Mathematical and Computer Modelling 52 (2010) 1982–1996

Contents lists available at ScienceDirect

Mathematical and Computer Modelling

journal homepage: www.elsevier.com/locate/mcm

# P systems and computational algebraic topology

Hepzibah A. Christinal [a,b], Daniel Díaz-Pernil [b,*], Pedro Real [b]

[a] Karunya University Coimbatore, Tamilnadu, India

[b] Research Group on Computational Topology and Applied Mathematics, Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012, Sevilla, Spain

ARTICLE INFO

*Article history:*
Received 23 September 2009
Received in revised form 25 December 2009
Accepted 31 January 2010

*Keywords:*
P systems
Membrane computing
Homology groups
Segmentation
Digital topology
Algebraic topology

ABSTRACT

Membrane Computing is a paradigm inspired from biological cellular communication. Membrane computing devices are called P systems. In this paper we calculate some algebraic-topological information of 2D and 3D images in a general and parallel manner using P systems. First, we present a new way to obtain the homology groups of 2D digital images in time logarithmic with respect to the input data involving an improvement with respect to the algorithms development by S. Peltier et al. Second, we obtain an edge-segmentation of 2D and 3D digital images in constant time with respect to the input data.



## 1. Introduction

Natural Computing studies new computational paradigms inspired by Nature. It abstracts the way in which Nature "computes", conceiving new computing models. There are several fields in Natural Computing that are now well established. Among them, Genetic Algorithms introduced by Holland [1] which is inspired by natural evolution and selection in order to find an optimal solution in a large set of feasible candidate solutions; Neural Networks, introduced by McCulloch and Pitts [2], which is based on the interconnections of neurons in the brain; or DNA-based molecular computing, that was initiated by Head [3]. Some years after, Adleman [4] published a solution to an instance of the Hamiltonian path problem by manipulating DNA strands in a lab.

Membrane Computing [5] is a new field in Natural Computing that is based on the assumption that the processes taking place within the compartmental structure of a living cell can be interpreted as computations. In particular, we focus on membranes, which are involved in many reactions taking place inside various compartments of a cell. Biological membranes are much more than mere barriers that define compartments, they act as selective channels of communication between different compartments as well as between the cell and its environment [6].

The computational devices in Membrane Computing are called P systems. Roughly speaking, a P system consists of a membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous,non-deterministic, maximally parallel manner. Since the seminal paper [7], different models of P systems have been studied. According to their architecture, these models can be split into two sets: cell-like P systems [8,9] and tissue-like P systems [10–14]. In cell-like P systems, membranes are hierarchically arranged in a tree-like structure. The inspiration for such architecture is the set of vesicles inside the cell. All of them perform their biological processes in parallel and life is the consequence of the harmonious conjunction of such processes.

In 1996, Chao and Nakayama connected Natural Computing and Algebraic Topology using Neural Networks [15] (extended Kohonen mapping). Some years after, Subramanian et al. presented in [16,17] two works where Digital Image

* Corresponding author.
*E-mail addresses:* hepzi@us.es (H.A. Christinal), sbdani@us.es (D. Díaz-Pernil), real@us.es (P. Real).

0895-7177/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mcm.2010.06.001

and Natural Computing were linked. In this paper we develop a new research line, where for the first time, the power and efficiency of P systems [18,19,8] are applied to topological process for 2D and 3D digital images.

The question of which membrane model is best to use in this paper is not very important since we use so few membranes in our solution. Moreover, it is not necessary to use a dynamic membrane structure, so there is no reason to use other variants of P systems with membrane division or membrane creation rules. In this paper, the first where computational algebraic topology and membrane systems are related, we use the original variant of P systems which we refer to as *basic P systems*.

First, we calculate the homology groups of binary 2D images. *Homology theory* is a branch of algebraic topology that attempts to distinguish between spaces by constructing algebraic invariants that reflect the connectivity properties of the space. The field has its origins in the work of Poincaré. Homology groups (related to the "different" $n$-dimensional holes, connected components, tunnels, cavities, etc., a geometric object has) are invariants from Algebraic Topology [20] which are frequently used in Digital Image Analysis [21–24] and Structural Pattern Recognition [25–30]. In some way, they reflect the topological nature of the object in terms of the number and characteristics of its holes. In a binary 2D image, the computation of homology groups can be reduced to a process of black and white connected components labeling. The different black connected components are the generators of the 0-dimensional homology group of the "black" part of the image whereas the closed "black" curves surrounding the different white connected components of the image are the generators of its 1-dimensional homology group.

Thanks to massive parallelism of the membrane devices, the time used to obtain the homology groups does not depend on the number of connected components or the number of holes, but only on their width. The necessary time to calculate the homology groups of 2D digital images with the basic P systems defined in section is logarithmic with respect to the input data ($\mathcal{O}(n)$). This involves an improvement with respect to the algorithm development by Peltier et al. in [31], where they use irregular graphs pyramids with a time complexity of $\mathcal{O}(n^{5/3})$.

Second, we realize the segmentation of 2D and 3D colored digital images. Segmentation in computer vision (see [32]), refers to the process of partitioning a digital image into multiple segments (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics.

The paper is structured as follows: first we present some preliminary concepts related to digital image processing and P systems. In the next section we present the definition of basic P systems with input. In Section 4, we design two systems to calculate $H_0$ and $H_1$ of any 2D image ($n \times n$) and present an specific $8 \times 8$ image to show how these systems work. In Section 5, we design a family of systems for edge-based segmentation in a 2D image ($n \times m$). At the end of this section, we introduce a family of P systems to obtain an edge-based segmentation of 3D images. Finally, we draw some conclusions and outline some future work.

## 2. Preliminaries

In this section we briefly recall some concepts related to digital image and P systems.

A 2D digital image can be considered as a matrix where one pixel in the image is an element of the matrix. There are two natural definitions for adjacency, 4-adjacency or 8-adjacency.

In 4-adjacency, given a pixel $K_{ij}$ (where $K = B \vee K = W$), the list of adjacent pixels to this is $\{K_{ij-1}, K_{ij+1}, K_{i-1j}, K_{i+1j}\}$ i.e. the pixels to the north, south, east, and west of $K_{ij}$ are called *adjacent* (we do not consider those pixels diagonal to $K_{ij}$ to be adjacent). For example:

$$\begin{array}{ccc} & B & \\ B & K & W \\ & W & \end{array}$$

In 8-adjacency, the list of adjacent pixels to $K_{ij}$ is $\{K_{i-1j-1}, K_{i-1j}, K_{i-1j+1}, K_{ij-1}, K_{ij+1}, K_{i+1j-1}, K_{i+1j}, K_{i+1j+1}\}$ i.e. the adjacent pixels to any pixel $K_{i,j}$ are those north, south, east, west of it and, moreover, we consider the diagonal pixels. For example:

$$\begin{array}{ccc} B & W & B \\ W & K & B \\ B & B & W. \end{array}$$

In this paper we work with 4-adjacency (for 2D images), because from a membrane computing point of view it is more complicated to design systems with small numbers of adjacent pixels.

For 3D images, the pixels are known as *voxels*. The considered matrix in this case is a 3 dimensional matrix, and the adjacency elected is 6-adjacency, i.e.; the adjacent voxels with regard to the given one are to the north, south, west, east, up and down. We never consider a voxel to be adjacent if it appears in a diagonal with respect to the considered voxel.

When we consider a colored digital image we define the *alphabet of colors associated with this image* $\mathcal{C} = \{a : a_{ij}$ is a pixel of the image $\wedge$ $a$ is a color$\}$. The size of this alphabet is given by the number of colors that appear in the image. Moreover, we can introduce an order in this alphabet which it is used in the Section 5 (image segmentation).

We define the *black graph (white graph) associated with the image*, $G_B = (V_B, E_B)$ $(G_W = (V_W, E_W))$, where the vertices are formed by the black (white) pixels and there exists an edge between two black (white) pixels if they are adjacent.

An *alphabet*, $\Sigma$, is a non empty set, whose elements are called *symbols*. An ordered sequence of symbols is a *string*. The number of symbols in a string $u$ is the *length* of the string, and it is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by $\lambda$. The set of strings of length $n$ built with symbols from the alphabet $\Sigma$ is denoted by $\Sigma^n$ and $\Sigma^* = \cup_{n\geq 0} \Sigma^n$. A *language* over $\Sigma$ is a subset from $\Sigma^*$.

