Homological processing of biomedical digital images: automation and certification*

Jónathan Heras, Gadea Mata, María Poza, and Julio Rubio

Department of Mathematics and Computer Science of University of La Rioja
{jonathan.heras, gadea.mata, maria.poza, julio.rubio}@unirioja.es

Abstract. In this paper a methodology to extract and compute homological information from biomedical images is proposed; automating some processes, up to now, manually done. The main features of our approach are the usage of several programming languages (*Java*, *Common Lisp* and *Haskell*) and the application of formal methods (namely theorem provers) to verify the correctness of some of the automated process. As case study to test the suitability of our approach, we have applied it to measure the number of synapses of a neuron.

1 Introduction

In this work a methodology to deal with digital images is presented. The main steps of our proposal are the following ones. First of all, we detect what kind of homological information is needed in a concrete problem of image processing. Secondly, the image is manipulated to get an image where the topological information is as explicit as possible. Afterwards, the size of the data is reduced, ensuring that no relevant information is lost during the process. Eventually, a computer algebra program is applied to compute the homological invariants of the image.

The objectives of this kind of research are twofold: (1) automating some tasks made up to now manually (or semi-automatically) by biologists and other experimental scientists: tracing, marking, counting, and so on; and, (2) verifying the correctness of the automated process.

Even if the latter objective could be considered as excessive, in fields where the accuracy standards are not so high as in mathematics or theoretical computer science, it is necessary to stress that the simplifications done by experimental scientists are based on solid (even if heuristic) previous experience. If they must trust computer programs, it is convenient to produce them in a reliable manner, in such a way that scientists could be confident of the results mechanically obtained.

In this paper we describe an instantiation of this methodology with the following features. The processed images are related to the synaptical structure in neurons [2]; and the topological invariant to be computed is the number of

* Partially supported by Ministerio de Ciencia e Innovación, project MTM2009-13842-C02-01, and by European Community FP7, STREP project ForMath, n. 243847.

connected components (useful to determine the evolution of the density of the occurrence of synapses in neurons, under the effect of some drugs).

The digital images obtained experimentally are handled by means of the *ImageJ Java* environment [24], producing a bitmap file where connected components should be counted. From the previous file an incidence matrix is constructed, which is processed through a *Haskell* program [17], to obtain a smaller matrix with the same homological information (we are using here Discrete Morse Theory, as is explained in [25]). The correctness of these *Haskell* programs are being analyzed by using the *Coq* proof assistant [5,3], and more specifically the *SSReflect* environment [9]. The matrices obtained by means of the *Haskell* programs are given as input to the *fKenzo* system [10,14], a user interface for the *Kenzo* program [7], which actually computes the homological information.

The rest of this paper is organized as follows. First of all, our concrete biomedical problem is stated in Section 2. The role played by Algebraic Topology in this problem is presented in Section 3. The process which is followed to reduce digital images keeping their homological information in a reliable manner is introduced in Section 4. Our methodology, as a whole, to automating and certifying the study of biomedical images is presented in Section 5. The paper ends with a section of Conclusions and Further work and the bibliography.

2 Digital Imaging for Synapse Counting

Synapses are the points of connection between neurons. The synapses have two sides: *presynaptic* and *postsynaptic*, these names indicate the direction of information flow, which is from "pre" to "post". The relevance of synapses comes from the fact that they are related to the computational capabilities of the brain.

The possibility of changing the number of synapses may be an important asset in the treatment of neurological diseases, such as Alzheimer, see [27]. Therefore, we can claim that an efficient, reliable and automatic method for counting synapses is instrumental in the study of tools which allow one to increase the number of synaptical contacts.

2.1 Counting synapses manually

Let us briefly explain the methodology which was followed, up to now, to count the number of synapses in a neuron. This process can be split into two parts: (1) obtaining several images of the same neuron in a concrete moment, and, (2) processing such images to count the number of synapses. We are mainly interested in the latter part.

In a nutshell, the process to obtain the images is as follows. Hippocampal neuron cultures are permeabilized and treated with two different primary markers, *bassoon* and *synapsin*. These antibodies recognize specifically the presynaptic structures. Then is necessary a secondary antibody couple attached to fluochromes (red and green) making this two synaptic proteins visible under the fluorescence microscope. Finally, the two markers are photographed in two gray scale images as can be seen in Figure 1. A more detailed description of this process is given in [6].

After obtaining the two images, they are processed in order to count the synapses of the neuron. To this aim, the image processing program *ImageJ* [24] is used.

