# Editing Distances Between Membrane Structures

Damián López and José M. Sempere

Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia, 46071 Valencia, Spain
{dlopez, jsempere}@dsic.upv.es

**Abstract.** In this work we propose an efficient solution to calculate the minimum editing distance between membrane structures of arbitrary P systems. We use a new model of tree automata based on multisets of states and symbols linked to the finite control. This new model accepts a set of trees with symmetries between their internal nodes (*mirrored trees*). Once we have calculated the editing distance between an arbitrary tree and an arbitrary multiset tree automaton, we can translate the classical operations of insertion, deletion and substitution into rule applications of membrane dissolving and membrane creation.

## 1 Introduction

One of the main components of P systems is the membrane structure. This structure evolves during the computation time due to the application of rules associated to the membranes. The membrane structure can be represented by a tree in which the internal nodes denote regions which have inner regions inside. The root of the tree is always associated to the skin membrane of the P system.

The relation between regions and trees has been recently strengthened by Freund *et al.* [7]. These authors have established that any recursively enumerable set of trees can be generated by a P system with active membranes and string objects. So, P systems can be viewed as tree generators.

In this work we use multiset tree automata to accept and handle the tree structures defined by P systems [16]. This model is an extension of classical tree automata [8] in which the states and symbols of the finite control form *multisets*. Multiset theory has been linked to parallel processing as showed in [2].

The main aspect we will solve in this work is the one related to *editing structural configurations* of P systems. Recently, Csuhaj-Varjú *et al.* [4] have proposed editing distances between configurations of P systems. Here, we restrict our solution only to the structural configuration of P systems, that is, the membrane structure underlying any P system configuration. The multiset tree automata model that we propose in this work will be useful to calculate the trees associated with membrane structures. Here we can take advantage of a previous work on editing distances between trees and tree automata [10].

The structure of this work is as follows. First we introduce basic definitions and notation about multisets, tree languages and automata and P systems. In section

R. Freund et al. (Eds.): WMC 2005, LNCS 3850, pp. 326–341, 2006.

3, we introduce the model of multiset tree automata, we define the relation of *mirroring* between trees and we establish some results between tree automata, multiset tree automata, and mirroring trees. In section 4, we use previous results about editing distances between trees and tree automata in order to solve the minimum editing distance between membrane structures. Finally, we state some conclusions and give some guidelines for future works.

## 2 Notation and Definitions

In the sequel we will provide some concepts from formal language theory, membrane computing, and multiset processing. Further details can be found in the books [15], [12], and [2].

**Multisets**

First, we will provide some definitions from multiset theory as exposed in [17].

**Definition 1.** *Let $D$ be a set. A multiset over $D$ is a pair $\langle D, f\rangle$ where $f : D \longrightarrow \mathbb{N}$ is a function.*

**Definition 2.** *Suppose that $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ are two multisets. The removal of multiset $B$ from $A$, denoted by $A \ominus B$, is the multiset $C = \langle D, h\rangle$ where for all $a \in D$ $h(a) = max(f(a) - g(a), 0)$.*

**Definition 3.** *Let $A = \langle D, f\rangle$ be a multiset; we will say that $A$ is* empty *if for all $a \in D$, $f(a) = 0$.*

**Definition 4.** *Let $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ be two multisets. Their sum, denoted by $A \oplus B$, is the multiset $C = \langle D, h\rangle$, where for all $a \in D$ $h(a) = f(a) + g(a)$.*

**Definition 5.** *Let $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ be two multisets. We will say that $A = B$ if for all $a \in D$ $f(a) = g(a)$.*

The number of elements that a multiset contains can be finite. In such case, the multiset will be finite too. The size of any multiset $M$, denoted by $|M|$ will be the number of elements that it contains. We are specially interested in the class of multisets that we call *bounded multisets*. They are multisets that hold the property that the sum of all the elements is bounded by a constant $n$. Formally, we will denote by $\mathcal{M}_n(D)$ the set of all multisets $\langle D, f\rangle$ such that $\sum_{a\in D} f(a) = n$.

A concept that is quite useful to work with sets and multisets is the *Parikh mapping*. Formally, a Parikh mapping can be viewed as the application $\Psi : D^* \rightarrow \mathbb{N}^n$ where $D = \{d_1, d_2, \ldots, d_n\}$ and $D^*$ is the set of strings defined by $D$. Given an element $x \in D^*$ we define $\Psi(x) = (\#_{d_1}(x), \ldots, \#_{d_n}(x))$ where $\#_{d_j}(x)$ denotes the number of occurrences of $d_j$ in $x$.

Later, we will use tuples of symbols and states as strings and we will apply the Parikh mapping as defined above.

**Tree Automata and Tree Languages**

Now, we will introduce some concepts from tree languages and automata as exposed in [3, 8]. First, a *ranked alphabet* is the pair $(V, r)$ where $V$ is an alphabet and $r$ is a finite relation in $V \times \mathbb{N}$. We denote by $V_n$ the subset $\{\sigma \in V \mid (\sigma, n) \in r\}$. Given $(V, r)$ we define $maxarity(V)$ as the maximum integer $n$ such that$(\sigma, n) \in r$.

For every ranked alphabet $(V, r)$, the set of trees over $V$, is denoted by $V^T$ and defined inductively as follows:

$a \in V^T$ for every $a \in V_0$,
$\sigma(t_1, \ldots, t_n) \in V^T$ whenever $\sigma \in V_n$ and $t_1, \ldots, t_n \in V^T$, $n > 0$,

and let a *tree language* over $V$ be defined as a subset of $V^T$.

