Discovering the Membrane Topology of Hyperdag P Systems

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

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

Summary. In an earlier paper, we presented an extension to the families of P systems, called hyperdag P Systems (hP systems), by proposing a new underlying topological structure based on the hierarchical dag structure (instead of trees or digraphs). In this paper, we develop building-block membrane algorithms for discovery of the global topological structure from the local cell point of view. In doing so, we propose more convenient operational modes and communication transfer modes, that depend only on each of the individual cell rules.

Finally, by extending our initial work done in visualization of hP system membranes with interconnections based on dag structures without transitive arcs, we propose several ways to represent all communication channels, including transitive ones, in the plane by 3D-folded (and possibly twisted) simple-closed regions.

1 Introduction

In this paper we continue our study [8]. Specifically, we are interested to validate the adequacy of our hP system model for describing a subset of fundamental distributed algorithms that present relevance to networking.

For several algorithms, especially Algorithms 1 and 5 below, we follow and extend to dags the approach used by Ciobanu *et al.* in [4, 3]. We also use traditional rewriting rules, without pseudo-code.

In this process, we identify areas where our initial model was not versatile enough and we propose corresponding adjustments, that can also be retrofitted to other models of the P family, such as the refinement of the rewrite and transfer modes. We also advocate the weak policy for priority rules [10], which we believe is closer to the actual task scheduling in operating systems.

This paper focuses on basic building blocks that are relevant for network discovery (see also [7]): broadcast, convergecast, flooding, and a simple synchronization solution, that highlights the versatility of the dag structure underlying hP systems.

We have earlier proposed an algorithm to visually represent hP systems, where the underlying cell structure was restricted to a canonical dag (i.e., without tran-

sitive arcs) [8]. Nodes were represented as simple closed regions on the plane (with possible nesting or overlaps) and communication channels by direct containment relationships of the regions. In this paper, we extend this planar representation by presenting several plausible solutions that enable us to visualize any hP system, modelled as an arbitrary dag, in the plane. Additionally, for these solutions, we discuss their advantages and limitations. Finally, in Section 6, we describe a new algorithm for representing general hP systems, where transitive arcs are not excluded.

2 Preliminaries

We assume that the reader is familiar with the basic terminology and notations [8]: relations, graphs, nodes (vertices), arcs, directed graphs, directed acyclic graphs (dags), canonical dags (dags without transitive arcs), trees, node height (number of arcs on the longest path to a descendant), topological order, set or multiset based hypergraphs, simple closed curves (Jordan curves), alphabets, strings and multisets over an alphabet.

We also assume familiarity with transition P systems and their planar representation [10] and with hyperdag P systems (hP systems) [8].

Without giving all functional details, we recall here the basic notations and the definition of hP systems. Given a set of objects O, we define the following sets of tagged objects: $O_{\uparrow} = \{o_{\uparrow} \mid o \in O\}$, $O_{\downarrow} = \{o_{\downarrow} \mid o \in O\}$, $O_{\leftrightarrow} = \{o_{\leftrightarrow} \mid o \in O\}$, $O_{go} = \{o_{go} \mid o \in O\}$, $O_{out} = \{o_{out} \mid o \in O\}$. Intuitively, the $_{\uparrow}$, $_{\downarrow}$, $_{\leftrightarrow}$ tags indicate objects that will be transferred to parents, children, siblings, respectively; the $_{go}$ tags indicate transfer to all neighbors (parents, children and siblings); the $_{out}$ tags indicate transfer to the environment.

Definition 1 (Hyperdag P systems). *An* hP system *of order m is a system $\Pi = (O, \sigma_1, \ldots, \sigma_m, \delta, I_{out})$, where:*

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

The dynamic operations of hP systems, i.e., the configuration changes via object rewriting and object transfer, are a natural extension of similar operations used by transition P systems and nP systems. Our earlier paper, [8], describes the dynamic behavior of hP systems, in more detail.

We measure the *runtime complexity* of a P system in terms of *P-steps*, where a P-step corresponds to a transition on a parallel P machine. If no more transitions are possible, the hP system halts. For halted hP systems, the *computational result* is the multiset of objects emitted *out* (to the "environment"), over all the time steps, from the output cells I_{out}. The *numerical result* is the set of vectors consisting of the object multiplicities in the multiset result. Within the family of P systems, two systems are *functionally equivalent* if they yield the same computational result.

Example 1. Figure 1 shows the structure of an hP system that models a computer network. Four computers are connected to "Ethernet Bus 1", the other four computers are connected to "Ethernet Bus 2", while two of the first group and two of the second group are at the same time connected to a wireless cell. In this figure we also suggest that "Ethernet Bus 1" and "Ethernet Bus 2" are themselves connected to a higher level communication hub, in a generalized hypergraph.

We have already shown, [8], that our hP systems can simulate any transition P system [10] and any bidirectional nP system [9], with the same number of steps and object transfers. To keep the arguments simple, we have only considered systems without additional features, such as dissolving membranes, priorities or polarities. However, our definition of hP systems can also be extended, as needed, with additional features, in a straightforward manner, and we do so in this paper.

Model refinements

- As initially defined [8], the rules are applied according to the current cell state s, in the rewrite mode $\alpha(s) \in \{min, par, max\}$, and the objects are sent out in the transfer mode $\beta(s) \in \{one, spread, repl\}$. In this paper, we propose a refinement to these modes and allow that *the rewrite and transfer modes to depend on the rule used* (instead of the state), as long as there are no conflicting requirements. We will highlight the cases where this modes extension is essential.
- We also consider rules with *priorities*, in their *weak* interpretation [10]. In the current paper, *lower numbers* (i.e., first enumerated) indicate *higher priority*. In the *weak* interpretation of the priority, rules are applied in decreasing order of their priorities — where a lower priority rule can only applied after all higher priority rules have been applied (as required by the rewriting modes). In contrast, in the *strong* interpretation, a lower priority rule cannot be applied at all, if a higher priority rule was applied. We will highlight the cases where the weak interpretation is required.

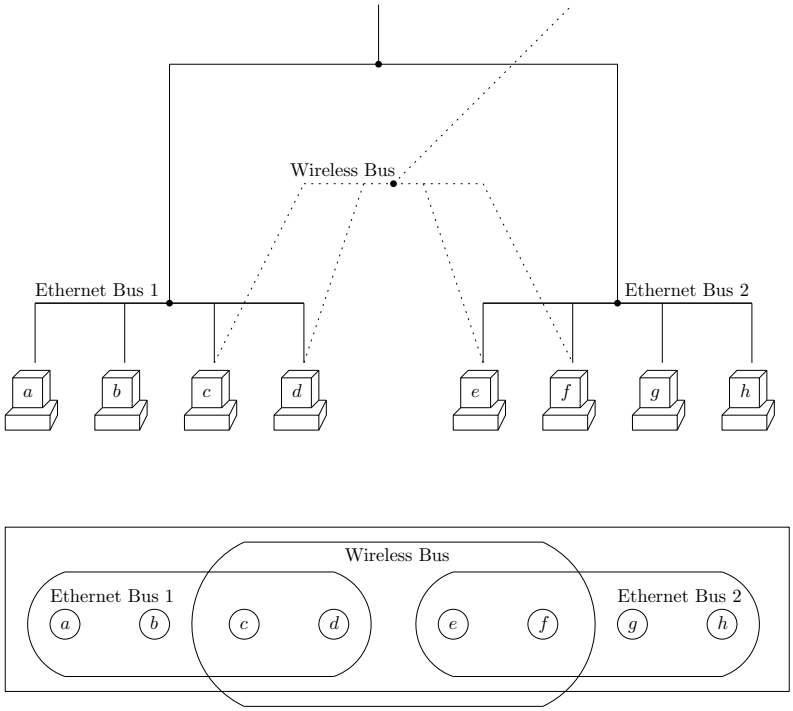

Fig. 1. A computer network and its corresponding hP representation.

