BFS Solution for Disjoint Paths in P Systems

Radu Nicolescu and Huiling Wu

Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
r.nicolescu@auckland.ac.nz, hwu065@aucklanduni.ac.nz

Summary. This paper continues the research on determining a maximum cardinality set of edge- and node-disjoint paths between a source cell and a target cell in P systems. We review the previous solution [3], based on depth-first search (DFS), and we propose a faster solution, based on breadth-first search (BFS), which leverages the parallel and distributed characteristics of P systems. The runtime complexity shows that, our BFS-based solution performs better than the DFS-based solution, in terms of P steps.

1 Introduction

P systems is a bio-inspired computational model, based on the way in which chemicals interact and cross cell membranes, introduced by Păun [16]. The essential specification of a P system includes a membrane structure, objects and rules. All cells evolve synchronously by applying rules in a non-deterministic and (potentially maximally) parallel manner. Thus, P systems is a strong candidate as a model for distributed and parallel computing.

Given a digraph G and two nodes, s and t, the disjoint paths problem aims to find the maximum number of s-to-t edge- or node-disjoint paths. There are many important applications that need to find alternative paths between two nodes, in all domains. Alternative paths are fundamental in biological remodelling, e.g., of nervous or vascular systems. Multipath routing can use all available bandwidth in computer networks. Disjoint paths are sought in streaming multi-core applications that are bandwidth sensitive to avoid sharing communication links between processors [17]. The maximum matching problem in a bipartite graph can also be transformed to the disjoint paths problem. In case of non-complete graphs, Byzantine Agreement requires at least $2k+1$ node-disjoint paths, between each pair of nodes to ensure that a distributed consensus can occur, with up to k failures [9].

It is interesting to design a native P system solution for the disjoint path problem. In this case, the input graph is the P system structure itself, not as data to a program. Also, the system is fully distributed, i.e. there is no central node and only local channels (between structural neighbours) are allowed. In 2010, Dinneen,

Kim and Nicolescu [3] proposed the first P solution, as a distributed version of the Ford-Fulkerson algorithm, based on depth-first search (DFS). This solution searches by visiting nodes sequentially, which is not always efficient. To exploit the parallel potential of P systems, we propose a faster P system solution—a distributed version of the Edmonds-Karp algorithm, which concurrently searches as many paths as possible in breadth-first search (BFS).

This paper is organized as follows. Section 2 defines a simplified P system, general enough to cover most basic families. Section 3 describes the disjoint paths problem and the strategies for finding disjoint paths in digraphs. Section 4 discusses the specifics of the disjoint paths problem in P systems. Section 5 reviews the previous DFS-based solution [3] and sets out our faster BFS-based solution. Section 6 presents the P system rules for the disjoint paths algorithm using BFS. Section 7 compares the performance of the BFS-based and DFS-based algorithms, in terms of P steps, and the relative performance of the BFS-based solution simulation on sequential vs. parallel (multi-core) hardware. Finally, Section 8 summarizes our work and highlights future work.

2 Preliminary

Essentially, a static P system is specified by the membrane structure, objects and rules. The membrane structure can be modeled as: a rooted tree (cell-like P systems [16]), a directed acyclic graph (hyperdag P systems [11], [12], [13]), or in a more general case, an arbitrary digraph (neural P systems [10], [14]). Usually, the objects are symbols from a given alphabet, but one can also consider strings or other more complex structures (such as tuples). P systems combine rewriting rules that change objects in the region and communication rules that move objects across membranes. Here, we define a simple P system, with *priorities*, *promoters* and *duplex* channels as a system, $\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_n, \delta)$, where:

1. O is a finite non-empty alphabet of *objects*;
2. $\sigma_1, \ldots, \sigma_n$ are cells, of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i)$, $1 \leq i \leq n$, where:
   - Q_i is a finite set of *states*;
   - $s_{i,0} \in Q_i$ is the *initial state*;
   - $w_{i,0} \in O^*$ is the *initial multiset* of objects;
   - R_i is a finite *ordered* set of rewriting/communication *rules* of the form: $s \ x \rightarrow_\alpha s' \ x' \ (y)_\beta|_z$, where: $s, s' \in Q_i$, $x, x', y, z \in O^*$, $\alpha \in \{min, max\}$, $\beta \in \{\uparrow, \downarrow, \updownarrow\}$.
3. δ is a set of *digraph* arcs on $\{1, 2, \ldots, n\}$, without symmetric arcs, representing *duplex* channels between cells.

The membrane structure is a digraph with duplex channels, so parents can send messages to children *and* children to parents, but the disjoint paths strictly follow the parent-child direction. Rules are prioritized and are applied in *weak priority* order [15].

The general form of a rule, which transforms state s to state s', is $s\ x \rightarrow_{\alpha} s'\ x'\ (y)_{\beta_{\gamma}}|_{z}$. This rule consumes multiset x, and then (after all applicable rules have consumed their left-hand objects) produces multiset x', in the same cell ("*here*"). Also, it produces multiset y and sends it, by *replication*, to all parents ("*up*"), to all children ("*down*"), or to all parents and children ("*up and down*"), according to the value of target indicator $\beta \in \{\uparrow, \downarrow, \updownarrow\}$ (effectively, here we use the *repl* communication mode, exclusively). $\alpha \in \{min, max\}$ describes the rewriting mode. In the *minimal* mode, an applicable rule is applied exactly once. In the *maximal* mode, an applicable rule is used as many times as possible and all rules with the same states s and s' can be applied in the maximally parallel manner. Finally, the optional z indicates a multiset of promoters, which are not consumed, but are required, when determining whether the rule can be applied.

3 Disjoint Paths

Given a *digraph*, $G = (V, E)$, a *source* node, $s \in V$, and a *target* node, $t \in V$, the edge- and node-disjoint paths problem looks for one of the largest sets of edge- and node-disjoint s-to-t paths. A set of paths is *edge-disjoint* or *node-disjoint* if they have no common arc or no common intermediate node. Note that node-disjoint paths are also edge-disjoint paths, but the converse is not true. Cormen et al. [1] give a more detailed presentation of the topics discussed in this section.

Figure 1 (a) shows two node-disjoint paths from 0 to 6, i.e. 0.3.6 and 0.1.4.6, which are also edge-disjoint. In this scenario, this is the maximum number of node-disjoint paths one can find. However, one could add to this set another path, 0.2.3.5.6, shown in Figure 1 (b), to obtain a set of three edge-disjoint paths.

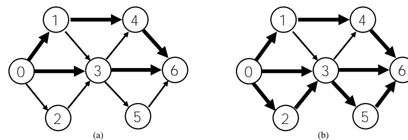

Fig. 1. Node- and edge- disjoint paths.

The maximum edge-disjoint paths problem can be transformed to a maximum flow problem by assigning unit capacity to each edge [5]. Given a set of already *established* edge- or node-disjoint paths P, we recall the definition of the *residual* digraph $G_r = (V_r, E_r)$:

- V_r = V and
- $E_r = (E \setminus E_P) \cup E'_P$, where E_P is the set of arcs (u, v) that appear in the P paths and $E'_P = \{(v, u) \mid (u, v) \in E_P\}$.

Briefly, the residual digraph is constructed by reversing the already established path arcs. An *augmenting* path is an s-to-t path in the residual digraph, G_r.

Augmenting paths are used to extend the existing set of established disjoint paths. If an augmenting arc reverses an existing path arc (also known as a *push-back* operation), then these two arcs "cancel" each other, due to zero total flow, and are discarded. The remaining path fragments are relinked to construct an extended set of disjoint paths. This round is repeated, starting with the new and larger set of established paths, until no more augmenting paths are found. A more detailed construction appears in Ford and Fulkerson maximal flow algorithm [5].