Given the tuple $l = \langle 1, 2, \ldots, k\rangle$ we will denote the set of permutations of $l$ by $perm(l)$. Let $t = \sigma(t_1, \ldots, t_n)$ be a tree over $V^T$, we will denote the set of permutations of $t$ at first level by $perm_1(t)$. Formally, $perm_1(t) = \{\sigma(t_{i_1}, \ldots, t_{i_n}) \mid \langle i_1, i_2, \ldots, i_n\rangle \in perm(\langle 1, 2, \ldots, n\rangle)\}$.

Let $\mathbb{N}^*$ be the set of finite strings of natural numbers, separated by dots, formed using the product as the composition rule and the empty word $\lambda$ as the identity. Let the prefix relation $\leq$ in $\mathbb{N}^*$ be defined by the condition that $u \leq v$ if and only if $u \cdot w = v$ for some $w \in \mathbb{N}^*$ $(u, v \in \mathbb{N}^*)$. A finite subset $D$ of $\mathbb{N}^*$ is called a *tree domain* if:

$$u \leq v, \text{ where } v \in D \text{ implies } u \in D, \text{ and}$$
$$u \cdot i \in D \text{ whenever } u \cdot j \in D \ (1 \leq i \leq j).$$

Each tree domain $D$ could be seen as an unlabelled tree whose nodes correspond to the elements of $D$ where the hierarchy relation is the prefix order. Thus, each tree $t$ over $V$ can be seen as an application $t : D \to V$. The set $D$ is called the *domain of the tree* $t$, and denoted by $dom(t)$. The elements of the tree domain $dom(t)$ are called *positions* or *nodes* of the tree $t$. We denote by $t(x)$ the label of a given node $x$ in $dom(t)$.

Let the level of $x \in dom(t)$ be denoted by $level(x)$. Intuitively, the level of a node measures its distance from the root of the tree. Then, we can define the depth of a tree $t$ as $depth(t) = max\{level(x) \mid x \in dom(t)\}$. In the same way, for any tree $t$, we denote the size of the tree by $|t|$ and the set of subtrees of $t$ (denoted with $Sub(t)$) as follows:

$$Sub(a) = \{a\} \text{ for all } a \in V_0,$$
$$Sub(t) = \{t\} \cup \bigcup_{i=1,\ldots,n} Sub(t_i) \text{ for } t = \sigma(t_1, \ldots, t_n) \ (n > 0).$$

For any set of trees $T$, $Sub(T) = \bigcup_{t \in T} Sub(t)$. Given a tree $t = \sigma(t_1, \ldots, t_n)$, the root of $t$ will be denoted as $root(t)$ and defined as $root(t) = \sigma$. If $t = a$, then $root(t) = a$. The successors of a tree $t = \sigma(t_1, \ldots, t_n)$ will be defined as $H^t = \langle root(t_1), \ldots, root(t_n)\rangle$.

**Definition 6.** *A* finite deterministic tree automaton *is defined by the tuple* $A = (Q, V, \delta, F)$*, where* $Q$ *is a finite set of states,* $V$ *is a ranked alphabet,* $Q \cap V = \emptyset$*,* $F \subseteq Q$ *is a set of final states, and* $\delta = \bigcup_{i:V_i \neq \emptyset} \delta_i$ *is a set of transition functions defined as follows:*

$$\begin{aligned} \delta_n : (V_n \times (Q \cup V_0)^n) \to Q \qquad & n > 0, \\ \delta_0(a) = a \qquad & \forall a \in V_0. \end{aligned}$$

Given the state $q \in Q$, we define the *ancestors* of the state $q$, denoted by $Anc(q)$, as the set of strings

$$Anc(q) = \{p_1 \dots p_n \mid p_i \in Q \cup V_0 \wedge \delta_n(\sigma, p_1, \dots, p_n) = q \in \delta\}.$$

From now on, we will refer to finite deterministic tree automata simply as *tree automata*. We suggest [3, 8] for other definitions on tree automata.

The transition function $\delta$ is extended to a function $\delta : V^T \to Q \cup V_0$ on trees as follows:

$$\begin{aligned} &\delta(a) = a \text{ for any } a \in V_0, \\ &\delta(t) = \delta_n(\sigma, \delta(t_1), \dots, \delta(t_n)) \text{ for } t = \sigma(t_1, \dots, t_n) \ (n > 0). \end{aligned}$$

Note that the symbol $\delta$ denotes both the set of transition functions of the automaton and the extension of these functions to operate on trees. In addition, one can observe that the tree automaton $A$ cannot accept any tree of depth zero.

