Determining the branchings of 3D structures from respective 2D projections

Jorge J. G. Leandro, Roberto M. Cesar Jr.
Institute of Mathematics and Statistics - USP
Department of Computer Science
Rua do Matão, 1010 São Paulo, SP
05508-090 Brazil
{jleandro,cesar}@vision.ime.usp.br

Luciano da Fontoura Costa
Instituto de Física de São Carlos - USP
Department of Physics and Informatics
Av. Trabalhador Sãocarlense, 400 São Carlos, SP
13560-970 Brazil
luciano@ifsc.usp.br

Abstract

This work describes a new framework for automatic extraction of 2D branching structures images obtained from 3D shapes, such as neurons and retinopathy images. The majority of methods for neuronal cell shape analysis that are based on the 2D contours of cells fall short of properly characterizing such cells because crossings among neuronal processes constrain the access of contour following algorithms to the innermost regions of the cell. The framework presented in this article addresses, possibly for the first time, the problem of determining the continuity along crossings, therefore granting to the contour following algorithm full access to all processes of the neuronal cell under analysis. First, the raw image is preprocessed so as to obtain an 8-connected, one-pixel wide skeleton as well as a set of seed pixels for each subtree and all the branching/crossing regions. Then, for each seed pixel, the algorithm labels all valid neighbors, until a branching/crossing region is reached, when a decision about the proper continuation is taken based on the tangent continuity. The algorithm has shown robustness for images with parallel segments and low densities of branching/crossing densities. The problem of too high densities of branching/crossing regions can be addressed by using a suitable data structure. Successful experimental results using real data (neural cell images) are presented.

1. Introduction

One of the most important implications of the intensive investigations in genetics and biomolecular sciences along the last decades has been the realization that more complete understanding and control of phenotipic features of individuals can not be fully achieved without effective characterization of such features as well as the consideration of influences of the external and internal environment where cells and tissues develop (e.g. [4]). Whole new areas, including *post-genomics*, *neuroinformatics*, and *systems biology* have ultimately derived from the intensifying efforts in investigating such fundamental questions.

Few biological systems provide such a challenging and rich laboratory for relating phenotypic characteristics to genetics and animal development as the nervous system. Indeed, the ever changing shapes of neurons are closely related not only to the genetic cell content, but also to external stimuli and internal biochemical and anatomical changes. Already important as a subsidy for diagnosis, the characterization of *neuronal shape* has progressively established itself as a key resource for several investigations in biology and neuroscience. Indeed, it is only by obtaining a precise and comprehensive representation and characterization of the shapes of the neuronal cells under analysis that more objective and quantitative efforts can be made so as to relate neuronal phenotype with genetics, phylogenetics, comparative neurology and animal development. Another important problem related to neuronal shape concerns the shape-function paradigm (e.g. [5]), which addresses how the structure of neuronal cells would be related to their respective function. By being prototypically complex, neuronal cell shapes also constitute a particularly interesting type of data for shape analysis.

Despite its key role in so many areas, it was only more recently that the endeavor of neuronal cell shape analysis started to establish itself as an important research area on itself. The intensification of related research efforts along the last years has significantly contributed to formalizing and developing a large number of measurements and models of neuronal shape, to the point that it becomes difficult to provide a comprehensive review of this area in the current work. Among the several approaches aimed at characterizing the geometry and connectivity of neuronal cells [2, 8, 9], those based on the contours of the cells provide particularly effective means for measuring and characterizing the respective shape as a consequence of the mapping from the

2D (or 3D) spaces where the cells were imaged into 1D parametric curves describing the outline of the cells (Figure 1). Among other possibilities, such curves can be used in order to infer the normal and/or tangent orientation fields along the cell contours, as well as the estimation of the respective curvature (e.g. [3]), which can provide particularly valuable information about the local degree of bending of the curve as well as its concavity. However, such approach is often limited by the presence of crossings between the neuronal processes, implying some regions of the cell to become inaccessible for contour extraction.