A *multiset over a set A* is a pair $(A, f)$ where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in A \mid f(x) > 0\}$ and its *size* is defined as $\sum_{x\in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite).

If $m = (A, f)$ is a finite multiset over $A$, then it will be denoted as $m = a_1^{f(a_1)} a_2^{f(a_2)} \cdots a_k^{f(a_k)}$, where $supp(m) = \{a_1, \ldots, a_k\}$, and for each element $a_i$, $f(a_i)$ is called the multiplicity of $a_i$.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For details, see [5].

## 3. Formal framework

We present a description, detailed but informal, of a model of *Cellular Computation with Membranes* introduced by Păun in [7]: *basic P systems*.

A P system has a set of membranes, organized by a membrane structure than can be viewed as a rooted tree. There is a *skin membrane* (root) which contain all other membranes inside it and which separates the system from the environment. Membranes that contain no other membranes are called *elemental membranes* (leaves of the membrane structure tree). The *regions* delimited by the membranes (the space between a membrane and the membranes it contains if any) can contain *objects* that can appear repeatedly. These objects can be transformed into other objects or move to an adjacent membrane, by means of *evolution rules* associated with a membrane.

We recursively define a membrane structure, $\mu$, as an element of the set *MS* as follows:

1. $[\,] \in MS$.
2. If $\mu_1, \ldots, \mu_k \in MS$ (with $k \geq 1$), then $[\mu_1 \ldots \mu_k] \in MS$.

Each pair of matching brackets appearing in $\mu$ is called a *membrane*. The pair of external brackets is called the *skin membrane*. And a pair of brackets without other brackets inside it is called an *elementary membrane*. The *degree* of $\mu$ is the number of membranes contained in $\mu$.

Let two membranes be $m_1$ and $m_2$ of $\mu$, such that the second one is contained in the first ($m_2 \subset m_1$). If there does not exists a membrane $m'$ contained in $m_1$ and containing $m_2$ ($m_2 \subset m' \subset m_1$) then these membranes are adjacent. In this case, we can say $m_1$ is the *parent membrane* of $m_2$, and $m_2$ is the *child membrane* of $m_1$.

We now informally present the *syntax* of the basic model of cell computing.

A *basic* P *system* of degree $q \geq 1$ is a tuple

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, (R_1, \rho_1), \ldots, (R_q, \rho_q), o_\Pi)$$

where:

- $\Gamma$ is a finite alphabet, whose elements are called objects.
- $\mu$ is a membrane structure. The membranes are labeled using natural numbers from 1 to $q$ (it means $\mu$ contain exactly $q$ membranes identified by its label).
- $\mathcal{M}_i$ is a finite multiset over $\Gamma$ associated with the membrane of the system with label $i \in \mathbb{N}$, for each $i = 1, \ldots, q$.
- $R_i$ is a finite set of evolution rules associated with the membranes in the system labeled by $i$, for each $i = 1, \ldots, q$. An evolution rule is a pair $(u, v)$, normally represented $u \to v$, where $u$ is a string over $\Gamma$ and $v = v'$ or $v = v'\delta$, $v'$ being a string over $\Gamma \times (\{here, out\} \cup \{in_i : i = 1, \ldots, q\})$.
- $\rho_i$ is a strict partial order over $R_i$, for each $i = 1, \ldots, q$, establishing a priority relation between rules of $R_i$.
- $o_\Pi$ is a natural number between 1 and $q$ indicating the output membrane of the system.

A *basic* P *system with input* of grade $q \geq 1$ is a tuple $(\Pi, \Sigma, i_\Pi)$, where:

- $\Pi$ is a basic P system with working alphabet $\Gamma$, with $q$ membranes labeled by $1, \ldots, q$ and initial associated multisets $\mathcal{M}_1, \ldots, \mathcal{M}_q$, respectively.
- $\Sigma$ is an (input) alphabet strictly contained in $\Gamma$.
- $\mathcal{M}_i$ is a multiset over alphabet $\Gamma \setminus \Sigma$, for each $i = 1, \ldots, q$.
- $i_\Pi$ is the label of a distinguished membrane representing the input membrane.

To describe the *semantics* of the model it is necessary to introduce the concept of a configuration of the system, and the notion of computation of the same.

A *configuration* of a basic P system $\Pi = (\Gamma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, (R_1, \rho_1), \ldots, (R_q, \rho_q), m)$ is a tuple $\left(\mu', M_{i_1}, \ldots, M_{i_t}\right)$ such as:

- $\mu'$ is the membrane structure obtained from $\mu$ when some membranes different to those labeled by $i_1, \ldots, i_t$ are eliminated. The skin membrane can not be eliminated.
- $M_{i_j}$ is a finite multiset over $\Gamma$, for each $j = 1, \ldots, t$.

The *initial configuration* of the basic P system, $\Pi = (\Gamma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, (R_1, \rho_1), \ldots, (R_q, \rho_q), m)$, is the tuple $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_q)$.

Let $(\Pi, \Sigma, i_\Pi)$ be a basic P system with input. Let $\Gamma$, $\mu$ and $\mathcal{M}_1, \ldots, \mathcal{M}_q$ be the working alphabet, the membrane structure and the initial multisets of $\Pi$. Let $m$ be a multiset over the input alphabet $\Sigma$. Then the initial configuration of the system with input $m$ is $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} + m, \ldots, \mathcal{M}_q)$.

The application of a rule $u \to v$ associated with a membrane with label $i$ presented in a configuration is realized as follows: the objects in $u$ are eliminated from the membrane with label $i$ (then, this membrane must contain sufficient objects in each membrane to apply this rule); then, for each pair $(s, out) \in v$ an object $s \in \Gamma$ is included in the parent membrane of $i$ (or it leaves the system if $i$ is the skin); for each $(s, here) \in v$ an object $s \in \Gamma$ is added to the membrane $i$; for each pair $(s, in_j) \in v$ an object $s \in \Gamma$ is introduced in the membrane $j$ (if the membrane $j$ is not a child membrane of $i$, then the rule can not be applied); at the end, if $\delta \in v$, then the membrane $i$ is dissolved; i.e., it is eliminated from the membrane structure, passing its objects to the first ascendent membrane not dissolved (the skin can not be dissolved).

In an other way, we interpret the priority relation between rules in a *strong sense*: this relation forbids the application of a rule if there is another rule with greater priority.

Given two configurations, $C$ and $C'$, of $\Pi$, we say that $C'$ is obtained from $C$ in a transition step, $C \Rightarrow_\Pi C'$, if the second is the result of applying in parallel (simultaneously) a maximal subset of evolution rules associated with all the membranes that appear in the membrane structure to configuration $C$. This multiset also indicates the number of times that each rule is applied. This subset must be maximal in the sense that when the application of rules is ended, any object that could evolve and by rule must not appear in any membrane of the system. Bearing in mind that for a configuration, usually there is more than one possible multiset of applicable rules. The transition steps are realized in a non-deterministic manner, that is a configuration of the system can have more than one following configuration.

A computation $\mathcal{C}$ of $\Pi$ is a succession (finite or infinite) of configurations, $\{C_i\}_{i<r}$, such that

- $C_0$ is the initial configuration of $\Pi$.
- $C_i \Rightarrow_\Pi C_{i+1}$, for all $i < r - 1$.
- Or $r \in \mathbb{N}^+$ (i.e., is a natural number distinct from zero) and there does not exist any rule that can be applied in any membrane of $C_{r-1}$ (in this case $\mathcal{C}$ is an halting computation, and it has realized $|\mathcal{C}| = r - 1$ steps and $C_{r-1}$ is its *stopped configuration*), or $r = \infty$ (in this case $\mathcal{C}$ is not an halting computation).

We say a computation $\mathcal{C}$ is *successful* if it is a halting computation and the output membrane $o_\Pi$ appears in $C_{r-1}$ as an elementary membrane.

Given a successful computation of $\Pi$, the output of this computation is codified by the content of the output membrane, for example the output of this computation could be defined as the size of the associated multiset of the output membrane of the system in its halting configuration. Then, a basic P system is a machine that generates the set of natural numbers, $N(\Pi)$, formed by the output of all successful computations of this system.