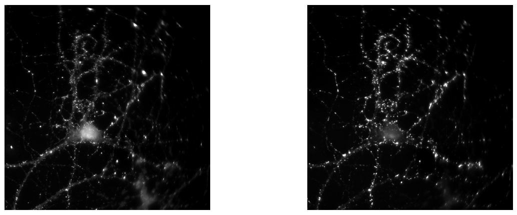


Fig. 1. Neuron with bassoon and synapsin antibody markers

The first step consists in overlapping the images with the two markers using the red channel for the image with the bassoon antibody marker and the green channel for the image with the synapsin one. The result, as can be seen in Figure 2, is an image with red, green and yellow points. In our case, we are interested in the *colocalization* of red and green point, that is yellow points, which are the candidates to be the synapses of the neuron.

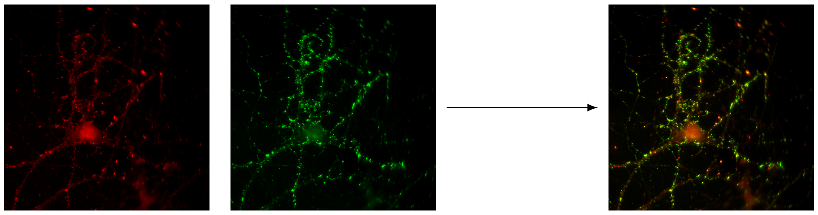


Fig. 2. Overlapping of both markers

Subsequently, using the image of the right side of Figure 2, the structure of the neuron is determined using the pencil tool of the ImageJ program. Finally, the synapses of the neuron (the yellow points which are inside the structure previously marked) are manually counted one by one.

2.2 The SynapsesCountJ plug-in

A manual counting of synapses is impractical since it implies a considerable time investment. Moreover, this process is not applied just over a neuron but a battery of neurons in order obtain results about the evolution of the density of the occurrence of synapses in neurons under the effect of some drugs. Therefore, we have undertaken the task of automating this counting process as much as possible.

As a first step, a new ImageJ plug-in called *SynapsesCountJ* [21] has been developed. This plug-in improves the interaction with the ImageJ system to count synapses. The *SynapsesCountJ* plug-in works as follows.

First of all, using the *NeuronJ* plug-in [22], we determine the dendrites of the neuron, that is, the branches of the neuron. This is a semi-automatic step since the *NeuronJ* plug-in is able to perform a neuron tracing just choosing two points of a dendrite. Therefore, the effort of determining the structure of a neuron is considerably reduced.

Afterwards, the images with the two markers and the one with the structure are overlapped using the *SynapsesCountJ* plug-in. In this case, we use the red channel for the image with the bassoon marker, the green channel for the image with the synapsin marker and the blue channel for the image with the neuron dendrites, see Figure 3.

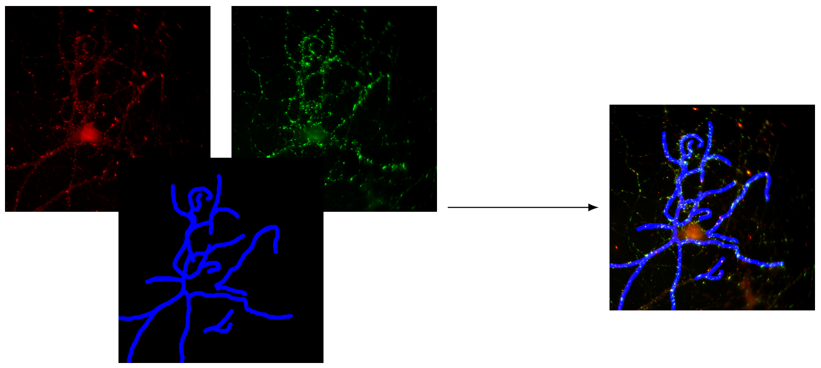


Fig. 3. Overlapping markers and structure

Now, we have an image where the synapse candidates are the white points. However, it is worth noting that the synapses are not only the fully white points, but also the ones whose color is close enough to white. So, *SynapsesCountJ* users must select a range of white values; then, the plug-in extracts the points in that range of values and inverts the colors in order to show the synapses as black points, see Figure 4.

At this point, the effort of counting the synapsis of a neuron is reduced to measure the number of connected components of the final image. To this aim several techniques can be applied, in particular we are going to use a well-known Algebraic Topology tool.