3 Basic algorithms for network discovery—without IDs

In this section and the following, we study several basic distributed algorithms for network discovery, adapted to hP systems. Essentially, all cells start in the same state and with the same or similar set of rules, but there are several different scenarios:

1. Initially, cells know nothing about the structure in which they are linked, and must even discover their local neighborhood (i.e., their parents, children, siblings), as well as some global model topology characteristics (such as various dag measures or shortest paths).
2. As above, but each cell has its own ID (identifier) and is allowed to have custom rules for this ID.
3. As above, each cell has its own ID and also knows the details of its immediate neighbors (parents, children and, optionally, siblings).

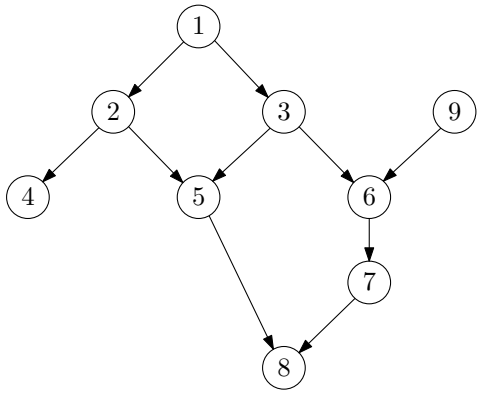

Fig. 2. Sample dag for illustrating our algorithms.

Algorithm 1: Broadcast to all descendants.

Precondition: Cells do not need any inbuilt knowledge about the network topology. All cells start in state s_0, with the same rules. The initiating cell has an additional object a, that is not present in any other cell.

Postcondition: All descendant cells are eventually visited and enter state s_1.

Rules:

1. $s_0 a \to s_1 a_{\downarrow}$, with $\alpha = min$, $\beta = repl$.
2. $s_1 a \to s_1$, with $\alpha = par$.

□

Proof. This is a *deterministic* algorithm. Rule 1 is applied exactly once, when the cell is in state s_0 and an a is available. This a is consumed, the cell enters state s_1 and another a is sent to all the children, replicated as necessary. Additional a's may appear in a cell, because, in a dag structure, a cell may have more than one parent. Rule 2 is applicable in state s_1 and silently discards any additional a's, without changing the state and without interacting with other cells. All a's will eventually disappear from the system—however, cells themselves may never know that the algorithm has completed and no other a's will come from its parents. By induction, all descendants will receive an a and enter state s_1. □

Remark 1.

- This broadcast algorithm can be initiated anywhere in the dag. However, it is probably most useful when initiated on a dag source, or on all sources at the same time (using the same object a or a different object for each source).

- This algorithm completes after $h + 1$ P-steps, where h is the *height* of the initiating node.
- State s_1 may be reached before the algorithm completes and cannot be used as a termination indicator.
- Several other broadcasting algorithms can be built in a similar manner, such as *broadcast to all ancestors* or *broadcast to all reachable cells* (ancestors and descendants).
- This algorithm family follows the approach used by Ciobanu *et al.* [4, 3], for tree based algorithms, called *Skin membrane broadcast* and *Generalized broadcast*.

Example 2. We illustrate the algorithm for broadcasting to all descendants, for the hP system shown in Figure 2.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	s_0a	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1	s_0a	s_0a	s_0	s_0	s_0	s_0	s_0	s_0
2	s_1	s_1	s_1	s_0a	s_0aa	s_0a	s_0	s_0	s_0
3	s_1	s_1	s_1	s_1	s_1a	s_1	s_0a	s_0a	s_0
4	s_1	s_1	s_1	s_1	s_1	s_1	s_1	s_1a	s_0
5	s_1	s_1	s_1	s_1	s_1	s_1	s_1	s_1	s_0

Algorithm 2: Counting all paths from a given ancestor.

Precondition: Cells do not need any inbuilt knowledge about the network topology. All cells start in state s_0 and with the same rules. The initiating cell has an additional object a, not present in any other cell.

Postcondition: All descendant cells are eventually visited, enter state s_1 and will have a number of b's equal to the number of distinct paths from the initiating cell.

Rules:

1. $s_0a \to s_1ba_{\downarrow}$, with $\alpha = par$, $\beta = repl$.
2. $s_1a \to s_1ba_{\downarrow}$, with $\alpha = par$, $\beta = repl$.

□

Proof. This is a *deterministic* algorithm. Rule 1 is applied when the cell is in state s_0 and an a is available. This a is consumed, the cell enters state s_1, a b is generated and another a is sent to all its children, replicated as necessary. Additional a's may appear in a cell, because, in a dag structure, a cell may have more than one parent. Rule 2 is similar to rule 1. State s_1 is similar to state s_0 and is not essential here, it appears here only to mark visited cells. The number of generated b's is equal to the number of received a's, which eventually will be equal to the number of paths from the initiating cell. All a's will eventually disappear from the

system—however, cells themselves may never know that the algorithm has completed, that no other a's will come from a parent and all paths have been counted. A more rigorous proof will proceed by induction. □

Remark 2.

- This algorithm completes after $h + 1$ P-steps, where h is the *height* of the initiating node.
- State s_1 may be reached before the algorithm completes and cannot be used as a termination indicator.
- Several other path counting algorithms can be built in a similar manner, such as the number of *paths to a given descendant*.

Example 3. We illustrate the algorithm for counting all paths from a given ancestor, for the hP system shown in Figure 2.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	s_0a	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1b	s_0a	s_0a	s_0	s_0	s_0	s_0	s_0	s_0
2	s_1b	s_1b	s_1b	s_0a	s_0aa	s_0a	s_0	s_0	s_0
3	s_1b	s_1b	s_1b	s_1b	s_1bb	s_1b	s_0a	s_0aa	s_0
4	s_1b	s_1b	s_1b	s_1b	s_1bb	s_1b	s_1b	s_1abb	s_0
5	s_1b	s_1b	s_1b	s_1b	s_1bb	s_1b	s_1b	s_1bbb	s_0

Algorithm 3: Counting the children of a given cell.

Precondition: Cells do not need any inbuilt knowledge about the network topology. The initiating cell and its children start in state s_0 and with the same rules. The initiating cell has an additional object a, not present in any other cell.

Postcondition: The initiating cell ends in state s_1 and with a number of c's equal to its child count. The child cells end in state s_1. As a side effect, other parents (if any) of these children will receive superfluous c's—however, these c's can be discarded, if needed (rules not shown here).

Rules:

1. $s_0a \to s_1p_{\downarrow}$, with $\alpha = min$, $\beta = repl$.
2. $s_0p \to s_1c_{\uparrow}$, with $\alpha = min$, $\beta = repl$.

□

Proof. This is a *deterministic* algorithm with a straightforward proof, not given here. □

Remark 3.

- This algorithm completes after 2 P-steps.
- Several other algorithms that enumerate the immediate neighborhood can be built in a similar manner, such as *counting parents*, *counting siblings*, *counting neighbors*.

Algorithm 4: Broadcast for counting all children.

Precondition: Cells do not need any inbuilt knowledge about the network topology. All cells start in state s_0 and with the same rules. The initiating cell has an additional object a, not present in any other cell.

Postcondition: Each descendant cell enters state s_1 and, eventually, will contain a number of c's equal to its child count.

Rules:

0. For state s_0:
   1) $s_0 a \to s_1 p_\downarrow$, with $\alpha = min$, $\beta = repl$.
   2) $s_0 p \to s_1 p_\downarrow c_\uparrow$, with $\alpha = min$, $\beta = repl$.
1. For state s_1:
   1) $s_1 p \to s_1$, with $\alpha = par$.

□

Proof. This a *deterministic* algorithm: the proof combines those from the broadcast algorithm (Algorithm 1) and the child counting algorithm (Algorithm 3). □

Remark 4.