Given a finite set of trees $T$, let the *subtree automaton* for $T$ be defined as $AB_T = (Q, V, \delta, F)$, where:

$$\begin{aligned} &Q = Sub(T), \\ &F = T, \\ &\delta_n(\sigma, u_1, \dots, u_n) = \sigma(u_1, \dots, u_n) \qquad && \sigma(u_1, \dots, u_n) \in Q, \\ &\delta_0(a) = a && a \in V_0. \end{aligned}$$

## P Systems

Finally, we will introduce some basic concepts from the theory of membrane systems taken from [12]. A general $P$ system of degree $m$ is a construct

$$\Pi = (V, T, C, \mu, w_1, \dots, w_m, (R_1, \rho_1), \dots, (R_m, \rho_m), i_0),$$

where:

- $V$ is an alphabet (the *objects*),
- $T \subseteq V$ (the *output alphabet*),
- $C \subseteq V$, $C \cap T = \emptyset$ (the *catalysts*),
- $\mu$ is a membrane structure consisting of $m$ membranes,
- $w_i$, $1 \leq i \leq m$ is a string representing a multiset over $V$ associated with the region $i$,

- $R_i$, $1 \leq i \leq m$ is a finite set of *evolution rules* over $V$ associated with the $i$th region and $\rho_i$ is a partial order relation over $R_i$ specifying a *priority*. An evolution rule is a pair $(u, v)$ (or $u \rightarrow v$), where $u$ is a string over $V$ and $v = v'$ or $v = v'\delta$, where $v'$ is a string over

$$\{a_{here}, a_{out}, a_{in_j} \mid a \in V, 1 \leq j \leq m\},$$

  and $\delta$ is an special symbol not in $V$ (it defines the *membrane dissolving action*),
- $i_0$ is a number between 1 and $m$ and it specifies the *output* membrane of $\Pi$, or $i_0 = \infty$ and in this case the output is read outside the system).

The language generated by $\Pi$ in external mode ($i_0 = \infty$) is denoted by $L(\Pi)$ and it is defined as the set of strings that can be defined by collecting the objects that leave the system by arranging them in the leaving order (if several objects leave the system at the same time, then all permutations are allowed). The set of numbers that represent the objects in the output membrane $i_0$ will be denote by $N(\Pi)$. Obviously, both sets $L(\Pi)$ and $N(\Pi)$ are defined only for *halting computations*.

Some kinds of P systems which have been proposed focus on the creation, division, and modification of membrane structures. There have been several works in which these operations have been proposed (see, for example, [1, 11, 12, 13]).

In the following, we enumerate some kinds of rules which are able to modify the membrane structure:

1. 2-division: $[_h a]_h \rightarrow [_{h'} b]_{h'} [_{h''} c]_{h''}$,
2. Creation: $a \rightarrow [_h b]_h$,
3. Dissolving: $[_h a]_h \rightarrow b$.

The power of P systems with the previous operations and other ones (e.g., *exocytosis, endocytosis*, etc.) has been widely studied in the literature.

## 3 Multiset Tree Automata and Mirrored Trees

We will extend some definitions of tree automata and tree languages over multisets. We will introduce the concept of multiset tree automata and then we will characterize the set of trees that they accept, as exposed in [16]. Observe that our approach is different from Csuhaj-Varjú *et al.* [5] and from Kudlek *et al.* [9] where the authors consider the case that *bags of objects* are analyzed by an abstract machine. Here, we do not consider *bags of (sub)trees* but we introduce bags of states and symbols in the finite control of the automata.

Given any tree automaton $A = (Q, V, \delta, F)$ and $\delta_n(\sigma, p_1, p_2, \ldots, p_n) \in \delta$, we can associate to $\delta_n$ the multiset $\langle Q \cup V_0, f \rangle \in \mathcal{M}_n(Q \cup V_0)$ where $f$ is defined by $\Psi(p_1 p_2 \ldots p_n)$. The multiset defined in such way will be denoted by $M_\Psi(\delta_n)$. Alternatively, we can define $M_\Psi(\delta_n)$ as $M_\Psi(p_1) \oplus M_\Psi(p_2) \oplus \cdots \oplus M_\Psi(p_n)$ where $M_\Psi(p_i) \in \mathcal{M}_1(Q \cup V_0)$ for all $1 \leq i \leq n$. Observe that if $\delta_n(\sigma, p_1, p_2, \ldots, p_n) \in \delta$, $\delta'_n(\sigma, p'_1, p'_2, \ldots, p'_n) \in \delta$, and $M_\Psi(\delta_n) = M_\Psi(\delta'_n)$ then $\delta_n$ and $\delta'_n$ are defined over

the same set of states and symbols but in different order (that is, the multiset induced by $\langle p_1 p_2 \dots p_n \rangle$ equals the one induced by $\langle p'_1 p'_2 \dots p'_n \rangle$).

Now, we can define a *multiset tree automaton* that performs a bottom-up parsing as in the tree automaton case.

**Definition 7.** *A* multiset tree automaton *is defined by the tuple* $MA = (Q, V, \delta, F)$*, where* $Q$ *is a finite set of states,* $V$ *is a ranked alphabet with maxarity*$(V) = n$*,* $Q \cap V = \emptyset$*,* $F \subseteq Q$ *is a set of final states, and* $\delta$ *is a set of transition functions defined as follows:*

$$\delta = \bigcup_{\substack{1 \le i \le n \\ V_i \neq \emptyset}} \delta_i,$$

$$\begin{aligned} \delta_i : (V_i \times \mathcal{M}_i(Q \cup V_0)) &\to \mathcal{P}(\mathcal{M}_1(Q)) \qquad i = 1, \dots, n, \\ \delta_0(a) = M_\Psi(a) &\in \mathcal{M}_1(Q \cup V_0) \qquad \forall a \in V_0. \end{aligned}$$

We can take notice that every tree automaton $A$ defines a multiset tree automaton $MA$ as follows

**Definition 8.** *Let* $A = (Q, V, \delta, F)$ *be a tree automaton. The multiset tree automaton induced by* $A$ *is defined by the tuple* $MA = (Q, V, \delta', F)$ *where each* $\delta'$ *is defined as follows:* $M_\Psi(r) \in \delta'_n(\sigma, M)$ *if* $\delta_n(\sigma, p_1, p_2, \dots, p_n) = r$ *and* $M_\Psi(\delta_n) = M$*.*

Observe that, in the general case, the multiset tree automaton induced by $A$ is non-deterministic.