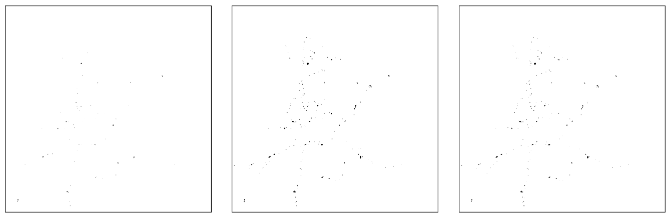


Fig. 4. Synapses using different ranges

3 The role of Algebraic Topology

In this section, we briefly provide the minimal standard background about Algebraic Topology needed in the rest of the paper. We mainly focus on definitions. Many good textbooks are available for these definitions and results about them, for instance [20].

3.1 Mathematical Preliminaries

Simplicial complexes are at the heart of Computational Algebraic Topology, since they give a concrete, combinatorial description of otherwise rather abstract objects which makes many important topological computations possible.

Let us start with some basic terminology. Let V be an ordered set, called the *vertex set*. An *(abstract) simplex* over V is any finite subset of V. An *(abstract) n-simplex* over V is a simplex over V whose cardinality is equal to $n+1$. Given a simplex α over V, we call subsets of α *faces* of α.

Definition 1 An *(ordered abstract) simplicial complex* over V is a set of simplices $\mathcal{K}$ over V such that it is closed by taking faces (subsets); that is to say, if $\alpha \in \mathcal{K}$ all the faces of α are in $\mathcal{K}$, too.

Let $\mathcal{K}$ be a simplicial complex. Then the set $S_n(\mathcal{K})$ of n-simplices of $\mathcal{K}$ is the set made of the simplices of cardinality $n+1$.

Let $\mathcal{K}$ be a simplicial complex over V. Let n and i be two integers such that $n \geq 1$ and $0 \leq i \leq n$. Then the *face operator* ∂_i^n is the linear map $\partial_i^n : S_n(\mathcal{K}) \to S_{n-1}(\mathcal{K})$, defined by: $\partial_i^n((v_0, \ldots, v_n)) = (v_0, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$; the i-th vertex of the simplex is removed, so that an $(n-1)$-simplex is obtained.

Now, we are going to introduce a central notion in Algebraic Topology. We assume as known the notions of ring, module over a ring and module morphism (see [16] for details). We will consider that the ground ring is $\mathbb{Z}$, the most common case in Algebraic Topology.

Definition 2 A *chain complex* C_* is a pair of sequences $(C_n, d_n)_{n\in\mathbb{Z}}$ where for every $n \in \mathbb{Z}$: the component C_n is a $\mathbb{Z}$-module (the *chain group of dimension*

n); the component d_n is a module morphism $d_n : C_n \to C_{n-1}$ (the *differential map*); and, the composition $d_{n-1}d_n$ is null, $d_{n-1}d_n = 0$.

The *n-homology group* of C_*, denoted by $H_n(C_*)$, is defined as the quotient $Ker\ d_n/Im\ d_{n+1}$.

Intuitively, homology groups measure n-dimensional holes in topological spaces. For instance, H_0, H_1 and H_2 measure respectively the number of connected components, the number of holes and the number of cavities of a space.

Once we have defined the notions of simplicial complexes and chain complexes, we can define the link between them.

Definition 3 Let $\mathcal{K}$ be a simplicial complex over V. Then *the chain complex $C_*(\mathcal{K})$ canonically associated with* $\mathcal{K}$ is defined as follows. The chain group $C_n(\mathcal{K})$ is the free $\mathbb{Z}$ module generated by $S_n(\mathcal{K})$. In addition, let $(v_0, \dots, v_n)$ be an n - simplex of $\mathcal{K}$, the differential of this simplex is defined as: $d_n := \sum_{i=0}^{n}(-1)^i\partial_i^n$.

The homology groups of a simplicial complex $\mathcal{K}$ are the ones of the chain complex $C_*(\mathcal{K})$: $H_n(\mathcal{K}) = H_n(C_*(\mathcal{K}))$.

The above definitions are classical notions from Algebraic Topology. However, in spite of being abstract concepts, they can be used to analyze digital images. There are several methods to construct a simplicial complex from a digital image [1]. Then, let us explain one of these methods to construct the simplicial complex associated with a monochromatic two dimensional image.

3.2 From Digital Images to Simplicial Complexes

Let $\mathcal{I}$ be a monochromatic image, to construct the simplicial complex associated with $\mathcal{I}$, denoted by $\mathcal{K}_\mathcal{I}$, we proceed as follows. Each black pixel of $\mathcal{I}$ is divided into two 2-simplexes (triangles), in turn each one of those 2-simplexes consists of three 1-simplexes (edges) and three 0-simplexes (vertices); the simplicial complex $\mathcal{K}_\mathcal{I}$ is the union of all those simplexes. An example can be seen in Figure 5; and a more detailed description of this process is explained in [11]. The homology groups of a 2D-image $\mathcal{I}$ are the ones of the simplicial complex $\mathcal{K}_\mathcal{I}$.