- This algorithm runs in $h + 1$ P-steps, where h is the *height* of the initiating cell.
- State s_1 may be reached before the algorithm completes its cleanup phase and cannot be used as a termination indicator.
- As a side effect, any parent of the visited children that is not a descendant of the initiating node will receive superfluous c's.
- Several other algorithms that broadcast a request to count the immediate neighborhood can be built in a similar manner, such as *broadcast for counting all parents*, *broadcast for counting all siblings*, *broadcast for counting all neighbors*.

Example 4. We illustrate the algorithm for counting all children via broadcasting, for the hP system shown in Figure 2.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	s_0a	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1	s_0p	s_0p	s_0	s_0	s_0	s_0	s_0	s_0
2	s_1cc	s_1	s_1	s_0p	s_0pp	s_0p	s_0	s_0	s_0
3	s_1cc	s_1cc	s_1cc	s_1	s_1p	s_1	s_0p	s_0p	s_0c
4	s_1cc	s_1cc	s_1cc	s_1	s_1c	s_1c	s_1c	s_1p	s_0c
5	s_1cc	s_1cc	s_1cc	s_1	s_1c	s_1c	s_1c	s_1	s_0c

Algorithm 5: Counting heights by flooding.

Precondition: Cells do not need any inbuilt knowledge about the network topology. All cells start in state s_0, with the same rules and have no initial object.

Postcondition: All cells end in state s_2. The number of t's in each cell equals the distance from a furthest descendant.

Rules:

0. For state s_0:
   1) $s_0 \to s_1ac_\uparrow$, $\alpha = min$, $\beta = repl$.
1. For state s_1, the rules will run under the following *priorities*, under the *weak interpretation*:
   1) $s_1ac \to s_1atc_\uparrow$, $\alpha = max$, $\beta = repl$.
   2) $s_1c \to s_1$, $\alpha = max$.
   3) $s_1a \to s_2$, $\alpha = min$.

□

Proof. Each cell emits a single object c to each of its parents in the first step. During successive active steps, a cell either: (a) uses rule 1.3 to enter the terminating state s_2 or (b) continues via rule 1.1 to forward one c up to each of its parents. In the latter case, since we have $\alpha = max$, and as enabled by the weak interpretation of priorities, rule 1.2 is further used to remove all remaining c's (if any), in the same step. The cell safely enters the end state s_2 when no more c's appear. Induction shows that the set of times that c's appear is consecutive: if a cell at $k > 1$ links away emitted a c, then there must be another cell at $k - 1$ links away emitting another c. Finally, the number of times rule 1.1 is applied is the number of times a cell receives at least one new c from below. These steps are tallied by occurrences of the object t. □

Remark 5.

- The time complexity of this quick algorithm is $h + 2$ P-steps, where h is the height of the dag. The two extra P-steps correspond to the initial step and the step to detect no more c's.

- This algorithm, like other distributed flooding based algorithms, requires that all cells start at the same time. Achieving this synchronization could be a non-trivial task—see Section 5.
- This algorithm follows the approach by Ciobanu *et al.* [4, 3], for the tree based algorithm called *Convergecast*. Here we prefer to use the term *flooding*, and use the term *convergecast* for a result accumulation triggered by an initial broadcast.
- This algorithm makes critical use of the *weak interpretation* for *priorities*.

Example 5. We illustrate the algorithm for counting heights by flooding, for the hP system shown in Figure 2.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1acc	s_1acc	s_1acc	s_1a	s_1ac	s_1ac	s_1ac	s_1a	s_1ac
2	s_1acct	s_1act	s_1acct	s_2	s_1at	s_1act	s_1at	s_2	s_1act
3	s_1acctt	s_1att	s_1actt	s_2	s_2t	s_1att	s_2t	s_2	s_1actt
4	s_1act^3	s_2tt	s_1at^3	s_2	s_2t	s_2tt	s_2t	s_2	s_1at^3
5	s_1at^4	s_2tt	s_2t^3	s_2	s_2t	s_2tt	s_2t	s_2	s_2t^3
6	s_2t^4	s_2tt	s_2t^3	s_2	s_2t	s_2tt	s_2t	s_2	s_2t^3

Algorithm 6: Counting nodes in a single-source dag.

Precondition: Cells do not need any inbuilt knowledge about the network topology. All cells start in state s_0, with the same rules. The initiating cell is the source of a single-sourced dag and has an additional object a, not present in any other cell.

Postcondition: Eventually, the initiating cell will contain a number of c's equal to the number of all its descendants, including itself, which is also the required node count.

Rules:

0. For state s_0:
   1) $s_0a \to s_3p_{\downarrow}c$, with $\alpha = min$, $\beta = repl$.
   2) $s_0p \to s_1p_{\downarrow}$, with $\alpha = min$, $\beta = repl$.
1. For state s_1:
   1) $s_1 \to s_2c_{\uparrow}$, with $\alpha = min$, $\beta = one$.
2. For state s_2:
   1) $s_2c \to s_2c_{\uparrow}$, with $\alpha = max$, $\beta = one$.
   2) $s_2p \to s_2$, with $\alpha = max$.

□

Proof. We prove that the source will eventually contain the k copies of object c, where k is the order of the single-source dag. The source cell will produce a copy

of c following rule 0.1. A non-source cell σ_i will send one c to a parent σ_j, where $j \in \delta^{-1}(i)$, because a node is at state s_1 during at most one P-step, by rule 1.1. A cell σ_i will forward up, using rule 2.1, additional c's to one of its parents, which will eventually arrive at the source. □

Remark 6.

- This algorithm takes up to $2h$ P-steps, where h is the *height* of the initiating cell.
- The end state s_3 is not halting, may be reached before the algorithm completes and cannot be used as a termination indicator.

Example 6. We illustrate the algorithm for counting nodes in a single-sourced dag via convergecast, for the hP system shown in Figure 2, after removing node 9.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
0	s_0a	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_3c	s_0p	s_0p	s_0	s_0	s_0	s_0	s_0
2	s_3c	s_1	s_1	s_0p	s_0pp	s_0p	s_0	s_0
3	s_3c^3	s_2	s_2	s_1	s_1p	s_1	s_0p	s_0p
4	s_3c^3	s_2c	s_2cc	s_2	s_2p	s_2	s_1	s_1p
5	s_3c^6	s_2	s_2	s_2	s_2	s_2c	s_2c	s_2p
6	s_3c^6	s_2	s_2c	s_2	s_2	s_2c	s_2	s_2
7	s_3c^7	s_2	s_2c	s_2	s_2	s_2	s_2	s_2
8	s_3c^8	s_2	s_2	s_2	s_2	s_2	s_2	s_2

4 Basic algorithms for network discovery—with IDs

In this section we assume each cell has an unique ID and the cells only know their own ID. Objects may be tagged with IDs to aid in communication.

Algorithm 7: Counting descendants by convergecast—with cell IDs.

Precondition: Cells do not need any inbuilt knowledge about the network topology. For each cell with index i, $1 \leq i \leq m$, the alphabet includes special ID objects c_i and $\bar{c_i}$. All cells start in state s_0 and have the same rules, except several similar, but custom specific, rules to process the IDs. The initiating cell has an additional object a, not present in any other cell.

Postcondition: All visited cells enter state s_1 and, eventually, each cell will contain exactly one $\bar{c_i}$ for each descendant cell with index i, including itself: the number of these objects is the required count.

Rules:

0. For state s_0 and cell i (these are custom rules, specific for each cell):
   1) $s_0 a \to s_1 p_{\downarrow} \bar{c}_i$, with $\alpha = min$, $\beta = repl$.
   2) $s_0 p \to s_1 p_{\downarrow} c_{i\uparrow} \bar{c}_i$, with $\alpha = min$, $\beta = repl$.
1. For state s_1, the rules will run under the following *priorities*:
   1) $s_1 c_j \bar{c}_j \to s_1 \bar{c}_j$, for $1 \leq j \leq m$, with $\alpha = max$.
   2) $s_1 \bar{c}_j \bar{c}_j \to s_1 \bar{c}_j$, for $1 \leq j \leq m$, with $\alpha = max$.
   3) $s_1 c_j \to s_1 c_{j\uparrow} \bar{c}_j$, for $1 \leq j \leq m$, with $\alpha = max$, $\beta = repl$.
   4) $s_1 p \to s_1$, with $\alpha = max$.