As in the case of tree automata, $\delta'$ could also be extended to operate on trees. Here, the automaton carries out a bottom-up parsing where the tuples of states and/or symbols are transformed by using the Parikh mapping $\Psi$ to obtain the multisets in $\mathcal{M}_n(Q \cup V_0)$. If the analysis is completed and $\delta'$ returns a multiset with at least one final state, then the input tree is accepted. So, $\delta'$ can be extended as follows:

$$\begin{aligned} &\delta'(a) = M_\Psi(a) \text{ for any } a \in V_0, \\ &\delta'(t) = \{M \in \delta'_n(\sigma, M_1 \oplus \dots \oplus M_n) \mid M_i \in \delta'(t_i)\ 1 \le i \le n\}, \\ &\qquad \text{for } t = \sigma(t_1, \dots, t_n)\ (n > 0). \end{aligned}$$

Formally, every multiset tree automaton $MA$ accepts the following language

$$L(MA) = \{t \in V^T \mid M_\Psi(q) \in \delta'(t), q \in F\}.$$

Another extension which will be useful is the one related to the ancestors of every state. So, we define $Anc_\Psi(q) = \{M \mid M_\Psi(q) \in \delta_n(\sigma, M)\}$. The following two results characterize the relation between the languages accepted by tree automata and the multiset tree automata induced by them.

**Theorem 1.** *(Sempere and López, [16]) Let* $A = (Q, V, \delta, F)$ *be a tree automaton,* $MA = (Q, V, \delta', F)$ *be the multiset tree automaton induced by* $A$ *and* $t = \sigma(t_1, \dots, t_n) \in V^T$*. If* $\delta(t) = q$*, then* $M_\Psi(q) \in \delta'(t)$*.*

**Corollary 1.** *(Sempere and López, [16]) Let $A = (Q, V, \delta, F)$ be a tree automaton and $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by $A$. If $t \in L(A)$, then $t \in L(MA)$.*

### Mirrored Equivalent Trees

We will introduce the concept of *mirroring* in tree structures as it was exposed in [16]. Informally speaking, two trees will be related by mirroring if some permutations at the structural level make the difference among them. For example, the trees of Figure 1 have identical subtrees except that some internal nodes have changed their order.

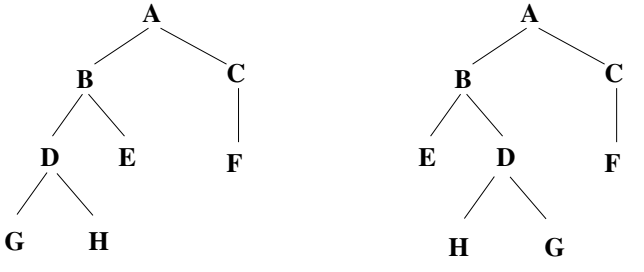


**Fig. 1.** Two mirrored trees

We propose a definition that relates all the trees with this mirroring property. For any other concepts used in this section, we refer to the previous section 2 on tree automata.

**Definition 9.** *Let $(V, r)$ be a ranked alphabet and $t$ and $s$ be two trees from $V^T$. We say that $t$ and $s$ are* mirror equivalent*, denoted by $t \bowtie s$, if one of the following conditions holds:*

1. $t = s = a \in V_0$,
2. $t \in perm_1(s)$,
3. $t = \sigma(t_1, \ldots, t_n)$, $s = \sigma(s_1, \ldots, s_n)$ *and there exists* $\langle s^1, s^2, \ldots, s^k \rangle \in perm(\langle s_1, s_2, \ldots, s_n \rangle)$ *such that* $t_i \bowtie s^i$ *for all* $1 \leq i \leq n$.

The following results characterize the set of trees accepted by a multiset tree automaton induced by a tree automaton.

**Theorem 2.** *(Sempere and López, [16]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $t = \sigma(t_1, \ldots, t_n) \in V^T$, and $s = \sigma(s_1, \ldots, s_n) \in V^T$. Let $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by $A$. If $t \bowtie s$, then $\delta'(t) = \delta'(s)$.*

Note that the converse result of this theorem is not generally true. For instance, consider the trees $t = \sigma(a)$ and $s = \sigma(a, \sigma(a))$ and the tree automaton with the following transition function:

$$\delta_1(\sigma, a) = q_1 \in F, \qquad \delta_2(\sigma, a, q_1) = q_1 \in F.$$

It is easy to see that $\delta'(t) = \delta'(s)$ but $t$ is not mirror equivalent to $s$.

**Corollary 2.** *(Sempere and López, [16]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $MA = (Q, V, \delta', F)$ the multiset tree automaton induced by $A$ and $t \in V^T$. If $t \in L(MA)$, then $s \in L(MA)$ for any $s \in V^T$ such that $t \bowtie s$.*

The last results were useful to propose an algorithm to determine whether two trees are mirror equivalent or not [16]: given two trees $s$ and $t$, we can establish in time $\mathcal{O}((min\{|t|, |s|\})^2)$ if $t \bowtie s$.