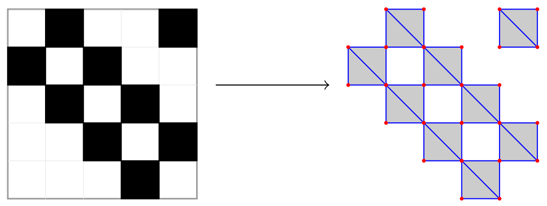


Fig. 5. A digital image and its simplicial complex representation

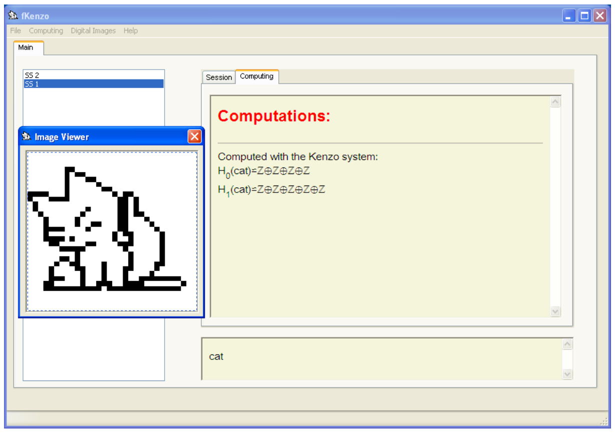
Eventually, we can interpret properties about a digital image from its homology groups. 2D-images are embedded in $\mathbb{R}^2$ then its homology groups vanish for dimensions greater than 2 and they are torsion-free from dimension 0 to dimension 1; that is, their homology groups are either null or a direct sum of $\mathbb{Z}$ components in dimensions 0 and 1. The number of $\mathbb{Z}$ components of the homology groups of dimension 0 and 1 measures respectively the number of connected components and the number of holes of the image. For instance, the homology groups of the image of Figure 5 are $H_0 = \mathbb{Z}\oplus\mathbb{Z}$ and $H_1 = \mathbb{Z}\oplus\mathbb{Z}\oplus\mathbb{Z}$; so, the image has two connected components and three holes. Therefore, applying the method presented in this subsection, we can reduce the problem of counting synapses of the images of Figure 4 to compute H_0 of such pictures.

We have presented a method to obtain a simplicial complex associated with a 2D-image, this process can be generalized to higher-dimensional images [23].

3.3 The *fKenzo* system

fKenzo [14] is a graphical user interface of the Kenzo system [7], a *Common Lisp* Computer Algebra system devoted to Algebraic Topology which was developed by Francis Sergeraert. The *fKenzo* system is organized through modules which can not only provide access to part of the Kenzo functionality, like the computation of homology groups of iterated loop spaces, but also include features which enhance the capabilities of Kenzo, like the connection with the GAP system to compute the homology of Eilenberg MacLane spaces of finite cyclic groups, a more detailed description about these additional features can be seen in [12].

One of the *fKenzo* modules allows one to analyze monochromatic digital images applying the methodology explained in the previous subsection. Figure 6 shows the computation of the homology groups of a small cat image in *fKenzo* which has 4 connected components and 5 holes. These properties are interpreted from the homology groups of the image as we have seen previously.



Computations:

Computed with the Kenzo system:
H_0(cat)=Z⊕Z⊕Z⊕Z
H_1(cat)=Z⊕Z⊕Z⊕Z⊕Z

cat

Fig. 6. Computation of Homology groups of a small cat

Therefore, using the *fKenzo* system we are able to compute the homology groups of images such as the ones presented in Figure 4. However, computing those homology groups can take a lot of time, due to the huge amount of information of the biomedical images; then, a safe reduction image processing which keeps the homological information is necessary.

4 Towards certification: methodology

The tool that we have used to deal with the problem of reducing digital images is based on *Discrete Morse Theory* [8]; in particular, we have worked in an algebraic setting of the Discrete Morse Theory which was described in [25]. The aim of Discrete Morse Theory is to find *simplicial collapses* that transform a simplicial complex $\mathcal{K}$ to a smaller complex which keeps the relevant information.

First of all, let us show the general idea with an example. The bases of the simplicial complex depicted in the left side of Figure 7 are made of 16 vertices, 32 edges and 16 triangles. So, the associated chain complex is

$$\cdots \leftarrow 0 \xleftarrow{d_0} \mathbb{Z}^{16} \xleftarrow{d_1} \mathbb{Z}^{32} \xleftarrow{d_2} \mathbb{Z}^{16} \xleftarrow{d_3} 0 \leftarrow \cdots$$

with the differential map as explained in Definition 3.