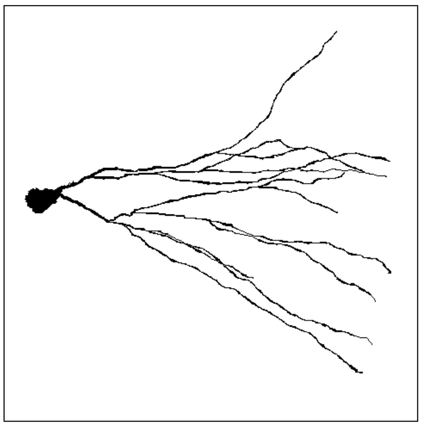


Figure 1. Example of neuron image considered in this work.

Usually, the contour [7] obtained from a *shape* is understood as a parameterized curve, from which several measurements, such as curvature, can be obtained. A typical shortcoming in extracting contours from branching structures consists in the fact that the chain-code-based contour following algorithm (Figure 2) can not traverse regions delimited by crossings (due to the 3D to 2D projection). As a result, only the outer contour of the cell is obtained, while the innermost structures remain unaccessible (fig 3). In other words, the contour following algorithm based in the chain-code can not deal with non-Jordan curves [1].

The current work is aimed precisely at solving such a problem, so that more complete parametric representations of the cell shape can be obtained and analyzed. This is achieved through the incorporation of several criteria, with particular emphasis given to ensuring the continuity of the tangent orientation as the means to identify the proper continuation of the neuronal processes as they go through crossing-points. The introduced algorithm represents the main original contribution of the present paper. To circumvent the aforementioned shortcoming in branching structures contour following, we propose an algorithm devised

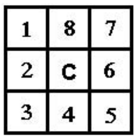


Figure 2. Chain-code illustration. The neighborhood of the pixel *c* is scanned according to the increasing sequence $1..8$.

to separate crossing branches within a branching structure, namely the *Branch Tracking Algorithm (BTA)*, so allowing the chain-code contour following algorithm to yield a proper contour for most neuronal shapes.

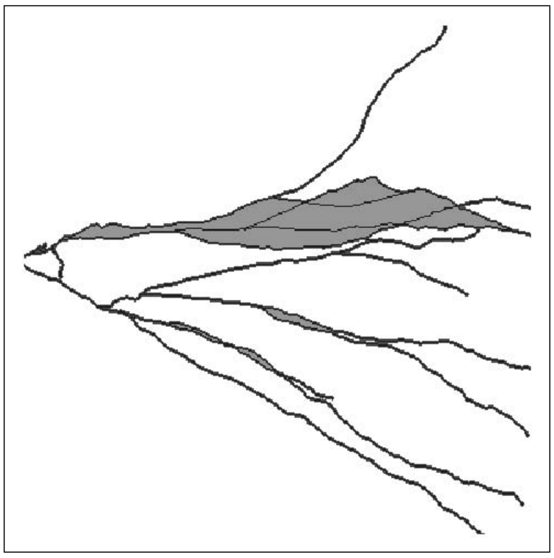


Figure 3. Neuron skeleton (black) and respective contour (dark gray). The light gray shaded areas are unreachable for the contour following algorithm based on the chain-code.

This paper is organized as follows. Section 2 presents an overview of the proposed framework, which is subsequently detailed in Section 3. Experimental results obtained from the application of the proposed algorithm to images of real neural cell are presented in Section 4. The paper is concluded with some discussion on the obtained results and our ongoing work in Section 5.

2. Concepts and Overview

In general, an image obtained by means of an imaging device such as a photograph camera is a two-dimensional projection from a three-dimensional shape, hence lacking information regarding depth. Particularly, when dealing with *complex shape images*, such as neurons and retinal images, depth information becomes crucial to set apart branches that seem to intersect one another. This is the main problem addressed in the present paper.

A **branching structure** is a binary image of the 8-connected one-wide pixel skeleton, obtained from a *shape image*, where *bifurcation regions* and *crossing regions* are present. A **branch** is the binary image comprised of a set of pixels starting from either the soma up to its termination or a bifurcation/crossing region up to its termination, whose object pixel coordinates may be parameterized as a smooth curve. A **bifurcation region** is the set of pixels where an inward branch splits into two outward branches, one of them in quite a distinct orientation. A **crossing region** is the set of pixels where an inward branch splits into $N > 2$ outward branches, with $N - 1$ outward branches in quite distinct orientations.