## 4. Calculating homology groups

Homology is a powerful topological invariant, which characterizes an object by its $p$-dimensional holes. Intuitively the 0-dimensional holes can be seen as connected components, 1-dimensional holes can be seen as tunnels and 2-dimensional holes as cavities. For example, the torus contains one 0-dimensional hole, two 1-dimensional holes (each of them are an edge cycle) and one 2-dimensional hole (the cavity enclosed by the entire surface of the torus).

In the following, we calculate homology groups, $H_0$ and $H_1$, of 2D digital images. We can encode images with multiple pixels forming a network of points of $\mathbb{N}^2$. Moreover, we suppose that each pixel is associated with one of the two possible colors, black or white. A black (or white) pixel in position $(i, j)$ in the image is encoded as an object $B_{ij}$ (or $W_{ij}$).

$H_0$ is the number of connected components formed by black pixels and $H_1$ is the number of the holes created by black pixels, that is the number of connected components of white pixels surrounded by black pixels.

### 4.1. A family of P systems to obtain $H_0$

The $H_0$ *problem* is the following: *Given a digital 2D image with pixels of two colors, black and white, determine the number of black connected components (0-dimensional holes) included in the image.*

Next, we shall give some notions about to prove that the $H_0$ problem can be solved in a logarithmic time with respect to the input data by a family of basic P systems. The functioning of a system of the family consists of the following stages:

1. *Removing stage*: We can divide this stage in two parts working in parallel:
   (a) Remove leaves of graph $G_B$, e.g. the single points of the black connected components.
   (b) Reduce the size of graph $G_B$.
2. *Output stage*: The system sends to the environment the right answer according to the results of the previous stage.

We shall define a family $\boldsymbol{\Pi_0} = \{\Pi(i) : i \in \mathbb{N}\}$ such that each system $\Pi(n)$ will solve all instances of graphs with $n^2$ pixels, provided that the appropriate input multiset is given.

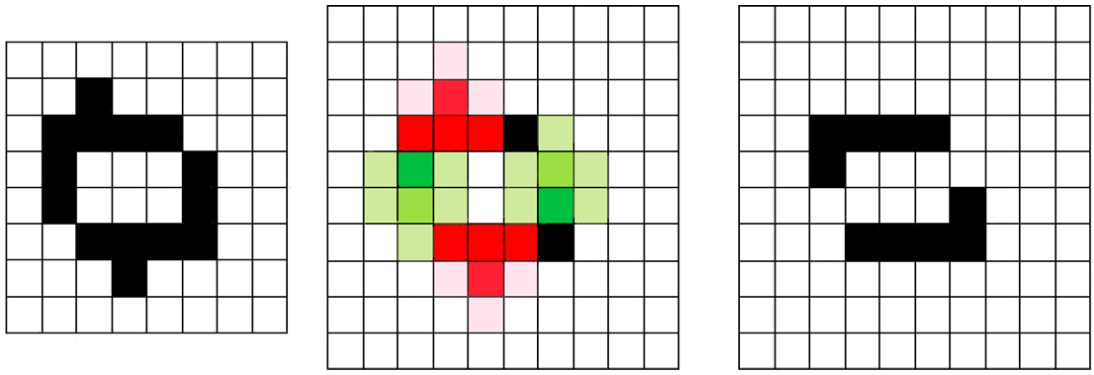

**Fig. 1.** Cutting branches of two black connected components.

We want to know the number of connected components of $G_B$. We will construct a basic P system for each image with $n^2$ pixels ($n \in \mathbb{N}$) where three types of elements appear: $B_{ij}$ encodes the black pixels, $W_{ij}$ encodes the white pixels and $G_{ij}$ encodes black connected components. One object $C$ is sent to the output membrane for each $G_{ij}$ that appear in the system.

*4.1.1. Definition of the family*

Next, we present a family of basic P systems where the initial configuration has one membrane. We shall address the resolution via an parallel algorithm, which consists of the following:

For each $n \in \mathbb{N}$ we consider the basic P system

$$\Pi_0(n) = (\Gamma, \Sigma, \mu, \mathcal{M}_0, \mathcal{M}_1, (R_0, \rho_0), (R_1, \rho_1), i_\Pi, o_\Pi)$$

defined as follows

(a) $\Gamma = \Sigma \cup \{G_{ij} : 1 \leq i, j \leq n\} \cup \{C\}$,
(b) $\Sigma = \{B_{ij}, W_{ij} : 1 \leq i, j \leq n\}$,
(c) $\mu = [[\ ]_1]_0$,
(d) $M_0 = \emptyset, M_1 = \{W_{ij} : (i = 0 \wedge 1 \leq j \leq n) \vee (i = n+1 \wedge 1 \leq j \leq n) \vee (j = 0 \wedge 1 \leq i \leq n) \vee (j = n+1 \wedge 1 \leq i \leq n)\}$,
(e) $R_0 = \emptyset, R_1$ is the following set of rules:

1.
$$\begin{array}{ccc} & W & K \\ W & B & B \\ & W & B \end{array} \rightarrow \begin{array}{ccc} & W & K \\ W & W & B \\ & W & B \end{array}$$

where $K = B$ or $K = W$. There exist 8 rules of this type depending the position of the black pixels (north, south, east, west) and the value of $K$. Formally, one of these rules is written as follows:

$$W_{i-1j}W_{ij-1}B_{ij}W_{ij+1}K_{i+1j-1}B_{i+1j}B_{i+1j+1} \rightarrow W_{i-1j}W_{ij-1}W_{ij}W_{ij+1}K_{i+1j-1}B_{i+1j}B_{i+1j+1}$$

for $1 \leq i, j \leq n-1$. The rest of the rules of this system can be formally written in a similar way to this rule.

2.
$$\begin{array}{ccc} & W & W \\ W & B & B \\ & W & W \end{array} \rightarrow \begin{array}{ccc} & W & W \\ W & W & B \\ & W & W. \end{array}$$

There exists 2 rules of this type depending the position of the white pixel in the center line with respect to two black pixels (left and right).

The two first types of rules eliminate the *simple points* of the black connected components, e.g. the black pixels that can be eliminated without dividing the connected components of the $G_B$, as we can see in Fig. 1.

The rules of types from 3 to 6 are used to reduce the dimensions of the black connected components with white pixels inside them, as we can see in Fig. 2 (left up square to rules of type 3, left down square to rules of type 4, right up square to rules of type 5 and right down square to rules of type 6).

3.
$$\begin{array}{ccc} & W & \\ W & B & B \\ & B & B \end{array} \rightarrow \begin{array}{ccc} & W & \\ W & W & B \\ & B & B. \end{array}$$

There are 4 rules of this type depending the position of the black pixels (north, south, east, west).

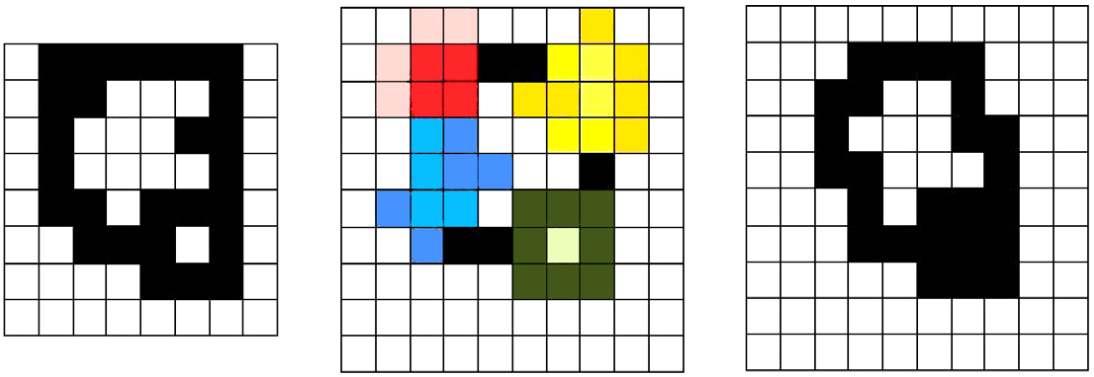

**Fig. 2.** Reducing black connected components with rules of types 5–8.

4.

$$\begin{array}{ccccc} & W & & & \\ W & B & B & B & \\ & B & W & W & \end{array} \quad \rightarrow \quad \begin{array}{cccc} & W & & \\ W & W & B & B \\ & B & B & W \\ & & W & \end{array}$$

There exists 4 rules of this type depending of the orientation of the black pixels. But, we only take two rules. We consider the left part of the rules. We take two rules where the black pixels are in the northeastern corner and the southeastern corner, respectively.