Example 1. Figure 2 illustrates a residual digraph and an augmenting path: (a) shows a digraph, where two edge-disjoint paths, 0.1.4.7 and 0.2.5.7, are present; (b) shows the residual digraph, formed by reversing path arcs; (c) shows an augmenting path, 0.3.5.2.6.7, which uses a reverse arc, $(5, 2)$; (d) discards the cancelling arcs, $(2, 5)$ and $(5, 2)$; (e) relinks the remaining path fragments, 0.1.4.7, 0.2, 5.7, 0.3.5 and 2.6.7, resulting in now three edge-disjoint paths, 0.1.4.7, 0.2.6.7 and 0.3.5.7.

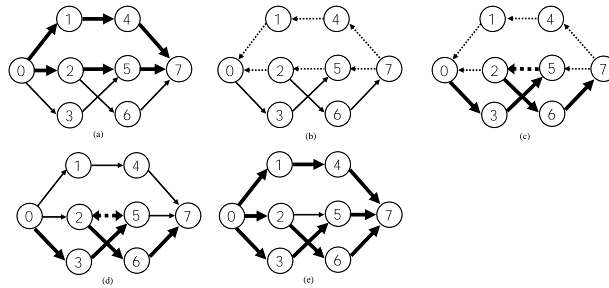

Fig. 2. Finding an augmenting path in the residual digraph.

The search for augmenting paths uses a search algorithm such as DFS (e.g., the Ford-Fulkerson algorithm) or BFS (e.g., the Edmonds-Karp algorithm). A *search path* in the residual graph (also known as a *tentative augmenting path*) starts from the source node and tries to reach the target node. A *successful search path* becomes a new *augmenting path* and is used (as previously explained) to increase the number of disjoint paths. Conceptually, this solves the edge-disjoint paths problem (at a high level). However, the node-disjoint paths require additional refinements—usually by *node splitting* [8]. Each *intermediate* node, v, is split into an *entry* node, v_1, and an *exit* node, v_2, linked by an arc (v_1, v_2). Arcs that in the original digraph, G, were directed into v are redirected into v_1 and arcs that were directed out of v are redirected out of v_2. Figure 3 illustrates this node-splitting procedure: (a) shows the original digraph and (b) the modified digraph, where all intermediate nodes are split—this is a bipartite digraph.

4 Disjoint Paths in P Systems

Classical algorithms use the digraph as data and keep global information. In contrast, our solutions are *fully distributed*. There is no central cell to convey global information among all cells, i.e. cells only communicate with their neighbors via local channels (between structural neighbours).

Unlike traditional programs, which keep full path information globally, our P systems solution records paths predecessors and successors locally in each cell, similar to distributed routing tables in computer networks. To construct such routing indicators, we assume that each cell σ_i is "blessed" with a unique *cell ID* object, ι_i, functioning as a *promoter*.

Although many versions of P systems accept cell division, we feel that this feature should not be used here and we intentionally discard it. Rather than actually splitting the intermediate P cells, we simulate this by ad-hoc cell rules. This approach could be in other distributed networks, where nodes cannot be split [3]. Essentially, node splitting prevents more than one unit flow to pass through an intermediate node [8].

In our case, node splitting can be simulated by: (i) constraining in and out flow capacities to one and (ii) having two *visited* markers for each cell, one for a *virtual entry* node and another for a *virtual exit* node, extending the visiting idea of classical search algorithms. Figure 3 illustrates a scenario when one cell, y, is visited *twice*, first on its entry and then on its exit node [3]. Assume that path $\pi = s.x.y.z.t$, is established. Consider a search path, τ, starting from cell, s, and reaching cell, y, in fact, y's entry node. This is allowed and y's entry node is marked as visited. However, to constrain its in-flow to one, y can only push-back τ on its in-flow arc, (x, y). Cell x's exit node becomes visited, x's out-flow becomes zero and τ continues on x's outgoing arc, (x, z). When τ reaches cell z, z's entry node becomes visited and z pushes τ back on its in-flow arc, (y, z). Cell y's exit node becomes visited, y's out-flow becomes zero and τ continues on y's outgoing arc, (y, t). When no other outgoing arc is present, the cell needs to push-back from its exit node to its entry node, which is only possible if its entry node is not visited. Finally, the search path, τ, reaches the target, t, and becomes $\tau = s.y.x.z.t$. After removing cancelling arcs and relinking the remaining ones, we have two node-disjoint paths, $s.x.z.t$ and $s.y.t$.

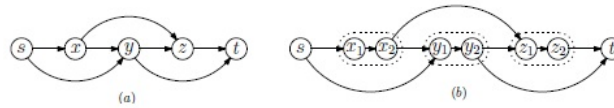

Fig. 3. Simulating node splitting [3].

5 Distributed DFS-based and BFS-based Solutions

As mentioned in Section 3, augmenting paths can be searched using DFS or BFS. Conceptually, DFS explores as far as possible along a single branch, before backtracking, while BFS explores as many branches as possible concurrently—P systems can exploit this parallelism.

5.1 Distributed DFS-based Strategy

Dinneen et al's DFS-based algorithms find disjoint paths in successive rounds [3].

Each round starts with a set of already *established disjoint paths*, which is empty at the start of the first round. The *source* cell, σ_s, starts to explore one of the untried branches. If the search path reaches the *target* cell, σ_t, it *confirms* to σ_s that a new augmenting paths was found; otherwise, it *backtracks*. While moving towards σ_s, the confirmation reshapes the existing paths and the newly found augmenting path, i.e. discarding cancelling arcs and relinking the rest, building a larger set of paths,

If σ_s receives the confirmation (one search path was successful, i.e. a new augmenting path was found), it broadcasts a *reset signal*, to prepare the next round. Otherwise, if the search fails, σ_s receives the backtrack. If there is an untried branch, the round is repeated. Otherwise, σ_s broadcasts a *finalize signal* to all cells and the search terminates.

This search algorithm is similar to a classical distributed DFS. Other more efficient distributed DFS algorithms [18] can be considered, but we do not follow this issue here.

5.2 Distributed BFS-based Strategy

Our BFS-based algorithms also work in successive rounds:

Each round starts with a set of already established disjoint paths, which is empty at the start of the first round. The source cell, σ_s, broadcasts a "wave", to find new augmenting paths. Current "frontier"cells send out *connect signals*. The cells which receive and *accept* these connect signals become the new frontier, by appending themselves at the end of current search paths. The advancing wave periodically sends *progress indicators* back to the source: (a) *connect acknowledgments* (at least one search path is still extending) and (b) *path confirmations* (at least one search path was successful, i.e. at least a new augmenting path was found). While travelling towards the source, each path confirmation reshapes the existing paths and the newly found augmenting path, creating a larger set of paths.

If no progress indicator arrives in the expected time, σ_s assumes that the search round ends. If at least one search path was successful (at least one augmenting path was found), σ_s broadcasts a *reset signal*, which prepares the next round, by resetting all cells (except the target). Otherwise, σ_s broadcasts a *finalize signal* to all cells and the search terminates.

In each round, an *intermediate* cell, σ_i, can be visited only once. Several search paths may try to visit the same intermediate cell *simultaneously*, but only one of them succeeds. Figure 4 (a) shows such a scenario: cells 1, 2 and 3 try to connect cell 4, in the same step; but only cell 1 succeeds, via arc $(1, 4)$. This *choice* operation is further described in Section 6.

The target cell, σ_t, faces a subtle decision problem. When several search paths arrive, simultaneously or sequentially, σ_t must quickly decide which augmenting path can be established and which one must be ignored (in the current round). We solve this problem using a *branch-cut* strategy. Given a search path, τ, its *branch ID* is the cell ID of its first intermediate cell after the source, taken by τ. Figure 4 (b) shows four potential paths arriving at cell 6: $\pi = 0.1.6$, $\tau_1 = 0.1.3.6$, $\tau_2 = 0.1.5.6$ and $\tau_3 = 0.2.4.6$; their branch IDs are 1, 1, 1 and 2, respectively. Paths π, τ_1 and τ_2 share the same branch ID, 1, and are incompatible. The following result is straightforward:

Proposition 1. *In any search round, search paths which share the same branch ID are incompatible; only one of them can be accepted.*

Therefore, the target cell accept or reject decision is based on branch ID. These *branch ID* operations are further described in Section 6.