The proposed approach involves two main steps:

- Preprocessing the original image through mathematical morphology transformations, yielding its 8-connected one-wide pixel skeleton image, its crossing regions image and a queue containing the origins of the branches, to be taken as the seeds for the *BTA*;
- Calling the *BTA* for each source point (i.e. each seed). Starting by the current seed, the algorithm labels iteratively its valid neighboring pixels, i.e still non-labeled pixels, whose vicinity equals two pixels until a *crossing/bifurcation* region is reached. The algorithm takes a decision to continue with the tracking procedure for the current branch. The *branching structure* topology itself provides the algorithm with proper information to decide among several possible outward branchings.

These steps are detailed in next section.

3. General Framework

3.1. Preprocessing

Firstly, the following preprocessing algorithm is performed in order to achieve the desired *branching structure* to be labeled:

BTA:Preprocessing
begin

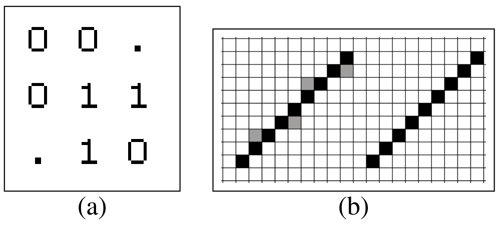


Figure 4. (a) Structuring Element for Hit-or-Miss filtering the prunned skeleton to yield an 8-connected one-pixel wide skeleton (b)Light gray pixels at the left hand should be removed yielding the non-redundant structure at the right hand.

- Thresholding the original gray level image to binarize it;
- Eroding the binary image by a disk structuring element to get rid of dendrites;
- Area Filtering the eroded image to segment only the neuron soma;
- Dilating the filtered neuron soma to retrieve its original size;
- Dilating the binary image in order to merge branch portions which are too close and almost parallel;
- Skeletonizing the binary image by morphological thinning [6];
- Pruning the skeleton (Figure 5-a) to get rid of noise structures [6];
- Building a 3×3 mask template for the Hit-or-Miss operation (Figure 4-a) to eliminate redundant information (Figure 4-b) and achieve an 8-connected one-pixel wide skeleton structure (Figure 5-b);
- Subtracting the soma from the 8-connected one-pixel-wide skeleton;
- Detecting end points twice. First in the 8-connected one-pixel wide skeleton structure *along* the soma segmented previously and then in the 8-connected one-pixel wide skeleton structure *without* the soma segmented previously. The difference between these two images should ultimately yields only the **origin** points to be used as source or seeds to feed the *BTA*, as depicted in (Figure5-c).
- Creating a neighborhood image, where each pixel value represents the number of object pixels in its 8 neighborhood. Henceforth this image will be referred to as *Crossing regions* image. (Figure 5-d).

end

It is worth mentioning that structure elements size for all morphological operations were empirically found for the set of processed images. Different sized images may require different structure element sizes.

3.2. Branches Tracking

For the branch tracking step, it is introduced the *Branch Tracking Algorithm (BTA)* to segment, that is to label, each neuron branch as a distinct object on its own. Roughly speaking, the *BTA* is comprised of two great loops, the outermost one associated to a Queue for seeds (source points), while the innermost one manages a Stack for valid neighbors, neighbors of neighbors and so forth. Objects, i.e. non-labeled and non-crossing pixels, are considered as being valid. Thus, for each dequeued seed, the *BTA* is called for a partial branch tree, by stacking all valid pixels pertaining to the same branch. These stacked pixels will be subsequently labeled until either the current branch termination pixel or a branching/crossing region is reached. Every time a branching/crossing region is reached, the *BTA* identifies the best branch to continue the labeling process, while enqueueing seeds for partial branch trees to be labeled in a recursive-like fashion, until no more branches stemming from a branch within the tree rooted at the current source seed is found.