## 4 Solving the Membrane Structure Recognition Problem

Recently, in [7], a way to generate trees by membrane systems has been proposed. Initially, any membrane structure can be represented by a tree taking the membrane structure as a hierarchical order between regions. Freund *et al.* [7] have taken advantage of a variant of P systems with active membranes and string objects. Active membranes have an electrical charge (*polarization*) together with a set of rules that allow the membrane to change polarizations, move objects (strings), dissolving the membrane, 2-dividing the membrane, etc. They have proved that any recursively enumerable tree language can be generated by a P system.

A way to recognize two identical membrane structures by taking advantage of tree representations was proposed in [16]. For example, let us see Figure 2, in which we represent a membrane structure with different trees.

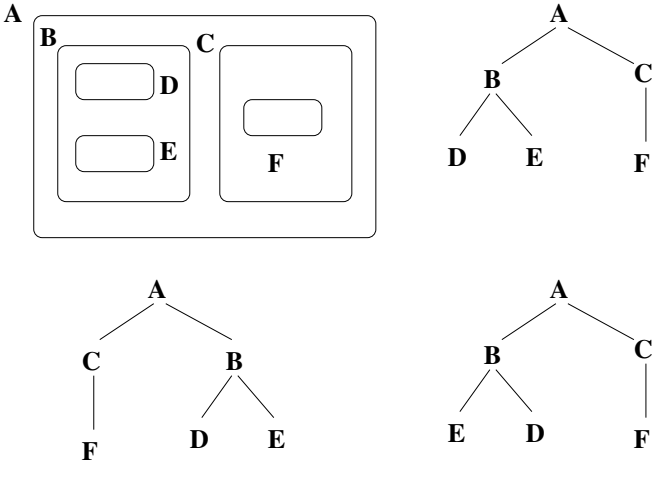


**Fig. 2.** A membrane structure together with different representations by trees

Obviously, the initial order of a membrane structure can be fixed. Anyway, whenever the system evolves (by membrane dissolving, division, creation, etc.) this order can be somehow ambiguous. Furthermore, the initial order of a P system is only a naming convention given that the membrane structure of any

P system can be renamed without changing its behavior due to the parallelism (observe that if this mechanism were sequential, then the ordering could be important for the final output).

The representation by trees could be essential for the analysis of the dynamic behavior of P systems. Whenever we work with trees to represent the membrane structure of a given P system, we can find a *mirroring effect*. Again, look at Figure 2: the three different trees proposed for the membrane structure have a mirroring property, that is, some subtrees at a given level of the tree have been permuted.

The method that we propose to establish if two membrane structures $\mu$ and $\mu\prime$ are identical is based on the algorithm proposed in [16]. First, we represent $\mu$ and $\mu\prime$ by $t$ and $s$ respectively. Then, we apply the proposed algorithm and, if $t \bowtie s$ we can affirm that $\mu$ and $\mu\prime$ are identical.

## 5 Editing Distances Between Membrane Structures

The study of relations between membrane structures is proposed in the sequel. The main problem we address is the following:

*Let $\mu$ and $\mu\prime$ be two membrane structures corresponding to arbitrary P systems. What is the minimum set of membrane rule applications needed to transform one into the other?*

The solution to the last problem can be approached by using multiset tree automata and editing distances between trees and tree automata. A previous work [10], considered the case of tree automata. Here, we will extend the previous results to multiset tree automata as described in previous sections.

First, we will describe the method employed in [10], in order to give the main components of the editing distance calculation.

Given a tree automaton $A = (Q, V, \delta, F)$ and a tree $t$, the distance between $t$ and $A$ can be established as the minimum in the set $\{D(t, q) \mid q \in F\}$, where $D(t, q)$ is the minimum distance of the tree $t$ to the state $q$. The distance $D(t, q)$ evaluates the number of operations needed to reduce the tree $t$ to the state $q$ according to the function $\delta$ in automata $A$. Some operations involved in the distance refer to operations for trees as *Insertion, Deletion* and *Substitution*. We consider the costs for these operations as exposed in [10]. Observe that these costs are usually defined by taking into account the sizes of the trees. So, the bigger tree involved in the operation, the bigger cost to handle it:

- Insertion
  $\forall a \in V \ I(a) = 1$
  $I(\sigma(t_1, t_2, \ldots, t_k)) = 1 + \sum_{\forall j} I(t_j)$,
- Deletion
  $\forall a \in V \ B(a) = 1$
  $B(\sigma(t_1, t_2, \ldots, t_k)) = 1 + \sum_{\forall j} B(t_j)$,

– Substitution
  $\forall a \in V \ S(a,a) = 0$
  $\forall a,b \in V \ S(a,b) = 1$
  $S(\sigma(t_1,t_2,\ldots,t_k),a) = B(\sigma(t_1,t_2,\ldots,t_k)) + I(a)$
  $S(a,\sigma(t_1,t_2,\ldots,t_k)) = B(a) + I(\sigma(t_1,t_2,\ldots,t_k))$.

So, the distance of every (sub)tree to a tree automaton will involve every ancestor of each state of the automata together with the substructures of the tree. If we have to reduce the structure $\sigma(s_1,s_2,\ldots,s_n)$ to the state $q$ such that $Anc(q)$ contains $\langle p_1,\ldots,p_m\rangle$, we will have to modify substructures $s_i$ or we will have to insert states $p_j$ at the minimum cost.

The edition cost of every tree to every state of the automaton can be calculated by considering the set of ancestors of the state and the set of successors of the tree. Then we can apply a dynamic programming scheme that takes into account previous calculations which can be stored in a distance matrix. For additional details of this method we refer the reader to [10].

