Exploring Computation Trees Associated with P Systems

Andrés Cordón-Franco, Miguel A. Gutiérrez-Naranjo,
Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez

Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
University of Sevilla,
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
{acordon, magutier, marper, ariscosn}@us.es

Abstract. Usually, the evolution of a P system generates a computation tree too large to be efficiently handled with present–day computers; moreover, different branches in this tree may differ significantly from a computational complexity point of view, that is, for the amount of time and storage necessary to reach a result. In this paper we propose a first approach to outline a strategy for selecting a suitable branch, in some sense, of the computation tree associated with a P system. To this end, we introduce the key notion of the *dependency graph* of a P system.

1 Introduction

The evolution of a non-deterministic computational device usually gives rise to a computation tree with several branches, potentially infinite. Moreover, even in the case of finite computation trees, the amount of information is often too big to be efficiently handled with present-day computers. This is why it is convenient to look for good strategies for exploring such computation trees.

In this paper, we address one of these non-deterministic computational devices: P systems (see [2, 3] or visit [4]). Our goal is to enrich the simulation of the evolution of a P system with a new component, usually called *strategy* or *search plan*, that *controls* the rules to be applied in each cellular step, in order to select a branch of the computation tree with a low (if possible, minimal) cost.

The ideal situation would be to have a mapping h^* assigning to each node n of a computation tree a number $h^*(n)$ which measures the minimum length of a path from node n to a leaf (that is, to a halting configuration). Then, the strategy would consist on selecting at each non-deterministic step of the computation a node corresponding to the least value of h^*. However, the drawback is the high computational cost of such a mapping.

In a more realistic situation, we look for a computable mapping being an *efficient estimation* of h^*, that will be referred to as a *heuristic function* or *evaluation function.*

The paper is organized as follows. In Section 2 we describe the P system variant used in this paper. Section 3 introduces the key notion of the *dependency* *graph* of a P system. Based upon this notion, in Section 4 we present a mapping h, which helps us to search for a short branch in the computation tree of a P system. We also prove that this mapping is, indeed, a heuristic function. Finally, in Section 5 we develop an illustrative example and we finish with some final remarks.

2 P Systems with a Fixed Number of Membranes

Let us now briefly introduce the P system variant we shall work with in this paper. We shall only use evolution and communication rules and we shall avoid division or dissolution rules; so, the number of membranes (indeed, the whole membrane structure) remains unchanged during the evolution of the system.

P systems will be represented as tuples $\Pi = (\Gamma, H, \mu, w_1, \dots, w_q, R)$, where:

- Γ is a finite alphabet (the working alphabet) whose elements are called objects.
- H is a finite set of labels for membranes.
- μ is a tree-like membrane structure of degree q, one-to-one labelled by the set H.
- $w_1, \dots, w_q$ are multisets over Γ describing the multisets of objects initially placed in membranes from μ.
- R is a finite set of developmental rules of the following forms:
  1. $[a \to v]_l$, where $a \in \Gamma$, $v \in \Gamma^*$ *(object evolution rules).*
     Internal rule: an object a can evolve to a multiset v inside a membrane labelled by l.
  2. $[a]_l \to [\ ]_l\, b$, where $a, b \in \Gamma$ *(send-out communication rules).*
     An object a can get out of a membrane labelled by l, possibly transformed in a new object b.
  3. $a[\ ]_l \to [b]_l$, where $a, b \in \Gamma$ *(send-in communication rules).*
     An object a can get into a membrane labelled by l, possibly transformed in a new object b.

Note that cooperation is not allowed, and priority or electrical charges are not considered either. Besides, let us observe that the rules of the system are associated with labels (e.g., the rule $[a \to v]_l$ is associated with the label $l \in H$).

The rules are applied according to the following principles. Object evolution rules are applied in a maximal parallel way (that is, all objects which can evolve by such rules must do it), while communication rules are used sequentially, in the sense that one membrane can be used by at most one rule of this type at one step.

Concerning the interaction with the user, we shall consider that the results of the computations are collected outside the system (note that objects can leave the system during the evolution, provided that *send-out* rules are applied in the skin membrane). In order to be able to handle this external output of the computations in a formal way, we add a new region to the membrane structure, called *environment*. In this way, the information about the contents of the environment can be included in the configurations of the system.

3 Dependency Graphs

Roughly speaking, transitions in P systems are performed by rules in which the occurrence of an element a_0 in a membrane m_0 produces the apparition of the element a_1 in a membrane m_1. In a certain sense, one can consider a dependency between the pair (a_0, m_0) and the pair (a_1, m_1). This dependency is not based on the initial configuration but on the set of rules of the P system.