In order to determine the branch to continue the labeling past a branching/crossing region, a *breadth-first search* has been adopted, which works by enqueueing all the pixels into an auxiliary queue while getting across the just detected branching/crossing region. This *breadth-first search* takes place until a stability condition is achieved, i.e. there are only non-crossing pixels enqueued, for a consecutive number C of times. This process is illustrated in figure 6 and table 1.

state	current	queue	B	Σ
0	$\emptyset$	a	1	1
1	a	b	0	0
2	b	c d $\check{a}$	0	0
3	c	d e $\check{d}\check{b}$	0	0
4	d	e $\check{c}$ f $\check{b}$	1	1
5	e	f g $\check{c}$	1	2
6	f	g $\check{d}$ h	1	3
7	g	h i $\check{e}$	0	0
8	h	i j $\check{f}$	0	0
9	i	j l m $\check{g}$	0	0
10	j	l m n $\check{h}$	0	0
11	l	m n o p $\check{m}\check{i}$	0	0
12	m	n o p $\check{l}\check{p}\check{i}$	0	0
13	n	o p q $\check{j}$	0	0
14	o	p q r $\check{l}$	0	0
15	p	q r $\check{l}$ s $\check{m}$	1	1
16	q	r s $\check{n}$ t	1	2
17	r	s t u $\check{o}$	1	3
18	s	t u $\check{p}$ v	1	4
19	t	u v $\check{q}$ x	1	5
20	u	v x y $\check{r}$	1	6
21	v	x y $\check{s}$ w	1	7
22	x	y w $\check{t}$ z	1	8

Table 1. States of the auxiliary queue. *B* is 1 if all pixels in a given state are non-crossing and 0 otherwise. Σ is increased by one if the respective *B* is on, and zeroed otherwise. Notice the check marks on some enqueued pixels; these checked pixels will be ignored, thus removed from the queue, as they have already been considered before.

Notice that scanning the neighborhood of the pixel **a** (state 0 in table 1) according to the chain-code defined in figure 2, we obtained the state 1. By setting the stability condition parameter C to 8, this procedure is repeated until state 22, when Σ equals C. At this point, the remaining pixels in the auxiliary queue, **y**, **w** and **z**, provide us with the terminations of the desired outward direction vectors. Strictly speaking, each corresponding direction vector origin should fulfill two requirements simultaneously: (i) be neighbor of the nearest crossing region pixel and (ii) have an existing path of valid pixels between it and the respective termination. For images presenting low branching/crossing region densities, the second requirement might be ignored, but not in the case of the figure 6-a. Figure 6-b shows the vectors obtained by considering both conditions (i) and (ii). All the vectors are then normalized and dot products between the inward direction vector v_0 and each outward direction vector are computed. Obviously, the outward vector for which the dot product result is maximum gives the proper direction to continue with the labeling, therefore the next pixel to be stacked. The remaining vectors are taken as side branches seeds to be enqueued for further processing. In the example described in table 1 and figure 6, the vector v_3 would give the direction toward which to continue the labeling, while vectors v_1 and v_2 would give directions for future consideration.

In brief, the *BTA* might be described as follows:

BTA:LabelTree

begin

- Initialize a stack for valid branch pixels and a queue for seeds.
- While queue is not empty