The main components used to calculate the distance of a tree $t$ to a multiset tree automaton $MA$ are the same as in the tree automata case with the following remarks:

1. The successors of any node in the tree are considered as a multiset instead of a sequence.
2. The ancestors of every state in the automaton form a multiset.
3. The editing costs for trees and states are the same as in the tree automata.
4. The calculation of the edit distance is performed by using a edition matrix which can be obtained by using a dynamic programming strategy with some differences which will be explained later.

We propose **Algorithm 1** which obtains the distance from a tree $t$ to a multiset tree automaton $MA$. Note that the target of the algorithm is to force the automaton to accept the tree. Therefore the set of edit operations is not fully needed. The algorithm use edit operations for substitution (reduction) of a tree to a state of the automaton, deletion of a (sub)tree and insertion of a state. Intuitively, the substitution of a tree by a state of the automaton could be seen as the substitution of the tree by the nearest tree that could be reduced to the state.

The error-correcting analysis method is shown in **Algorithm 1**. First the cost of the basic operations are obtained (i.e., insertion cost of a state and deletion of a subtree). Each of the calculations carried out are stored in a distance matrix indexed by the set of subtrees and the set of states of the automaton. This matrix is first initialized and the basic distances are stored. Distances between symbols in $V_0$ and between any symbol and any state of the automaton are also considered.

Note that the key problem of the algorithm is to find, for any subtree $t' = \sigma(t_1,\ldots,t_n)$ of $t$ and any transition $\delta(\sigma,M) = M_\Psi(p)$, with $M \in Anc_\Psi(p)$, the matching of minimum cost between each $t_i$ and the states and symbols in $M$. This problem can be reduced to the *minimum cost maximum matching* or

**Algorithm 1.** Algorithm to obtain the minimum distance from a tree $t$ to the nearest tree in $L(MA)$.

**Input:**

A multiset tree automaton $A = (Q, V, \delta, F)$.
A tree $t$.

**Output:**

Edit distance from $t$ to the automaton $A$.

**Method:**

/* initialization */
$\forall t' \in Sub(t) \quad B[t'] = |t'| \quad end\forall$
$\forall a \in V_0 \quad I[a] = 1 \quad end\forall$
$\forall q \in Q$
  $I[q] = min\{|t'| \ : \ \delta(t') = q\}$
  $\forall t' \in Sub(t)$
    $D[t', q] = \infty$
  $end\forall$
$end\forall$
$\forall a, b \in V_0$
  $D[a, b] = \begin{cases} 1 & \text{if } a \neq b \\ 0 & \text{otherwise} \end{cases}$
  $D[a, q] = 1 + I[q] \ : \ q \in Q$
$end\forall$
/* iteration */
$\forall t' = \sigma(t'_1, \ldots, t'_n) \in Sub(t)$ /* postorder traverse */
  $\forall \delta(\sigma, M) = M_\Psi(p)$
    $D[t', p] = min(D[t', p], MMC(t', \delta(\sigma, M)))$
  $end\forall$
$end\forall$
Return$(min\{D[t', q] \ : \ q \in F\})$
EndMethod:

*maximum bipartite matching* problem [14]. It is known that this problem can be solved in polinomial time by reducing it to the *minimum cost maximum flow* (MCMF) problem (see also [14]). This scheme is similar to the one proposed in [18] where the author considers distances between unordered trees.

Briefly, MCMF looks for obtaining, for a given graph $G = (V, E)$ in which functions *capacity* and *cost* are defined among the edges, the best way (with lower cost) to send the maximum flow between two nodes of the graph. The flow has to take into account the capacity constraint. The cost function measures the penalization of each unit of flow. Several solutions have been implemented to solve this problem and their complexities depend on the number of nodes $n$ and the number of edges of the graph $m$. A proper algorithm for our purposes could be the one by Edmons and Karp [6] because its complexity depends only on the number of nodes of the graph ($\mathcal{O}(n^3)$).

Given a tree $t = \sigma(t_1, \ldots, t_n)$ and a transition $M_\Psi(p) \in \delta(\sigma, M)$, the minimum cost matching between $t_i$ and the states in $M$ can be obtained by the subroutine

$MMC$. First, this subroutine builds the directed graph from the parameters and set the proper capacities and costs functions among the edges. Then, a general solution could be run in order to solve the matching. The subroutine is shown in **Algorithm 2**.

Intuitively, each successor tree and each state (namely nodes $t_i$ and $q_j$ respectively) have their own nodes in the graph. Each node in one set is connected with all the nodes in the other. These connections model the reduction (substitution) of each tree to each state. Therefore, the capacity of these edges is set to 1 (these edges can be used only once) and the cost is set to the distance between each tree and each state. Note that this distance is always available due to the postorder traverse of the tree.

**Algorithm 2.** $MMC$ Subroutine to obtain the Maximum Matching of Minimum Cost.

**Input:**

A multiset tree automaton transition $\delta(\sigma, M) = M_\Psi(p)$.
A tree $t = \sigma(t_1, \ldots, t_n)$.

**Output:**

Minimum cost of the maximum match between $\{t_1, \ldots, t_n\}$ and $M$.