We can reduce the amount of information keeping the homological properties using *admissible discrete vector fields.* Vector fields are a tool to cancel "useless" information. If we want to design an admissible discrete vector field, we can decide the only allowed vectors are oriented leftward or downward because it is enough to avoid loops. In the example, the vector field can be seen in the second image of Figure 7. There are only two critical cells, remain one vertex and one edge. So, the reduced chain complex is

$$\cdots \leftarrow 0 \xleftarrow{d_0} \mathbb{Z} \xleftarrow{d_1} \mathbb{Z} \xleftarrow{d_2} 0 \leftarrow \cdots$$

with the null map between both copies of $\mathbb{Z}$. Therefore, $H_0 = \mathbb{Z}$ (one connected component) and $H_1 = \mathbb{Z}$ (one hole).

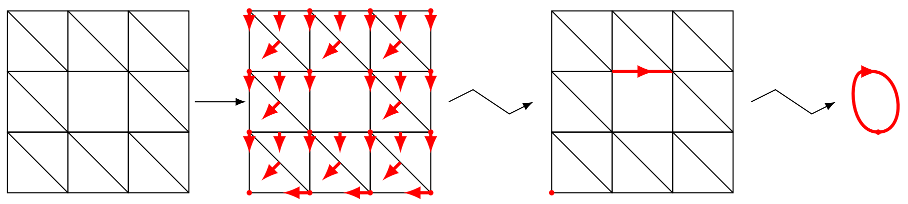


Fig. 7. Simplicial Complex with an admissible discrete vector field

4.1 Basic notions on Algebraic Discrete Morse Theory

Definition 4 An *algebraic cellular complex (ACC)* $C = (C_n, d_n, \beta_n)_{n\in\mathbb{Z}}$ is a chain complex $(C_n, d_n)_{n\in\mathbb{Z}}$ with a distinguished basis $\{\beta_n\}_{n\in\mathbb{Z}}$ in every chain group.

Let C be an ACC. A $(p-1)-$cell σ is said to be a *face* of a p-cell τ if the coefficient of α in $d_p\tau$ is non-null. It is a *regular face* if this coefficient is 1 or -1.

Definition 5 A *discrete vector field* V on an ACC $C = (C_p, d_p, \beta_p)_{p\in\mathbb{Z}}$ is a collection of pairs $V = \{(\sigma_i, \tau_i)\}_{i\in\beta}$ satisfying the conditions: every σ_i is some p-cell, in which case the other corresponding component τ_i is a $(p+1)$-cell; every component σ_i is a regular face of the corresponding component τ_i; and a cell of C appears at most one time in the vector field.

A cell χ which does not appear in a discrete vector field $V = \{(\sigma_i, \tau_i)\}_{i\in\beta}$ is called a *critical cell*.

However, not all the vector fields are able to reduce chain complexes (for instance the ones that produce loops), we need an additional admissibility property.

Definition 6 If $V = \{(\sigma_i, \tau_i)\}_{i\in\beta}$ is a vector field on an ACC $C = (C_p, d_p, \beta_p)_{p\in\mathbb{Z}}$, a *$V$-path of degree p* is a sequence $\pi = ((\sigma_{i_k}, \tau_{i_k}))_{0\leq k<m}$ satisfying: every pair $((\sigma_{i_k}, \tau_{i_k}))$ is a component of the vector field V and the cell τ_{i_k} is a p-cell; for every $0 < k < m$, the component σ_{i_k} is a face of $\tau_{i_{k-1}}$, non necessarily regular, but different from $\sigma_{i_{k-1}}$.

A discrete vector field V on an algebraic cellular complex $C = (C_p, d_p, \beta_p)_{p\in\mathbb{Z}}$ is *admissible* if for every $p \in \mathbb{Z}$, a function $\lambda_p : \beta_p \to \mathbb{Z}$ is provided satisfying the following property: every V-path starting from $\sigma \in \beta_p$ has a length bounded by $\lambda_p(\sigma)$.

Once we have defined the notion of a discrete vector field, we can explain that an admissible discrete vector field, V on an ACC C, defines a reduction generated by the critical p-cells. Firstly, let us introduce the notion of reduction.