□

Proof. Assume that δ is the underlying dag relation. For each cell σ_i, consider the sets $C_i = \{c_j \mid j \in \delta^*(i)\}$, $\bar{C}_i = \{\bar{c}_j \mid j \in \delta^*(i)\}$, which consist of ID objects matching σ_i's children. By induction on the dag height, we prove that each visited cell σ_i will eventually contain the set $\bar{C}_i$, and, if it is not the initiating cell, will also send up all elements of the set C_i, possibly with some duplicates (up to all its parents). The base case, height $h = 0$, is satisfied by rule 0.1, if σ_i is the initiator, or by rule 0.2, otherwise. For cell σ_i at height $h + 1$, by induction, each child cell σ_k sends up C_k, possibly with some duplicates. By rules 0.1 and 0.2, cell σ_i further acquires one $\bar{c}_i$ and, if not the initiator, sends up one c_i. From its children, cell σ_i acquires the multiset C'_i, consisting of all the elements of the set $\bigcup_{k \in \delta(i)} C_k = C_i \setminus c_i$, possibly with some duplications. Rule 1.3 sends up one copy of each element of multiset C'_i and records a barred copy of it. Rule 1.2 halves the number of duplicates in multiset C'_i. Rule 1.1 filters out duplicates in multiset C'_i, if a barred copy already exists. Rule 1.4 clears all p's, which are not needed anymore. □

Remark 7.

- Other counting algorithms can be built in a similar manner, such as *counting ancestors*, *counting siblings*, *counting sources* or *counting sinks*.
- The end state s_1 is not halting, it may be reached before the algorithm completes and cannot be used as a termination indicator.
- As a side effect, any parent of the visited children that is not a descendant of the initiating node may receive superfluous c_i's.
- This algorithm works under both *strong* and *weak* interpretation of *priorities*.

Example 7. We illustrate the algorithm for counting descendants via convergecast using cell IDs, for the hP system shown in Figure 2.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	s_0a	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1c_1	s_0p	s_0p	s_0	s_0	s_0	s_0	s_0	s_0
2	$s_1c_2c_3$ $\bar{c_1}$	s_1 $\bar{c_2}$	s_1 $\bar{c_3}$	s_0p	s_0pp	s_0p	s_0	s_0	s_0
3	s_1 $\bar{c_1}\bar{c_2}\bar{c_3}$	$s_1c_4c_5$ $\bar{c_2}$	$s_1c_5c_6$ $\bar{c_3}$	s_1 $\bar{c_4}$	s_1p $\bar{c_5}$	s_1 $\bar{c_6}$	s_0p	s_0p	s_0c_6
4	$s_1c_4c_5c_5c_6$ $\bar{c_1}\bar{c_2}\bar{c_3}$	s_1 $\bar{c_2}\bar{c_4}\bar{c_5}$	s_1 $\bar{c_3}\bar{c_5}\bar{c_6}$	s_1 $\bar{c_4}$	s_1c_8 $\bar{c_5}$	s_1c_7 $\bar{c_6}$	s_1c_8 $\bar{c_7}$	s_1p $\bar{c_8}$	s_0c_6
5	s_1 $\bar{c_1}\bar{c_2}\bar{c_3}\bar{c_4}\bar{c_5}\bar{c_5}\bar{c_6}$	s_1c_8 $\bar{c_2}\bar{c_4}\bar{c_5}$	$s_1c_7c_8$ $\bar{c_3}\bar{c_5}\bar{c_6}$	s_1 $\bar{c_4}$	s_1 $\bar{c_5}\bar{c_8}$	s_1c_8 $\bar{c_6}\bar{c_7}$	s_1 $\bar{c_7}\bar{c_8}$	s_1 $\bar{c_8}$	$s_0c_6c_7$
6	$s_1c_7c_8c_8$ $\bar{c_1}\bar{c_2}\bar{c_3}\bar{c_4}\bar{c_5}\bar{c_6}$	s_1 $\bar{c_2}\bar{c_4}\bar{c_5}\bar{c_8}$	s_1c_8 $\bar{c_3}\bar{c_5}\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_4}$	s_1 $\bar{c_5}\bar{c_8}$	s_1 $\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_7}\bar{c_8}$	s_1 $\bar{c_8}$	$s_0c_6c_7c_8$
7	s_1 $\bar{c_1}\bar{c_2}\bar{c_3}\bar{c_4}\bar{c_5}\bar{c_6}\bar{c_7}\bar{c_8}\bar{c_8}$	s_1 $\bar{c_2}\bar{c_4}\bar{c_5}\bar{c_8}$	s_1 $\bar{c_3}\bar{c_5}\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_4}$	s_1 $\bar{c_5}\bar{c_8}$	s_1 $\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_7}\bar{c_8}$	s_1 $\bar{c_8}$	$s_0c_6c_7c_8$
8	s_1 $\bar{c_1}\bar{c_2}\bar{c_3}\bar{c_4}\bar{c_5}\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_2}\bar{c_4}\bar{c_5}\bar{c_8}$	s_1 $\bar{c_3}\bar{c_5}\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_4}$	s_1 $\bar{c_5}\bar{c_8}$	s_1 $\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_7}\bar{c_8}$	s_1 $\bar{c_8}$	$s_0c_6c_7c_8$

Algorithm 8: Shortest paths from a given cell.

Precondition: Cells do not need any inbuilt knowledge about the network topology. For each cells with indices i, j, $1 \leq i, j \leq m$, the alphabet includes special ID objects: p_i, $\bar{p_i}$, $\bar{c_i}$, x_{ij}. All cells start in state s_0 and have the same rules, except several similar but custom specific rules to process the IDs. The initiating cell has an additional object a, not present in any other cell.

Postcondition: This algorithm builds a *shortest paths* spanning tree, that is a breadth-first tree rooted at the initiating cell and preserving this dag's relation δ. Each visited cell σ_i, except the initiating cell, will contain one $\bar{p_k}$, indicating its parent σ_k in the spanning tree. Each visited cell σ_i will also contain one $\bar{c_j}$ for each σ_j that is a child of σ_i in the spanning tree, i.e., it will contain all elements of the set $\{\bar{c_j} \mid (i,j) \in \delta, \sigma_j \text{ contains } \bar{p_i}\}$.

Rules:

0. For state s_0 and cell i (custom rules, specific for cell i):
   1) $s_0a \to s_1p_{i\downarrow}$, with $\alpha = min$, $\beta = repl$.
   2) $s_0p_j \to s_1\bar{p_j}p_{i\downarrow}x_{ji\uparrow}$, for $1 \leq j \leq m$, with $\alpha = min$, $\beta = repl$.
   3) $s_0x_{kj} \to s_0$, for $1 \leq k, j \leq m, k \neq i$, with $\alpha = max$.
1. For state s_1 and cell i (custom rules, specific for cell i):
   1) $s_1x_{ij} \to s_1\bar{c_j}$, for $1 \leq j \leq m$, with $\alpha = max$.
   2) $s_1p_j \to s_1$, for $1 \leq j \leq m$, with $\alpha = max$.
   3) $s_1x_{kj} \to s_1$, for $1 \leq k, j \leq m, k \neq i$, with $\alpha = max$.

