Semi-Automated Dendrogram Generation for Neural Shape Analysis

ROBERTO MARCONDES CESAR JUNIOR
LUCIANO DA FONTOURA COSTA

Cybernetic Vision Research Group
GII - IFSC - University of São Paulo - São Carlos, SP
Caixa Postal 369, 13560-970, Brazil
e-mails: {pinda,luciano}@ifqsc.sc.usp.br http://www.ifqsc.sc.usp.br/visao

Abstract. This work introduces a new framework for shape characterization of neural cells by semi-automated generation of dendrograms, which are abstract structures describing different aspects associated to the branching structure of dendrites in neurons. Particularly, the dendrograms generated by the proposed method includes the length of each dendritic segment. The new method, which is based on multiscale curvature contour segmentation and syntactic shape analysis, is robust and greatly improve the tiresome task of dendrogram extraction, traditionally performed by human operators. Results corroborating the effectiveness of the technique are also presented.

Keywords: shape analysis; multiscale curvature; neuromorphometry; syntactic methods; dendrograms

1 Introduction

The analysis of the shape of neurons and neural structures has received growing attention as a consequence of the recent advances in computer vision techniques as well as substantial performance enhancements in off-the-shelf microprocessor systems. Largely overlooked by neuroscientists until the recent past, neuromorphometric studies of neural cells are essential for the proper characterization of constraints imposed by the physical boundaries of cell membrane over the functional properties of the respective neuron [Costa (1995)]. Among the many measures that have been considered for neural shape characterization [Costa (1997)], the dendrogram [Poznanski (1992); Schutter & Bower (1994); Turner *et al.* (1995); Velte & Miller (1995)] presents special importance because of the comprehensiveness it allows for the representation of the features of neural cells. Basically, dendrograms are binary trees representing the pattern of dendritic arborization of neurons, with their nodes corresponding to the branches in such arborizations. It should be observed that binary trees are indeed enough to represent such arborizations, since it is statistically impossible to have a triple branch, no mattering how close the departing segments may be (the triple branches eventually appearing in spatially sampled images can be split into two binary branches). Such a basic structure has often been extended to include additional information about the neural cells, such as the length and diameter of the dendritic segments, which are represented by the arches of the tree, the density of ion channels and average value of the electrical parameters along the segment, and even the axonal structure. Indeed, by including the angles between dendritic segments, dendrograms can be extended in order to express almost the totality of the large scale information contained in the 3D shape of most diverse neural cells. Of course, the full representation of the microscopic features of neurons, such as the exact position and type of channels not to mention the precise shape and position of every spine characterizing most excitatory cells, would be largely incompatible with the overall data structure and philosophy underlying dendrograms.

Frequently obtained from measures made painstakingly by human operators, dendrograms tend to present a certain degree of subjectiveness, not to mention the long time implied by such a process, which has been traditionally performed by human operators. The principal objective of the present paper consists precisely in developing a semi-automated technique for facilitating the generation of dendrograms from images of neurons obtained through optical microscopy. In addition to the branching structure, the considered dendrograms should also incorporate the length of every dendritic segment. Contrary to initial expectations, the problem of dendrogram extraction is complicated by ambiguities, as it will be discussed later in the present paper. Furthermore, the very process of determining the branch points is rather critical because of the variety of angles and curvatures in which such derivations can appear. Actually, the approach described here has only been possible as a consequence of recent developments in multi-scale contour analysis [Cesar & Costa (1996); Cesar & Costa (1997)] allowing

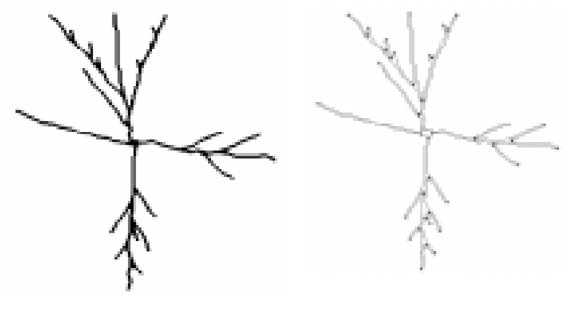


Figure (1): (a) Artificial neural cell; (b) Its respective contour.