**Method:**

/* construction of the graph */
Let $G = (V, E)$ where:
$V = \{t_1, \ldots, t_n\} \cup M \cup \{s, ss, iq, dt\}$
$(t_i, q_j) \in E,\ \forall q_j \in M;\ i : 1..n$
$(t_i, dt) \in E,\ i : 1..n$
$(iq, q_j) \in E,\ \forall q_j \in M$
$(s, t_i) \in E,\ i : 1..n$
$(q_j, ss) \in E,\ \forall q_j \in M$
$(s, iq), (dt, ss) \in E$
/* set capacities of each edge */
$c(t_i, q_j) = 1,\ \forall q_j \in M;\ i : 1..n$
$c(t_i, dt) = 1,\ i : 1..n$
$c(iq, q_j) = \#_{q_j}(M),\ i : 1..n$
$c(s, t_i) = 1,\ i : 1..n$
$c(q_j, ss) = \#_{q_j}(M),\ \forall q_j \in M$
$c(s, iq) = |M|,\ c(dt, ss) = n$
/* set cost of each edge */
$d(t_i, q_j) = D[t_i, q_j],\ \forall q_j \in M;\ i : 1..n$
$d(t_i, dt) = B[t_i],\ i : 1..n$
$d(iq, q_j) = I[q_j],\ \forall q_j \in M$
$d(s, t_i) = 0,\ i : 1..n$
$d(q_j, ss) = 0,\ \forall q_j \in M$
$d(s, iq) = 0$
$d(dt, ss) = 0$
Return($MinCostMaxFlow(G)$)

EndMethod:

The set of edit operations we consider also takes into account the insertion of a state. The node $iq$ and the connections between this node and the nodes $q_j$ model the insertion operation. Thus, the cost of these edges is set to the insertion cost of the state. Note that the number of insertions of each state is bounded by the number of occurrences of the state in $M$, therefore, the capacities of these edges is set to this value.

In the same way, in order to model the deletion of trees, the node $dt$ and the connections with the successor trees are considered in the graph. Each tree can be deleted only once, therefore the capacity of these edges is set to 1. Obviously, the cost of these edges is set to the cost of deleting the corresponding tree.

The construction of the graph also considers a source node $s$. This node is connected to the tree nodes, with connectivity 1 and cost 0 (these edges must be selected without cost). The node $s$ is also connected to the node $iq$ and the cost of this edge is set to 0. Note that the number of state insertions is bounded by the number of states, therefore, the capacity of this edge is set to $|M|$. The cost of this connection is set to 0.

Finally, the graph construction considers a sink node $ss$. This node is connected with the state nodes $q_j$ with cost 0. Note that the edition process aims to fit the set of successors with the multiset of ancestors, thus, the capacity of the edges must be set to the number of occurrences of each state. The node $dt$ is also connected with the node $ss$ with cost 0. This edge models the tree deletions, therefore, the capacity of the connection must be set to the number of trees that can be deleted.

*Example 1.* Let us consider the tree $t = \sigma(\sigma(b, \sigma(a, \sigma(a, b), a)), \sigma(a, \sigma(a, a)))$ and the automaton defined by the following transition functions with $q_3 \in F$:

$$\delta(\sigma, aq_1a) = q_1, \quad \delta(\sigma, bq_2) = q_2, \quad \delta(\sigma, aa) = q_1,$$
$$\delta(\sigma, b) = q_2, \qquad \delta(\sigma, q_1q_2) = q_3.$$

First, the insertion and deletion costs are obtained. They are shown in the following tables

| $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ |
|---|---|---|---|---|---|
| 3 | 6 | 8 | 3 | 5 | 14 |

Deletion costs

| $q_1$ | $q_2$ | $q_3$ |
|---|---|---|
| 3 | 2 | 6 |

Insertion costs

Then, the editing process considers the first postorder subtree $\sigma(a, b)$ and the first transition $\delta(\sigma, aq_1a) = q_1$. The process starts with the construction of the graph shown in Figure 3.

Solid lines in Figure 3 show the minimum cost matching. The distance is stored in the matrix of distances. Note that this cost is improved when the transition $\delta(\sigma, aa)= q_1$ is considered. The following table shows an intermediate state of the matrix.

| $D_A$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ |
|---|---|---|---|---|---|---|
| $q_1$ | 1 | 1 | | | | |
| $q_2$ | 1 | 3 | | | | |
| $q_3$ | 7 | | | | | |

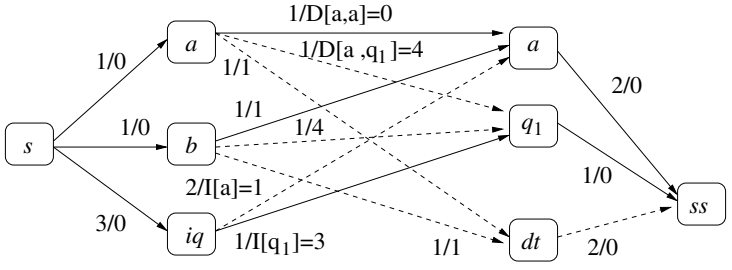


**Fig. 3.** Underlying graph to obtain the distance of the first postorder subtree to the first transition of the automaton. Edge labels show the capacity/cost. Solid lines show the best matching.

