

- If the membrane structure of a considered P system is a digraph, then synchronization time t is $e + 2r + 7 \leq t \leq 3e + 7$.

Our empirical work shows that the radius of the BFS spanning tree is often as small as the radius of its host graph and we expect, more often than not, the synchronization time to be closer to $e + 2r + 7$ than to $3e + 7$ for arbitrary digraph-based P systems.

Finally, we mention a couple open problems for the future. We would like a theoretical proof based on properties of random trees of why it seems that the our gain in performance is independent of the order of the trees considered. The current FSSP solution is designed for digraph-based P systems with *duplex* channels. Another remaining open problem is to obtain an efficient FSSP solution that synchronizes strongly connected digraphs using *simplex* channels.

Acknowledgments

The authors wish to thank Ionuţ-Mihai Niculescu for providing us with some early empirical statistics on random graphs and to acknowledge the University of Auckland FRDF grant 9843/3626216 to assist our research.

References

[1] A. Alhazov, M. Margenstern, and S. Verlan. Fast synchronization in P systems. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Workshop on Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 2008.

[2] R. Balzer. An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, 10(1):22–42, 1967.

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

[4] A. Berthiaume, T. Bittner, L. Perkovic, A. Settle, and J. Simon. Bounding the firing synchronization problem on a ring. *Theor. Comput. Sci.*, 320(2-3):213–228, 2004.

[5] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu. Faster synchronization in P systems. *International Journal of Natural Computing*, pages 1–15, 2011.

[6] E. Goto. A minimal time solution of the firing squad problem. Course notes for Applied Mathematics 298, pages 52–59, Harvard University, 1962.

[7] J. J. Grefenstette. Network structure and the firing squad synchronization problem. *J. Comput. Syst. Sci.*, 26(1):139–152, 1983.

[8] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, Aug 2008.

[9] M. Ionescu and D. Sburlan. On P systems with promoters/inhibitors. *J. UCS*, 10(5):581–599, 2004.

[10] K. Kobayashi. The firing squad synchronization problem for a class of polyautomata networks. *J. Comput. Syst. Sci.*, 17(3):300–318, 1978.

[11] E. F. Moore. The firing squad synchronization problem. In E. Moore, editor, *Sequential Machines, Selected Papers*, pages 213–214. Addison-Wesley, Reading MA., 1964.

[12] F. R. Moore and G. G. Langdon. A generalized firing squad problem. *Information and Control*, 12(3):212–220, 1968.

[13] Y. Nishitani and N. Honda. The firing squad synchronization problem for graphs. *Theor. Comput. Sci.*, 14:39–61, 1981.

[14] G. Păun. Introduction to membrane computing. In G. Ciobanu, M. J. Pérez-Jiménez, and G. Păun, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 1–42. Springer-Verlag, 2006.

[15] H. Schmid and T. Worsch. The firing squad synchronization problem with many generals for one-dimensional CA. In J.-J. Lévy, E. W. Mayr, and J. C. Mitchell, editors, *IFIP TCS*, pages 111–124. Kluwer, 2004.

[16] W. A. Stein et al. *Sage Mathematics Software (Version 4.6)*. The Sage Development Team, 2010. `http://www.sagemath.org`.

[17] H. Umeo, N. Kamikawa, K. Nishioka, and S. Akiguchi. Generalized firing squad synchronization protocols for one-dimensional cellular automata—a survey. *Acta Physica Polonica B Proceedings Supplement*, 3(2):267–289, 2010.

[18] A. Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9(1):66–78, 1966.

[19] E. W. Weisstein. Prüfer code, from MathWorld—a Wolfram web resource. `http://mathworld.wolfram.com/PrueferCode.html`, [Online; accessed 8-April-2011].

[20] T. Williams, C. Kelley, and many others. Gnuplot 4.2: an interactive plotting program. `http://gnuplot.sourceforge.net/`, March 2009.