the accurate determination of the branch points and extremities along the dendritic arborization, which is done by considering the curvature along such structures. Yet, in spite of its versatility and accuracy, some points may still be overlooked, thus implying the interactive involvment of an operator in order to validate the automatically extracted representation. A reasonably versatile graphic-interactive interface has been especially developed in order to assist this process. Preliminary assessments have indicated that operator intervention is required only in order to alter less than 20% of the obtained branch points and extremities. The problem of ambiguity has been addressed in terms of a grammatical approach where the branches are parsed according to decreasing distance between their respective extremities.

The present article begins by discussing the problems involved in dendrogram generation and proceeds by describing the curvature-based contour analysis used for the determination of the branch points and extremities, the graphic-interactive interface, and the parsing strategy used to solve the ambiguities during the dendrogram extraction. Once the dendrogram has been extracted, its is possible to straightforwardly obtain the skeleton of the respective cell, which is also described and commented upon. Experimental results corroborating the effectiveness of the overall approach are presented and discussed, and the prospects for further developments identified.

2 The Problem of Dendrogram Generation

The method introduced in this paper is based on contour analysis of neural cells. In order to gain insight into the different steps involved in the method, a dendritic whole branch will be used as example throughout the paper. Figure 1(a) presents the shape of a (synthetic) neural cell, while Figure 1(b) presents its respective outline. The right-most dendritic whole branch, which is shown in Figure 2, will be used as an example in the introduction and discussion of the algorithms.

Although the binary trees underlying dendrograms can be straightforwardly extracted from the one-pixel wide skeletons of the respective cells, such skeletons are often difficult to be determined because the cell body is irregular and thick (especially near the soma). The main problem is that thinning algorithms are broadly known to be relatively complex and do not guarantee optimal performance. Instead of relying on skeletons, the present approach has considered the analysis of the dendritic branches directly in terms of the outer boundaries of the cell, as illustrated in Figure 1(b) and 2.

Two special kind of critical or *dominant points* along such contours have been considered in the present article: the extremities, identified by the prefix *e*, and the branch points, identified by *b*, which are characterized as belonging to convex and concave portions of the contour (the concavity can be inferred

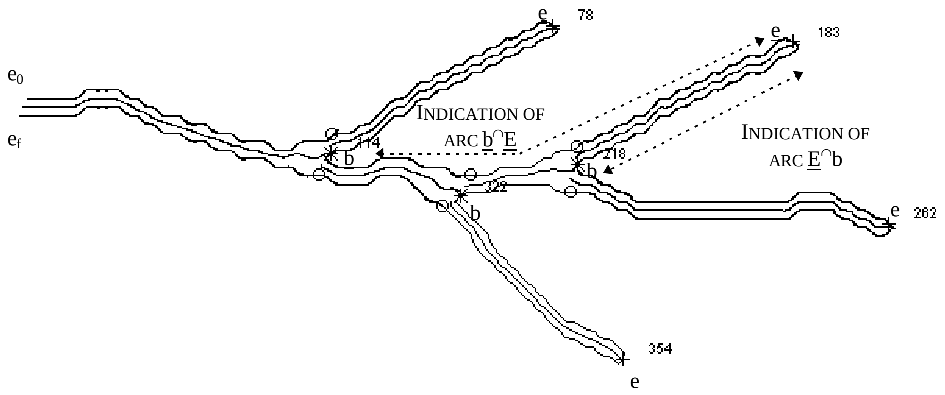

e 78
e 183
e_0
e_f
INDICATION OF ARC $\underline{b}^\frown\underline{E}$
114
b
218
b
INDICATION OF ARC $\underline{E}^\frown b$
322
b
e 262
354
e

Figure (2): The outer boundaries of a dendritic whole branch. Extremities correspond to the symbol “e” and are indicated by “+” in the Figure; branch points to the symbol “b” and are indicated by “*”; and secondary branch points are indicated by “○”.