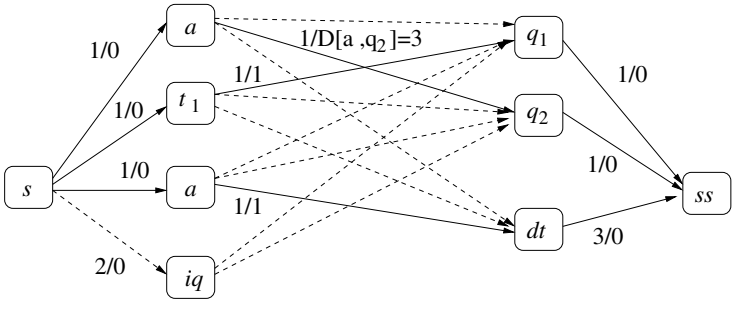


**Fig. 4.** Underlying graph to obtain the distance of the second postorder subtree to the transition of the automaton $\delta(\sigma, q_1 q_2) = q_3$. Solid lines show the best matching.

We now compute the distance of the second postorder subtree, $\sigma(a, \sigma(a, b), a)$, to the state $q_3$. The underlying graph is shown in Figure 4. The best matching is indicated in solid lines.

Observe that the minimum editing distance that we have calculated can be established in terms of operations which have a translation into membrane rules. Let us consider that $\mu$ is the membrane structure which is accepted by the multiset tree automaton $MA$ and $\mu'$ is represented by a tree $t$. We have the following correspondences between edition operations and membrane rules:

1. Insertion of state $q$
   Let us suppose that the insertion is produced to match the ancestors of a state $p$. The minimum tree that can be reduced to $q$ is $t_j$. The operations needed to achieve this goal in the membrane structure are membrane creation at region $p$ in order to obtain membrane structure $t_j$.
2. Reduction of tree $t_i$ to state $q$
   Let us suppose that the tree which can be reduced to $q$ with a minimal cost is $t_j$, according with the $\delta$ function of the automaton. The operations needed to make this reduction are the ones involved to transform $t_i$ to $t_j$ at a

region $k$. These operations consider again membrane creation and dissolving depending on the operations involved in the minimum distance from $t_i$ to $t_j$.
3. Substitution of $a$ by $b$
   The region $a$ is dissolved and created with a new label.
4. Deletion of tree $t_i$
   Let us suppose that $t_i$ is a membrane structure at region $k$. The deletion consists of several membrane dissolving of structure $t_i$.

## 6 Conclusions and Future Work

We have proposed a method to calculate the minimum number of membrane rules needed to transform a membrane structure into a different one. The number of rules needed, if so, establishes an editing distance between P systems by taking into account only membrane modifications. This measure can provide new definitions about *structural confluence* in P systems, that is, structural agreement during evolution.

Observe that we have worked with a simplified version of P systems. That is, the objects inside any region do not influence the editing distance. A future research will consider how the objects can be taken into account to calculate the editing distance.

## Acknowledgements

Work supported by the Spanish CICYT under contract TIC2003-09319-C03-02. The authors are grateful to the anonymous referees which have made several sharp remarks on the first version of this work. Special thanks are given to Mario Pérez-Jiménez for useful discussions during the *3rd Brainstorming Week on Membrane Computing* which was held in Sevilla from 31st January to 4th February 2005.

## References

1. A. Alhazov, T.O. Ishdorj: Membrane operations in P systems with active membranes. In *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho Caparrini, eds.). TR 01/04 of RGNC. Sevilla University, 2004, 37–844.
2. C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa: *Multiset Processing. Mathematical, Computer Science, Molecular Computing Points of View.* LNCS 2235, Springer, Berlin, 2001.
3. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi: Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (1997).
4. E. Csuhaj-Varjú, A. Di Nola, Gh. Păun, M. Pérez-Jiménez, G. Vaszil: Editing configurations of P systems. *Proc. Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, RGNC Report 01/2005, 131–155.

5. E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana: Multiset automata. In [2], 69–83.
6. J. Edmonds, R.M. Karp: Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19, 2 (1972), 248–264.
7. R. Freund, M. Oswald, A. Păun: P systems generating trees. In *Pre-proceedings of Fifth Workshop on Membrane Computing, WMC5, Milano, June 2004* (G. Mauri, Gh. Păun, C. Zandron, eds.), MolCoNet project IST-2001-32008, 2004, 221–232.
8. F. Gécseg, M. Steinby: Tree languages. In vol. 3 of [15], 1–69.
9. M. Kudlek, C. Martín-Vide, Gh. Păun: Towards a formal macroset theory. In [2], 123–133.
10. D. López, J.M. Sempere, P. García: Error correcting analysis for tree languages. *International Journal of Pattern Recognition and Artificial Intelligence*, 14, 3 (2000), 357–368.
11. A. Păun: On P systems with active membranes. In: *Proceedings of Conference on Unconventionals Models of Computation*, Brussels, 2000, 187-201.
12. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
13. Gh. Păun, Y. Suzuki, H. Tanaka, T. Yokomori: On the power of membrane division in P systems. *Theoretical Computer Sci.*, 324, 1 (2004), 61–85.
14. R.L. Rivest T.H. Cormen, C.E. Leiserson, C. Stein: *Introduction to Algorithms.* MIT Press and McGraw-Hill, second edition, 2001.
15. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages.* Springer, Berlin, 1997.
16. J.M. Sempere, D. López: Recognizing membrane structures with tree automata. In *Proceedings of the 3rd Brainstorming Week on Membrane Computing 2005* (M.A. Gutirrez Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, eds.), RGNC Report 01/2005 Research Group on Natural Computing, Sevilla University, Fénix Editora, 2005, 305–316.
17. A. Syropoulos: Mathematics of multisets. In [2], 347–358.
18. K. Zhang: A constrained edit distance between unordered labelled trees. *Algorithmica*, 15 (1996), 205–222.