□

Proof. It is clear that every visited cell σ_i, except the initiating cell, contains one $\bar{p_k}$ where $k \in \delta^{-1}(i)$ from rule 0.2. By a node's height, we prove that a cell σ_i will

contain the set $C_i = \{\bar{c_j} \mid (i,j) \in \delta, \sigma_j \text{ contains } \bar{p_i}\}$. For height 0, $C_i = \emptyset$ is true since a sink σ_i does not have any children to receive an x_{ji}—see rule 0.2. For a cell σ_i of height greater than 0, first observe that rule 1.1 is only applied if rule 0.2 has been applied for a child cell j. Thus, C_i contains all $\bar{c_j}$ such that (i,j) is in the spanning tree. Those x_{kj}'s are removed by rule 0.3, and x_{ij}'s that are not converted to $\bar{c_j}$ are removed by rule 1.3. □

Remark 8.

- For this algorithm, cells need additional symbols, see the precondition.
- This algorithm takes $h+1$ P-steps, where h is the *height* of the initiating cell.
- The end state s_1 is not halting, it may be reached before the algorithm completes and cannot be used as a termination indicator.
- As a side effect, any parent of the visited children that is not a descendant of the initiating node will receive superfluous x_{ij}'s, but they are removed by rule 0.3.
- The rules for state s_0 make effective use of our rewrite mode refinement: rules 0.1 and 0.2 use $\alpha = min$, while rule 0.3 uses $\alpha = max$.
- Provided that arcs are associated with weights, this algorithm can be extended into a distributed version of the *Bellman-Ford algorithm* [7].

Example 8. We illustrate the algorithm for counting nodes in a single-source dag via convergecast, for the hP system shown in Figure 2. The thick arrows in Figure 3 show the resulting spanning tree.

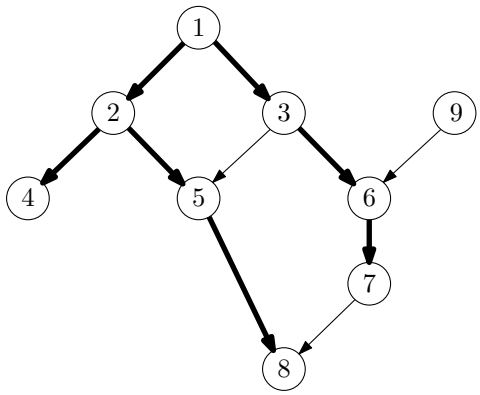

Fig. 3. A spanning tree created by the shortest paths algorithm (Algorithm 8).

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	s_0a	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1	s_0p_1	s_0p_1	s_0	s_0	s_0	s_0	s_0	s_0
2	$s_1x_{12}x_{13}$	$s_1\bar{p_1}$	$s_1\bar{p_1}$	s_0p_2	$s_0p_2p_3$	s_0p_3	s_0	s_0	s_0
3	$s_1\bar{c_2}\bar{c_3}$	$s_1\bar{p_1}x_{24}x_{25}$	$s_1\bar{p_1}x_{25}x_{36}$	$s_1\bar{p_2}$	$s_1p_3\bar{p_2}$	$s_1\bar{p_3}$	s_0p_6	s_0p_5	s_0x_{36}
4	$s_1\bar{c_2}\bar{c_3}$	$s_1\bar{p_1}\bar{c_4}\bar{c_5}$	$s_1\bar{p_1}\bar{c_6}$	$s_1\bar{p_2}$	$s_1\bar{p_2}x_{58}$	$s_1\bar{p_3}x_{67}$	$s_1\bar{p_6}x_{58}$	$s_1p_7\bar{p_5}$	s_0
5	$s_1\bar{c_2}\bar{c_3}$	$s_1\bar{p_1}\bar{c_4}\bar{c_5}$	$s_1\bar{p_1}\bar{c_6}$	$s_1\bar{p_2}$	$s_1\bar{p_2}\bar{c_8}$	$s_1\bar{p_3}\bar{c_7}$	$s_1\bar{p_6}$	$s_1\bar{p_5}$	s_0

5 The Firing-Squad-Synchronization-Problem (FSSP)

More sophisticated network algorithms can be built on the fundamental building blocks discussed in the previous sections.

For a given hP system, with cells $\sigma_1, \ldots, \sigma_m$, we now consider the problem of synchronizing a set of cells $F \subseteq \{\sigma_1, \ldots, \sigma_m\}$, where all cells in the set F synchronize by entering a designated firing state *simultaneously* and (of course) *for the first time*.

There are several ways to solve the problem. In the simplest scenario, we are allowed to extend the structure. The tree structures allow only limited extensions, that are not useful in solving this problem. However, dag structures (or more general models) allow extensions that greatly simplify the solution to this problem and other similar problems, to the point that they become "trivial". We take this as an additional argument supporting the introduction of dag structures in the context of P systems.

Here, we consider only the first scenario, in which we may extend the structure. We select an arbitrary subset of *squad* cells, $F \subseteq \{\sigma_1, \ldots, \sigma_m\}$, that we wish to synchronize (possibly the whole set), and an arbitrary *commander* cell $\sigma_c \in F$. As a simple solution to this problem, we add an external cell, called *sergeant*, to the existing hP system and additional communication channels from the sergeant to all cells in the set F. The commander initiates the synchronization process by sending a notification to the sergeant. When the sergeant receives this notification, the sergeant sends commands to all cells in the set F, which prompts the cells to synchronize by entering the firing state. The algorithm below does not consider the sergeant as part of the firing squad. However, with a simple extension (not shown here), we can also cover the case when the sergeant is to be part of the firing squad.

Algorithm 9: A dag synchronization algorithm.

Precondition: We are given an hP system with m cells $\sigma_1, \ldots, \sigma_m$, a squad subset $F \subseteq \{\sigma_1, \ldots, \sigma_m\}$, and a commander cell $\sigma_c \in F$. We extend the underlying dag structure by adding a new sergeant cell σ_{m+1} and additional communication channels from σ_{m+1}, as parent, to σ_i, as child, for each $i \in F \subseteq X$.

All cells start in the state s_0 and have the same rules. The state s_1 is the firing state. Initially, the sergeant σ_{m+1} has an object c, the commander σ_c has an object

a, and all other cells have no object.

Postcondition: All cells in the set F enter the state s_1, simultaneously and for the first time, after 3 P-steps.

Rules:

0. For state s_0, the rules will run under the following *priorities*:
   1) $s_0 a \to s_0 b_\uparrow$, with $\alpha = min$, $\beta = repl$.
   2) $s_0 bc \to s_0 f_\downarrow$, with $\alpha = min$, $\beta = repl$.
   3) $s_0 b \to s_0$, with $\alpha = min$.
   4) $s_0 f \to s_1$ with $\alpha = min$.

□

Proof. At step 1, the commander sends a b notifier to all its parents, including the newly created sergeant, via rule 0.1. At step 2, the sergeant sends the firing command f to all squad cells, using rule 0.2. All other commander's parents clear their b notifiers at step 2, using rule 0.3. At step 3, all squad cells enter the firing state s_1, using rule 0.4. This algorithm will work under both weak and strong interpretations of priorities. □

Example 9. We illustrate the algorithm for synchronizing the hP system shown in Figure 4. This hP system consists of 7 cells $\{\sigma_1, \ldots, \sigma_7\}$, $F = \{\sigma_1, \ldots, \sigma_5\}$ and σ_5 is the commander. The actual system structure is irrelevant in this case and was replaced by a blob that circumscribes the cells $\sigma_1, \ldots, \sigma_7$. In the diagram, this structure has already been extended by the sergeant cell σ_8 and the required communication channels.