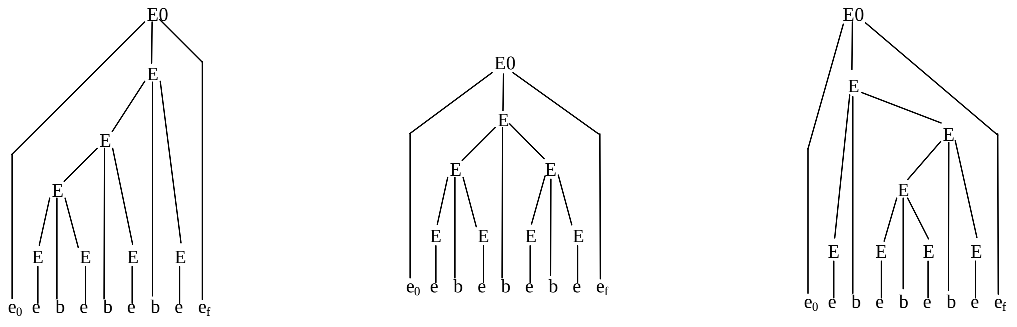


Figure (3): Three different parsing trees for the same string.

from the sign of the curvature at the contour points). Assuming that such points have been correctly identified, it is possible to think of the problem of dendrogram extraction in terms of a grammatical approach, more specifically through the parsing of a basic sequence defining the underlying structural atoms in dendrograms, namely the sequence extremity/branch-point/extremity. However, this parsing can not be performed directly, since there are ambiguities in such grammatical representations of dendritic whole branches. For instance, the same sequence as that describing the ramifications in Figure 2, namely $e_0ebebebee_f$, may have different parsing trees, which would imply different binary structures, as illustrated in Figure 3. The solution to such problems are described in the following sections.

3 Contour Segmentation

The methods presented in this section are based on the multiscale curvature methods (derived from the *curvegram*) introduced in [Cesar & Costa (1996); Cesar & Costa (1997)]. The framework presented in this work starts by representing the neural cell in terms of its outline contour, which is defined by a parametric curve $C(t) = (x(t), y(t))$, where the parameter t usually stands for the arc length parametrization of C. A complex representation of the contour is obtained through the definition of a complex signal $u(t)$, i.e. $u(t) = x(t) + i\,y(t)$. The Fourier transform pair of $u(t)$ is given in (1) and (2).

$$U(f) = F\{u(t)\} = \int_{-\infty}^{\infty} u(t)\, e^{-i\,2\pi f t}\, dt \qquad (1)$$

$$u(t) = F^{-1}\{U(f)\} = \int_{-\infty}^{\infty} U(f)\, e^{i\,2\pi f t}\, dt \qquad (2)$$

As it is well known, the Fourier transform presents the interesting *derivative property* (3).

$$\frac{d^{(n)}u}{dt^{(n)}}(t) \Leftrightarrow (i2\pi f)^n U(f) \qquad (3)$$

In particular, the first and the second derivatives of $u(t)$ can be defined in terms of $U(f)$ as expressed by equations (4) and (5).

$$\dot{u}(t) = \mathrm{F}^{-1}\left\{\dot{U}(f)\right\} = \mathrm{F}^{-1}\left\{i\,2\pi f\,U(f)\right\} \qquad (4)$$

$$\ddot{u}(t) = \mathrm{F}^{-1}\left\{\ddot{U}(f)\right\} = \mathrm{F}^{-1}\left\{-\,(2\pi f)^2\,U(f)\right\} \qquad (5)$$