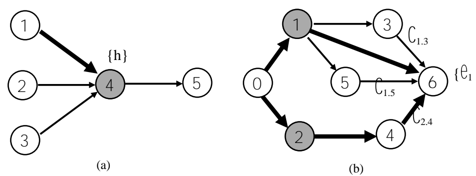

Fig. 4. BFS challenges. (a) A choice must be made between several search paths connecting the same cell (4), (b) Search paths sharing the same branch ID are incompatible.

6 P System Rules for Disjoint Paths Using BFS

The P system rules for edge- and node-disjoint paths are slightly different, due to the simulated node-splitting approach, but the basic principle is the same. We first discuss the edge-disjoint and then the changes required to cover the node-disjoint.

6.1 Rules for Edge-disjoint Paths

Algorithm 1 (P system algorithm for edge-disjoint paths)

Input: All cells start with the *same set of rules* and *without any topological awareness* (they do not even know their local neighbours). All cells start in the

same initial state. Initially, each cell, σ_i, contains a *cell ID* object, ι_i, which is *immutable* and used as a *promoter*. Additionally, the source cell, σ_s, and the target cell, σ_t, are decorated with objects, a and z, respectively.

Output: All cells end in the *same final state*. On completion, all cells are *empty*, with the following exceptions: (1) The source cell, σ_s, and the target cell, σ_t, are still decorated with objects, a and z, respectively; (2) The cells on *edge-disjoint paths* contain path link objects, for *predecessors*, p_j, and for *successors*, s_k.

We use the following six states: S_0, the initial state; S_1, the quiescent state; S_2, the frontier state; S_3, for previous frontier cells; S_4, the final state; and S_5, a special state for the target cell.

Initially, all cells are in the initial state, S_0. When each cell produces a catalyst-like object, it enters the quiescent state, S_1. When cells in S_1 accept connect signals, they enter the frontier state, S_2, except the target which changes directly to S_5. Cells on the frontier send connect signals to neighbors and then change to S_3, to receive and relay progress indicators. Specifically, the target remains in S_5, after accepting the first connect signal (because it is always waiting to be connected), until it receives the finalize signal. When the search finishes, all cells transit to the final state, S_4. Figure 5 shows all state transitions.

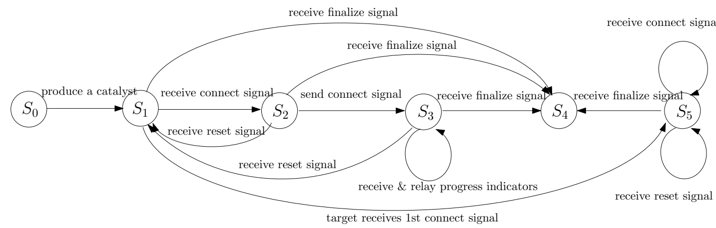

Fig. 5. State-chart of BFS-based algorithm.

We use these symbols to describe our edge-disjoint implementation:

- a indicates the source cell.
- z indicates the target cell.
- d indicates, in the source cell, that an augmenting path was found in the current round (it appears in the source cell).
- e_j records, in the target cell, the branch ID of a successful augmenting path (i.e. σ_j is the first cell after the source, in this augmenting path).
- c_s is the connect signal sent by the source cell, σ_s, to its children.
- $c_{j.k}$ is the connect signal sent by an intermediate cell, σ_k, to its children; j is the branch ID.
- $l_{j.k}$ is the connect signal sent by an intermediate cell, σ_k, to its parents; j is the branch ID.
- r_j is the connect acknowledgment sent to cell, σ_j.

- $f_{j.k}$ is the path confirmation of a successful augmenting path, sent by cell σ_j to cell σ_k.
- h is a catalyst object in each cell.
- o is a signal broadcast by the source cell, σ_s, to make each cell produce one catalyst object.
- u indicates the first intermediate cell after the source, which is produced on receiving the connect signal, c_s.
- b is the reset signal which starts a new round.
- g is the finalize signal which terminates the search.
- t_j indicates that cell σ_j is a predecessor on a search path (recorded when a cell accepts a connect signal).
- p_j is a disjoint path predecessor (recorded when a successful augmenting path is confirmed).
- s_j is a disjoint path successor (recorded when a successful augmenting path is confirmed).
- w, v implement a source cell timer to wait for the first response or confirmation.
- x, y implement another source cell timer to wait for the periodically relayed response or confirmation.

We next present the rules and briefly explain them.

0. Rules in state S_1:
   1 $S_0\ a \rightarrow_{\texttt{min}} S_1\ ah(o)_{\downarrow}$
   2 $S_0\ o \rightarrow_{\texttt{min}} S_1\ h(o)_{\downarrow}$
   3 $S_0\ o \rightarrow_{\texttt{max}} S_1$

1. Rules in state S_1:
   1 $S_1\ o \rightarrow_{\texttt{max}} S_1$
   2 $S_1\ d \rightarrow_{\texttt{max}} S_1$
   3 $S_1\ b \rightarrow_{\texttt{max}} S_1$
   4 $S_1\ e_j \rightarrow_{\texttt{max}} S_1$
   5 $S_1\ g \rightarrow_{\texttt{min}} S_4\ (g)_{\downarrow}$
   6 $S_1\ v \rightarrow_{\texttt{max}} S_1$
   7 $S_1\ w \rightarrow_{\texttt{max}} S_1$
   8 $S_1\ x \rightarrow_{\texttt{max}} S_1$
   9 $S_1\ y \rightarrow_{\texttt{max}} S_1$
   10 $S_1\ f_{j.k} \rightarrow_{\texttt{max}} S_1$
   11 $S_1\ t_j \rightarrow_{\texttt{max}} S_1$
   12 $S_1\ r_j \rightarrow_{\texttt{max}} S_1$
   13 $S_1\ a \rightarrow_{\texttt{min}} S_2\ a$
   14 $S_1\ c_j p_j \rightarrow_{\texttt{min}} S_1\ up_j$
   15 $S_1\ c_{j.k} p_k \rightarrow_{\texttt{min}} S_1\ p_k$
   16 $S_1\ zhc_{j.k} \rightarrow_{\texttt{min}} S_5\ zhp_k e_j (f_{i.k})_{\updownarrow}|_{\iota_i}$
   17 $S_1\ zhc_j \rightarrow_{\texttt{min}} S_5\ zhup_j (f_{i.j})_{\updownarrow}|_{\iota_i}$
   18 $S_1\ hl_{j.k} s_k \rightarrow_{\texttt{min}} S_2\ ht_k e_j s_k\ (r_k)_{\updownarrow}$

19 $S_1\ hc_j \rightarrow_{\mathtt{min}} S_2\ hut_j\ (r_j)_{\updownarrow}$
20 $S_1\ hc_{j.k} \rightarrow_{\mathtt{min}} S_2\ ht_k e_j\ (r_k)_{\updownarrow}$

2. Rules in state S_2:
   1 $S_2\ b \rightarrow_{\mathtt{min}} S_1 (b)_{\downarrow}$
   2 $S_2\ g \rightarrow_{\mathtt{min}} S_4 (g)_{\downarrow}$
   3 $S_2\ ah \rightarrow_{\mathtt{min}} S_3\ ahw(c_i)_{\downarrow}|_{\iota_i}$
   4 $S_2\ he_j \rightarrow_{\mathtt{min}} S_3\ he_j(l_{j.i})_{\uparrow}\ (c_{j.i})_{\downarrow}|_{\iota_i}$
   5 $S_2\ hu \rightarrow_{\mathtt{min}} S_3\ hu(l_{i.i})_{\uparrow}\ (c_{i.i})_{\downarrow}|_{\iota_i}$
   6 $S_2\ f_{j.k} \rightarrow_{\mathtt{max}} S_2$
   7 $S_2\ c_{j.k} \rightarrow_{\mathtt{max}} S_2$
   8 $S_2\ l_{j.k} \rightarrow_{\mathtt{max}} S_2$

3. Rules for state S_3:
   1 $S_3\ b \rightarrow_{\mathtt{min}} S_1 (b)_{\downarrow}$
   2 $S_3\ g \rightarrow_{\mathtt{min}} S_4 (g)_{\downarrow}$
   3 $S_3\ axyyf_{j.i} \rightarrow_{\mathtt{min}} S_3\ ads_j x|_{\iota_i}$
   4 $S_3\ axyyr_i \rightarrow_{\mathtt{min}} S_3\ ax|_{\iota_i}$
   5 $S_3\ axyyyf_{j.i} \rightarrow_{\mathtt{min}} S_3\ ads_j x|_{\iota_i}$
   6 $S_3\ axyyyr_i \rightarrow_{\mathtt{min}} S_3\ ax|_{\iota_i}$
   7 $S_3\ adxyyy \rightarrow_{\mathtt{min}} S_1\ a(b)_{\downarrow}$
   8 $S_3\ axyyy \rightarrow_{\mathtt{min}} S_4\ a(g)_{\downarrow}$
   9 $S_3\ awvv \rightarrow_{\mathtt{min}} S_4\ a(g)_{\downarrow}$
   10 $S_3\ awvf_{j.i} \rightarrow_{\mathtt{min}} S_3\ ads_j x|_{\iota_i}$
   11 $S_3\ awvr_i \rightarrow_{\mathtt{min}} S_3\ ax|_{\iota_i}$
   12 $S_3\ x \rightarrow_{\mathtt{min}} S_3\ y$
   13 $S_3\ t_j f_{k.i} \rightarrow_{\mathtt{min}} S_3\ p_j s_k (f_{i.j})_{\updownarrow}|_{\iota_i}$
   14 $S_3\ af_{j.i} \rightarrow_{\mathtt{min}} S_3\ as_j|_{\iota_i}$
   15 $S_3\ p_j s_j \rightarrow_{\mathtt{min}} S_3$
   16 $S_3\ r_i t_j \rightarrow_{\mathtt{min}} S_3\ t_j (r_j)_{\updownarrow}|_{\iota_i}$
   17 $S_3\ w \rightarrow_{\mathtt{min}} S_3\ wv$
   18 $S_3\ r_j \rightarrow_{\mathtt{max}} S_3$
   19 $S_3\ c_{j.k} \rightarrow_{\mathtt{max}} S_3$
   20 $S_3\ f_{j.k} \rightarrow_{\mathtt{max}} S_3$
   21 $S_3\ l_{j.k} \rightarrow_{\mathtt{max}} S_3$

4. Rules for state S_4:
   1 $S_4\ g \rightarrow_{\mathtt{max}} S_4$
   2 $S_4\ e_j \rightarrow_{\mathtt{max}} S_4$
   3 $S_4\ f_{j.k} \rightarrow_{\mathtt{max}} S_4$
   4 $S_4\ c_{j.k} \rightarrow_{\mathtt{max}} S_4$
   5 $S_4\ l_{j.k} \rightarrow_{\mathtt{max}} S_4$
   6 $S_4\ t_j \rightarrow_{\mathtt{max}} S_4$
   7 $S_4\ r_j \rightarrow_{\mathtt{max}} S_4$
   8 $S_4\ w \rightarrow_{\mathtt{max}} S_4$

9 $S_4\ v \to_{\texttt{max}} S_4$
10 $S_4\ u \to_{\texttt{max}} S_4$
11 $S_4\ h \to_{\texttt{max}} S_4$
12 $S_4\ o \to_{\texttt{max}} S_4$

5. Rules for state S_5:
1 $S_5\ c_j p_j \to_{\texttt{min}} S_5\ p_j$
2 $S_5\ c_{j.k} \to_{\texttt{min}} S_5|_{e_j}$
3 $S_5\ c_{j.k} p_k \to_{\texttt{min}} S_5\ p_k$
4 $S_5\ hc_{j.k} \to_{\texttt{min}} S_5\ hp_k e_j (f_{i.k})_{\updownarrow}|_{\iota_i}$
5 $S_5\ hc_j \to_{\texttt{min}} S_5\ hp_j (f_{i.j})_{\updownarrow}|_{\iota_i}$
6 $S_5\ g \to_{\texttt{max}} S_4$
7 $S_5\ b \to_{\texttt{max}} S_5$
8 $S_5\ f_{j.k} \to_{\texttt{max}} S_5$
9 $S_5\ l_{j.k} \to_{\texttt{max}} S_5$
10 $S_5\ t_j \to_{\texttt{max}} S_5$
11 $S_5\ r_j \to_{\texttt{max}} S_5$
12 $S_5\ u \to_{\texttt{max}} S_5$

The following paragraphs outline how these rules are used by each major cell group: the source cell, frontier cells, other intermediate cells and the target cell.

Scripts for the source cell: In the initial state S_0, the source cell, σ_s, indicated by the special object a, starts by broadcasting an object, o, to all cells and enters S_1 (rule 0.1); each receiving cell creates a local catalyst-like object, h, and enters S_1 (rule 0.2).

Next, cell σ_s enters S_2 (rule 1.13) and starts the search wave via connection requests, c_s (rule 2.3). Then, the source cell σ_s changes to state S_3 and uses timers to wait (a) one step for the the *first* progress indicators (rules 3.10, 3.11, 3.17), and (b) two steps for further *relayed* progress indicators (rules 3.3, 3.4, 3.12). If no progress indicator arrives when the timer overflows, cell σ_s waits *one more step* (rules 3.5, 3.6). If still no expected progress indicator arrives, cell σ_s assumes the round has ended. If an augmenting path was found in the current round, σ_s broadcasts a reset signal b to reset all cells (except the target σ_t) to S_1 (rule 3.7). Otherwise, σ_s broadcasts a finalize signal, g, which prompts all cells to enter S_4 (rules 3.8, 3.9).

It is interesting to note why the source cell needs to wait for one more step, even when the timer overflows. An intermediate cell filters connect signals, using rules 1.14–15, which have higher priority than the rules to accept a connect signal, i.e. rules 1.18–20. The rules to accept a connect signal cannot apply in the same step because of the different target states. For example, in Figure 6, path 0.2.4.6.7.9 is found in the first round. In the second round, search paths 0.1.4 and 0.3.5 attempt to connect to cell 6. Cell 6 discards cell 4's connect signal, following the higher-priority rule 1.15 and then, in the next step, accepts cell 5's connect signal, using rule 1.20. In this case, the source cell needs an extra one-step delay, to receive the

relayed connect acknowledgment from cell 6. All unacceptable signals are discarded in one step, so a one-step delay is enough.

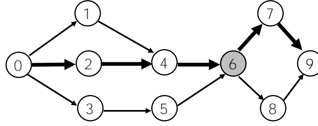

Fig. 6. A particular case requiring a delayed connect acknowledgment.

Scripts for a frontier cell: An intermediate cell, σ_i, if it is unvisited in this round, accepts exactly one connect signal and discards the rest; otherwise, it discards all connect signals. By accepting one connect signal, σ_i enters S_2 and becomes a frontier cell to send connect signals. When σ_i sends its connect signals, the frontier advances.

An intermediate cell, σ_i, may receive connect signals: (a) c_s, connect signals sent by the source cell, σ_s, to its children; (b) $c_{j.k}$, connect signals sent by a frontier cell, σ_k, to its children; (c) $l_{j.k}$, connect signals sent by a frontier cell, σ_k, to its parents. Received connect signals are checked for *acceptability*: (a) a c_s or $c_{j.k}$ connect signal is *acceptable* if it does not come from an established path predecessor, which corresponds to a forward operation (rules 1.14, 1.15, 1.19, 1.20); (b) a $l_{j.k}$ connect signal is *acceptable* if it comes from an established path successor, which corresponds to a push-back operation (rule 1.18).