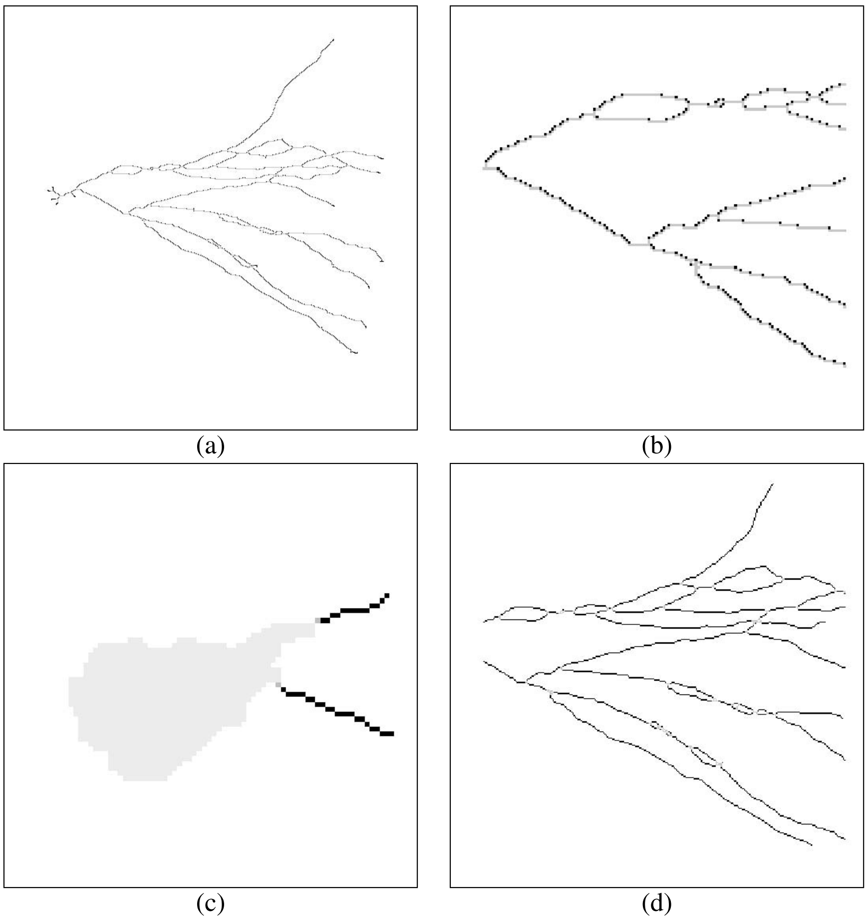


Figure 5. Preprocessing results: (a)Pruned skeleton. (b)The darkest pixels were removed by the Hit-or-Miss filtering yielding the 8-connected one-pixel wide skeleton shown in lighter gray level. (c) Soma (light gray), seed (dark gray) and skeleton(black). (d)Crossing regions (light gray) and skeleton (black).

  - Call **BTA:LabelBranch**.

- Check conditions to stack only valid neighbor pixels, that is object and non-labeled pixels.

end

The **BTA:LabelTree** algorithm makes a call to the **BTA:LabelBranch** procedure, which is defined below:

BTA:LabelBranch
begin

- Pop the current pixel.
- Keep track of the j^{th} preceding pixel to compute the inward direction vector when arriving at a *crossing/bifurcation* region, where j may be a parameter. Usually, it suffices to take the 4^{th} pixel, that is $j = 4$.
- If the current pixel is valid, i.e. it is an object, non-labeled and outside a crossing region, then label it.
- Probe the current pixel neighborhood to check if there is a *crossing/bifurcation* region.
- In case there is any inward crossing region pixel in the neighborhood, compute the inward normal direction vector.
- In case there is any inward crossing region pixel in the neighborhood, enqueue all neighbors iteratively, until reaching a number of non-crossing pixels outward the branches for a consecutive number c of times, thus providing a stability condition.