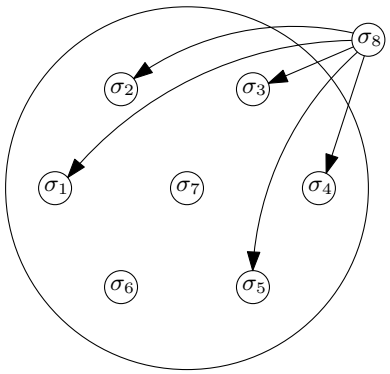

Fig. 4. An hP system for the synchronization algorithm (Algorithm 9), extended by the sergeant cell σ_8 and the required communication channels.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
0	s_0	s_0	s_0	s_0	s_0a	s_0	s_0	s_0c
1	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0bc
2	s_0f	s_0f	s_0f	s_0f	s_0f	s_0	s_0	s_0
3	s_1	s_1	s_1	s_1	s_1	s_0	s_0	s_0

Bernardini *et al.* present a deterministic solution for transition P systems with polarizations and priorities [2], which works in time $4N + 2H$, where N and H are the number of tree nodes and tree height, respectively. Alhazov *et al.* present another deterministic solution for P systems with promoters and inhibitors [1], which works in time $3H$, where H is the tree height.

In a follow-up paper, [5], we provide a more constrained solution, that does not need structural extension, and covers both hP and nP systems. Our deterministic solution uses rules applied under the weak priority scheme and works in time $6R$, where R is the radius of the commander in the underlying dag/digraph.

6 Planar representation

We define a *simple region* as the interior of a simple closed curve (Jordan curve). By default, all our regions will be delimited by simple closed curves that are also smooth, with the possible exception of a finite number of points. This additional assumption is not strictly needed, but simplifies our arguments.

A simple region R_j is *directly contained* in a simple region R_i, if $R_j \subset R_i$ and there is no simple region R_k, such that $R_j \subset R_k \subset R_i$ (where $\subset$ denotes strict inclusion).

It is well known that any transition P system has a planar Venn-like representation, with a 1:1 mapping between its tree nodes and a set of hierarchically nested simple regions. Conversely, any single rooted set of hierarchically nested simple regions can be interpreted as a tree, which can further form the structural basis of a number of transition P systems.

We have already shown that this planar representation can be generalized for hP systems based on canonical dags (i.e., without transitive arcs) and arbitrary sets of simple regions (not necessarily nested), while still maintaining a 1:1 mapping between dag nodes and simple regions [8].

Specifically, any hP system structurally based on a canonical dag can be intensionally represented by a set of simple regions, where direct containment denotes a parent-child relation. The converse is also true, any set of simple regions can be interpreted as a canonical dag, which can further form the structural basis of a number of hP systems.

We will now provide several solutions to our open question [8]: How to represent the other dags, that do contain transitive arcs? First, we discuss a negative result. First, a counter-example that appeals to the intuition, and then a theorem with a brief proof.

Example 10. Consider the dag (a) of Figure 5, where nodes $1, 2, 3$ are to be represented by simple regions R_1, R_2, R_3, respectively. We consider the following three candidate representations: (e), (f) and (g). However, none of them properly match the dag (a), they only match dags obtained from (a) by removing one of its arcs:

(e) represents the dag (b), obtained from (a) by removing the arc $(1, 3)$;
(f) represents the dag (c), obtained from (a) by removing the arc $(1, 2)$;
(g) represents the dag (d), obtained from (a) by removing the arc $(2, 3)$.

Theorem 1. *Dags with transitive arcs cannot be planarly represented by simple regions, with a 1:1 mapping between nodes and regions.*

Proof. Consider again the counter-example in Example 10. The existence of arcs $(2, 3), (1, 2)$ requires that $R_3 \subset R_2 \subset R_1$. This means that R_3 cannot be directly contained in R_1, as required by the arc $(1, 3)$. □

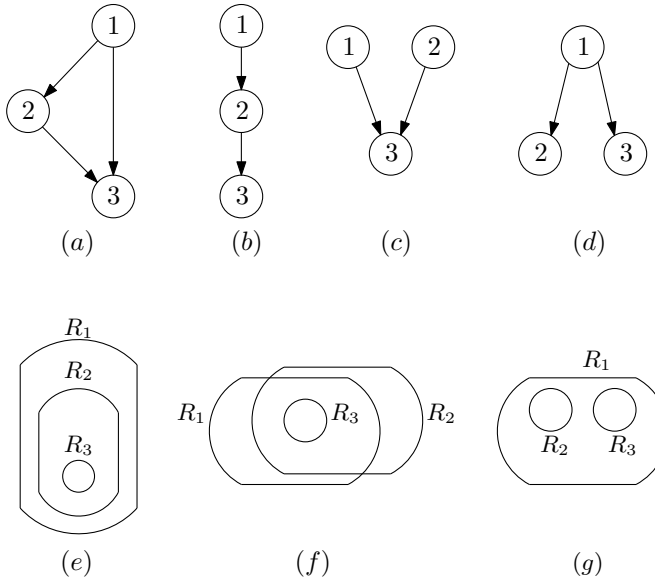

Fig. 5. A counter-example for planar representation of non-canonical dags.

It is clear, in view of this negative result, that we must somehow relax the requirements, if we want to obtain meaningful representations for general hP systems, based on dag structure that may contain transitive arcs. We consider in turn five tentative solutions.

6.1 Solution I: Self-intersecting curves

We drop the requirement of mapping nodes to simple regions delimited by simple closed curves. We now allow self-intersecting closed curves with inward folds. A node can be represented as the union of *subregions*: first, a base simple region, and, next, zero, one or more other simple regions, which are delimited by inward folds of base region's contour (therefore included in the base region). For this solution, we say that there is an arc (i, j) in the dag if and only if a subregion of R_i directly contains region R_j, where regions R_i, R_j represent nodes i, j in the dag, respectively.

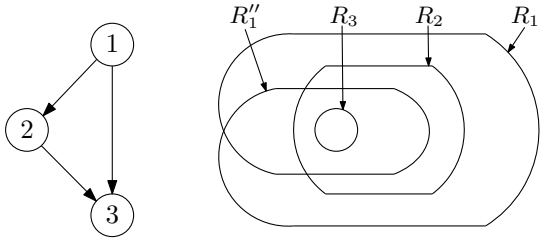

Fig. 6. Solution I: R_1 is delimited by a self-intersecting closed curve.

Example 11. The region R_1 in Figure 6 is delimited by a self-intersecting closed curve with an inward fold that defines the inner R_1'' subregion. Note the following relations:

- $R_1 = R_1 \cup R_1''$, thus R_1'' is a subregion of R_1;
- R_1 directly contains R_2, which indicates the arc $(1, 2)$;
- R_2 directly contains R_3, which indicates the arc $(2, 3)$;
- R_1'' directly contains R_3, which indicates the transitive arc $(1, 3)$, because R_1'' is a subregion of R_1.

Remark 9. It is difficult to visualize a cell that is modelled by a self-intersecting curve. Therefore, this approach does not seem adequate.

6.2 Solution II: Distinct regions

We drop the requirement of a 1:1 mapping between dag nodes and regions. Specifically, we accept that a node may be represented by the union of one or more distinct simple regions, here called *subregions*. Again, as in Solution I, an arc (i, j) is in the dag if and only if a subregion of R_i directly contains region R_j.

Example 12. In Figure 7, the simple region R_1 is the union of two simple regions, R_1' and R_1'', connected by a dotted line. Note the following relations:

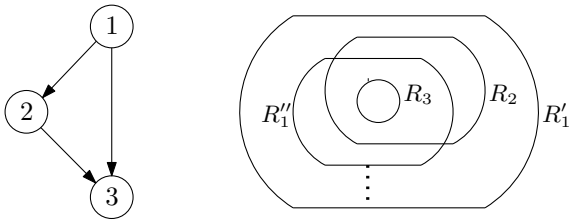

Fig. 7. Solution II: R_1 is the union of two simple regions, R_1' and R_1''.

- $R_1 = R_1' \cup R_1''$, thus R_1' and R_1'' are subregions of R_1;
- R_1' directly contains R_2, which indicates the arc $(1, 2)$, because R_1' is a subregion of R_1;
- R_2 directly contains R_3, which indicates the arc $(2, 3)$;
- R_1'' directly contains R_3, which indicates the transitive arc $(1, 3)$, because R_1'' is a subregion of R_1.