More formally, the rules in the P system model presented above can be reformulated as follows:

$$(a_0, m_1) \rightarrow (a_1, m_2)(a_2, m_2) \dots (a_n, m_2)$$

The occurrence of the element a_0 in membrane m_1 triggers the rule and produces the apparition of the multiset $a_1 a_2 \dots a_n$ into membrane m_2.

Obviously, if $m_1 \neq m_2$, then we have a communication rule. In this case, n must be equal to 1 and both membranes must be adjacent (one membrane is the father of the other one). If m_1 is the father of m_2, then we have a send-in communication rule, and if the opposite holds, then we have a send-out communication rule. On the other hand, if $m_1 = m_2$, then we have an *evolution* rule.

The pair (a_0, m_1) stands for the *left side* of the rule and the multiset of pairs $(a_1, m_2)(a_2, m_2) \dots (a_n, m_2)$ stands for the *right side* of the rule.

Next, we define the *dependency graph* of a P system based on this new representation of the rules.

Definition 1. *The* dependency graph *of a P system Π is a pair $G_\Pi = (V_\Pi, E_\Pi)$ such that V_Π is the set of all the pairs (z, m), where z is a symbol of the alphabet of Π and m is a label of a membrane (or env for the environment), and E_Π is the set of all ordered pairs (arcs) of elements of V_Π, $((z_1, m_1), (z_2, m_2))$, such that (z_1, m_1) is the left side of a rule and (z_2, m_2) belongs to the right side of that rule.*

The distance between two nodes of the dependency graph is defined as usual.

Definition 2. *A* path of length *n from a vertex x to a vertex y in a directed graph $G = (V, E)$ is a finite sequence $v_0, v_1, \dots, v_n$ of vertices such that $v_0 = x$, $v_n = y$, and $(v_i, v_{i+1}) \in E$ for $i = 0, \dots, n-1$. The sequence of vertices with a single vertex is also considered a path. If there is a path γ from x to y, we say that y is* reachable *from x (via γ). The* distance *between two vertices v_1 and v_2, $d(v_1, v_2)$, is the length of the shortest path from v_1 to v_2, or infinite, if v_2 is not reachable from v_1.*

We illustrate the definition of dependency graph with an example. Let us consider the next toy P system, with alphabet $\Gamma = \{a, b, c, d, v, w, z, yes\}$, membrane structure $\mu = [\ [\ ]_e]_s$ and set of rules:

Rule 1: $[\, a \rightarrow zb \,]_e$ **Rule 6:** $[\, a \,]_s \rightarrow [\ ]_s\, yes$
Rule 2: $[\, a \,]_e \rightarrow [\ ]_e\, a$ **Rule 7:** $[\, z \,]_e \rightarrow [\ ]_e\, a$
Rule 3: $[\, b \,]_e \rightarrow [\,]_e\, c$ **Rule 8:** $z\, [\ ]_e \rightarrow [\, d \,]_e$
Rule 4: $[\, a \rightarrow z^2 v \,]_s$ **Rule 9:** $[\, v \rightarrow w \,]_s$
Rule 5: $[\, z \,]_s \rightarrow [\ ]_s\, z$ **Rule 10:** $[\, w \,]_s \rightarrow [\ ]_s\, yes$

In order to construct the corresponding dependency graph, we have to consider the set of regions that are determined by the membrane structure of the P system. Since the elements can be sent out of the system (rules **5**, **6** and **10**), we have to consider three regions: $\{e, s, env\}$. Then, with the new representation, the rules can be rewritten as follows:

Rule 1: $(a, e) \rightarrow (z, e)(b, e)$ **Rule 6:** $(a, s) \rightarrow (yes, env)$
Rule 2: $(a, e) \rightarrow (a, s)$ **Rule 7:** $(z, e) \rightarrow (a, s)$
Rule 3: $(b, e) \rightarrow (c, s)$ **Rule 8:** $(z, s) \rightarrow (d, e)$
Rule 4: $(a, s) \rightarrow (z, s)^2(v, s)$ **Rule 9:** $(v, s) \rightarrow (w, s)$
Rule 5: $(z, s) \rightarrow (z, env)$ **Rule 10:** $(w, s) \rightarrow (yes, env)$