Definition 7 A *reduction* between two chain complexes C_* and D_* (which is denoted by $C_* \Rrightarrow D_*$) is a triple (f, g, h) where: (a) the components $f : C_* \to D_*$ and $g : D_* \to C_*$ are chain complex morphisms; (b) the component $h : C_* \to C_{*+1}$ is a graded group morphism of degree $+1$; and (cc) the following relations are satisfied: (1) $fg = id_{D_*}$; (2) $d_{C_*}h + hd_{C_*} = id_{C_*} - gf$; (3) $fh = 0$; (4) $hg = 0$; and (5) $hh = 0$.

Let $C_* \Rrightarrow D_*$ be a reduction, then C_* and D_* have canonically isomorphic homology groups, see [26].

Theorem 1 (Vector-Field Reduction [25]) Let $C = (C_p, d_p\beta_p)_{p\in\mathbb{Z}}$ be an ACC and $V = \{\sigma_i, \beta_i\}_{i\in\beta}$ be an admissible discrete vector field on C. Then the vector field V defines a canonical reduction $(C_p, d_p) \Rrightarrow (C_p^c, d'_p)$ where $C_p^c = \mathbb{Z}\left[\beta_p^c\right]$ is the free $\mathbb{Z}-$module generated by the critical cells.

With this theorem, we are able to work with the chain complex generated by the critical cells which is much smaller than the initial chain complex, knowing that homological properties are preserved. Let us note that the larger is the number of vectors which compose the vector field the smaller is the reduced chain complex. So, we are interested in creating a vector field with many vectors as possible.

4.2 Implementation, Testing and Certification

For algebraic cellular complexes coming from actual digital images, we can summarize the reduction problem of an $ACC\ C = (C_n, d_n, \beta_n)_{n \in \mathbb{Z}}$ by a vector field as follows. It is worth noting that even the bigger digital images have always a finite number of components, hence an ACC coming from a digital image always have a finite number of components in each degree. In addition, the differential map d_n of the ACC can be represented by a finite matrix M_n. Then, the problem of computing homology groups is translated to a problem of diagonalizing those matrices, see [28].

From M_n, an admissible vector field V can be constructed. Subsequently, using M_n and the vector field V, a new matrix $\widehat{M}_n$ (smaller than M_n) is obtained. Thanks to the matrix $\widehat{M}_n$ the homology groups of C can be computed much faster. In [25] the algorithms that compute both the admissible vector field and the reduced matrix are explained; here, we just state them (Algorithm 2 invokes Algorithm 1).

Algorithm 1
Input: an integer matrix M.
Output: an admissible vector field V.

Algorithm 2
Input: an integer matrix M.
Output: a reduced matrix $\widehat{M}$.

These algorithms have been implemented as *Haskell* programs [17]. We have chosen *Haskell* because both the code and the way of working is similar to the ones of the *Coq* formal proof management system [5], which will be used to certify the correctness of the programs.

After implementing Algorithm 2, and all its subalgorithms, we have spent a long time testing our programs, as usual in software development. To this aim, we have done two types of testing. On the one hand, an automatic testing with examples generated from random images has been performed. On the other hand, we have also tested our programs with *QuickCheck* [4], a tool which allows one to automatically test properties about programs.

Let us focus on the testing performed via *QuickCheck*. The way of working is as follows: firstly, a specification of the properties that the programs must fulfill is given. Then, the system tests those properties with cases randomly generated. For instance, we can define the following property.

```
> prop-admissible-vf M = (int-matrix M) ==> (admissible (vectorCvd M)) ✠
> \emph{QuickCheck} prop-admissible-vf ✠
OK, passed 100 tests
```

The above code must be read as follows. The first line states the property called it `prop-admissible-vf`, that we want to test; in this case, we are interested in checking that given an integer matrix `M` the function `vectorCvd`, which implements Algorithm 1, builds an admissible discrete vector field (`admissible` is a test function that returns `True` if its argument is an admissible discrete vector field and `False` otherwise). The second line is the *QuickCheck* invocation to test the property `prop-admissible-vf`. The last line presents the result produced by *QuickCheck*, which means that all the cases generated by *QuickCheck* satisfy the property `prop-admissible-vf`. In this way, all the functions implemented in *Haskell* have been tested.

The usage of *QuickCheck* can be considered as a good starting point towards the formal verification of our programs. On the one hand, a specification of the properties which must be satisfied by our programs is given (a necessary step in the formalization process). On the other hand, before trying a formal verification of our programs (a quite difficult task) we are testing them, a process which can be useful in order to detect bugs.

Finally, the last step is the formalisation of the implementation of our algorithm in the *Coq* Theorem Prover [5] using the *SSReflect* environment [9]. Once this task is accomplished, we can claim that our programs are correct; that is to say, they always produce the expected result. The formalisation of our programs is an ongoing work; so, here we are only introducing some remarks.