5.

$$\begin{array}{cccc} W & W & W & \\ B & B & B & W \\ B & W & B & \\ & W & & \end{array} \quad \rightarrow \quad \begin{array}{cccc} W & W & W & \\ B & W & W & W \\ B & B & B & \\ & W & & \end{array}$$

There exists 8 rules of this type depending of the orientation of the black pixels and the position of the white pixel in the second line with respect to the black pixels.

6.

$$\begin{array}{ccc} B & B & W \\ B & W & B \\ B & B & B \end{array} \quad \rightarrow \quad \begin{array}{ccc} B & B & W \\ B & B & B \\ B & B & B. \end{array}$$

There exists 8 rules of this type depending of the orientation of the white pixels with respect to the position of the black pixels.

7.

$$\begin{array}{cccc} & W & W & \\ W & B & B & W \\ W & B & B & W \\ & W & W & \end{array} \quad \rightarrow \quad \begin{array}{cccc} & W & W & \\ W & B & W & W \\ W & W & W & W \\ & W & W & \end{array}$$

8.

$$\begin{array}{ccc} & W & \\ W & B & W \\ & W & \end{array} \quad \rightarrow \quad \begin{array}{ccc} & W & \\ W & G & W \\ & W & \end{array}$$

9.

$$G \rightarrow W\,(C, out).$$

This first type of rules sends an object $C$ to the skin membrane and removes the object $G$ and adds the object $W$ to the membrane of the system.

(f) $\rho_0 = \rho_1 = \emptyset$,
(g) $i_\Pi = 1$,
(h) $o_\Pi = 0$.

*4.1.2. Overview of the computation*

Given an image whose size is $n^2$, there exists a system $\Pi_0(n)$ which works in a parallel manner as follows: First, the system eliminates branches of the black connected components that appear in the image using at most 4 steps. For this, the system uses rules of type 1 and 2. Secondly, the system reduces the size of the black connected components from four directions: north, south, east, west. The system takes the rules of types 3–7 to realize this task. The system needs a logarithmic number of steps with respect to the size of the biggest black connected component to reduce each component to a single black pixel. In fact, the system realizes these task simultaneously. But, it is better if we explain each task separately. The number of rules working in a computation step depends on the size of the black connected components in the image. Finally, it uses rules of type 8 to identify the connected component and rules of type 9 to send one object $C$ for each component to the membrane 0.

*4.1.3. Sketch of the verification*

Next, we show in an informal way that the family built above solves the $H_0$ problem in logarithmic time with respect to the input data. A formal proof would be too long to show in this paper.

Let $I$ be a digital image 2D with two colors, black and white, and $|I| = n^2$. Let $\Pi_0(n)$ be a basic P system of the family $\Pi_0$.

**Proposition 1.** *For each black connected component in I the application of rules of types* 1–7 *does not change the number of black connected components*

*Proof sketch.* It is easy to verify manually that each of the rules of types 1–7 cannot cause a black connected component to split into two or more separate components nor cause separate components to join together.

Now, we confirm that the systems of the family reduce all these components to only one black pixel. But first, we give some previous results.

**Lemma 2.** *For each convex black connected component,* $\mathbb{C}$ *of I, the application of the first six types of rules reduce* $\mathbb{C}$ *to a single pixel in a logarithmic time with respect to the size of* $\mathbb{C}$.

*Proof sketch.* We reduce $\mathbb{C}$ in a parallel manner using the following methods:

- *Distant corners*: We consider two corners of a black connected component that are in the same row or column and a distance of more than 1 pixel apart. We use rules of type 3 when we have compact corners. We eliminate one pixel in one (or both) corner(s) (see the top left corner of Fig. 2). If the black connected components are one pixel thick (a stretch corner) the rules of type 4 move the corner pixel one step closer to the center of the component (see the top right corner of Fig. 2).
- *Near corners*: Again, we consider two corners in the same row or column. If the distance between the two corners is equal to one pixel we apply rules of type 5 which reduce the size of the black connected components. In the case where three corners are closed we can not apply type 5 rules. So instead we take the rules of type 6 to compact this part of the black component.
- *Single points*: We use the rules of types 1 and 2 to eliminate branches of the graph $G_B$ as we can see in the Fig. 1.

We illustrate with an example image containing all white pixels except for the black outline of a square with sides 1 pixel thick. Step by step, the rules of the system increase the number of corners of the black connected component (by eroding the corners). So the number of rules that can be applied to each configuration increase until the corners are very near. We can work all the sides of the component at the same time. But, the rules of type 4 only work with the north and east sides of the square. So, we avoid non halting computations. Then, we reduce each black connected component to a single pixel in linear time with respect to the diameter of this component.

**Lemma 3.** *For each concave black connected component,* $\mathbb{C}$ *of I, the application of the first six types of rules reduce* $\mathbb{C}$ *to a single pixel in a logarithmic time with respect to the size of* $\mathbb{C}$.

*Proof sketch.* We use again the methods showed in the previous lemma. But, while system erode the east borders[1] of the convex areas of each black connected component, it may also be growing from its west (southwestern or northwestern) borders of the concave areas. Then, system may duplicate the works in some areas of each black connected component. It is not a problem because system works in logarithmic time with respect to the size of the component.

The consequence of the two previous lemmas is the following results.

**Proposition 4.** *For each black connected component,* $\mathbb{C}$ *of I, the application of the first six types of rules reduce* $\mathbb{C}$ *to only one pixel in logarithmic time with respect to the size of this.*

**Proposition 5.** *We change each black connected component with a single pixel by one green pixel in a computation step.*

**Proof.** Obvious by application of rules 7. □

**Proposition 6.** *For each green pixel that appears in membrane* 1 *during the computation, we send an object C to the output membrane using only two computation steps.*

[1] In fact, system erodes the southeastern and northeastern borders by action of rule of type 4.

**Proof.** We apply one rule of type 8 for each object $G_{ij}$ $(1 \leq i, j \leq n)$ that appears in membrane 1 along the computation. Then, we obtain an object $C$ in membrane 1 for each object $G_{ij}$. In the next step, there is in the membrane 2 a number of objects $C$ equal to the number of green pixels that arrived in membrane 1 during the computation (by action of the rules of type 9). □

As a consequence of the above results, we can formulate the following.

**Theorem 7.** *Let $I$ be a digital 2D image with two colors, black and white. $\Pi_0(n)$ sends to the output membrane a number of objects $C$ equal to the number of black connected components in the image.*

### *4.2. A family of P systems to obtain $H_1$*

The $H_1$ *problem* is the following: *Given a digital 2D image with pixels of two colors, black and white, determine the number of connected components formed by white pixels inside of a black connected component (1-dimensional holes) including in the image.*

Next, we shall give some notions about how to prove that the $H_1$ problem can be solved in a logarithmic time with respect to the input data by a family of basic P systems. The functioning of a system of the family consists in the following stages:

1. *Removing stage*: We work in a similar way to the $H_0$ problem. We divide this stage into two parts working in parallel:
   (a) Remove leaves of graph $G_B$, e.g. the single points of the black connected components.
   (b) Reduce the size of graph $G_B$.
2. *Output stage*: The system sends to the environment the right answer according to the results of the previous stage.

We shall define a family $\boldsymbol{\Pi_1} = \{\Pi(i) : i \in \mathbb{N}\}$ such that each system $\Pi(n)$ will solve all instances of graphs with $n^2$ pixels, provided that the appropriate input multiset is given.

We want to know the number of connected components of $G_W$ included in one $G_B$. We will construct a basic P system for each image with $n^2$ pixels $(n \in \mathbb{N})$ where three types of elements appear: $B_{ij}$ encodes the black pixels, $W_{ij}$ encodes the white pixels and $G_{ij}$ encodes black connected components. One object $C$ is sent to the output membrane for each $G_{ij}$ that appears in the system.