The differentiation process defined by Equations 4 e 5 acts like high-pass filters, which can increase the high-frequencies noise commonly encountered in digital contours. In order to cope with such a problem, the differentiated spectra should be filtered with a low-pass filter to control the noise. By taking the gaussian $G_a(f)$ as low-pass filter, where $G_a(f) = exp(-(af)^2/2)$, the differentiated signals $\dot{u}(t,a)$ and $\ddot{u}(t,a)$ filtered at scale a can be given by (6) and (7).

$$\dot{u}(t,a) = \mathrm{F}^{-1}\left\{\dot{U}(f)\,G_a(f)\right\} \qquad (6)$$

$$\ddot{u}(t,a) = \mathrm{F}^{-1}\left\{\ddot{U}(f)\,G_a(f)\right\} \qquad (7)$$

The multiscale curvature description of $C(t)$, namely the *curvegram* [Cesar & Costa (1997)], is given by equation (8).

$$k(t,a) = \frac{-\,\mathrm{Im}\left\{\dot{u}(t,a)\,\ddot{u}^*(t,a)\right\}}{\left|\dot{u}(t,a)\right|^3} \qquad (8)$$

The curvegram $k(t,a)$ describes the curvature of $C(t)$ analyzed at scale a. The segmentation of the neural contour is carried out by choosing an appropriate scale a_0, followed by two threshold operations, one for the detection of the extremities of the neurons (defined by negative minima of curvature, which characterize convexities, assuming that the contour is traversed clockwise), and other for the detection of the branch points (defined by positive maxima of curvature characterizing concavities, assuming that the contour is traversed clockwise). It is important to note that, if the contour is traversed counterclockwise, it suffices to define the extremities as positive maxima and the branch points as negative minima. Furthermore, it is assumed that there is a minimum neighborhood around each dominant point (be it an extremity or a branch point) inside which there must be only one dominant point. This assumption implies a post-processing step in order to verify the minimum neighborhood around each dominant point. In the case that there is more than one of such points, the algorithm should choose only one. In our implementation, the algorithm chooses the more central dominant point among the points inside a common minimum neighborhood. The algorithm for dominant point detection can be summarized as follows:

```
1. Algorithm 1: dominant point detection
    2. Calculate $k(t,a_0)$, as defined by Equation 8, where $a_0$ is a constant;
    3. Find all branch points defined as $\mathbf{B} = \{u(t) \mid k(t,a_0) > T_B$, and $k(t,a_0)$ is a local positive maximum$\}$;
    4. Find all extremities defined as $\mathbf{E} = \{u(t) \mid k(t,a_0) < T_E$, and $k(t,a_0)$ is a local negative minimum$\}$;
    5. Filter $\mathbf{E}$ and $\mathbf{B}$, so that the minimum neighborhood around each extremity (or branch point) contain only one extremity (or branch point);
6. End.
```

Algorithm 1 requires the specification of four parameters, the analyzing scale a_0, the thresholds T_E and T_B and the size of the minimum neighborhood for dominant points. In all experiments, the minimum neighborhood for the digitized version of the contour has been set as three points on each side of the dominant point. The other three parameters are set by a semi-automated procedure, as explained in the following section.

4 GUI's and the Semi-Automated Approach

As it was explained at the end of Section 10, the segmentation algorithm for dominant point detection requires the specification of three parameters, the analyzing scale and the thresholds for the detection of extremities and branch points. These parameters are set by three semi-automated procedures that may require operator interventions. A fourth semi-automated procedure is also carried out, which asks the operator for final adjustments in the contour partitioning, both for the inclusion of missing and exclusion of false dominant points detected by **Algorithm 1**. The four procedures use the MatLab GUI resources ("Graphical User Interfaces"), and work as described in the following.