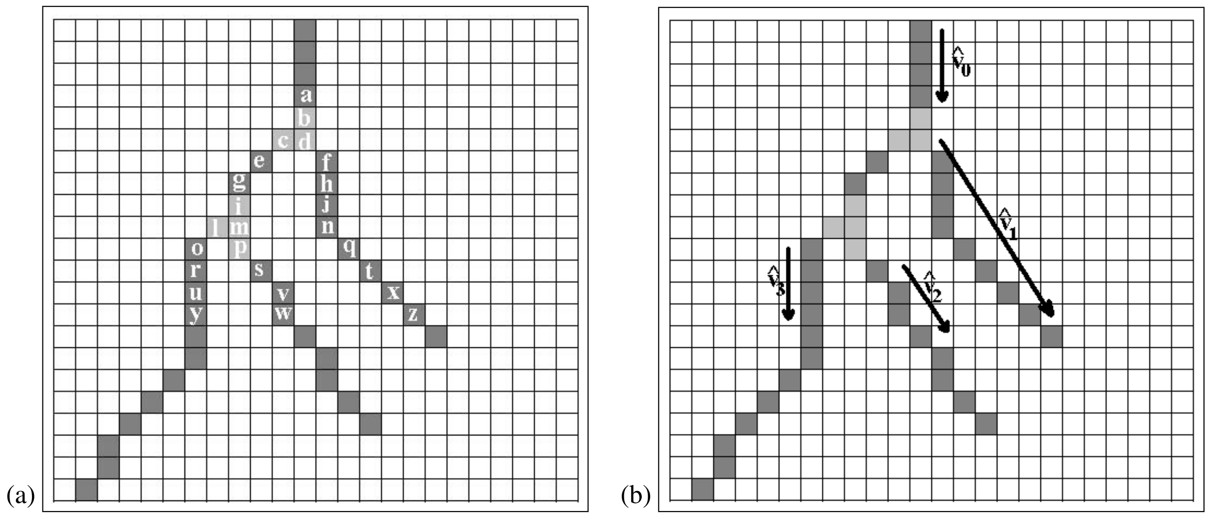


Figure 6. A typical critical region and the corresponding direction vectors obtained from it. (a) Close critical regions (light gray). The *breadth-first search* starts as the pixel *a* is reached and stops when only pixels *y, w* and *z* remain in the auxiliary queue. (b) the obtained direction vectors.

- These just reached non-crossing pixels will be the terminations of our outward direction vectors.
- For each termination pixel, take its nearest pixel inside the *crossing/bifurcation* region and compute the respective normalized direction vector.
- Now calculate the dot product between each outward direction v_i vector and the inward direction vector v_0 and take the k_{th} index related to the vector v_k that gives rise to the largest dot product result, i.e:

$$k = argmax_i(< v_0, v_i >) \tag{1}$$

- If there are only two outward branches, push the pixel related to the k_{th} outward normal direction vector onto the stack and enqueue the other.
- Reinitialize the auxiliary variables in order to retrieve the ante-penultimate valid pixel coordinates for the next inward direction vector.
- If there are more than two outward branches, push the pixel related to the k_{th} outward normal direction vector onto the stack and enqueue all the remaining non-crossing vectors that are not in opposite orientations.

end

4. Results

The algorithms described so far have been implemented as *Matlab* scripts. The mathematical morphology operations have been applied using the *Mathematical Morphology Toolbox* by *SDC* (`http://www.mmorph.com/`). The method has been evaluated and results obtained by labeling rat hippocampal cells from public available *Southampton Archive* (`http://www.compneuro.org/CDROM/nmorph/index/topindex_tn.html`). Some results are presented in figure 7. The left column shows the original images while the right column presents the respective labeling results, shown as connected components with the same gray-level as label. New labels are assigned to dendrite segments born from branches. The algorithm is able to distinguish branches from crossing, which is reflected by the correct assigned labels for the outward segments from such structures. Notice how parallelism and high densities of branching/crossing regions imply the *BTA* to label the same branch with different labels.
For the 3 test images shown in figure 7, it has been visually identified 40 bifurcations and 3 clear-cut crossings. All the bifurcations were correctly labeled and so were the 3 crossings without parallelism occurrencies.

5. Concluding Remarks

The proposed approach starts by extracting the skeleton of the $2D$ projection of a $3D$ branching structure. Any skeletonization method may be applied to obtain the skeleton and incorporated (e.g. morphological thinning, exact dilations, medial axis transform). The important issue is to