#### *4.2.1. Definition of the family*

For each $n \in \mathbb{N}$ we consider the basic P system

$$\Pi_1(n) = (\Gamma, \Sigma, \mu, \mathcal{M}_0, \mathcal{M}_1, (R_0, \rho_0), (R_1, \rho_1), i_\Pi, o_\Pi)$$

defined as follows

(a) $\Gamma = \Sigma \cup \{G_{ij} : 1 \leq i, j \leq n\} \cup \{C\}$,
(b) $\Sigma = \{B_{ij}, W_{ij} : 1 \leq i, j \leq n\}$,
(c) $\mu = [[\ \ ]_1]_0$,
(d) $M_0 = \emptyset, M_1 = \{W_{ij} : (i = 0 \wedge 1 \leq j \leq n) \vee (i = n+1 \wedge 1 \leq j \leq n) \vee (j = 0 \wedge 1 \leq i \leq n) \vee (j = n+1 \wedge 1 \leq i \leq n)\}$,
(e) $R_0 = \emptyset, R_1$ is the following set of rules:

1.
$$\begin{array}{ccccc} & W & K & & W\ K \\ W & B & B & \to\ W & W\ B \\ & W & B & & W\ B \end{array}$$
2.
$$\begin{array}{ccccc} & W & W & & W\ W \\ W & B & B & \to\ W & W\ B \\ & W & W & & W\ W \end{array}$$
3.
$$\begin{array}{ccccc} & W & & & W \\ W & B & B & \to\ W & W\ B \\ & B & B & & B\ B \end{array}$$
4.
$$\begin{array}{ccccc} & W & & & W & & \\ W & B & B & B & W & W & B & B \\ & B & W & W & \to & B & B & W \\ & & W & & & & W & \end{array}$$
5.
$$\begin{array}{ccccc} W & W & W & & W & W & W & \\ B & B & B & W & B & W & W & W \\ B & W & B & \to & B & B & B & \\ & W & & & & W & & \end{array}$$

6.
$$\begin{matrix} B & B & K \\ B & W & B \\ B & B & B \end{matrix} \rightarrow \begin{matrix} B & B & K \\ B & G & B \\ B & B & B \end{matrix}$$
There exists 8 rules of this type depending on the orientation of the white pixels with respect to the position of the black pixels. When we find a hole(white connected component) with only one white pixel in membrane 1 system trade this white pixel against a green pixel with the membrane 0.

7.
$$G \rightarrow B\,(C, out).$$
This first type of rules sends an object $C$ to the skin membrane and removes the object $G$ and adds the object $B$ to the membrane of the system.

(f) $\rho_0 = \rho_1 = \emptyset$,
(g) $i_{\Pi} = 1$,
(h) $o_{\Pi} = 0$.

*4.2.2. Overview of the computation*

Given an image whose size is $n^2$, there exists a system $\Pi_1(n)$ which works in a parallel manner as follows: First, the system eliminates branches of the black connected components that appear in the image using at most 4 steps. For this, the system uses rules of type 1 and 2. Secondly, the system reduces the size of the hole from four directions: north, south, east, west. The system takes the rules of types 3 to 5 to realize this task. The system needs a logarithmic number of steps with respect to the size of the biggest hole to reduce each one. In fact, the system realizes these task simultaneously. But, it is better if we explain each task separately. The number of rules working in a computation step depending of the size of the holes in the image. Finally, it uses the rules of type 6 to identify the holes with only one white pixel and rules of type 7 to send an object $C$ for each hole to the membrane 0.

### 4.3. Complexity and necessary resources

Given that the size of the input data is $O(n^2)$, the amount of necessary resources to define the systems of our two families and the complexity of our solutions to calculate the homology groups can be observed in the following table:

| Homology groups 2D images problem | | |
|---|---|---|
| **Complexity** | **$H_0$** | **$H_1$** |
| Number of steps of a computation | $O(n)$ | $O(n)$ |
| **Necessary resources** | **$H_0$** | **$H_1$** |
| Size of the alphabet | $3n^2 + 1$ | $3n^2 + 1$ |
| Initial number of membranes | 2 | 2 |
| Initial number of objects | $4n - 3$ | $4n - 3$ |
| Number of rules | $O(n^2)$ | $O(n^2)$ |
| Upper bound for the length of the rules | 24 | 22 |

We can observe the necessary time to calculate the homology groups of a 2D digital image is logarithmic with respect to the input data ($\mathcal{O}(n)$). This is an improvement with respect to the algorithms development by Peltier et al. in [31], where they use irregular graphs pyramids with a time complexity of $\mathcal{O}(n^{5/3})$.

### 4.4. Examples

First, we are going to obtain the number of different black connected components for the images given in Fig. 3 using the system $\Pi_0(8)$.

After a logarithmic number of steps with respect to the input data, the output data appears in the output membrane of the system. This output is encoded in the images shown in Fig. 4.

Using the system from Section 4.2, in a logarithmic number of steps with respect to the input data, the number of white connected components inside of black connected components will appear in the output membrane. This is shown in Fig. 5 where the outputs (generated by each of the images in Fig. 3) encoded by the elements that appear in the output membrane. $H_1$ is given by the number of green pixels (objects $G$).

## 5. Segmenting digital images in constant time

In this section, we segment images based on edge-based segmentation. A region inside of a digital images is given by a set of adjacent pixels with the same colors. So, we can divide a digital image into regions, some of them with the same colors. We want to find boundaries of regions which are sufficiently different from each other. We define two families of

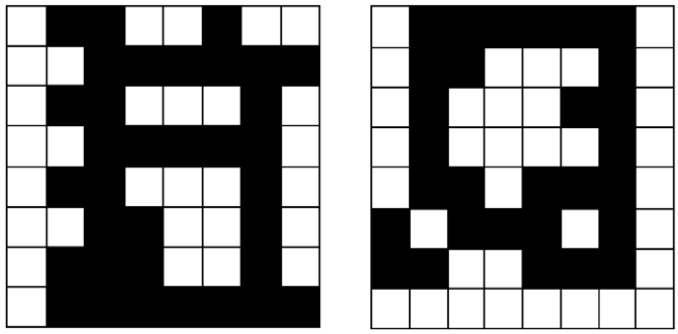

**Fig. 3.** Two input images to check.

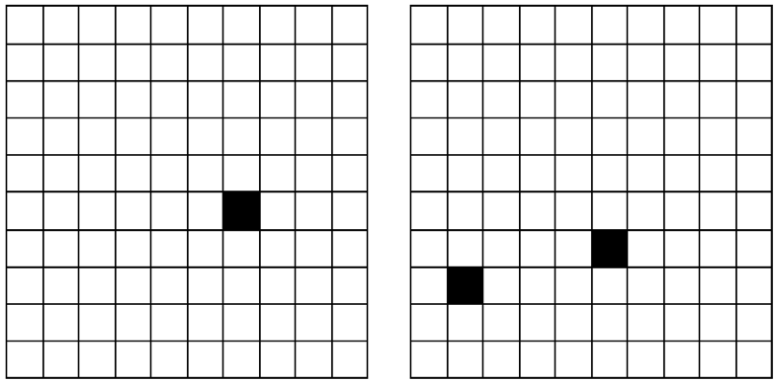

**Fig. 4.** Number of black connected components of the input images.

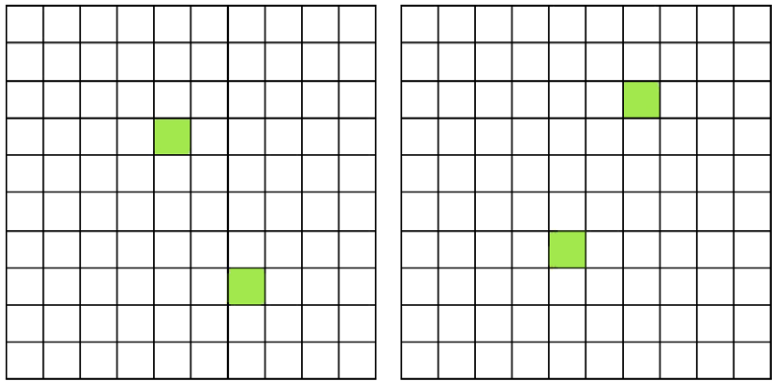

**Fig. 5.** Number of white holes (green pixels) of the images in Fig. 3. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

P systems for edge-based segmentation, one of them to segment 2D images and then adapt these systems to segment 3D images. We consider an alphabet of colors, $\mathcal{C}$, with an order. The result of the segmentation can be different depending on the order considered in $\mathcal{C}$.