Therefore, the dependency graph of Π, $G_\Pi = (V_\Pi, E_\Pi)$, is defined by the following sets:

$$V_\Pi = \left\{ \begin{array}{l} (a, e),\ (b, e),\ (c, e),\ (d, e),\ (v, e),\ (w, e),\ (z, e),\ (yes, e), \\ (a, s),\ (b, s),\ (c, s),\ (d, s),\ (v, s),\ (w, s),\ (z, s),\ (yes, s), \\ (a, env),\ (b, env),\ (c, env),\ (d, env), \\ (v, env),\ (w, env),\ (z, env),\ (yes, env) \end{array} \right\},$$

$$E_\Pi = \left\{ \begin{array}{l} ((a, e), (z, e)), ((a, e), (b, e)), ((a, e), (a, s)), \\ ((a, s), (z, s)), ((a, s), (v, s)), ((a, s), (yes, env)), \\ ((b, e), (c, s)), \\ ((z, e), (a, s)), ((z, s), (z, env)), ((z, s), (d, e)), \\ ((v, s), (w, s)), \\ ((w, s), (yes, env)) \end{array} \right\}.$$

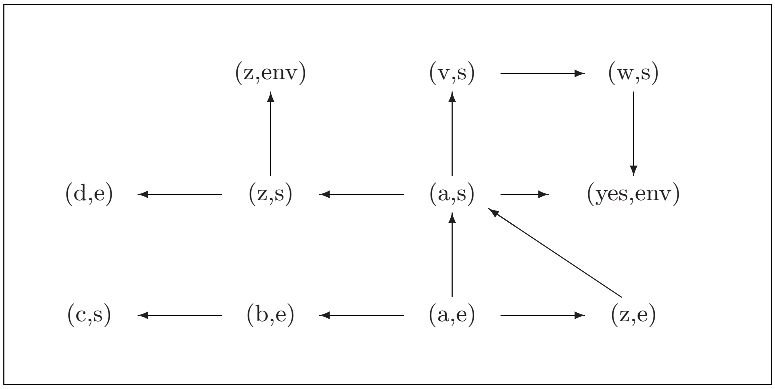
$$\begin{tikzcd}
 & (z,env) & (v,s) \arrow[r] & (w,s) \arrow[d] \\
(d,e) & (z,s) \arrow[l] \arrow[u] & (a,s) \arrow[l] \arrow[u] \arrow[r] & (yes,env) \\
(c,s) & (b,e) \arrow[l] & (a,e) \arrow[l] \arrow[u] \arrow[r] & (z,e) \arrow[ul]
\end{tikzcd}$$

Fig. 1. The dependency graph

The set V_Π of vertices has 24 elements, but 13 of them are isolated: only 11 vertices occur in some arc. Figure 1 shows the connected vertices of the graph. Note that the dependency graph only depends on the membrane structure and on the set of rules of the P system, and not on the elements of the membranes at the initial configuration. This example will be further studied in Section 5.

4 Heuristics

The simulation of a P system with current computers is a quite complex task. P systems are intrinsically non-deterministic computational devices and therefore their computation trees are difficult to store and handle with one-processor (or a bounded number of processors) computers. A possible way to overtake this shortcoming is to follow only one branch of the associated computation tree in each run of the simulator. Choosing which branch must be explored is not an easy decision, as different branches may differ significantly from a computational complexity point of view.

In order to select the best branch of the computation tree, the ideal situation would be to have a mapping h^* assigning to every node a number which indicates the minimum length of a path from this node to a leaf. Unfortunately, such a mapping usually has a high computational cost.

In a more realistic situation, we look for a mapping h being an *efficient estimation* of h^*, in the following sense:

– h assigns to each node n of a computation tree a number $h(n)$ verifying $h(n) \leq h^*(n)$; and
– the value $h(n)$ can be calculated with a *low* computational cost.

Then, $h(n)$ is a real-valued function over the nodes that, in some cases, provides a lower bound of the number of cellular steps needed to reach a halting configuration from node n. Such a mapping h will be referred to as a *heuristic function* or *evaluation function*. The aim of this section is to present a heuristic function h based on the notion of dependency graph of a P system.

In general, given a configuration of a non-deterministic P system, there exist several essentially different halting configurations which can be reached. However, in this first approach to the use of heuristics, we shall only consider a family of P system that behaves *well* in the following sense. Despite of the non-determinism, every computation of a P system in the family halts (i.e., there are no infinite branches in the computation tree) and, starting from a given initial configuration, all computations send out to the environment the same answer, just at the last step.

Consequently, when simulating such a P system, no matter which branch of the associated computation tree we follow, we will get the same answer, but the computational cost measured as the number of steps in the computation may vary widely. Hence, it is interesting to define a heuristic function measuring, in some sense, how far a given configuration is from a halting configuration in the computation tree.