Cell σ_i becomes a frontier cell by accepting either: (1) a connect signal, c_s, from σ_s (rules 1.14, 1.19), in this case, cell σ_i (a) generates an u, indicating that it is the first intermediate cell on the current search path (the first after cell σ_s); (b) records its predecessor on the search path, σ_s, as t_s; and (c) sends a connect acknowledgment, r_s, back to cell σ_s; or (2) a connect signal, $c_{j.k}$ or $l_{j.k}$ from σ_k (rules 1.15, 1.18, 1.20), in this case, cell σ_i (a) records the branch ID, j, as e_j; (b) records its predecessor on the search path, σ_k, as t_k; and (c) sends a connect acknowledgment, r_k, back to cell σ_k.

Then, as a frontier cell, σ_i sends connect signals to neighbors and changes to state S_3: (1) if cell σ_i is marked by an u object, it uses its own ID, i, as the branch ID to further generate connect signals, $c_{i.i}$ or $l_{i.i}$ (rule 2.5); (2) otherwise, σ_i uses the recorded e_j as the branch ID to further generate connect signals, $c_{j.i}$ or $l_{j.i}$ (rule 2.4).

Consider the scenario when several connect signals arrive *simultaneously* in an *unvisited* cell, σ_i (see Figure 4 (a)). Cell σ_i makes a (conceptually random) *choice* and selects exactly one of the acceptable connect signals, thus deciding which search path can follow through. To solve this choice problem, we use an object, h, which functions like a catalyst [15]. Object h is immediately consumed by the rule which *accepts* the connect signal, therefore no other connect signal is accepted (rules 1.16–20). Next, the catalyst, h, is recreated, but the cell also

changes its state, thus it cannot accept another connect signal (not in the same search round).

Scripts for other intermediate cell: A previous frontier cell, σ_i, relays progress indicators: connect acknowledgments, r_i (rule 3.16) and path confirmations, $f_{k.i}$ (rule 3.13). On receiving path confirmations, σ_i transforms a temporary path predecessor, t_j, into an established path predecessor, p_j, and records the path successors, s_k. In the next step, cell σ_i discards matching predecessor and successor objects (i.e. referring to the same cell), e.g., σ_i may already contain (from a previous round) another predecessor-successor pair, $p_{j'}$, $s_{k'}$. If $j = k'$, then p_j and $s_{k'}$ are deleted, as one end of the cancelling arc pair, (j, i) and (i, j); similarly, if $k = j'$, then s_k and $p_{j'}$ are deleted (rule 3.15).

Scripts for the target cell: The target cell, σ_t, accepts either (1) a connect signal from σ_s, c_s, if it does not come from an established path predecessor (rules 1.17, 5.1, 5.5), or (2) a connect signal from a frontier cell σ_k, $c_{j.k}$ (rules 1.16, 5.2, 5.3, 5.4), which indicates the different branch ID (rule 5.2) and does not come from an established path predecessor (rule 5.3). In case (1), cell σ_t: (a) generates an u, indicating that it is the second cell on a search path (the first after cell σ_s); (b) records its predecessor on the search path, σ_s, as p_s; and (c) sends a path confirmation $f_{t.s}$, back to cell σ_s. In case (2), cell σ_t: (a) records the branch ID, j, as e_j; (b) records its predecessor on the search path, σ_k, as p_k; and (c) sends a path confirmation, $f_{t.k}$, back to cell σ_k.

This branch-cut strategy is illustrated in Figure 4 (b). It shows an established path, $\pi = 0.1.6$, whose branch ID is recorded as e_1. Consider the fate of other search paths, $\tau_1 = 0.1.3.6$, $\tau_2 = 0.1.5.6$, and $\tau_3 = 0.2.4.6$, which attempt to reach the target 6, later in the same round. τ_1 sends the connect signal $c_{1.3}$, which is rejected. τ_2 sends the connect signal $c_{1.5}$, which is also rejected. τ_3 sends the connect signal $c_{2.4}$, which is accepted. To summarize, in this example round, two augmenting paths are established, π and τ_3; other attempts are properly ignored.

It is important that recording objects e_i are used as *promoters*, which enable rules, without being consumed [7]. Otherwise, objects e_i can be consumed before completing their role; e.g., the rejection of τ_1 would consume e_1 and there would be nothing left to reject τ_2.

Example 2. Table 7 shows Algorithm 1 tracing fragments for stages (a), (c) and (e) of Figure 2, illustrating how our P system solution works. In all stages, each cell, σ_i, contains a promoter object, ι_i, as the cell ID; the source cell, σ_0, and the target cell, σ_7, are decorated by objects, a and z, respectively. The catalyst object, h, remains in each cell after it is produced, until the cell enters the final state, S_4.

In stage 1(a), the two established paths, 0.1.4.7 and 0.2.5.7, are recorded by the following cell contents: $\sigma_0 : \{s_1, s_2\}$, $\sigma_1 : \{p_0, s_4\}$, $\sigma_2 : \{p_0, s_5\}$, $\sigma_4 : \{p_1, s_7\}$, $\sigma_5 : \{p_2, s_7\}$, $\sigma_7 : \{p_4, p_5\}$. In the source cell σ_0, xy^3 is a timer to wait for the relayed progress indicators, which currently overflows. The object d indicates that an augmenting path was found in the current round, so in the next step, the source cell, σ_0, broadcasts a reset signal to all cells to start a new round. Cells σ_1, σ_2, and

Table 7. Algorithm 1 tracing fragments for stages (a), (c) and (e) of Figure 2.

Stage\Cell	σ_0	σ_1	σ_2	σ_3
1(a)	$S_3\ \iota_0 adhs_1s_2xy^3$	$S_3\ \iota_1 hp_0s_4u$	$S_3\ \iota_2 hp_0s_5u$	$S_3\ \iota_3 ht_0u$
1(c)	$S_3\ \iota_0 ahs_1s_2xy$	$S_1\ \iota_1 hp_0s_4u^2$	$S_3\ \iota_2 e_3hp_0r_3s_5t_5u^2$	$S_3\ \iota_3 hr_3t_0u^2$
1(e)	$S_4\ \iota_0 as_1s_2s_3$	$S_4\ \iota_1 p_0s_4$	$S_4\ \iota_2 p_0s_6$	$S_4\ \iota_3 p_0s_5$
Stage\Cell	σ_4	σ_5	σ_6	σ_7
1(a)	$S_3\ \iota_4 e_1hp_1s_7$	$S_3\ \iota_5 e_2hp_2s_7$	$S_3\ \iota_6 e_2ht_2$	$S_5\ \iota_7 e_1e_2hp_4p_5z$
1(c)	$S_1\ \iota_4 f_{7.6}hp_1s_7$	$S_3\ \iota_5 e_3f_{7.6}hp_2s_7t_3$	$S_3\ \iota_6 e_3f_{7.6}ht_2$	$S_5\ \iota_7 e_1e_2e_3hp_4p_5p_6r_3z$
1(e)	$S_4\ \iota_4 p_1s_7$	$S_4\ \iota_5 p_3s_7$	$S_4\ \iota_6 p_2s_7$	$S_4\ \iota_7 p_4p_5p_6z$

σ_3 have objects, u, indicating that they are the first intermediate cells after the source, while cells $\sigma_4, \sigma_5, \sigma_6$ contain objects, e_j, which mean they should include j as the branch ID when sending connect signals. The target cell, σ_7, records the already used branch IDs, e_1 and e_2.

In stage 1(c), the successful search path 0.3.5.2.6.7 is recorded as: $\sigma_3 : \{t_0\}$, $\sigma_5 : \{t_3\}$, $\sigma_2 : \{t_5\}$, $\sigma_6 : \{t_2\}$, $\sigma_7 : \{p_6\}$ (the target records p_6 directly). The target cell σ_7 also records the branch ID of the newly successful path, e_3, and sends back a path confirmation $f_{7.6}$ to all its neighbors. In cell σ_3, the objects, r_3 and t_0, indicate that the connect acknowledgment needs to be relayed to the source cell σ_0. Thus, in the next step, cell σ_0 receives a connect acknowledgment from cell σ_3 and resets the timer.

In stage 1(e), all cells enter the final state S_4 and there are three established paths, 0.1.4.7, 0.2.6.7 and 0.3.5.7, which are recorded as: $\sigma_0 : \{s_1, s_2, s_3\}$, $\sigma_1 : \{p_0, s_4\}$, $\sigma_2 : \{p_0, s_6\}$, $\sigma_3 : \{p_0, s_5\}$, $\sigma_4 : \{p_1, s_7\}$, $\sigma_5 : \{p_3, s_7\}$, $\sigma_6 : \{p_2, s_7\}$, $\sigma_7 : \{p_4, p_5, p_6\}$.

The preceding arguments indicate a bisimulation relation between our BFS-based algorithm and the classical Edmonds and Karp BFS-based algorithm for edge-disjoint paths [4]. The following theorem encapsulates all these arguments:

Theorem 1. *When Algorithm 1 terminates, path predecessor and successor objects listed in its output section indicate a maximal cardinality set of edge-disjoint paths.*

6.2 Rules for Node-disjoint Paths

Algorithm 2 (P system algorithm for node-disjoint paths)

Input: As in the edge-disjoint paths algorithm of Algorithm 1.

Output: Similar to in the edge-disjoint paths algorithm. However, the predecessor and successor objects indicate *node-disjoint paths*, instead of edge-disjoint paths.

To simulate node splitting, the node-disjoint version uses additional symbols (as before, rules assume that cell σ_i is the current cell):

- m indicates that the "entry node is visited".
- n indicates that the "exit node is visited".

- q indicates that this cell's in-flow and out-flow is one (or, equivalently, that this cell is in an already established or confirmed path).
- $t_{j.k}$ indicates cell σ_i's predecessor, σ_j, on a search path, recorded after it receives the connect acknowledgment from cell σ_i's successor, σ_k (before receiving this acknowledgment, σ_i's predecessor is temporarily recorded as t_j.)
- $r_{j.k}$ is a connect acknowledgment sent by cell σ_j to cell σ_k.

0. Rules in state S_1:
   1 $S_0\ a \rightarrow_{\texttt{min}} S_1\ ah(o)_{\downarrow}$
   2 $S_0\ o \rightarrow_{\texttt{min}} S_1\ h(o)_{\downarrow}$
   3 $S_0\ o \rightarrow_{\texttt{max}} S_1$

1. Rules in state S_1:
   1 $S_1\ o \rightarrow_{\texttt{max}} S_1$
   2 $S_1\ d \rightarrow_{\texttt{max}} S_1$
   3 $S_1\ b \rightarrow_{\texttt{max}} S_1$
   4 $S_1\ e_j \rightarrow_{\texttt{max}} S_1$
   5 $S_1\ g \rightarrow_{\texttt{min}} S_4\ (g)_{\downarrow}$
   6 $S_1\ v \rightarrow_{\texttt{max}} S_1$
   7 $S_1\ w \rightarrow_{\texttt{max}} S_1$
   8 $S_1\ u \rightarrow_{\texttt{max}} S_1$
   9 $S_1\ m \rightarrow_{\texttt{max}} S_1$
   10 $S_1\ n \rightarrow_{\texttt{max}} S_1$
   11 $S_1\ f_{j.k} \rightarrow_{\texttt{max}} S_1$
   12 $S_1\ t_{j.k} \rightarrow_{\texttt{max}} S_1$
   13 $S_1\ t_j \rightarrow_{\texttt{max}} S_1$
   14 $S_1\ r_{j.k} \rightarrow_{\texttt{max}} S_1$
   15 $S_1\ a \rightarrow_{\texttt{min}} S_2\ a$
   16 $S_1\ c_j p_j \rightarrow_{\texttt{min}} S_1\ p_j$
   17 $S_1\ c_{j.k} p_k \rightarrow_{\texttt{min}} S_1\ p_k$
   18 $S_1\ zhc_{j.k} \rightarrow_{\texttt{min}} S_5\ zhp_k e_j (f_{i.k})_{\updownarrow}|_{\iota_i}$
   19 $S_1\ zhc_j \rightarrow_{\texttt{min}} S_5\ zhp_j (f_{i.j})_{\updownarrow}|_{\iota_i}$
   20 $S_1\ hl_{j.k} s_k \rightarrow_{\texttt{min}} S_2\ ht_k e_j s_k n\ (r_{i.k})_{\updownarrow}|_{\iota_i}$
   21 $S_1\ hc_{j.k} q \rightarrow_{\texttt{min}} S_2\ ht_k e_j mq\ (r_{i.k})_{\updownarrow}|_{\iota_i}$
   22 $S_1\ hc_j \rightarrow_{\texttt{min}} S_2\ hut_j\ (r_{i.j})_{\updownarrow}|_{\iota_i}$
   23 $S_1\ hc_{j.k} \rightarrow_{\texttt{min}} S_2\ ht_k e_j\ (r_{i.k})_{\updownarrow}|_{\iota_i}$

2. Rules in state S_2:
   1 $S_2\ b \rightarrow_{\texttt{min}} S_1 (b)_{\downarrow}$
   2 $S_2\ g \rightarrow_{\texttt{min}} S_4 (g)_{\downarrow}$
   3 $S_2\ ah \rightarrow_{\texttt{min}} S_3\ ahw(c_i)_{\downarrow}|_{\iota_i}$
   4 $S_2\ he_j m \rightarrow_{\texttt{min}} S_3\ he_j m\ (l_{j.i})_{\uparrow}|_{\iota_i}$
   5 $S_2\ he_j n \rightarrow_{\texttt{min}} S_3\ he_j n\ (l_{j.i})_{\uparrow}\ (c_{j.i})_{\downarrow}|_{\iota_i}$
   6 $S_2\ he_j \rightarrow_{\texttt{min}} S_3\ he_j\ (l_{j.i})_{\uparrow}\ (c_{j.i})_{\downarrow}|_{\iota_i}$
   7 $S_2\ hu \rightarrow_{\texttt{min}} S_3\ hu(l_{i.i})_{\uparrow}\ (c_{i.i})_{\downarrow}|_{\iota_i}$

8 $S_2\ f_{j.k} \rightarrow_{\texttt{max}} S_2$
9 $S_2\ c_{j.k} \rightarrow_{\texttt{max}} S_2$
10 $S_2\ l_{j.k} \rightarrow_{\texttt{max}} S_2$

3. Rules in state S_3:
1 $S_3\ b \rightarrow_{\texttt{min}} S_1 (b)_{\downarrow}$
2 $S_3\ g \rightarrow_{\texttt{min}} S_4 (g)_{\downarrow}$
3 $S_3\ hml_{j.k}s_k \rightarrow_{\texttt{min}} S_3\ hmnt_k e_j s_k\ (r_{i.k})_{\updownarrow}|_{\iota_i}$
4 $S_3\ he_j mn \rightarrow_{\texttt{min}} S_3\ hwe_j\ (l_{j.i})_{\uparrow}\ (c_{j.i})_{\downarrow}|_{\iota_i}$
5 $S_3\ axyyf_{j.i} \rightarrow_{\texttt{min}} S_3\ ads_j x|_{\iota_i}$
6 $S_3\ axyyr_{j.i} \rightarrow_{\texttt{min}} S_3\ ax|_{\iota_i}$
7 $S_3\ axyyyf_{j.i} \rightarrow_{\texttt{min}} S_3\ ads_j x|_{\iota_i}$
8 $S_3\ axyyyr_{j.i} \rightarrow_{\texttt{min}} S_3\ ax|_{\iota_i}$
9 $S_3\ adxyyy \rightarrow_{\texttt{min}} S_1\ a\ (b)_{\downarrow}|_{\iota_i}$
10 $S_3\ axyyy \rightarrow_{\texttt{min}} S_4\ a\ (g)_{\downarrow}|_{\iota_i}$
11 $S_3\ awvv \rightarrow_{\texttt{min}} S_4\ a(g)_{\downarrow}|_{\iota_i}$
12 $S_3\ awvf_{j.i} \rightarrow_{\texttt{min}} S_3\ ads_j x|_{\iota_i}$
13 $S_3\ awvr_{j.i} \rightarrow_{\texttt{min}} S_3\ ax|_{\iota_i}$
14 $S_3\ x \rightarrow_{\texttt{min}} S_3\ xy$
15 $S_3\ t_{j.k}f_{k.i} \rightarrow_{\texttt{min}} S_3\ p_j s_k q\ (f_{i.j})_{\updownarrow}|_{\iota_i}$
16 $S_3\ t_j f_{k.i} \rightarrow_{\texttt{min}} S_3\ p_j s_k q\ (f_{i.j})_{\updownarrow}|_{\iota_i}$
17 $S_3\ af_{j.i} \rightarrow_{\texttt{min}} S_3\ as_j|_{\iota_i}$
18 $S_3\ p_j s_j q \rightarrow_{\texttt{min}} S_3$
19 $S_3\ r_{k.i}t_{j.k} \rightarrow_{\texttt{min}} S_3\ t_{j.k}\ (r_{i.j})_{\updownarrow}|_{\iota_i}$
20 $S_3\ t_j r_{k.i} \rightarrow_{\texttt{min}} S_3\ t_{j.k}\ (r_{i.j})_{\updownarrow}|_{\iota_i}$
21 $S_3\ t_{j.l}r_{k.i} \rightarrow_{\texttt{min}} S_3\ t_{j.l}t_{j.k}\ (r_{i.j})_{\updownarrow}|_{\iota_i}$
22 $S_3\ w \rightarrow_{\texttt{min}} S_3\ wv$
23 $S_3\ ar_{j.i} \rightarrow_{\texttt{max}} S_3\ a|_{\iota_i}$
24 $S_3\ c_{j.k} \rightarrow_{\texttt{max}} S_3$
25 $S_3\ f_{j.k} \rightarrow_{\texttt{max}} S_3$
26 $S_3\ l_{j.k} \rightarrow_{\texttt{max}} S_3$

4. Rules in state S_4:
1 $S_4\ g \rightarrow_{\texttt{max}} S_4$
2 $S_4\ e_j \rightarrow_{\texttt{max}} S_4$
3 $S_4\ q \rightarrow_{\texttt{max}} S_4$
4 $S_4\ f_{j.k} \rightarrow_{\texttt{max}} S_4$
5 $S_4\ c_{j.k} \rightarrow_{\texttt{max}} S_4$
6 $S_4\ l_{j.k} \rightarrow_{\texttt{max}} S_4$
7 $S_4\ t_{j.k} \rightarrow_{\texttt{max}} S_4$
8 $S_4\ t_j \rightarrow_{\texttt{max}} S_4$
9 $S_4\ r_{j.k} \rightarrow_{\texttt{max}} S_4$
10 $S_4\ w \rightarrow_{\texttt{max}} S_4$
11 $S_4\ v \rightarrow_{\texttt{max}} S_4$
12 $S_4\ u \rightarrow_{\texttt{max}} S_4$

13 $S_4\ m \to_{\texttt{max}} S_4$
14 $S_4\ n \to_{\texttt{max}} S_4$
15 $S_4\ h \to_{\texttt{max}} S_4$
16 $S_4\ o \to_{\texttt{max}} S_4$

5. Rules in state S_5:
   1 $S_5\ c_j p_j \to_{\texttt{min}} S_5\ p_j$
   2 $S_5\ c_{j.k} \to_{\texttt{min}} S_5|_{e_j}$
   3 $S_5\ c_{j.k} p_k \to_{\texttt{min}} S_5\ p_k$
   4 $S_5\ hc_{j.k} \to_{\texttt{min}} S_5\ hp_k e_j (f_{i.k})_{\updownarrow}|_{\iota_i}$
   5 $S_5\ hc_j \to_{\texttt{min}} S_5\ hp_j (f_{i.j})_{\updownarrow}|_{\iota_i}$
   6 $S_5\ g \to_{\texttt{max}} S_4$
   7 $S_5\ b \to_{\texttt{max}} S_5$
   8 $S_5\ f_{j.k} \to_{\texttt{max}} S_5$
   9 $S_5\ l_{j.k} \to_{\texttt{max}} S_5$
   10 $S_5\ t_{j.k} \to_{\texttt{max}} S_5$
   11 $S_5\ t_j \to_{\texttt{max}} S_5$
   12 $S_5\ r_{j.k} \to_{\texttt{max}} S_5$
   13 $S_5\ u \to_{\texttt{max}} S_5$

When a cell, σ_i, is first reached by a search path, then both its "entry node" and "exit node" become *visited*. If this search path is successful, then σ_i is marked by one object q (rules 3.15, 3.16). In a subsequent round, new search paths can visit σ_i (1) via an incoming arc (forward mode); (2) via an outgoing arc, in the reverse direction (push-back mode) or (3) on both ways. When a search path visits σ_i via an incoming arc, it marks σ_i with one object, m, indicating a visited entry node (rule 1.21); in this case, the search path can only continue with a push-back (rule 2.4). When a search path visits σ_i via an outgoing arc, it marks the cell with one object, n, indicating a visited exit node (rule 1.20); in this case, the search path continues with all other possible arcs (rule 2.5), i.e. all forward searches and also a push-back on its current in-flow arc. A cell which has a visited entry node is in state S_3, but it can be later revisited by its exit node. Thus, in S_3, we provide extra rules to accept and send connect signals (rules 3.3, 3.4).

Cell, σ_i, can be visited at most once on each of its entry or exit nodes; but, it can be visited both on its entry and exit nodes, in which case it has two temporary predecessors (which simulate the node-splitting technique). In Figure 8, the search path, 0.4.5.2.1.8.9.3.2.6.7.10, has visited cell 2 twice, once on its "entry" node and again on its "exit" node. Cell 2 has two temporary predecessors, cells 5 and 3, and receives progress indicators from two successors, cells 1 and 6. Progress indicators relayed by cell 6 must be further relayed to cell 3 and progress indicators relayed by cell 1 must be further relayed to cell 5. To make the right choice, each cell records matching predecessor-successor pairs, e.g., cell 2 records the pairs $t_{5.1}$ and $t_{3.6}$. For example, when the progress indicator $r_{1.2}$ or $f_{1.2}$ arrives, cell 2 knows to forward it to the correct predecessor, cell 5. When the progress indicator $r_{6.2}$ or $f_{6.2}$ arrives, cell 2 knows to forward it to the correct predecessor, cell 3.

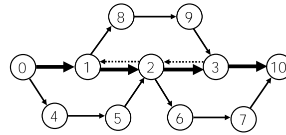

Fig. 8. An example of node-disjoint paths.

The following theorem sums up all these arguments:

Theorem 2. *When Algorithm 2 ends, path predecessors and successors objects mentioned in its output section indicate a maximal cardinality set of node-disjoint paths.*

7 Performance of BFS-based Solutions

Consider a simple P system with n cells, $m = |\delta|$ arcs, where $f_e =$ the maximum number of edge-disjoint paths, $f_n =$ the maximum number of node-disjoint paths and $d =$ the outdegree of the source cell. Dinneen at al. show that the DFS-based algorithms for edge- and node-disjoint paths run in $O(mn)$ P steps [3]. A closer inspection, not detailed here, shows that this upper bound can be improved.