Remark 10. In Example 12, a dotted line connects two regions belonging to the same node. It is difficult to see the significance of such dotted lines in the world of cells. Widening these dotted lines could create self-intersecting curves—a solution which we have already rejected. Two distinct simple regions should represent two distinct cells, not just one. Therefore, this approach does not seem adequate either.

6.3 Solution III: Flaps

We again require simple regions, but we imagine that our representation is an infinitesimally thin "sandwich" of several superimposed layers, up to one distinct layer for each node (see Figure 8b). Initially, each region is a simple region that is conceptually partitioned into a *base subregion* (at some bottom layer) and zero, one or more other *flap subregions*, that appear as flaps attached to the base. These flaps are then folded, in the three-dimensional space, to other "sandwich" layers (see Figure 8c). The idea is that orthogonal projections of the regions corresponding to destinations of transitive arcs, which cannot be contained directly in the base region, will be directly contained in such subregions (or vice-versa). Because the thin tethered strip that was used for flapping is not relevant, it is represented by dots (see Figure 8d). As in the previous solutions, an arc (i, j) is in the dag if and only if a subregion S_k of R_i directly contains region R_j.

Superficially, this representation looks similar to Figure 7. However, its interpretation is totally different, it is now a flattened three-dimensional object. We can visualize this by imagining a living organism that has been totally flattened by a roller-compactor (apologies for the "gory" image).

We next give a constructive algorithm that takes as input a dag (X, δ) and produces a set of overlapping regions $\{R_k \mid k \in X\}$, such that $(i, j) \in \delta$ if and only if a subregion of R_i directly contains R_j.

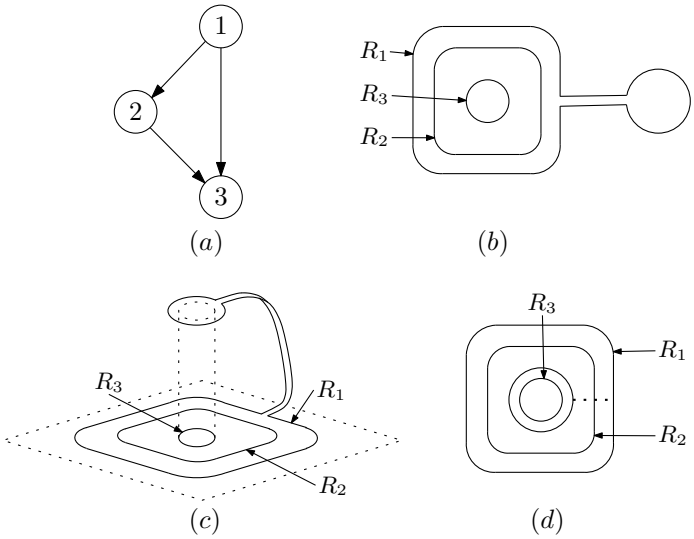

Fig. 8. The process described in Solution III.

Algorithm 10: DagToRegions.

Input: dag (X, δ).
Output: flattened regions $\{R_k \mid k \in X\}$.

Step 1: Reorder the nodes of the dag (X, δ) to be in reverse topological order. (That is, sink nodes come before source nodes.)

Step 2: For each node i in δ ordered as in step 1 do:
If i is a sink:
Create a new region R_i disjoint from all previous regions.
Otherwise:
Create a base region of R_i by creating a simple closed region properly containing the union of all regions R_j such that $(i, j) \in \delta$. Further, for any transitive arc (i, j) create a flap subregion that directly contains R_j and attach it with a strip to the edge of the base region.

□

Remark 11. In the set constructed by this algorithm, if two or more transitive arcs are incident to a node j then the respective flaps (without tethers) may share the same projected subregion directly containing region R_j.

Example 13. Figure 9 shows an input dag with 6 nodes, 3 transitive arcs and its corresponding planar region representation. Note the reverse topological order is $6, 5, 4, 3, 2, 1$ and the regions R_1 and R_2 use the same flap subregions containing the region R_6.

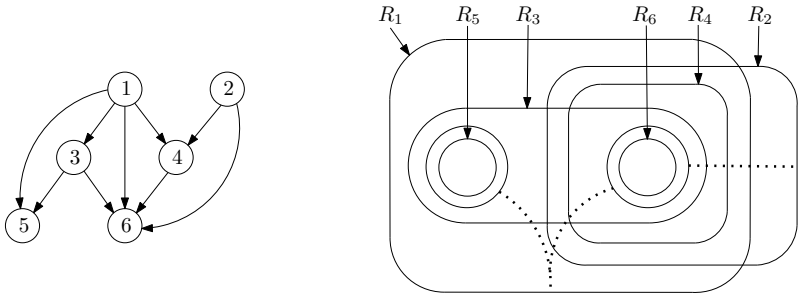

Fig. 9. Illustration of Algorithm 10.

Theorem 2. *Every dag with transitive arcs can be represented by a set of regions with folded flaps, with a 1:1 mapping between nodes and regions.*

Proof. We show by induction on the order of the dags that we can always produce a corresponding planar representation. First, note that any dag can be recursively constructed by adding a new node i and arcs incident from i to existing nodes. Note that Algorithm 10 builds planar representations from sink nodes (induction base case) to source nodes (inductive case). Hence, any dag has at least one folded planar representation, depending on the topological order used. We omit the details of how to ensure non-arcs; this can be easily achieved by adding "spikes" to the regions—see our first paper for representing non-transitive dags [8]. □

Theorem 3. *Every set of regions with folded flaps can be represented by a dag with transitive arcs, with a 1:1 mapping between nodes and regions.*

Proof. We show how to produce a unique dag from a folded planar representation. The first step is to label each region R_k, which will correspond to node $k \in X$ of a dag (X, δ). We add an arc (i, j) to δ if an only if a subregion of R_i directly contains the region R_j. □

Remark 12. One could imagine an additional constraint, that nodes, like cells, need to differentiate between its outside and inside or, in a planar representation, between up and down. We can relate this to membrane polarity, but we refrain from using this idea here, because it can conflict with the already accepted role of polarities in P systems. It is clear that, looking at our example, this solution does not take into account this *sense of direction.*

For example, considering the scenario of Figure (9), regions R_3, R_2 and R'_1 (the base subregion of R_1) can be stacked "properly", i.e., with the bottom side of R_3 on the top side of R_2 and the bottom side of R_2 on the top side of R'_1. However, the top side of R''_1 (the flap of R_1) will improperly sit on the top side of R_3, or, vice-versa, the bottom side of R''_1 will improperly sit on the bottom side of R_3.

Can we improve this? The answer follows.

6.4 Solution IV: Flaps with half-twists

This is a variation of Solution III, that additionally takes proper care of the outside/inside (or up/down) directions. We achieve this by introducing half-twists (as used to build Moebius strips), of which at most one half-twist is needed for each simple region.

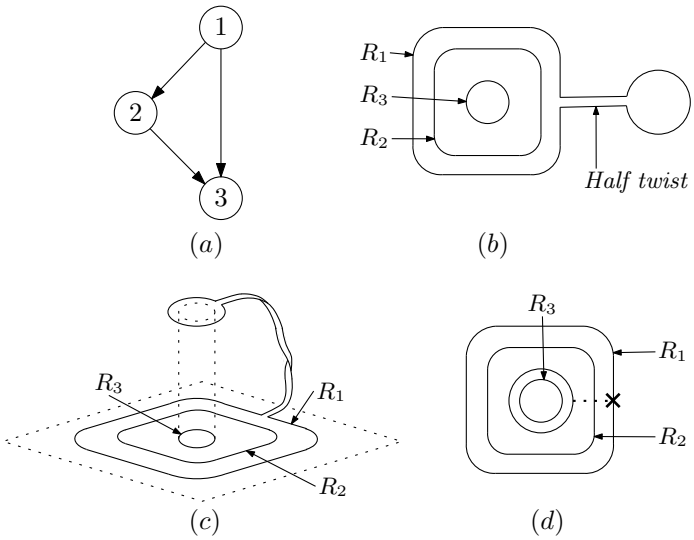

Fig. 10. The process described in Solution IV.

Example 14. Figure 10 describes this process.

(a) a given dag with three nodes, $1, 2$ and 3;
(b) three simple regions, R_1, R_2 and R_3, still in the same plane;
(c) R_1 flapped and half-twisted in three-dimensional space;
(d) final "roller-compacted" representation, where dots represent the thin strip that was flapped, and the mark $\times$ a possible location of the half-twist.

Corollary 1. *Dags with transitive arcs can be represented by regions with half-twisted flaps, with a 1:1 mapping between nodes and regions.*

Proof. Since half-twisted flaps are folded flaps, the projection of the boundary of the base and flaps used for a region is the same region as given in the proof of Theorems 2 and 3, provided we always twist a fold above its base. □

Remark 13. This solution solves all our concerns here and seems the best, taking into account the impossibility result (Theorem 1).

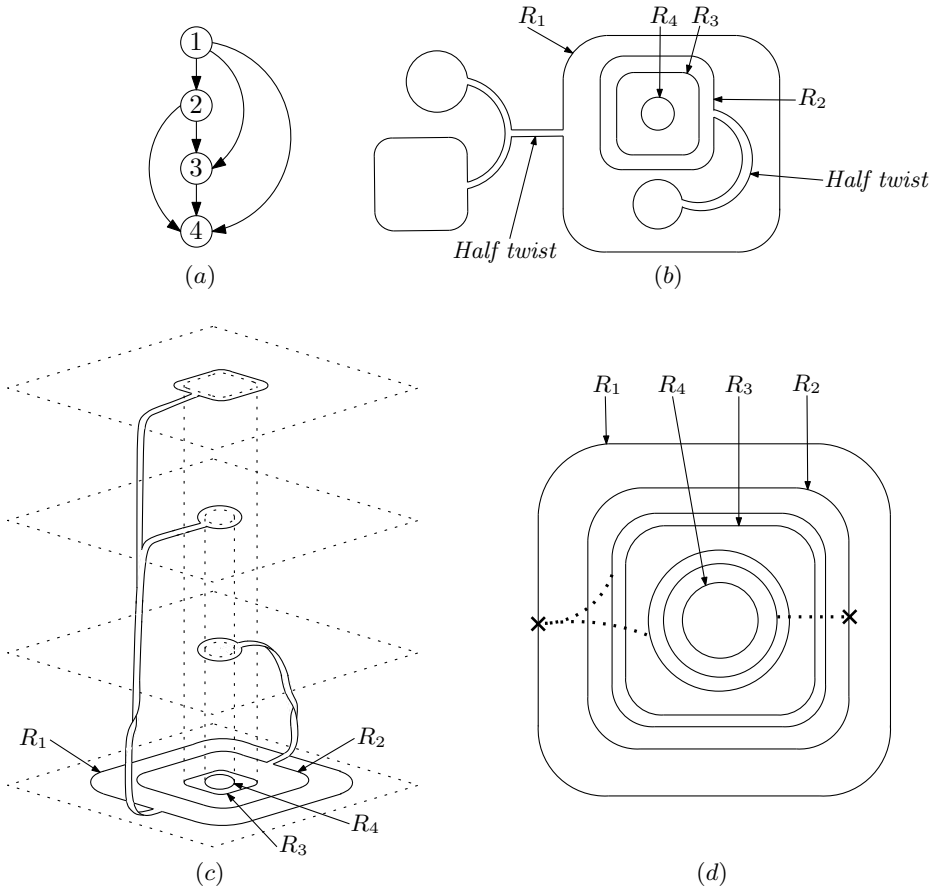

Fig. 11. The process described in Solution IV.

6.5 Solution V: Moebius strips

To be complete, we mention another possible solution, which removes any distinction between up and down sides. This representation can be obtained by representing membranes by (connected) Moebius strips.

Perhaps interestingly, Solutions IV and V seem to suggest links (obviously superficial, but still links) to modern applications of topology (Moebius strips and ladders, knot theory) to molecular biology, for example, see [6].

7 Conclusions

In this paper we have presented several concrete examples of hP systems for the discovery of basic membrane structure. Our primary goal was to show that, with

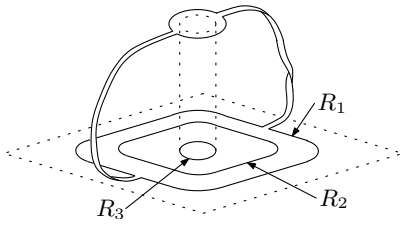

Fig. 12. The process described in Solution IV.

the correct model in terms of operational and communication modes, we could present simple algorithms. Our secondary goal was to obtain reasonably efficient algorithms.

We first started with cases, where the cells could be anonymous, and showed, among other things, how an hP system could (a) broadcast to descendants, (b) count paths between cells, (c) count children and descendants, and (d) determine cell heights. We then provided examples where we allowed each cell to know its own ID and use it as a communication marker. This model is highlighted by our algorithm that computes all the shortest paths from a given source cell —a simplified version of the distributed Bellman-Ford algorithm, with all unity weights. For each of our nontrivial algorithms, we illustrated the hP system computations on a fixed dag, providing step-by-step traces.

We then moved onto a simple solution that can be used to synchronize a subset (possibly all) of the states of the membrane's cells. We presented a fast trivial solution that requires structural extensions, which are straightforward with dags, but not applicable to trees. In a related paper [5], we describe a sophisticated solution that works on dags without extending the structure.

Finally, we focused on visualizing hP systems in the plane. We presented a natural model, using folded simple closed regions to model the membrane interconnections, including the transitive channels, as specified by an arbitrary dag structure of a hP system.

As with most ongoing projects, there are several open problems regarding practical computing using P systems and their extended models. We end by mentioning just a few, closely related to the development of fundamental algorithms for discovery of membrane topology.

- In terms of using membrane computing as a model for realistic networking, is there a natural way to route a message between cells (not necessarily connected directly) using messages, tagged by addressing identifiers, in analogy to the way messages are routed on the internet, with dynamically created routing information?

- What are the system requirements to model fault tolerant computing? The tree structure seems to fail here, because a single node failure can disconnect the graph and make consensus impossible. Is the dag structure versatile enough?
- Do we have the correct mix of rewrite and transfer modes for membrane computing? For example, in which situations can we exploit parallelism and in which scenarios are we forced to sequentially apply rewrite rules?

Acknowledgements

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

References

1. A. Alhazov, M. Margenstern, and S. Verlan. Fast synchronization in P systems. In David W. Corne, Pierluigi Frisco, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Workshop on Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 2008.
2. 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.
3. G., R. Desai, and A. Kumar. Membrane systems and distributed computing. In Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *WMC-CdeA*, volume 2597 of *Lecture Notes in Computer Science*, pages 187–202. Springer, 2002.
4. G. Ciobanu. Distributed algorithms over communicating membrane systems. *Biosystems*, 70(2):123–133, 2003.
5. M.J. Dinneen, Y.-B. Kim, and R. Nicolescu. The firing squad problem revisited (work in progress). Technical report, The University of Auckland, 2009.
6. E. Flapan. *When Topology Meets Chemistry: A Topological Look at Molecular Chirality*. Cambridge University Press, 2000.
7. N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
8. R. Nicolescu, M.J. Dinneen, and Y.-B. Kim. Structured modelling with hyperdag P systems: Part A. Report CDMTCS-342, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland, New Zealand, December 2008.
9. Gh. Păun. *Membrane Computing-An Introduction*. Springer-Verlag, 2002.
10. Gh. Păun. Introduction to membrane computing. In Gabriel Ciobanu, Mario J. Pérez-Jiménez, and Gheorghe Păun, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 1–42. Springer, 2006.