In order to define such a heuristic mapping, we introduce the concept of *L-configuration of a P system.* As above, we codify the information as ordered pairs *(element, membrane).*

Definition 3. *An* L-configuration*, LC, associated with a given configuration, C, is a multiset of pairs (z, m) such that, for every object z of the alphabet and for every membrane m, the multiplicity of z in m in the configuration C is equal to the multiplicity of the pair (z, m) in LC.*

Next we define a mapping h which helps to select a good branch in each choice point of the computation tree associated with a P system.

Definition 4. *Let Π be a P system. Let LC be an L-configuration of Π, and z the object sent out to the environment at the end of the computation. The heuristic function h is defined as*

$$h(LC) = \min\{\, d(v, (z, env)) \mid v \in LC\}$$

where d is the distance in the dependency graph.

We finish this section with a theorem which states that the mapping h verifies the required property.

Theorem 1. *Let h^* be a function mapping a configuration C onto the number of steps of the shortest path in the computation tree from C to a halting configuration. For each configuration C, we have $h(LC) \leq h^*(C)$, where LC is the L-configuration associated with C.*

Proof. The basic idea of the proof is that the *transition* between two adjacent vertices in the dependency graph needs at least one evolution step of the P system. Keeping this in mind, the proof is quite natural.

We will reason by *reductio ad absurdum.* Suppose that there exists a configuration C_0 such that $h^*(C_0) < h(LC_0)$ and let C^* be a halting configuration such that from C_0 to C^* there are $h^*(C_0)$ evolution steps. Let z be the element sent out in the last step of the computation. Since C^* is a halting configuration, (z, env) belongs to the L-configuration LC^* and

$$h^*(C_0) < h(LC_0) = \min\{\, d(v, (z, env)) \mid v \in LC_0\}$$

In particular, for all $v \in LC_0$ such that (z, env) is reachable from v, $h^*(C_0)$ is less than the length of the shortest path from v to (z, env) in the dependency graph. That is, the distance from any element of the L-configuration LC_0 to (z, env) is larger than the number of computation steps of the P system from C_0 to a halting configuration.

But this is impossible since, as remarked above, the transition between two adjacent vertices in the dependency graph needs at least one computation step of the P system. □

5 Example

To illustrate the definition of the heuristic function h, we consider again the example described in Section 3. Figure 2 shows the computation tree associated with the evolution of this P system. In the first step of the computation two new configurations are possible. We use the function h to select one of them in order to reach a halting configuration in the minimum number of steps.

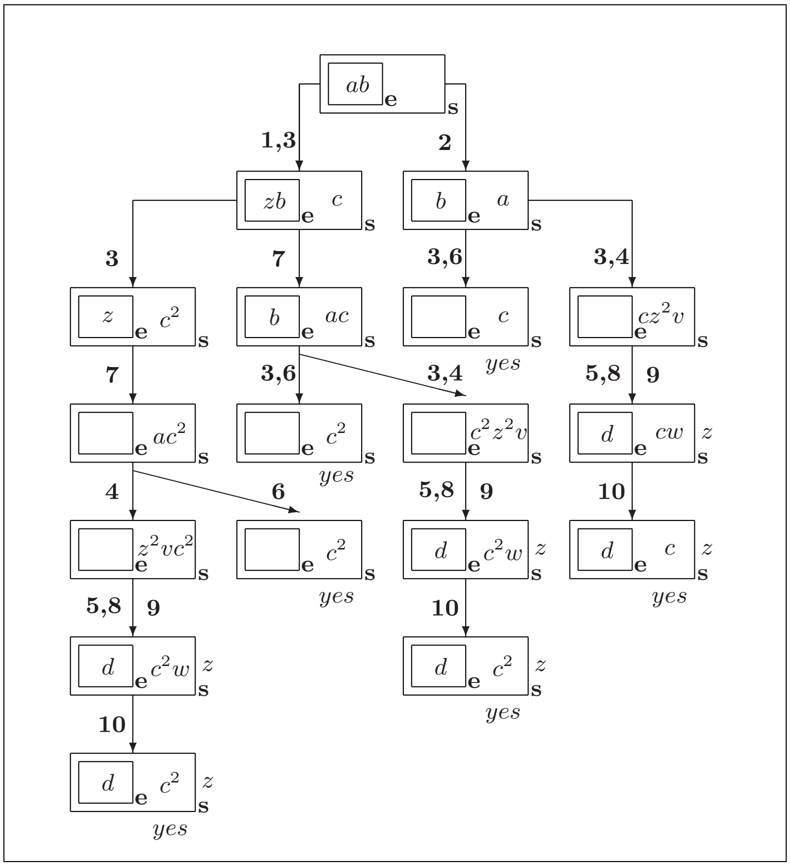

Fig. 2. The computation tree

The two possible configurations to proceed with are:

$$\begin{aligned} C_1 &\equiv [\ [\,zb\,]_e\, c]_s, \ \text{ by applying rules } \mathbf{1} \text{ and } \mathbf{3}, \\ C_2 &\equiv [\ [\,b\,]_e\, a]_s, \quad \text{by applying rule } \mathbf{2}. \end{aligned}$$