The first step in our formalisation consists in translating our *Haskell* programs into the *Coq* language. In spite of the fact that both languages are close, some changes are necessary in some functions, for instance, adding termination conditions. In addition, it is also necessary to define the test functions which allow us to specify the properties that our programs must satisfy (the `admissible` function is an example of this kind of functions). Eventually, we state the lemmas that ensure the correctness of our programs. For instance, we are interesting in proving properties like the following one.

SSReflect Lemma 1 Let `M` be an integer matrix, then `(vectorCvd M)` builds and admissible vector field.

```
Lemma admissible-vf:
        forall M, (int-matrix M) -> (admissible (vectorCvd M))
```

It is worth noting that the above lemma is very similar to the *QuickCheck* property that we have tested. As we have said previously, this formalisation process is work in progress and just some parts of the programs have been verified up to now.

5 Automating and Certifying the Workflow

Let us summarize our workflow, depicted in Figure 5 to study biomedical images related to synaptical structures. We must remark two things in this process: on the one hand, a wide variety of programs and programming languages are used in this process; on the other hand, several steps have been formalized using theorem proving tools improving in this way the reliability of our approach.

$$\begin{tikzcd}
\text{Biomedical image} \arrow[r, "\mathit{SynapsesCountJ}"] & \text{Processed image} \arrow[l, bend left, "\text{properties}"] \arrow[r, "\mathit{fKenzo}"] & \text{Chain complex} \arrow[r, "\mathit{fKenzo}"] \arrow[d, dotted, "\mathit{fKenzo}"] & \text{Matrix} \arrow[d, "\mathit{Haskell}"] \\
& & \text{Homology} \arrow[ul, "\text{interpretation}"] & \text{Reduced matrix} \arrow[l, "\mathit{fKenzo}"']
\end{tikzcd}$$

Fig. 8. Workflow of biomedical images study

First of all, the biomedical images related to synaptical structures are processed using the *SynapsesCountJ* plug-in. Namely, from three images of the same neuron (representing the neuron with two different antibody markers and the structure of the neuron), a monochromatic image representing the synapses of the neuron is returned.

The construction of the chain complex associated with the processed image is performed by *fKenzo* in three steps: (1) build the simplicial complex associated with the digital image; (2) construct the simplicial set canonically related to the simplicial complex; and, (3) build the chain complex associated with the simplicial set. All these steps have been formalized in the *ACL2* theorem prover [18]; namely, the certification of both first and second steps is given in [13]; and the verification of the third step was presented in [19].

At this point we could compute the homology groups of the image using the *fKenzo* system; however, as this option can take a lot of time, we use the alternative route. Namely, an integer matrix is obtained from the chain complex. In particular, as we are interested in computing the 0-homology group of the image in order to measure the number of its connected components, we only need the matrix M_1 which represents the differential map d_1 of the chain complex associated with the digital image. This step is performed thanks to *fKenzo* and is formalized in the *Coq* proof system, see [15].

The matrix associated with the chain complex is reduced thanks to the *Haskell* programs presented in Section 4. As we have said previously the formalization of this step is work in progress and is undertaken with the *Coq* proof system.

Once that we have obtained the reduced matrix, we can compute the homology groups thanks to *fKenzo*. The formalization of this step remains as further work. In particular, we obtain the 0-homology group which is interpreted as the number of connected components of the processed image. Finally, from the number of connected components, we obtain the number of synapses of the original biomedical image.

6 Conclusions and Further work

The main contribution of this paper is the presentation of a methodology to study digital images from a topological point of view. Our aim was to automate the processes carried out previously by a team of biologists when studying structural synaptical properties (see [6]). In addition to this *computational* aspect, we have also proposed a way of certifying the correctness of (some of) the automated steps, by using theorem provers as *Coq* or *ACL2*.

As already explained previously, the verification of our *Haskell* programs by means of *Coq/SSReflect* is still an ongoing work (it is a difficult and time consuming task). A second line of further research consists in finding more advanced applications of our homological tools in the biomedical imaging context, since up to now, only the counting of connected components (in other words, the computation of the 0-th homology group) has been used.

References

1. R. Ayala, E. Domínguez, A. Francés, and A. Quintero. Homotopy in digital spaces. *Discrete Applied Mathematics*, 125:3–24, 2003.
2. M. Bear, B. Connors, and M. Paradiso. *Neuroscience: Exploring the Brain.* Lippincott Williams & Wilkins, 2006.
3. Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development, Coq'Art: the Calculus of Inductive Constructions.* Springer-Verlag, 2004.
4. K. Claessen and J. Hughes. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *ACM SIGPLAN Notices*, pages 268–279. ACM Press, 2000.
5. Coq development team. The Coq Proof Assistant Reference Manual, version 8.3. Technical report, 2010.
6. G. Cuesto et al. Phosphoinositide-3-Kinase Activation Controls Synaptogenesis and Spinogenesis in Hippocampal Neurons. *The Journal of Neuroscience*, 31(8):2721–2733, 2011.
7. X. Dousson, J. Rubio, F. Sergeraert, and Y. Siret. The Kenzo program. Institut Fourier, Grenoble, 1998. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`.
8. R. Forman. Morse theory for cell complexes. *Advances in Mathematics*, 134:90–145, 1998.
9. G. Gonthier and A. Mahboubi. A Small Scale Reflection Extension for the Coq system. Technical report, Microsoft Research INRIA, 2009. `http://hal.inria.fr/inria-00258384`.
10. J. Heras. The *fKenzo* program. University of La Rioja, 2010. `http://www.unirioja.es/cu/joheras/fKenzo/`.
11. J. Heras. *Mathematical Knowledge Management in Algebraic Topology.* PhD thesis, University of La Rioja, 2011.

12. J. Heras, V. Pascual, A. Romero, and J. Rubio. Integrating multiple sources to answer questions in Algebraic Topology. In *Proceedings of the 9th International Conference on Mathematical Knowledge Management (MKM'10)*, volume 6167 of *Lectures Notes in Artificial Intelligence*, pages 331–335. Springer-Verlag, 2010.
13. J. Heras, V. Pascual, and J. Rubio. Proving with ACL2 the correctness of simplicial sets in the Kenzo system. In *Proceedings of the 20th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'2010)*, volume 6564 of *Lectures Notes in Computer Science*, pages 37–51. Springer-Verlag, 2011.
14. J. Heras, V. Pascual, J. Rubio, and F. Sergeraert. *fKenzo*: A user interface for computations in Algebraic Topology. *Journal of Symbolic Computation*, 46:685–698, 2011.
15. J. Heras, M. Poza, M. Dénès, and L. Rideau. Incidence simplicial matrices formalized in Coq/SSReflect. In *Proceedings 18th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning (Calculemus'2011)*, 2011. `http://wiki.portal.chalmers.se/cse/pmwiki.php/ForMath/PapersAndSlides`.
16. N. Jacobson. *Basic Algebra II*. W. H. Freeman and Company, 2nd edition, 1989.
17. S. P. Jones et al. The Haskell 98 language and libraries: The revised report. *Journal of Functional Programming*, 13(1):0–255, 2003. `http://www.haskell.org`.
18. M. Kaufmann and J. S. Moore. ACL2. `http://www.cs.utexas.edu/users/moore/acl2/`.
19. L. Lambán, F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. Applying ACL2 to the Formalization of Algebraic Topology: Simplicial Polynomials. In *Proceedings of the Interactive Theorem Proving 2011 (ITP'2011)*, 2011. `http://wiki.portal.chalmers.se/cse/uploads/ForMath/aafatsp`.
20. S. MacLane. *Homology*. Springer, 1963.
21. G. Mata. SynapsesCountJ. University of La Rioja, 2011. `http://imagejdocu.tudor.lu/doku.php?id=plugin:utilities:synapsescountj:start`.
22. E. Meijering et al. Design and Validation of a Tool for Neurite Tracing and Analysis in Fluorescence Microscopy Images. *Cytometry Part A*, 58(2):167–176, 2004.
23. D. Orden and F. Santos. Asymptotically efficient triangulations of the d-cube. *Discrete and Computational Geometry*, 30(4):509–528, 2003.
24. W. S. Rasband. ImageJ: Image Processing and Analysis in Java, 2003. `http://rsb.info.nih.gov/ij/`.
25. A. Romero and F. Sergeraert. Discrete Vector Fields and Fundamental Algebraic Topology, 2010. `http://arxiv.org/abs/1005.5685v1`.
26. J. Rubio and F. Sergeraert. Constructive Homological Algebra and Applications, Lecture Notes Summer School on Mathematics, Algorithms, and Proofs. University of Genova, 2006. `http://www-fourier.ujf-grenoble.fr/~sergerar/Papers/Genova-Lecture-Notes.pdf`.
27. D. J. Selkoe. Alzheimer's disease is a synaptic failure. *Science*, 298(5594):789–791, 2002.
28. O. Veblen. *Analysis Situs*. AMS Coll. Publ., 1931.