### 5.1. A family of P systems for a 2D segmentation

The *edge-based segmentation of the 2D digital image problem (2D-ES problem)* is the following: *Given a digital 2D image with pixels of (possibly) different colors, obtain the boundaries of regions in that image.*

Next, we shall give some notions about to prove that the *2D-ES problem* can be solved in a constant time with respect to the input data by a family of basic P systems. The functioning of a system of the family consists of the following stages:

1. *First marking pixels stage*: We mark some of the pixels belong to a boundary between two regions (edge pixels).

2. *Second marking pixels stage*: We add pixels adjacent to two edge pixels with the same color as them and adjacent to another pixel with different color to the edge pixels.
3. *Output stage*: The system sends to the environment all the marked pixels. This stage works in parallel to the previous stages.

We shall define a family $\boldsymbol{\Pi_3} = \{\Pi(n, m) : n, m \in \mathbb{N}\}$ such that each system $\Pi_3(n, m)$ will solve all instances of graphs with $n^2$ pixels, provided that the appropriate input multiset is given.

We define a family of P systems to do the edge-based segmentation to 2D images.

*5.1.1. Definition of the family*

For each $n, m \in \mathbb{N}$ we consider the basic P system

$$\Pi_3(n, m) = (\Gamma, \Sigma, \mu, \mathcal{M}_0, \mathcal{M}_1, (R_0, \rho_0), (R_1, \rho_1), i_\Pi, o_\Pi),$$

defined as follows:

(a) $\Gamma = \Sigma \cup \{\bar{a}_{ij} : 1 \leq i \leq n,\ 1 \leq j \leq m\} \cup \{A_{ij} : A \in \mathcal{C},\ 1 \leq i \leq n,\ 1 \leq j \leq m\}$,
(b) $\Sigma = \{a_{ij} : a \in \mathcal{C},\ 1 \leq i \leq n,\ 1 \leq j \leq m\}$
(c) $\mu = [[\ \ ]_1]_0$,
(d) $\mathcal{M}_0 = \mathcal{M}_1 = \emptyset$,
(e) $R_0$ is the following set of rules:

1.

$$a_{ij}b_{kl} \to a_{ij}A_{ij}b_{kl}$$

for $a, b \in \mathcal{C},\ a < b,\ 1 \leq i, k \leq n,\ 1 \leq j, l \leq m$ and $(i, j),\ (k, l)$ are adjacent pixels.

These rules are used when the image has two adjacent pixels with different associated colors (border pixels). Then, the pixel with less associated color is marked and the system creates an object representing this marked pixel (edge pixel).

2.

$$\bar{a}_{ij}a_{ij+1}\bar{a}_{i+1j+1}b_{i+1j} \to \bar{a}_{ij}\bar{a}_{ij+1}A_{ij+1}\bar{a}_{i+1j+1}b_{i+1j}$$

for $a, b \in \mathcal{C}, a < b, 1 \leq i \leq n-1,\ 1 \leq j \leq m-1$.

$$\bar{a}_{ij}a_{i-1j}\bar{a}_{i-1j+1}b_{ij+1} \to \bar{a}_{ij}\bar{a}_{i-1j}A_{i-1j}\bar{a}_{i-1j+1}b_{ij+1}$$

for $a, b \in \mathcal{C}, a < b, 2 \leq i \leq n,\ 1 \leq j \leq m-1$.

$$\bar{a}_{ij}a_{ij+1}\bar{a}_{i-1j+1}b_{i-1j} \to \bar{a}_{ij}\bar{a}_{ij+1}A_{ij+1}\bar{a}_{i-1j+1}b_{i-1j}$$

for $a, b \in \mathcal{C}, a < b, 2 \leq i \leq n,\ 1 \leq j \leq m-1$.

$$\bar{a}_{ij}a_{i+1j}\bar{a}_{i+1j+1}b_{ij+1} \to \bar{a}_{ij}\bar{a}_{i+1j}A_{i+1j}\bar{a}_{i+1j+1}b_{ij+1}$$

for $a, b \in \mathcal{C}, a < b, 1 \leq i \leq n-1,\ 1 \leq j \leq m-1$.

The rules mark with a bar the pixels which are adjacent to two same color pixels and which were marked before, but with the condition that the marked objects are adjacent to another pixel with a different color. Moreover, an edge object representing the last marked pixel is created.

3.

$$A_{ij} \to (A_{ij}, in_2)$$

for $1 \leq i \leq n,\ 1 \leq j \leq m$.

This rule is used to send the edge pixels to the output membrane.

$R_1 = \emptyset$
(f) $\rho_0 = \rho_1 = \emptyset$
(g) $i_\Pi = 0$
(h) $o_\Pi = 1$.

*5.1.2. An overview of the computation*

Input objects $a_{ij}$ encoding the colored pixels from a 2D image are placed in the input membrane. Rules of type 1, in a parallel manner, identify the border pixels and create the edge pixels. These rules need 4 steps to operate. After the second step, rules of type 2 can be used in parallel with the type 1 rules. The system needs 4 steps to create the new edge objects adjacent each of the previous border pixels, as explained above. The system could apply the two first types of rules simultaneously in some configurations, but it always applies the same number of these types of rules since this number is given by edge pixels (we consider 4-adjacency). Finally, rules of type 3 are applied to edge pixels ($A_{ij}$) that are in membrane 1. So, with one more step we have all the edge pixels in the output membrane. Thus, we need at most only 9 steps to obtain an edge-based segmentation for an $n \times m$ image. Then, we can conclude we have given an efficient solution of the edge-segmentation problem for 2D images in a constant time.

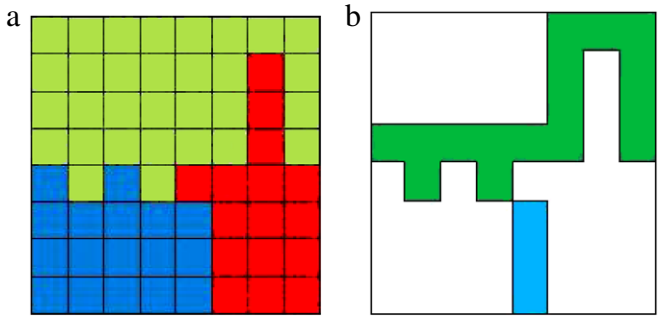


**Fig. 6.** (a) Input image; (b) Edge-segmentation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

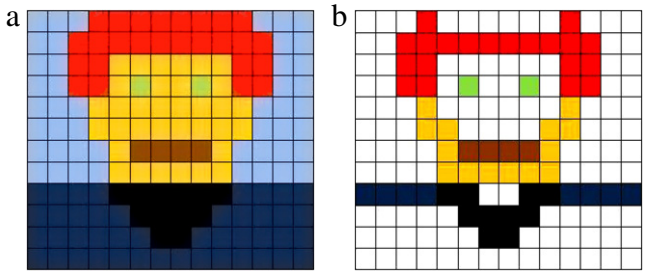


**Fig. 7.** (a) Input image; (b) Edge-segmentation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

*5.1.3. Complexity and necessary resources*

Given that the size of the input data is $O(n \cdot m)$ and $|\mathcal{C}| = k$, the amount of necessary resources for defining the systems of our family and the complexity of our problem can be observed in the following table:

| 2D-ES problem | |
|---|---|
| **Complexity** | |
| Number of steps of a computation | 9 |
| **Necessary resources** | |
| Size of the alphabet | $k \cdot n \cdot m$ |
| Initial number of membranes | 2 |
| Initial number of objects | 0 |
| Number of rules | $O(n \cdot m \cdot k^2)$ |
| Upper bound for the length of the rules | 9 |

*5.1.4. Examples*

First, we consider the $8 \times 8$ image given in Fig. 6(a), where three colors appear, green, blue and red. When we consider this order and apply system $\Pi(8, 8)$ we obtain the image 6(b).

Consider Fig. 7(a) ($12 \times 14$). The order of the colors in this case is the following: red, green, brown, orange, black, blue and light blue. Applying the system $\Pi_3(12, 14)$ we obtain the Fig. 7(b).