Procedure 1 (scale setting): A graphical window presents the original and the gaussian low-pass filtered contours to the user, which controls the amount of smoothness of the contour by a sliding control bar. There is a trade-off in this process since the filtered contour should be as smooth as possible but without rough distortions compared to the original shape. This procedure has a memory in the sense that the initial degree of smoothness is set up equal to the mean value scales of the two last executions of the program. It has been verified that such an approach is quite robust in the sense that the operator is seldom required to make adjustments. The same strategy is adopted by the *procedures 2* and *3*. The analyzing scale a_0 (**algorithm 1**) is taken to be the standard deviation of the gaussian filter corresponding to the smoothed contour;

Procedure 2 (extremity threshold setting): This procedure is analogous to *Procedure 1*, in the sense that a graphical window presents the contour with the extremities that are initially found indicated along the contour. The user is also prompted with a sliding control bar that allows him (or her) to control the threshold value. As the operator changes the threshold values, extremities are created and/or deleted, a process that is shown on-line in the graphical window. These extremities are found by step 4 of **algorithm 1**, where a initial value of T_E is set *a priori*. Again, this initial value is the average threshold that was the best choice in the last two executions of the program, i.e. the threshold that the program learned. Similar robust results are obtained by this strategy, and the operator rarely has to work hard in order to find a nice threshold.

Procedure 3 (branch point threshold setting): This procedure is the same as *Procedure 4*, except that *step 3* of **algorithm 1** is used in order to define the branch points.

Procedure 4 (final adjustment of dominant points): Although the contour partitioning is generally robust, false and missing dominant points are likely to result. *Procedure 4* allows the operator to check the segmented contour in order to correct such possible mistaken dominant points. The segmented contour is presented to the user in a graphical window, and two different symbols are used in order to distinguish extremities from branch points. The user is presented to a menu window that allows him (her) to perform any of the following 4 operations: include an extremity, include a branch point, delete an extremity, and delete a branch point. The operator executes the final adjustment by selecting the operation in the menu and pointing the desired point on the segmented contour. Finally,

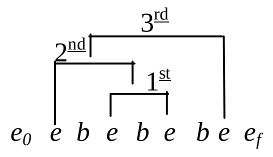


Figure (4): Correct derivation.

procedure 4 allows the operator to define each dendritic whole branch, each of which generally starting at the soma, by marking the initial and the final points of the whole branch. Each of the selected whole branches is analyzed in order to generate its respective dendrogram.

5 Parsing

The goal behind the parsing step is to derive a parsing tree that reflects the structure of the neural dendritic whole branch. The parsing step starts by labeling the dominant points obtained by the contour partitioning algorithm described in Sections 10 and 11, so that the initial and final points of the whole are labeled with the symbols "e_0" and "e_f", respectively; each extremity point is labeled by the symbol "*e*" and each branch point by the symbol "*b*". Therefore, we define a grammar G = (V_N, V_T, P, S), where V_N = {*E*}, V_T = { e_0, e_f, *e*, *b*}, S = {*E0*} and P consists of the following productions:

1. $E0 \rightarrow e_0\ E\ \ e_f$
2. $E \rightarrow E\ b\ E$
3. $E \rightarrow e$

As an illustration, the corresponding symbols have been indicated in the segmented whole branch shown in Figure 2. As it can be seen, the string representing that sub-tree is: "$e_0ebebebee_f$". The bottom-up parsing procedure is (seemingly) very simple, due to the simplicity of the grammar. A preliminary version of such an algorithm is:

1.**Algorithm 2:** Parsing (Version 1)

2. Change all the occurrences of "*e*" by "*E*", through the application of rule 3;

3. Change the occurrences of "*E b E*" by "*E*", through the application of rule 2;

4. Change the occurrence of "$e_0\ E\ \ e_f$" by "*E0*", through the application of rule 1;

5.End.

Nevertheless, there is an important problem associated with step 3 of **algorithm 2** because of the fact that different derivations may lead to the same string, which means that the order of application of rule 2 defines different parsing trees (e.g. see Figure 3). As a means of circumventing this ambiguity, a heuristic is applied in step 3 of **algorithm 2**. In order to understand how this works, imagine that each application of rule 2 of the grammar during the parsing procedure is equivalent to cut out a sub-branch corresponding to the respective "*EbE*" that is being substituted by "*E*". