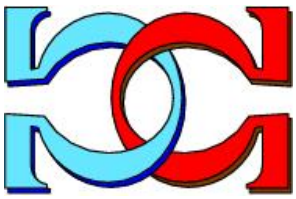

CDMTCS
Research
Report
Series

Towards Practical P Systems: Discovery Algorithms

Sonny Datt
Michael J. Dinneen
Department of Computer Science,
University of Auckland,
Auckland, New Zealand

CDMTCS-414
December 2011

Centre for Discrete Mathematics and
Theoretical Computer Science

Towards Practical P Systems: Discovery Algorithms

Sonny Datt and Michael J. Dinneen
Department of Computer Science
The University of Auckland, Private Bag 92019
Auckland, New Zealand
ndat001@ec.auckland.ac.nz, mjd@cs.auckland.ac.nz

Abstract

In this paper we offer an introduction to P Systems, a relatively new computational model inspired by biology. Our paper covers four of the more popular *state-capable* systems and provides a brief explanation of each, including at least one problem that each specific P System solves faster than a standard Turing machine (some NP-Complete problems are solved in polynomial *P Steps*). After determining common elements across the systems we go on to describe discovery algorithms: a high level approach to solving topology problems within P Systems. Our description includes a mapping from our generic or high level system to the low level rule sets for each of the P Systems covered in this paper. We also include some documentation on the usage of our approach with an example and some implications of the model itself.

1 Discovery Algorithms and P Systems

P Systems (a field of Membrane Computing) are parallel or distributed computational models inspired by living cells in biology which have been influenced by the needs of Computer Science. P Systems were originally conceived as purely theoretical objects: a recursive membrane or cell structure, i.e. each membrane had a set of 'child' membranes (similar to a rooted tree) which could each have their own set of children in turn. In addition to the sub-membranes, each cell stores within it a multiset of objects. Membranes or cells operate under a set of rules which govern their evolution, including communication which may be targeted (e.g. send-to-parent). Some P Systems also have dynamic restructure rules which allow the system itself to change form relatively quickly during the computation. But, arguably, the most powerful aspect of a P System is its parallelism: individual cells work under an intra-parallel model (within the cell itself) and cells interact with each other in an inter-parallel manner across the system (interact with one another simultaneously). We note that a P System is a parallel computing model

that does not require a problem be broken into independent sub-problems to be solved (as in matrix multiplication).

Păun's original system gained interest quickly as it solved SAT [11] in polynomial *P Steps* [13]—the natural measure of time complexity for this computational model. Since Păun's original proposal of a very limited system, many more complex or specifically defined systems have emerged. Some examples include the Simple P Module, Hyperdag P System and the Neural-like Tissue P System. We intend to define discovery rules to work within any of the models described in this paper, which will allow us to describe discovery algorithms to determine the topology of a system.

Discovery algorithms refer to a form of computation that is performed on a P System. They allow you to gain (topological) information about the P System in real time using only local information. Such algorithms hold purpose both in theoretical approaches (as a proof of concept) and foreseeable practical applications—since it means we can alter a given P System to perform both its original task and the task described by our discovery model. This will continue Dinneen, Kim and Nicolescu's work towards practical P Systems [16, 17].

In this paper we begin with preliminaries and then follow up with the motivations for our work. We then present a discussion of our decision to work with state-based systems and then offer a brief survey of the more popular systems to date. Our survey includes a description for each system in a separate section along with relevant comments. We then present our new model and a mapping from it to each of the P Systems mentioned in this paper. Our general model is very basic, while maintaining Turing completeness, lacking cell identifications (IDs) and only allowing for undirected communications. Solving a problem within the general model will allow the problem to be solved on any of the P System models described in this thesis which is advantageous for various reasons as explored in Section 3. We conclude with the set of requirements for a discovery algorithm to run within a P System which will be helpful when extending our work.

2 Preliminaries

We discuss some common elements across various P Systems here. This section is informal and is intended to familiarise the reader with various concepts before continuing. For a formal definition of any of the terms used in this section we refer the interested reader to Păun's introductory textbook [19]. However, complete formal definitions for each individual P System that we cover are included in their relevant section.

All P System models we present are defined as tuples containing four key pieces of information: the system's alphabet, the set of membranes, the collection(s) of evolution rules and the structural information. The system's alphabet is a set of objects that the P System uses to communicate or compute. The set of membranes may contain additional information such as initial objects or state sets including each cell's current state. The collection of evolution rules is sometimes ordered to indicate some form of priority. For some types of P System models this collection is distributed such that each cell has its own. Finally, the structural information is given as a set of relations that show which cells are adjacent to one another.

The rules generally take the form of a 'trigger'; the specification of what objects are required for a rule to be performed, and the outcome. The rules may or may not involve a corresponding state change. Cells can take on particular states, which will govern or limit which rules may be applied at any given time during the computation. A computation on a P System is the sequence of configurations, all the elements of the system as they are at any given point in time, from the starting configuration to the halting configuration. The complete definition of a P System (a tuple that contains all the information mentioned above) provides a starting configuration. We can move to the next configuration of the system by applying all rules which can be applied within the system, concurrently. This move to the next configuration is considered a *P Step*. A P System's computation is complete when there is no next configuration (or the next and current configurations are the same).

We note that membranes are the borders of cells, however when speaking about either we refer to the structure itself so the terms are interchangeable. At times it is easier to refer to communication across membranes and objects being received by cells. We refer to the neighbourhood of a cell in a P System as the set of cells which are adjacent to it in the structure, either as a parent or a child. When applying a rule within a particular cell that rule must be in one of three forms: *min*, *max* or *par*. These terms hold different meanings across various systems and will be explained when describing particular models. In general a min rule is a rule that is only applied once regardless of how many times it could be applied, a max rule is applied as many times as possible concurrently and a par rule is some limited application that is not as broad as max. For example, suppose we have a cell that contains within it five a objects, and three b objects. Also suppose the cell has the rule R: take the pair ab and produce a c; if R were a min rule we would expect the cell, after a single *P Step*, to contain four a objects, two b objects and a single c object; however if R was a max rule, after one *P Step* the cell would contain two a objects, no b objects and three c objects.

Some of the systems described have the ability to control which cells they send information to, i.e. specifically limiting information to parent(s) or children. We will use the following notation when specifying 'send-to' upon directed graph structures: Let M denote a multiset of objects:

- $M_{\uparrow}$ denotes a send to parent or encasing membrane, also referred to as $M_{out!}$ (a send to all external cells) in deterministic variations Păun's original system [19].
- $M_{\downarrow}$ denotes a send to children or (immediately) contained membranes (or the root of the child sub-tree), also referred to as $M_{in!}$ in deterministic variations of Păun's original system (a send to all internal cells) in [19].
- $M_{\updownarrow}$ denotes a send to both parents and children (or the neighbourhood of the cell performing the send).
- M (or lack of direction) denotes objects that are created and kept by the cell performing the task.

Our goal is to obtain a high level language that we can use to define generic rule sets. These sets can then be used to determine the underlying topology of commonly

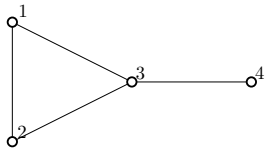

Figure 1: A graph $G = (V, E)$; $V = \{1, 2, 3, 4\}$; $E = \{ \{1,2\}, \{1,3\}, \{2,3\}, \{3,4\} \}$.

used P Systems. To create our rule set we will assume a general set of capabilities for all systems which will be altered (or limited) as we explore particular P Systems throughout this paper. We will consider topologies as undirected graphs, an example of which is shown in Figure 1, further understanding of graphs can be gained from an introductory graph theory textbook [6].

3 Motivations

We examine four P Systems in this paper to determine similarities and define discovery algorithms that can be implemented across all the systems we consider, including some we do not (such as Dual P Systems [1] and Population P Systems [3]). Discovery algorithms could be considered the equivalent of a high level programming language for P Systems (we describe this notion clearly in Section 9.3). As with many high level languages some efficiency may be lost when using them. Since discovery algorithms are high level, they will allow for a generalised way of communicating ideas, and it will also mean that algorithms can be presented in a standardised form that is more like pseudo-code than a formal rule set. We believe working with a restricted set of rules to appease all systems will help to show the power of each system in its own right. It may very well lead to more formal classes of P Systems. By formalising the discovery approach we can also learn some of the limitations of such an approach to solving problems.

In the past, P Systems have been considered only as theoretical concepts that expect the author of any paper to define (or create) the specific system in which their rule set, program in a sense, is to be run. This definition includes the topology so it may be difficult to see the motivation for our work. To describe some of the motivations we will consider the systems described thus far in two different groups: the first is the set of systems which allow for real-time generation or dissolution of cells; the second are those that do not. For the first group the motivation is clear. We may want to know the underlying structure of the system we have created. Perhaps further research of the growth patterns of P Systems will help us to better understand their problem solving power. For the second group the motivation is not immediately clear, until we consider that P Systems (while beautiful theoretical objects) have great practical potential. Nicolescu *et al.* have already shown how their Hyperdag P System can be used as a tool to model network hierarchies [16, 17]. Discovering the topology of a network in real-time when performing a task is a better approach than assuming a network structure. This could lead to more efficient implementations (for example, a broadcast would complete much faster if it was performed from the centre of the network); knowing the topology and allowing your

algorithm to work with delays or even faulty hardware will lead to more robust solutions.

4 State Based Systems

We work with P Systems that use cell-states: cells operate under a restricted rule set (effectively) since triggers involve the cell being in a certain state, similar to if-statements. We assume this is advantageous for two main reasons: firstly, because a state change procedure could be viewed as a Turing machine configuration sequence (when defining algorithms); secondly, because using systems with states ensures we do not accept an obvious loss of efficiency when removing excess objects when we simulate states. A state change procedure, (such as: when in state 1, produce some output and move to state 2, then produce some output and move to state 3, etc.) can be likened to a standard Turing machine algorithm that does action A, then moves to action B and then to action C, etc.; thirdly, because state based systems imply an order of precedence upon the rule set.

Open Problem 1 *Can states be simulated on* all *non-state-based P System models?*

On first glance it may appear as though states can be simulated with objects. Simulating states seems like a good idea, as it would allow discovery algorithms to work on more P Systems and would also allow a cell to be in multiple states at once—allowing entire algorithms to run concurrently. But the simulation of states is not attempted in this thesis. We do not go so far as to deny the possibility of an object based state-simulated system, but we do note the difficulties in creating one. Since we are simulating states with objects, there are two approaches we can take: the first is to designate a single object as a state-representing object and the second to designate a set of unique objects to denote a set of different states. In the former case we encounter the problem of conflict, while in the latter case we encounter a problem concerning efficiency or correctness.

If we were to designate a single object to denote states, then we would have to have multiple copies of this state-object to denote different states, specifically: one state-object implies state 1, two state-objects denotes state 2, and k state-objects denotes state k. This leads to a conflict: suppose we have a cell in state 6 containing objects a and b which has at least two evolution rules (where s is our state-object): $s^5a \to s^6b$ and $s^6b \to s^7b^2$; as we are in state 6 we expect that the b will be consumed and the cell will switch to state 7 to create a pair of b objects for itself. However, there is nothing to stop the state 5 rule from being applied as there are sufficient state-objects and an a object within the cell. This conflict cannot be resolved deterministically in Păun's original P System [19].

Alternatively we could simulate k states using k different objects instead of one object. By doing this we lose the concept of applying rules with priority and the ability to run max rules in a natural manner. Priority issues could lead to incorrectly functioning procedures. State based P System models have built within them a priority on rules to ensure that rules are applied correctly: if two rules can be applied and they result in different states, one of the rules takes precedence, and is applied instead of the other. When simulating states, there is no way to ensure precedence, so both rules are applied which leads to contradicting state settings. It is possible that with wise design procedures such cases could be avoided, so this merely introduces a difficulty.

The main concern with simulating multiple states with distinct objects is that of efficiency. In a state-based model, if we reach a certain state we can define a set of rules that would remove every object within the cell, as these rules can be set to max and applied concurrently it costs a single *P step*. However, in a simulated system we only have a single state-object, so removing m objects will take m steps. We note that duplicating state-objects before running the clean-up rules will allow max rules to be applied in their natural sense. Nevertheless, by having multiple state-objects within a cell, if the state changes we would need to ensure consistency. Ultimately, simulating states is difficult and appears to introduce many problems we could avoid by working with models that allow for states.

We note that state-like functionality can be implemented with the introduction of promoters and inhibitors [2], inhibitors stop particular rules from being applied whereas promoters encourage certain rules to be applied before others. However this functionality is by design and is not formally written as a mapping or translation between a state-based rule set to a promoter and inhibitors rule set without states. We put forward the following conjecture.

Conjecture 2 *There exists a rule set mapping from state-based models to non-state-based models which allow for* inhibitors *and* promoters*.*

It is important to note that proving Conjecture 2 will allow discovery algorithms to be performed on non-state based systems that allow for inhibitors and promoters as well. We assume state-based models when continuing with our work.

5 System Overviews

5.1 Păun's Original System

Păun's original system assumes one membrane that is recursive in nature. The outer-most membrane (also known as the skin) contains other membranes which may contain other membranes and so forth. Each membrane can only communicate with the cells/membranes neighbouring it, i.e. children or its parent membrane. The membranes communicate by sending objects. The original cells of the system have a unique identifier, but this identifier may not remain unique through the computational process as cells can be generated or dissolved in real-time. This model can is a rooted tree structure, where the root of the tree is the outer-most membrane, with each embedded membrane as a sub-tree (defined recursively on the structure). We note that a membrane without any children is known as an elementary membrane (or a leaf in the tree structure).

The root of the tree, or skin membrane interacts with the environment, which might also be considered as the output tape of the system. A mathematical definition follows that is consistent with [16]:

Definition 3 *A* ***P System*** *is a tuple* $\Pi = (\Sigma, \delta, w_1, \ldots, w_m, R_1, \ldots, R_m)$ *of degree (order)* m*, where:*

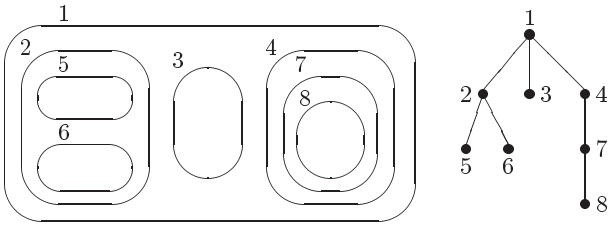

Figure 2: Păun's original system: A membrane structure and its associated rooted tree [20].

- Σ *is a finite non-empty alphabet of* objects*; a subset* $T \subseteq \Sigma$ *can be defined as an output alphabet.*
- δ *is a membrane structure consisting of* m *membranes, with the membranes labelled by the elements of the set* $\{1, 2, \ldots, m\}$*. This structure can be viewed as a rooted tree with a set of parent-child relations.*
- $w_i, 1 \leq i \leq m$ *are multisets over* Σ *associated with the membranes* $1, 2, \ldots, m$ *of* Π*.*
- $R_i, 1 \leq i \leq m$*, are finite sets of evolution rules over* Σ*, each* R_i *is associated with the region* i *of* Π*;* R_i *is ordered by a priority relation (on the rules of* R_i*). An evolution rule is of the form* $u \rightarrow v$*, where* u *is a string over* Σ *and* v *is a string representing a multiset over* $\{a, a_{\downarrow}, a_{\uparrow} \mid a \in \Sigma*\}$*.*

We note that a region is the term used to denote an outer membrane and all encased membranes. On the tree-model a region is the entire subtree. However, when working with rules which perform a send-to region, we send the objects to the root of the region only. We do not have direct access to the child-membranes.

As mentioned earlier, Păun's original P System has the ability to alter its structure in real-time. These operations are included as rules within the system and require some sort of 'trigger'. Păun's original system allows separate rules for each of the membranes; a system in which all the rules are the same is known as a *simple* system. As our goal is to produce the most general algorithms possible, we will assume the simple model when performing our 'mapping' (it is clear that models that allow for individual rule sets can be run with the initial configuration such that all the rules across cells are identical; the converse is not possible).

The P System's ability to reproduce cells was the fundamental part of Păun's linear *P step* solution to SAT [13]. The algorithm, basically, generates all 2^n possible variable assignments and has the original formula tested within each cell simultaneously. It takes only linear steps to generate all 2^n cells then the satisfiability-test is performed in a single step. The appropriate result is then, sent to the root cell (or the environment) by transmitting up the tree of height $O(n)$.

Păun's original system lacks the ability to bind cell IDs onto objects. This means that it is impossible to specify ownership of an object using the object itself. This was not necessary in Păun's model because of the parent-child relations, but does impose

a restriction on our generic algorithms when computing topologies. We also note that Păun's original system does not have states in the sense we use them, but the system does allow for *polarity* which is better described in [20]; polarity allows a cell to be positive, neutral or negative and depending on the polarity a certain subset of the full rule set is applied, resulting in output for the cell (or neighbouring cells) and a possible polarity change. We view this as a limited state based system, thus if our discovery algorithm requires fewer than four states then we can run it on Păun's original system, otherwise we cannot without a way to simulate states which was discussed earlier.

Another alternative for simulating states might be the concept of inhibitors or promoters [22]. This still does not account for the priorities built into a state based model, but provides a strong indication that state simulation may be possible.

In all other aspects, Păun's original system was the basis of what was described in Section 2. So, Păun's original system is capable of bi-directional communication, performs all actions in parallel, and can generate objects as required.

5.2 Hyperdag P System

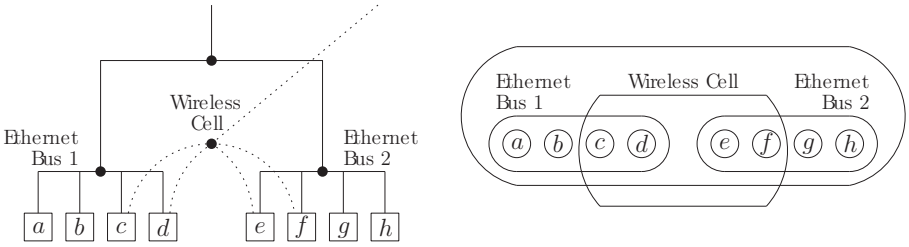

Figure 3: A network as a directed acyclic graph, notice that c, d, e and f have two parents [9].

The Hyperdag P System is hierarchical in nature and can be used to model a network as shown in Figure 3 (or any other structure in which leaves receive information from multiple parents).

Definition 4 *A **Hyperdag P System** is a tuple* $\Pi_h = (O, \sigma_1, \ldots, \sigma_m, \delta, I_{out})$*, where:*

- O *is a finite non-empty alphabet of objects;*
- $\sigma_i, 1 \leq i \leq m$ *are cells of the form* $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i)$*, where:*
  - Q_i *is a finite set (of states),*
  - $s_{i,0} \in Q_i$ *is the initial state,*
  - $w_{i,0} \in O^*$ *is the initial multiset of objects,*
  - P_i *is a finite set of multiset rewrite rules of the form:*
    $sx \rightarrow s'x'u_{\uparrow}v_{\downarrow}t_{\updownarrow}w_{\leftrightarrow}y_{go}z_{out}$,
    where $s, s' \in Q_i$, x, x', $u_{\uparrow}$, $v_{\downarrow}$, $t_{\updownarrow}$, $w_{\leftrightarrow}$, y_{go}, $z_{out} \in O^*$, *with the restriction that* $z_{out} = \lambda$ *for all* $i \in \{1, \ldots, m\} \setminus I_{out}$.

- δ *is a set of directed acyclic graph (DAG) parent-child arcs on* $\{1, \ldots, m\}$, *i.e,* $\delta \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$, *where* δ *is a DAG, representing bidirectional (symmetric) communication channels between cells;*

- $I_{out} \subseteq \{1, \ldots, m\}$ *indicates the output cells, the only cells allowed to send objects to the 'environment'.*

The rule definition is quite complex as it allows you to send directly to the environment (denoted by z_{out}). As this is specific to the Hyperdag P System and Tissue P System we do not use either of these features.

The hierarchical structure allows the relatively easy determination of the number of paths between any two cells in the system. This particular use has clear applications to network design problems. The algorithm starts from some root vertex (or from all root vertices depending on the starting configuration), then sends some object to all descendants. When a descendant receives objects, it forwards a copy of each of those objects to its descendants. The procedure stops when there are no new descendants. At the end of the computation each of the cells in the system knows how many paths there are from the starting cell to itself [18].

Hyperdag P Systems allow for the generation or dissolution of cells during a computation, provided that the newly created cell does not conflict with the DAG structure of the system. The Hyperdag P System assumes duplex communication (the ability to send data in both directions across the single parent-child relation), so our discovery algorithms will perform correctly upon the system since the cell can immediately inform its neighbours that it received information. The Hyperdag system also maintains the ability to generate objects as required, is state-capable and all rules are applied in parallel.

5.3 Neural-like Tissue P System

The Tissue P System is a model in which the underlying structure of the system can be considered as a directed graph. This is how the neural pathways are modelled in the brain [12] and offer a more complex structure than Păun's original system. Formally, we define this as follows [15]:

Definition 5 *A* ***Tissue P System*** *is a tuple* $\Pi_t = (O, \sigma_1, \ldots, \sigma_m, syn, i_{out})$, *where:*

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

- $\sigma_i, 1 \leq i \leq m$ *are cells, of the form* $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i), 1 \leq i \leq m$, *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 of the cell,*
  - R_i *is a finite set of rules of the form:* $sw \rightarrow s'xy_{go}z_{out}$, *where* $s, s' \in Q_i, w, x \in O^*$, $y_{go} \in (O \times \{go\})^*$, *and* $z_{out} \in (O \times \{out\})^*$, *with the restriction that* $z_{out} = \lambda$ *for all* $i \in \{1, \ldots, m\}$ *different from* I_{out}.

- $syn \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$ *(synapses among cells).*
- $i_{out} \in \{1, \ldots, m\}$ *indicates the output cells.*

The rule definition allows us to send directly to the environment (denoted by z_{out}); as this is specific to the Hyperdag P System and Tissue P System we do not use either of these features. The min, max and par specifications are applied in a global sense for the Tissue P System. Min specifies a single rule is to be applied once, par designates a single rule which is applied as many times as possible and max denotes that all the rules are applied concurrently as many times as possible.

A variation of Tissue P Systems solved 3-COL [11], a well known NP-Complete problem, in a linear number of *P Steps* [5]. The process for solving the problem is to generate a system that has enough cells to simulate all possible 3-colourings of the graph and then ensure validity by 'stabilising' via the environment. Ultimately the cost to generate the cells is linear and the cost to stabilise is constant as we only need to check three colours in this case.

The tissue system does not limit our model since all rules are applied in parallel, the system is state-capable and objects are created as required by the rules. However, we are hoping to determine underlying structure, so the tissue system's synapse relations must be symmetric i.e. if there is an arc from cell c_i to c_j then there must also be an arc from cell c_j to c_i. We encourage further work in running algorithms in the discovery sense on strongly connected directed underlying structures.

5.4 Simple P Module

The Simple P Module is arguably the most unique system we examine. The term 'simple' in its name appears inconsistent with the standard P System naming conventions, but makes sense when it is understood that P modules are designed to 'inter-lock' with one another to create greater systems. Each module performs a certain task and to perform some combination of tasks it is possible to join the various systems together to gain the full procedure. The Simple P Module does not have the inter-locking feature, making it simple. The name Simple P Module was coined by Dinneen, Kim and Nicolescu in [7], thus we will maintain it as such here.

Definition 6 *A* Simple P Module *is a system* $\Pi_m = (O, K, \delta)$*, where:*

- O *is a finite non-empty alphabet of objects;*
- $K = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ *is a finite set of (internal)* cells*;*
- δ *is a binary relation on* K*, without reflexive or symmetric arcs, which represents a set of parent-child structural arcs between existing cells, with* duplex *communication capability. Each cell,* $\sigma_i \in K$*, has the initial form* $\sigma_i = (Q_i, s_0, w_0, R_i)$ *and general form* $\sigma_i = (Q_i, s, w, R_i)$*, where:*
  - Q_i *is a finite set of* states*;*
  - $s_0 \in Q_i$ *is the* initial state*;* $s \in Q_i$ *is the* current state*;*

- $w_0 \in O^*$ *is the* initial multiset *of objects;* $w \in O^*$ *is the* current multiset *of objects;*
- R_i *is a finite* ordered *collection of multiset rewriting* rules *of the general form:* $s\ x \rightarrow_\alpha s'\ x'\ (u)_{\beta_\gamma}$*, where* $s, s' \in Q$*,* $x, x' \in O^*$*,* $u \in O^*$*,* $\alpha \in \{min, max\}$*,* $\beta \in \{\uparrow, \downarrow, \updownarrow, self\}$*,* $\gamma \in \{\texttt{one}, \texttt{spread}, \texttt{repl}\} \cup K$*. If* $u = \lambda$*, denoting the* empty string *of objects, this rule can be abbreviated as* $s\ x \rightarrow_\alpha s'\ x'$*. The application of a rule takes two sub-steps, after which the cell's current state* s *and multi-set of objects* x *is replaced by* s' *and* x'*, respectively, while* u *is a message which is sent as specified by the transfer operator* β_γ*.*

The rules given by the collection R_i are applied in the *weak priority* order [21]. For a cell $\sigma_i = (Q_i, t, w, R_i)$, a rule $s\ x \rightarrow_\alpha s'\ x'\ (u)_{\beta_\gamma} \in R_i$ is *applicable* if $t = s$ and $x \subseteq w$. Additionally, if $s\ x \rightarrow_\alpha s'\ x'\ (u)_{\beta_\gamma}$ is the first applicable rule, then each subsequent applicable rule's target state (i.e. state indicated in the right-hand side) must be s'.

We note that the Simple P Module has the ability to specify how to generate objects (specifically γ), γ is used to denote either: `one`, create only a single object for a particular neighbour $k \in K$; or `spread` in which we generate a set of objects and then spread them across specific recipients; or `repl` in which we replicate a set of objects for all recipients. We will only use the replicate feature.

The Simple P Module appears to be the most flexible. The (complete) P Module has various applications to software engineering problems, most notably the Byzantine agreement problem [8].

The Simple P Module cannot generate or dissolve cells during a computation. However, it is state-capable, has bi-directional communication, applies all rules in parallel and creates objects as required.

6 Discovery Rules

In this section we formalise our discovery algorithm rule sets and provide an initial mapping for our model to each of the systems covered thus far. For each of our rules we write the equivalent rule for each of the systems explored in this paper and offer an explanation of the rule being applied on its native system to show the reader that the outcome of the command across all systems is the same. Once again we have ensured this is possible by selecting only a subset of the full rule set for any given system.

6.1 Starting Configuration

To generalise the algorithms further we will assume that all computations start from a single cell which then spread across the system (hence discover the farther cells as opposed to assuming their existence). We refer to it as a 'single source start' since we start at a single source cell to begin the computation. There are various advantages to this approach, one of which is the ability to invoke the algorithms within another P System computation. All of the discovery algorithms will include the presentation of the different objects required to perform it. So if the starting objects for a particular

discovery algorithm and all the objects required by the computation of that algorithm are reserved, then we could have it run independently from any other algorithm (after some trigger) to gain topology information before continuing some other computation (we describe this notion clearly in Section 9.3). It will be assumed that the starting cell is the same as the skin, I_{out}, or an output cell from I_{out} if I_{out} is a set containing multiple elements for our systems. This assumption ensures the results of the computation can be obtained from the starting cell after some number of *P steps* (though it is better to ensure that all cells have the output information so that you can start with any cell in the system).

What follows is a starting configuration for any discovery algorithm and a mapping to each of the four systems stated thus far. The rule mapping is covered in Section 6.2.

The starting configuration for any discovery algorithm will be defined as:

Definition 7 *A P System single starting source is a tuple* $\Pi_s = (O_s,\ S_s,\ M_s,\ s,\ R_s)$ *where* O_s *is the alphabet required for the algorithm to run,* S_s *is the set of states required for the algorithm to run,* M_s *is the multiset of objects held by the starting cell,* s *is the initial state of the starting cell and* $R_s = \{r_0,\ r_1,\ \ldots,\ r_m\}$ *where* $r_i = sa \rightarrow s'a'a''_{\updownarrow}$ *where* $s, s' \in S_s,\ a, a', a'' \in O_s^*$*, is the rule set for all the cells in the system and* $m \in \mathbb{N}$*.*

We note that λ is the empty string, used to denote the empty set in the P System descriptions (following convention) and that s_0 is the initial state—which is assumed in all models.

Păun's original system $\Pi = (\Sigma = \Sigma_s,\ \delta,\ w_1 = M_s,\ w_2 = \lambda,\ \ldots,\ w_m = \lambda,\ R_i = R_s$ for all $i)$.

Hyperdag P System $\Pi_h = (O = \Sigma_s,\ \sigma_1, \ldots, \sigma_m,\ \delta,\ I_{out} = \{\sigma_1\})$ where $\sigma_1 = (S_s,\ s,\ M_s,\ R_s)$, and $\sigma_i = (S,\ s_0,\ \lambda,\ R_s)$, $2 \leq i \leq m$.

Tissue P System $\Pi_t = (O = \Sigma_s,\ \sigma_1,\ \ldots,\ \sigma_m,\ syn,\ i_{out} = \{\sigma_1\})$ where: $\sigma_1 = (S_s,\ s,\ M_s,\ R_s)$ and $\sigma_i = (S_s,\ s_0,\ \lambda,\ R_s)$, $2 \leq i \leq m$.

Simple P Module $\Pi_m = (O = \Sigma_s,\ K = \{\sigma_1 = (S_s,\ s,\ M_s,\ R_s),\ \sigma_i = (S_s, s_0,\ \lambda, R_s),\ 2 \leq i \leq m\},\ \delta)$.

6.2 Rules

All the rules in the discovery algorithms we work with are deterministic and a rule must be in one of two forms: min or max. We always replicate the object for each of the neighbouring cells such that all neighbours receive the same multiset. The multiset of objects on the left-hand side of the rule is consumed during the application of the rule. In order to maintain the original set we would need to specify it as an outcome. The left-hand side of the rule indicates a 'trigger' and the right-hand side of the rule is the set of outcomes.

6.2.1 Rule Mapping

We will map the aforementioned definition ($sM \rightarrow s'M'M''_{\updownarrow}$) and comment on specifics related to each system.

Păun's original system $sM \rightarrow s'M'M''_{\updownarrow}$ where $s, s' \in S_s$ and $M, M', M'' \subset \Sigma_s^*$.

In Păun's original system we are limited to three states, so s and s' are really from the set $S_s = \{positive,\ negative,\ neutral\}$. We also recognise that variants of Păun's original system allow for a rule that produces objects, both for the cell applying the rule and all neighbours of the cell simultaneously (off the same trigger). However, if we wanted to be consistent with the original system described in this paper we would require two *P Steps*. We would have to define two rules, the first creates $M' \cup K$ for the cell itself, the second creates M'' for all neighbours using K as a trigger, where $K \subset \Sigma_s$ (the K multiset should be unique to ensure a deterministic solution). We note that Păun's original system does not allow for min rules in deterministic variations.

Hyperdag P System $s\ M \rightarrow s'M'M''_{\updownarrow}$ where $s, s' \in S_s$ and $M, M', M'' \subset \Sigma_s^*$.

The Hyperdag P System allows for our simplified rule to be mapped without any difficulties. We recognise that some variants of the Hyperdag P System allow for min or max rules, if that is the case then we will assume max mode for all rules.

Tissue P System $s\ M \rightarrow s'M'M''_{\updownarrow}$ where $s, s' \in S_s$ and $M, M', M'' \subset \Sigma_s^*$.

Similar to the Hyperdag P System, the Tissue P System allows for our simplified rule to be mapped without any difficulties either.

Simple P Module $s\ M \rightarrow_{\alpha} s'M'M''_{\updownarrow}$ where $s, s' \in S_s$, $M, M', M'' \subset \Sigma_s^*$ and $\alpha = \texttt{max}$.

Here, α is always set to max in our generic rule set because at least one of the other systems does not support the min function.

We note that the alphabet we define for the single source start system may be mapped onto the original alphabet for the system it is being performed upon, we only require that the alphabet elements are unique.

7 An Example

We offer a simple 'flooding' example from the discovery paper [18] in our format.

Pre-conditions: Every cell in the network is in the initial state s_0 except for one cell that is also in s_0 but has an initial multiset made up of a single a.

Post-conditions: Every cell in the system is reached and is put into state s_1.

The following P System single source start implements this: $\Pi_s = (\{a\}, \{s_0, s_1\}, \{a\}, \{s_0\}, R_s = \{s_0 a \rightarrow s_1 a_{\updownarrow}, s_1 a \rightarrow s_1\})$

It is clear that the above starting configuration and rule set will eventuate in a stable, or halted, system in which every reachable cell is in state s_1. A small example is demonstrated in Figure 4.

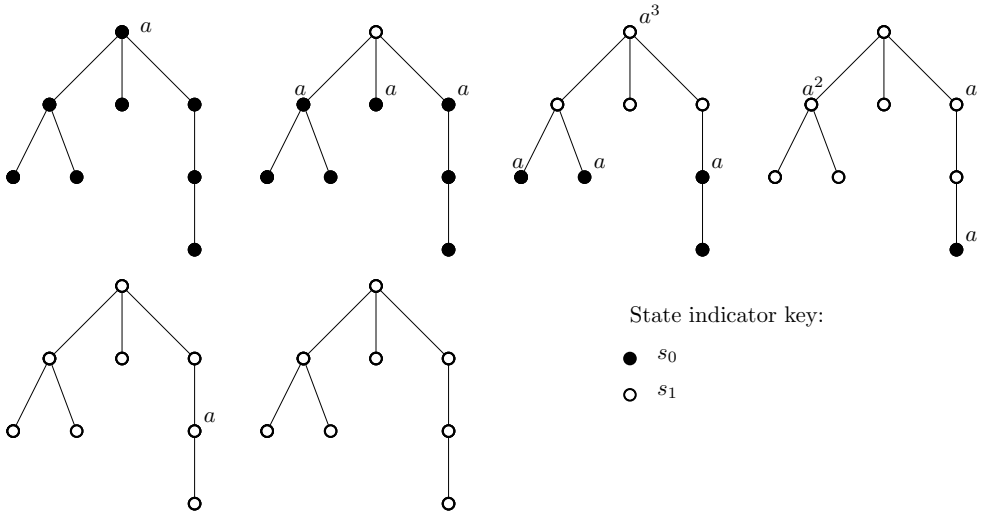

Figure 4: Example computation: the sequence of configurations of the P System.

8 System Requirements

To discover topological information on a P System Π that we have not covered explicitly we require that Π possesses certain properties.

Bi-directional Communication The system should allow for bi-directional communication. This is to ensure we are generating the underlying structure of the system.

Parallel Rule Application This ensures our running times in *P Steps* will be accurate. It also allows us to gain the benefits of working with a P System.

States The system must be state based for the reasons we explored in Section 4.

(Free) Object Generation Free object generation may not be realistic in some models (the Tissue System for example); however, the free generation of objects is required for our algorithms to run. By free generation of objects we mean that we can consume a single copy of an a object and generate k copies of a objects, where $k \in \mathbb{N}$. A more realistic model may ensure a conservation of some sort, and may stop our algorithms from running.

Determinism The ability to ensure a deterministic outcome.

Notice that we do not require cell identification, the ability to generate or dissolve cells in the computation.

We question whether or not discovery algorithms can be extended to directed systems. Many of the algorithms defined in this thesis assume echoed feedback to ensure correctness.

Open Problem 8 *Can discovery algorithms be extended to strongly connected systems?*

We accept that it is certainly possible if the goal is merely to distribute information, but when considering echoed feedback we conjecture that cell IDs will be required.

Conjecture 9 *Strongly connected systems will require cell identification to perform discovery algorithms.*

The reasoning behind Conjecture 9 is simple. Any given cell will need to be able to determine that one of its own neighbours has returned information. Since we do not have the ability to assume feedback within a fixed number of P steps, we will need some form of identifier to be able to tell if one of the child cells is communicating with it (or has sent feedback via an alternate path).

9 Implications of Our Model

We have built the framework that we will use to present solutions to topology problems considered in future work. As we have created a new model we ensure it is Turing complete and describe how the model can be mapped to any of the four models considered in this chapter. This section also explains why we can refer to our discovery model as a discovery algorithm with slight modification of terminology.

9.1 Turing Completeness

As we have defined a new system we feel it is important to ensure computational power.

Theorem 10 *Păun's original system is Turing complete without the ability to generate nor dissolve cells.*

Proof. It was shown that a *Counter Automaton* can simulate a Turing machine provided it has at least two counters [14]. It has also been shown that a single cell version of Păun's original system could simulate a Turing complete Counter Automata [10]. Hence proving that Păun's original system is Turing complete without the ability to generate nor dissolve cells. □

Lemma 11 *All non-state-based rule sets* $S_n = \{r_1, r_2, \ldots, r_k\}$*, where* $r_i = M_i \to M_i' M_{i,\updownarrow}''$ *for* $1 \leq i \leq k$*, can be mapped to corresponding state-based rule sets* $S_s = \{r_1', r_2', \ldots, r_k'\}$ *where* $r_i' = s_i N_i \to s_i' N_i' N_{i,\updownarrow}''$.

Proof. Let s be a state. There exists the following mapping:

For $1 \leq i \leq k$, r_i maps to r_i' such that $r_i = M \to M_i' M_{i,\updownarrow}''$ and $r_i' = sM_i \to sM_i' M_{i,\updownarrow}''$.

The order is maintained and as there is only a single state s in the S_s set no state-change occurs perfectly mimicking the S_n set. □

Theorem 12 *For every single cell* Π *(Păun's original system) which does not involve the generation nor dissolution of cells, there exists a* Π_s *system that will produce the same outcome.*

Proof. Let the system $\Pi = (\Sigma, \delta, w_1, R)$, given in Subsection 5.1 be a single cell version of Păun's original system which does not involve the generation nor dissolution of cells. There exists a mapping of the Π system to the $\Pi_s = (O_s, S_s, M_s, s, R_s)$ system. Where $O_s = \Sigma$, $S_s = \{s_0\}$ such that s_0 is the state introduced as in Lemma 11, $M_s = w_1$, $s = s_0$, $R_s = R$ by Lemma 11. Therefore Π_s performs all of Π's functions. □

Theorem 13 *The Π_s system is Turing complete.*

Proof. Theorem 10 tells us that single cell versions of Păun's original system are Turing complete. Theorem 12 allows us to map the set of all single cell versions of Păun's original system to our Π_s system. Therefore, by transitivity, the Π_s system is also Turing complete. □

9.2 Generalisation

By showing a clear mapping from our system to other systems in Section 6 we have ensured that any solution that runs on the generic Π_s model we have created will run upon any of the aforementioned models described in this paper. This mapping is only in one direction as the four models described have additional functionality we have omitted to ensure portability.

However, findings that hold for our system will hold for any system that incorporates all the following attributes: the system requirements (see Section 8); anonymity (lacks cell IDs); and a starting configuration in which only a single cell has a multiset of starting objects. This means our work has direct implications upon Păun's original system, the Hyperdag P System, the Tissue P System and the Simple P Module, when a single cell starting configuration is assumed and the rule set does not trigger cell generation. We note that results can vary depending upon whether the P System has a cyclic or acyclic underlying structure.

9.3 Model to Algorithm

We have created a discovery algorithm model, but in the general sense, an algorithm refers to a program to be run on an *Universal Turing Machine*. There is no concept of a machine that takes a P System and 'runs' it as such. For P Systems, the model itself is the computational device—similar to a particular Turing machine to complete some task. However, we can incorporate the functionality of a discovery system within any of the four models discussed in this paper with an *augmentation procedure*, by augmentation procedure we mean we can take a predefined P System and alter its definition such that it can obtain the output of the discovery model as a sub-solution. We define this procedure generically since the description is clear.

Let Π be one of the four systems discussed in this paper.

Let Π_s be our discovery system.

1. Add to the alphabet of the Π such that the Π_s alphabet is mappable in a disjoint manner. Also add an additional k objects to Π's alphabet, where k is the number of states used by Π.

2. Select one of the cell's starting multiset of objects (ideally one of the output cells) and alter it such that it also starts with the initial multiset for the discovery system Π_s. Recall that the discovery system starts all computations from a single source point.

3. Increase the states used by the Π system by the number of states used by the Π_s system.

4. Amend the Pi_s ruleset to all the rulesets' within Π. It is important to note here that because the starting state(s) is the original state(s) of the Π system the rules cannot be triggered until we make the following changes:

   (a) The states of the system have been segregated into two groups, specifically the original states and the discovery states. We need to alter one of the rules in the original system Π to trigger the discovery algorithm. This can be done by sending the single cell decided in item 2. with the initial multiset into the initial state of the discovery model.

   (b) Once this rule as been added (or altered) we need to ensure that the rest of the system performs the discovery procedure. To do this we need to ensure the initial state rules of the Π_s system take precedent and also store a reference to the previous state the cell was in (this will allow us to return the cell to the original state before the discovery algorithm was invoked). So the ruleset for any given cell will increase by km rules where k is the number of states used by the cell, and m is the number of rules that assume the initial state in Π_s. Where each rule will produce the desired outcome with respect to the Π_s system and a state indicator.

   (c) Once the discovery algorithm completes we need to ensure that the system returns to the state-configuration it was in before the discovery procedure was invoked. To do this we need to know when the discovery algorithm would halt and upon reaching its halting configuration consume the state indicator object and return each cell to the original state.

The description above has omitted a lot of detail, but the general idea is conveyed to show that defining a procedure P within the discovery model will allow P to work within another system. Or allow us to 'run' the 'algorithm' P within another system. While this is not the same as running a program upon an universal Turing machine, the slight modification of terminology does not introduce ambiguity for this field as there is not concept of a machine that 'runs' P Systems.

10 Summary

In this paper we have provided a framework for discovery algorithms for determining topologies in P Systems. To produce the framework we survey some of the most popular P Systems to determine common elements and comment on each of them individually.

We then go on to describe how our framework can be applied to all of the P Systems mentioned in this paper as well as commenting on some of the implied limitations of our model and how one might avoid them. To complete our framework we ensure that our re-stated rule set does not hinder Turing completeness for Păun's original system and then go over an example rule set that merely floods a P System with information. We also list system requirements and state some open problems relating to our work. We then comment on the implications of any findings using our model. This model formalises discovery algorithms so that we may both define generalisable solutions and learn of any limitations of the approach to solving problems. For further information about our generic model and concrete examples for determining structural information of a P system, we refer the reader to the first author's masters thesis [4].

References

[1] O. Agrigoroaiei and G. Ciobanu. Dual P systems. In D. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 95–107. Springer Berlin / Heidelberg, 2009.

[2] A. Alhazov, M. Margenstern, and S. Verlan. Fast synchronization in P systems. In D. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 118–128. Springer Berlin / Heidelberg, 2009.

[3] F. Bernardini and M. Gheorghe. Population P systems. *Journal of Universal Computer Science*, 10(5):509–539, 2004.

[4] S. Datt. *Membrane Topological Discovery Algorithms for P Systems.* Msc thesis, University of Auckland, New Zealand, 2011.

[5] D. Díaz-Pernil, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. R.-N. nez. A uniform family of tissue P systems with cell division solving 3-COL in a linear time. *Theoretical Computer Science*, 404:76–87, 2008.

[6] M. J. Dinneen, G. Gimel'farb, and M. C. Wilson. *Introduction to Algorithms, Data Structures and Formal Languages.* Pearson, New Zealand, 2004.

[7] M. J. Dinneen, Y. Kim, and R. Nicolescu. Edge and node-disjoint paths in P systems. *Electronic Proceedings in Theoretical Computer Science*, 4x:116–136, 2010. http://eptcs.org/.

[8] M. J. Dinneen, Y. Kim, and R. Nicolescu. P systems and the Byzantine agreement. *Journal of Logic and Algebraic Programming*, 79(6):334 – 349, 2010.

[9] M. J. Dinneen, Y. Kim, and R. Nicolescu. Toward practical P systems for distributed computing. *Seria Matematica-Informatica (Honor of Solomon Marcus on the Occasion of his 85th Anniversary)*, pages 23–34, 2010.

[10] P. Frisco and H. J. Hoogeboom. Simulating counter automata by P systems with symport/antiport. In *Revised Papers from the International Workshop on Membrane Computing*, pages 288–301, London, UK, 2003. Springer-Verlag.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W H Freeman & Co, 1979.

[12] M. S. Gazzaniga, R. B. Ivry, and G. R. Mangun. *Cognitive Neuroscience: The Biology of the Mind*. W. W. Norton & Company, 2002.

[13] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and F. J. Romero-Campero. A uniform solution to SAT using membrane creation. *Theoretical Computer Science*, 371:54–61, February 2007.

[14] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[15] C. Martín-Vide, G. Păun, J. Pazos, and A. Rodríguez-Patón. Tissue P systems. *Theoretical Computer Science*, 296:295–326, March 2003.

[16] R. Nicolescu, M. J. Dinneen, and Y. Kim. Structured modelling with hyperdag P systems: Part A. Report CDMTCS-342, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, New Zealand, 2008.

[17] R. Nicolescu, M. J. Dinneen, and Y. Kim. Structured modelling with hyperdag P systems: Part B. Report CDMTCS-373, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, New Zealand, 2009.

[18] R. Nicolescu, M. J. Dinneen, and Y. Kim. Discovering the membrane topology of hyperdag P systems. In G. Păun, M. J. Pérez-Jiménez, A. R.-N. nez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 5957 of *Lecture Notes in Computer Science*, pages 410–435. Springer Berlin / Heidelberg, 2010.

[19] G. Păun. *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.

[20] G. Păun. P systems with active membranes: attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics*, 6:75–90, January 2001.

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

[22] G. Păun and R. A. Păun. Membrane computing as a framework for modeling economic processes. In *Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 11–, Washington, DC, USA, 2005. IEEE Computer Society.