The information of these configurations can be represented by their associated L-configurations:

$$\begin{aligned} LC_1 &= \{(z,e),(b,e),(c,s)\},\\ LC_2 &= \{(b,e),(a,s)\}. \end{aligned}$$

In this example, the halting configurations are characterized by sending the object *yes* to the environment; that is, a configuration is a halting one if and only if (yes, env) is an element of its associated L-configuration. In order to decide which branch we must follow in the computation tree, we compute $h(LC_i)$, for $i = 1, 2$:

$$\begin{aligned} h(LC_1) &= \min\{\, d(v,(yes,env)) \mid v \in LC_1\}\\ &= d((z,e),(yes,env))\\ &= 2, \end{aligned}$$

$$\begin{aligned} h(LC_2) &= \min\{\, d(v,(yes,env)) \mid v \in LC_2\}\\ &= d((a,s),(yes,env))\\ &= 1. \end{aligned}$$

We have $h(LC_2) < h(LC_1)$, so, according to the heuristic h, we should follow the computation from the configuration C_2 to reach a halting configuration. Figure 2 shows the whole computation tree and it can be checked that, in the next evolution step, a halting configuration is reached from C_2 and this is the shortest path from the initial configuration to a halting one.

If we consider the subtree dominated by the configuration $C_1 \equiv [\,[\,zb\,]_e\, c]_s$, that is, if we consider C_1 as initial configuration, the function h can help to find the shortest computation. As Figure 2 shows, starting from C_1 two new configurations are possible:

$$\begin{aligned} C_{11} &\equiv [\,[\,z\,]_e\, c^2]_s, \text{ by applying rule } \mathbf{3},\\ C_{12} &\equiv [\,[\,b\,]_e\, ac]_s, \text{ by applying rule } \mathbf{7}. \end{aligned}$$

Their associated L-configurations are:

$$\begin{aligned} LC_{11} &= \{(z,e),(c,s),(c,s))\},\\ LC_{12} &= \{(b,e),(a,s),(c,s)\}. \end{aligned}$$

In order to decide which branch we must follow in the computation tree, we compute $h(LC_{1i})$ for $i = 1, 2$:

$$\begin{aligned} h(LC_{11}) &= \min\{\, d(v,(yes,env)) \mid v \in LC_{11}\}\\ &= d((z,e),(yes,env))\\ &= 2, \end{aligned}$$

$$\begin{aligned} h(LC_{12}) &= \min\{\, d(v,(yes,env)) \mid v \in LC_{12}\}\\ &= d((a,s),(yes,env))\\ &= 1. \end{aligned}$$

So, we should follow C_{12} to obtain the shortest computation. In fact, starting from C_{12}, a halting configuration is reached in the next step.

6 Conclusions and Further Research

In this paper we have presented a first approach towards an *efficient strategy* for searching in the computation tree associated with the evolution of a P system. Our aim is to select a branch with a low computational cost, measured as the number of steps needed to reach a halting configuration.

We propose a heuristic function h being an estimation of this computational cost, based on the notion of *dependency graph* of a P system.

We have illustrated the use of this heuristic function with an example belonging to a special family of P systems that, in a specific sense, "behaves well". Nonetheless, the search plan proposed here can be generalized, in a rather natural way, to other (less restrictive) variants of P system. In particular, extending the notion of *dependency graph* to P systems where the number of membranes can vary during the evolution (e.g., P systems with active membranes) seems to be a very promising question.

Another interesting topic is to study the transformation of configurations as a pre-computation phase, when the system itself is built. In this framework several important topics appear, such as the edit-distance between configurations, normal forms, reachability, etc.

Acknowledgement

The support for this research through the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds, is gratefully acknowledged.

References

1. A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, Weak metrics on configurations of a P system. In Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, and F. Sancho-Caparrini, (eds.), *Proceedings of the Second Brainstorming Week on Membrane Computing*, Report RGNC 01/04, University of Sevilla, 2004, 139–151.
2. Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143.
3. Gh. Păun, *Membrane Computing. An introduction*. Springer-Verlag, 2002.
4. `http://psystems.disco.unimib.it/`