Theorem 3. *The DFS-based algorithms run in $O(md)$ P steps, in both the edge- and node-disjoint cases.*

We show that our algorithms run asymptotically faster ($f_e, f_n \leq d$):

Theorem 4. *Our BFS-based algorithms run in at most $B(m, f) = (3m + 5)f + 4m + 6$ P steps, i.e. $O(mf)$, where $f = f_e$, in the edge-disjoint case, and $f = f_n$, in the node-disjoint case.*

Proof. 1. Initially, the source cell broadcasts a "catalyst" in one step.

2. Then, the algorithm repeatedly searches augmenting paths. First, consider the rounds where augmenting paths are found. In each round, each cell on the search path takes two steps to proceed, i.e. one step to accept a signal and one more step to send connect signals. Each search path spans at most m arcs, thus it takes at most $2m$ steps to reach its end (with or without reaching the target). All search paths in a round proceed in parallel. After the last augmenting path in a round was found, it takes at most m steps to confirm to the source. After receiving the last confirmation signal, the source cell waits four steps (to ensure that it is the last) and then takes one step to broadcast a reset signal. Therefore, each round, where augmenting paths are found, takes at most $3m + 5$ steps. At least one augmenting path is found in each round, so the total number of search rounds is at most f.

3. Next, consider the last search round, where no more augmenting paths are found. This case is similar, but not identical, to the preceding case. Each cell on the search path takes two steps to proceed, so it takes at most $2m$ steps to search augmenting paths. The connect acknowledgment from the end cell of the search path takes at most m steps to arrive at the source. The source waits for three or four steps for the time-out: three steps, if it does not receive any progress indicators; and four steps, otherwise. Then, the source cell broadcast a finalize signal, which takes at most m steps to reach all cells.
4. Finally, all cells take one final step, to clear all irrelevant objects, and the algorithm terminates.

To summarize, the algorithm runs in at most $(3m+5)f+4m+6$ steps and its asymptotic runtime complexity is $O(mf)$.

Table 9 compares the asymptotic complexity of our BFS-based algorithms against some well-known maximum flow BFS-based algorithms. Our BFS-based algorithms are faster, because they leverage the potentially unbounded parallelism inherent in P systems.

Table 9. Asymptotic worst-case complexity: classical BFS-based algorithms (steps), P system DFS-based algorithms [3] (P steps) and our P system BFS-based algorithms (P steps).

Edmonds-Karp [4]	$O(m^2n)$ steps
Dinic [2]	$O(mn^2)$ steps
Goldberg and Tarjan [6]	$O(nm \log n^2/m)$ steps
P System DFS-based [3]	$O(md)$ P steps
P System BFS-based [*here*]	$O(mf)$ P steps

Theorem 4 indicatess the worst-case upper bound, not the typical case. A typical search path does not use all m arcs. Also, the algorithm frequently finds more than one augmenting paths in the same search round, thus the number of rounds is typically much smaller than f. Therefore, the average runtime is probably much less than than the upper bound indicated by Theorem 4. Empirical results, obtained with our in-house simulator (still under development) support this observation.

Table 11, empirically compares the performance of our BFS-based algorithms against the DFS-based algorithms [3], for the scenarios of Figure 10. The empirical results show that BFS-based algorithms take fewer P steps than DFS-based algorithms. The performance is, as expected, influenced by the number of nodes and the density of the digraph. Typically, the ratio of BFS:DFS decreases even more, with the complexity of the digraph. We conclude that, the empirical complexity is substantially smaller than the asymptotic worst-case complexity indicated by Theorem 4.

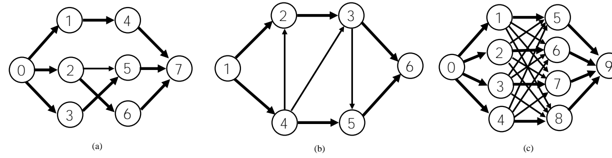

Fig. 10. Empirical tests of BFS-based and DFS-based algorithms.

Table 11. Empirical complexity of BFS-based and DFS-based algorithms (P steps).

Test Case	m	$f = f_e, f_n$	$B(m, f)$	BFS Empirical Complexity		DFS Empirical Complexity	
				Edge-disjoint	Node-disjoint	Edge-disjoint	Node-disjoint
(a)	10	3	151	44	45	63	62
(b)	9	2	106	24	24	61	59
(c)	24	4	410	66	75	241	194

8 Conclusions

We proposed the first BFS-based P system solutions for the edge- and node-disjoint paths problems. As expected, because of potentially unlimited parallelism inherent in P systems, our P system algorithms compare favourably with the traditional BFS-based algorithms. Empirical results show that, in terms of P steps, our BFS-based algorithms outperform the previously introduced DFS-based algorithms [3].

Several interesting questions and directions remain open. Can we solve this problem using a restricted P system without states, without sacrificing the current descriptive and performance complexity? What is the average complexity of our BFS-based algorithms? How much can we speedup the existing DFS-based algorithms, by use more efficient distributed DFS algorithms? An interesting avenue is to investigate a limited BFS design, in fact, a mixed BFS-DFS solution, which combines the advantages of both BFS and DFS. Finally, another direction is to investigate disjoint paths solutions on P systems with asynchronous semantics, where additional speedup is expected.

Acknowledgments

The authors wish to thank Tudor Balanescu, Michael J. Dinneen, Yun-Bum Kim, John Morris and three anonymous reviewers, for valuable comments and feedback that helped us improve the paper.

References

1. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. The MIT Press, 3rd edn. (2009)

2. Dinic, E.A.: Algorithm for solution of a problem of maximum flow in a network with power estimation. Soviet Math. Dokl., 11, 1277–1280 (1970)
3. Dinneen, M.J., Kim, Y.B., Nicolescu, R.: Edge- and node-disjoint paths in P systems. Electronic Proc. in TCS, 40, 121–141 (2010)
4. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM, 19(2), 248–264 (1972)
5. Ford, L.R., Jr., Fulkerson, D.R.: Maximal flow through a network. Canadian Journal of Mathematics, 8, 399–404 (1956)
6. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum flow problem. J. ACM, 35(4), 921–940 (1988)
7. Ionescu, M., Sburlan, D.: On p systems with promoters/inhibitors. J. Universal Computer Science, 10(5), 581–599 (2004)
8. Kozen, D.C.: The Design and Analysis of Algorithms. Springer, New York, NY, USA (1991)
9. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
10. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theor. Comput. Sci. 296(2), 295–326 (2003)
11. Nicolescu, R., Dinneen, M.J., Kim, Y.B.: 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), http://www.cs.auckland.ac.nz/CDMTCS/researchreports/342hyperdagA.pdf
12. Nicolescu, R., Dinneen, M.J., Kim, Y.B.: Structured modelling with hyperdag P systems: Part B. Report CDMTCS-373, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland, New Zealand (October 2009), http://www.cs.auckland.ac.nz/CDMTCS//researchreports/373hP_B.pdf
13. Nicolescu, R., Dinneen, M.J., Kim, Y.B.: Towards structured modelling with hyperdag P systems. Intern. J. Computers, Comm. and Control, 2, 209–222 (2010)
14. Păun, G.: Membrane Computing: An Introduction. Springer, New York, Inc., Secaucus, NJ, USA (2002)
15. Păun, G.: Introduction to membrane computing. In: Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.) Applications of Membrane Computing, pp. 1–42. Natural Computing Series, Springer (2006)
16. Păun, G., Centre, T., Science, C.: Computing with membranes. J. Computer and System Sciences, 61, 108–143 (1998)
17. Seo, D., Thottethodi, M.: Disjoint-path routing: Efficient communication for streaming applications. In: IPDPS. pp. 1–12. IEEE (2009)
18. Tel, G.: Introduction to Distributed Algorithms. Cambridge Univ. Press (2000)