*5.2. A family of P systems for 3D segmenting*

In this section, the input data are voxels ($(i, j, k) \in \mathbb{N}^3$) encoded by the elements $a_{ijk}$, with $a \in \mathcal{C}$ ($|\mathcal{C}| = h$). The 3D-adjacency consider in this paper is the 6-adjacency.

The *edge-based segmentation of the 3D digital image problem (3D-ES problem)* is the following: *Given a digital 3D image with voxels of (possibly) different colors, obtain the boundaries of regions in that image.*

Next, we shall give some notions about to prove that the *3D-ES problem* can be solved in a constant time with respect to the input data by a family of basic P systems. The functioning of a system of the family consists of the following stages:

1. *First marking voxels stage*: We mark some of the voxels belong to a boundary between two regions (edge voxels).

2. *Second marking voxels stage*: We add voxels adjacent to two edge voxels with the same color as them and adjacent to other voxel with different color to the edge voxels.
3. *output stage*: The system sends to the environment all the marked voxels. This stage works in parallel to the previous stages.

We shall define a family $\boldsymbol{\Pi_4} = \{\Pi(n, m, l) : n, m, l \in \mathbb{N}\}$ such that each system $\Pi_4(n, m, l)$ will solve all instances of graphs with $n^2$ voxels, provided that the appropriate input multiset is given.

Then, we can define a family of basic P systems to do an edge-based segmentation to any 3D image.

*5.2.1. Definition of the family*

For each $n, m, l \in \mathbb{N}$ we consider the system

$$\Pi_4(n, m, l) = (\Gamma, \Sigma, \mu, \mathcal{M}_0, \mathcal{M}_1, (R_0, \rho_0), (R_1, \rho_1), i_\Pi, o_\Pi)$$

defined as follows

(a) $\Gamma = \Sigma \cup \{A_{ijk} : 1 \leq i \leq n,\ 1 \leq j \leq m,\ 1 \leq k \leq l,\ a \in \mathcal{C}\}$,
(b) $\Sigma = \{a_{ijk} : a \in \mathcal{C},\ 1 \leq i \leq n,\ 1 \leq j \leq m,\ 1 \leq k \leq l\}$
(c) $\mu = [[\ \ ]_1]_0$,
(d) $\mathcal{M}_0 = \mathcal{M}_1 = \emptyset$,
(e) $R_0$ is the following set of rules:

1.
$$a_{i_1j_1k_1}b_{i_2j_2k_2} \to \bar{a}_{i_1j_1k_1}A_{i_1j_1k_1}b_{i_2j_2k_2}$$
for $1 \leq i_1, i_2 \leq n$, $1 \leq j_1, j_2 \leq m$, $1 \leq k_1, k_2 \leq l$, $(i_1, j_1, k_1)$ and $(i_2, j_2, k_2)$ adjacent voxels and finally, $a, b \in \mathcal{C}$ with $a < b$.

These rules are used when an image has two adjacent border voxels. Then, the system creates an object representing the voxel with the less associated color (edge voxel).

2.
$$\bar{a}_{ijk}a_{ij+1k}\bar{a}_{i+1j+1k}b_{i+1jk} \to \bar{a}_{ijk}\bar{a}_{ij+1k}A_{ij+1k}\bar{a}_{i+1j+1k}b_{i+1jk}$$
for $a, b \in \mathcal{C}$, $a < b$, $1 \leq i \leq n-1,\ 1 \leq j \leq m-1$, $1 \leq k \leq l$.
$$\bar{a}_{ijk}a_{i-1jk}\bar{a}_{i-1j+1k}b_{ij+1k} \to \bar{a}_{ijk}\bar{a}_{i-1jk}A_{i-1jk}\bar{a}_{i-1j+1k}b_{ij+1k}$$
for $a, b \in \mathcal{C}$, $a < b$, $2 \leq i \leq n,\ 1 \leq j \leq m-1$, $1 \leq k \leq l$.
$$\bar{a}_{ijk}a_{ij+1k}\bar{a}_{i-1j+1k}b_{i-1jk} \to \bar{a}_{ijk}\bar{a}_{ij+1k}A_{ij+1k}\bar{a}_{i-1j+1k}b_{i-1jk}$$
for $a, b \in \mathcal{C}$, $a < b$, $2 \leq i \leq n,\ 1 \leq j \leq m-1$, $1 \leq k \leq l$.
$$\bar{a}_{ijk}a_{i+1jk}\bar{a}_{i+1j+1k}b_{ij+1k} \to \bar{a}_{ijk}\bar{a}_{i+1jk}A_{i+1jk}\bar{a}_{i+1j+1k}b_{ij+1k}$$
for $a, b \in \mathcal{C}$, $a < b$, $1 \leq i \leq n-1,\ 1 \leq j \leq m-1$, $1 \leq k \leq l$.
$$\bar{a}_{ijk}a_{ijk+1}\bar{a}_{i+1jk+1}b_{i+1jk} \to \bar{a}_{ijk}\bar{a}_{ijk+1}A_{ijk+1}\bar{a}_{i+1jk+1}b_{i+1jk}$$
for $a, b \in \mathcal{C}$, $a < b$, $1 \leq i \leq n-1,\ 1 \leq j \leq m$, $1 \leq k \leq l-1$.
$$\bar{a}_{ijk}a_{i-1jk}\bar{a}_{i-1jk+1}b_{ijk+1} \to \bar{a}_{ijk}\bar{a}_{i-1jk}A_{i-1jk}\bar{a}_{i-1jk+1}b_{ijk+1}$$
for $a, b \in \mathcal{C}$, $a < b$, $2 \leq i \leq n,\ 1 \leq j \leq m$, $1 \leq k \leq l-1$.
$$\bar{a}_{ijk}a_{ijk+1}\bar{a}_{i-1jk+1}b_{i-1jk} \to \bar{a}_{ijk}\bar{a}_{ijk+1}A_{ijk+1}\bar{a}_{i-1jk+1}b_{i-1jk}$$
for $a, b \in \mathcal{C}$, $a < b$, $2 \leq i \leq n,\ 1 \leq j \leq m$, $1 \leq k \leq l-1$.
$$\bar{a}_{ijk}a_{i+1jk}\bar{a}_{i+1jk+1}b_{ijk+1} \to \bar{a}_{ijk}\bar{a}_{i+1jk}A_{i+1jk}\bar{a}_{i+1jk+1}b_{ijk+1}$$
for $a, b \in \mathcal{C}$, $a < b$, $1 \leq i \leq n-1,\ 1 \leq j \leq m$, $1 \leq k \leq l-1$.
$$\bar{a}_{ijk}a_{ij+1k}\bar{a}_{ij+1k+1}b_{ijk+1} \to \bar{a}_{ijk}\bar{a}_{ij+1k}A_{ij+1k}\bar{a}_{ij+1k+1}b_{ijk+1}$$
for $a, b \in \mathcal{C}$, $a < b$, $1 \leq i \leq n,\ 1 \leq j \leq m-1$, $1 \leq k \leq l-1$.
$$\bar{a}_{ijk}a_{ijk-1}\bar{a}_{ij+1k-1}b_{ij+1k} \to \bar{a}_{ijk}\bar{a}_{ijk-1}A_{ijk-1}\bar{a}_{ij+1k-1}b_{ij+1k}$$
for $a, b \in \mathcal{C}$, $a < b$, $1 \leq i \leq n,\ 1 \leq j \leq m-1$, $2 \leq k \leq l$.
$$\bar{a}_{ijk}a_{ij+1k}\bar{a}_{ij+1k-1}b_{ijk-1} \to \bar{a}_{ijk}\bar{a}_{ij+1k}A_{ij+1k}\bar{a}_{ij+1k-1}b_{ijk-1}$$

for $a, b \in \mathcal{C}$, $a < b$, $1 \leq i \leq n,\ 1 \leq j \leq m-1, 2 \leq k \leq l$.

$$\bar{a}_{ijk}a_{ijk+1}\bar{a}_{ij+1k+1}b_{ij+1k} \rightarrow \bar{a}_{ijk}\bar{a}_{ijk+1}A_{ijk+1}\bar{a}_{ij+1k+1}b_{ij+1k}$$

for $a, b \in \mathcal{C}$, $a < b$, $1 \leq i \leq n,\ 1 \leq j \leq m-1, 1 \leq k \leq l-1$.

The rules marked with a bar are the voxels which are adjacent to two identically colored voxels and which were marked before, but with the condition that the marked objects are adjacent to other voxels with a different color. Moreover, an edge object representing the last marked voxel is created.

3.

$$A_{ijk} \rightarrow (A_{ijk}, in_0)$$

for $1 \leq i \leq n,\ 1 \leq j \leq m, 1 \leq k \leq l$.

These rules are used to send the edge voxels to the output membrane.

$R_1 = \emptyset$

(f) $\rho_0 = \rho_1 = \emptyset$

(g) $i_\Pi = 1$

(h) $o_\Pi = 0$.

*5.2.2. An overview of the computation*

Input objects $a_{ijk}$ encode the colored voxels from a 3D image that are placed in the input membrane. The system begins to work on these objects. Rules of type 1, in a parallel manner, identify the border voxels and create the edge voxels. These rules needs at most 6 steps to mark all the possible border voxels. From the second computation step, the system can apply the rules of type 2 in a parallel manner with the rules of type 1. Using at most 12 other steps we create the new edge voxels adjacent to two previous border voxels as we explained above. The system could apply the first two types of rules simultaneously in some configurations, but it always applies the same number of these two types of rules because this number is given by edge voxels (we consider 6-adjacency). Finally, the third type of rules are applied in the following step and edge voxels ($A_{ijk}$) are sent to the skin membrane. So, with one step more we will have all the edge voxels in the output membrane. We only need 18 steps to obtain an edge-based segmentation for a 3D image. Then, we can conclude the edge-segmentation problem for 3D images is resolved here in a constant time with respect to the number of voxels.

*5.2.3. Complexity and necessary resources*

Taking account that the size of the input data is $O(n \cdot m \cdot k)$ and $|\mathcal{C}| = h$, the amount of necessary resources for defining the systems of our family and the complexity of our problem can be observed in the following table:

| 3D-ES problem | |
|---|---|
| **Complexity** | |
| Number of steps of a computation | 19 |
| **Necessary resources** | |
| Size of the alphabet | $2 \cdot k \cdot n \cdot m \cdot h$ |
| Initial number of membranes | 2 |
| Initial number of objects | 0 |
| Number of rules | $O(n \cdot m \cdot k \cdot h^2)$ |
| Upper bound for the length of the rules | 9 |

## 6. Conclusions and future work

We have showed in this paper how we can efficiently obtain the homological groups to 2D images using P systems. Moreover, we have computed the edge-segmentation of 2D digital images and we have described the family of P systems to realize the edge-segmentation of 3D images.

In the future we wish to use P systems to obtain more homological information: homology groups, spanning trees, homology gradient vector field, etc. Until now, this homological information is typically calculated using sequential algorithms or, in the best case, partially parallel algorithms. Then, we could use P systems in some research fields where the homological information is very important: 2D, 3D and 4D images, robotics,etc. Also we wish to use P system simulators to check these systems and create new programs to obtain better solutions, because the actual simulators are educational software. Finally, another possible interesting research direction from a computational point of view is to use some variants of P systems such as Tissue-like P systems or Spiking Neural P systems to obtain topological information.

## Acknowledgements

The first author acknowledges the support of the project "Computational Topology and Applied Mathematics" PAICYT research project FQM-296. The second author acknowledges the support of the project of Excellence of the Junta de Andalucía, grant P08-TIC-04200. The third author acknowledges the support of the project MTM2006-03722 of the Ministerio Español de Educación y Ciencia and the project PO6-TIC-02268 of Excellence of Junta de Andalucía.

## References

[1] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.
[2] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics 5 (1943) 115–133.
[3] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, Bulletin of Mathematical Biology 49 (1987) 737–759.
[4] L.M. Adleman, Molecular computations of solutions to combinatorial problems, Science 226 (1994) 1021–1024.
[5] Gh. Păun, Membrane Computing. An Introduction, Springer-Verlag, Berlin, 2002.
[6] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter, Molecular Biology of the Cell, 4th edition, Garland Science, 2002.
[7] Gh. Păun, Computing with membranes, Journal of Computer and System Sciences 61 (1) (2000) 108–143.
[8] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, A fast P system for finding a balanced 2-partition, Soft Computing 9 (9) (2005) 673–678.
[9] D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, A logarithmic bound for solving Subset Sum with P systems, in: Lecture Notes in Computer Science, vol. 4860, 2007, pp. 257–270.
[10] C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón, Tissue P systems, Theoretical Computer Science 296 (2003) 295–326.
[11] Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, Tissue P System with cell division. Second Brainstorming Week on Membrane Computing, Sevilla, Report RGNC 01/2004, (2004), pp. 380–386.
[12] D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, A linear time solution to the partition problem in a cellular tissue-like model, Journal of Computational and Theoretical Nanoscience 5 (7) (2010) 884–889.
[13] D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, Solving Subset Sum in linear time by using tissue P systems with cell division, in: J. Mira, J.R. Alvarez (Eds.), Bio-inspired Modeling of Cognitive Tasks, in: Lecture Notes in Computer Science, vol. 4527, 2007, pp. 170–179.
[14] D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, A uniform family of tissue P systems with cell division solving 3-COL in a linear time, Theoretical Computer Science 404 (1–2) (2008) 76–87.
[15] J. Chao, J. Nakayama, Cubical singular simples model for 3D objects and fast computation of homology groups, in: Proc. ICPR'96, IEEE, 1996, pp. 190–194.
[16] R. Ceterchi, M Madhu, G. Paun, K.G. Subramanian, Array-rewriting P systems, Natural Computing 2 (2003) 229–249.
[17] P.H. Chandra, K.G. Subramanian, On picture arrays generated by P systems, in: R. Freund, G. Lojka, M. Oswald, Gh. Paun (Eds.) Pre-Proc. of Sixth International Workshop on Membrane Computing, WMC6, Vienna, June 18–21, 2005, pp. 282–288.
[18] M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, F.J. Romero–Campero, A linear solution for QSAT with Membrane Creation, in: Lecture Notes in Computer Science, vol. 3850, 2006, pp. 241–252.
[19] Gh. Păun, Computing with membranes: attacking **NP**-complete problems, in: I. Antoniou, C. Calude, M.J. Dinneen (Eds.), In Unconventional Models of Computation, UMC'2K, 2000, pp. 94–115.
[20] A. Hatcher, Algebraic Topology, Cambridge Univ. Press, Cambridge, UK, 2001.
[21] R. González-Díaz, P. Real, On the cohomology of 3D digital images, Discrete Applied Mathematics 147 (2005) 245–263.
[22] R. González-Díaz, B. Medrano, P. Real, J. Sánchez, Reusing Integer Homology Information of Binary Digital Images, in: Lecture Notes in Computer Science, vol. 4245, 2006, pp. 199–210.
[23] H. Molina-Abril, P. Real, Advanced Homological Information on 3D Digital volumes, in: Lecture Notes in Computer Science, vol. 5342, 2008, pp. 361–371.
[24] P. Real, H. Molina-Abril, W. Kropatsch, Homological tree-based strategies for image analysis, in: Computer Analysis and Image Patterns, CAIP, 2009.
[25] R. González-Díaz, P. Real, Computation of cohomology operations of finite simplicial complexes, Homology Homotopy and Applications 2 (2003) 83–93.
[26] R. González-Díaz, P. Real, Towards Digital Cohomology, in: Lecture Notes in Computer Science, vol. 2886, 2003, pp. 92–101.
[27] R. González-Díaz, B. Medrano, P. Real, J. Sanchez-Pelaez, Algebraic Topological Analysis of Time-sequence of Digital Images, in: Lecture Notes in Computer Science, vol. 3718, 2005, pp. 208–219.
[28] P. Real, An algorithm computing homotopy groups, Mathematics and Computers in Simulation 42 (4–6) (1996) 461–465.
[29] P. Real, Homological perturbation theory and associativity, in: Homology, Homotopy and Applications, 2000, pp. 51–88.
[30] F. Sergeraert, The computability problem in algebraic topology, Advances in Mathematics 104 (1994) 1–29.
[31] S. Peltier, A. Ion, Y. Haxhimusa, W.G. Kropatsch, G. Damiand, Computing homology group generators of images using irregular graph pyramids, in: Lecture Notes in Computer Science, vol. 4538, 2007, pp. 283–294.
[32] Linda G. Shapiro, George C. Stockman, Computer Vision, 2001.