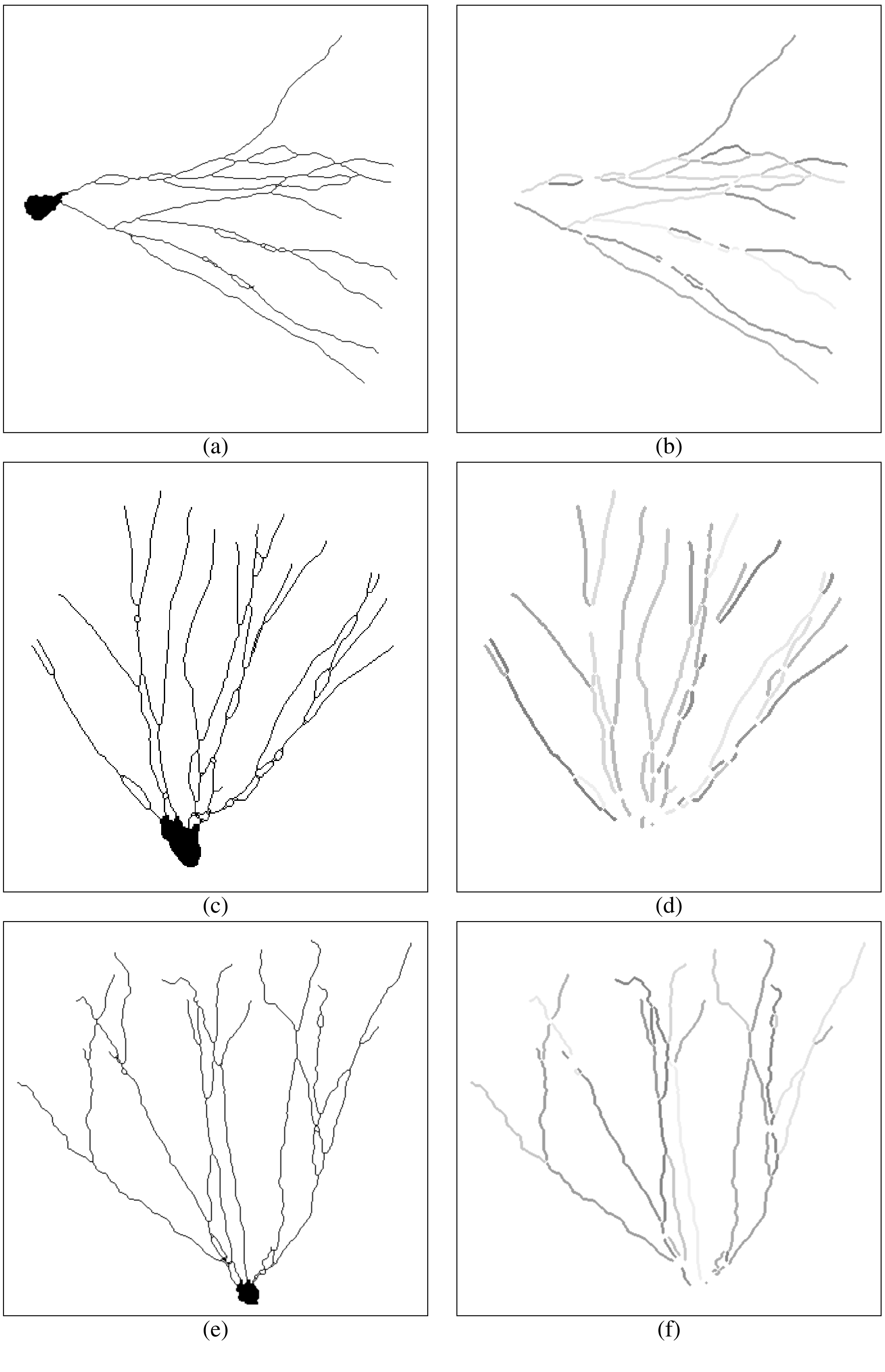
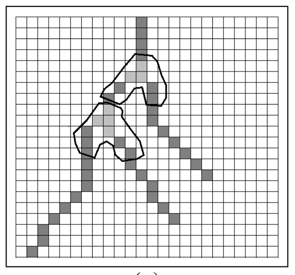

(a) (b) (c) (d) (e) (f)

Figure 7. Results. Original images (a), (c) and (e). Labeled images (b),(d) and (f). A different gray level value has been used to label each different segment in (b),(d) and (f). The set of used gray level values was chosen so as to enhance the contrast among segments. Also, these images have been dilated by a 1-sized cross structuring element in order to improve their visualization.

have a suitable skeleton as input to the method. The preprocessing parameters have been empirically chosen and depend on the adopted skeletonization algorithm.
Despite the good results presented in the previous section, there are still some pending cases deserving special attention in future research. In general, the major pitfalls encountered during *BTA* running were related to different particular structure topologies, leading to high crossing/bifurcation regions densities, superposition of branches and/or branches parallelism.

As pointed out in the *BTA* description, in case of images with high crossing/bifurcation regions density, both conditions should be accomplished in order to get the direction vectors origins, otherwise shortcomings may occur, such as missing some branches. In the example presented in figure (6), notice that for all the termination pixels **y**, **w** and **z** the nearest branching/crossing region pixel is **p**. Hence, by considering only the proximity condition, instead of the obtained vectors v_1, v_2 and v_3, all vectors origins would lie in **s**, i.e. the nearest non-crossing pixel which is neighbor of **p**. It is plain to see that, as a consequence, only pixel **s** would be stacked for continuing the labeling process. However there are not valid paths between **s** and **y** and between **s** and **z**. For this reason, the incoming branch would have its continuation from pixel **s** on, instead of pixel **f** as expected. As a result, the *BTA* would miss two branches. This is a straightforward consequence of the *breadth-first search agglutinating effect*, since the *breadth-first search* has been implemented by using a queue data structure, which is memoryless regarding the shape topology. In an effort to accomplish the stability condition, the algorithm has clumped both bifurcation/crossing regions (Figure 8-(a)) into only one (Figure 8-(b)). A possible solution for this problem is to implement the *breadth-first search* by using a tree data structure, so as to keep memory of the topology at the critical region. As soon as the stability condition would be accomplished, the *BTA* would just get the tree leaves, which would host the *direction vectors terminations*, and ask the tree for the *direction vectors origins* by climbing the tree from each termination up its nearest non-crossing bifurcation/crossing neighbor. The tree itself would provide us properly with the valid paths between terminations and origins.

Particularly, it has been observed in rat hippocampal cells that too close branches parallelism may occur with high frequencies. This might create short cycles along the structure leading to the *BTA* incorrectly labeling the branches. As a solution, the image has been dilated just before it has been skeletonized. By doing so, short cycles would be shrunk into a branching/crossing region. In future works, we intend to proceed a multiscale analysis to circumvent this shortcoming.


(a)

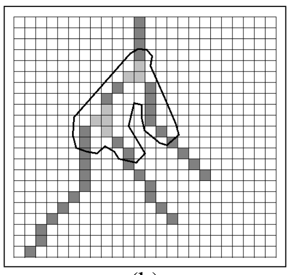

(b)

Figure 8. (a) Two separate regions (b) Two regions clumped into one.

Acknowledgements

RMC is grateful to FAPESP (2005/00587-5), CNPq (300722/98-2, 474596/2004-4, 491323/2005-0) and CAPES for financial support. LDFC is grateful to CNPq(308231/03-1) and FAPESP for sponsorship.
The authors are grateful to Southampton Archive for the rat hippocampal cells images hosted at: `http://www.compneuro.org/CDROM/nmorph/index/topindex_tn.html`.

References

[1] M. F. Carmo. *Differential geometry of curves and surface*. Prentice-Hall, Englewood Cliffs, N.J., 1976.

[2] F. Caserta, H. Stanley, W. Eldred, G. Daccord, R. Haussman, and J. Nittmann. Physical mechanisms underlying neurite outgrowth: A quantitative analysis of neuronal shape. *Physical Review Letters*, 64(1):95–98, 1990.

[3] R. M. Cesar-Jr. and L. F. Costa. Neural cell classification using wavelets and multiscale curvature. *Biological Cybernetics*, 79(4):347–360, 1998.

[4] L. da F. Costa. Bioinformatics: Perspectives for the future. *Genetic and Molecular Biology*, 3(4):564–574, 2004.

[5] L. da F. Costa. Morphological complex networks: Can individual morphology determine the general connectivity and dynamics of networks? 2005. http://xxx.lanl.gov/abs/q-bio.MN/0503041.

[6] E. R. Dougherty and R. A. Lotufo. *Hands-On Morphological Image Processing*. SPIE-International Society for Optical Engine, 2003.

[7] G. T. Herman. *Geometry of Digital Spaces*. Birkhauser Boston, 1998.

[8] H. Jelinek and E. Fernandez. Neurones and fractals: how reliable and useful are calculations of fractal dimensions? *Journal of Neuroscience Methods*, 81(1-2):9–18, 1998.

[9] K. Morigiwa, M. Tauci, and Y. Fukuda. Fractal analysis of ganglion cell dendritic branching patterns of the rat and cat retinae. *Neuroscience Research Suppl.*, 10:S131–S140, 1989.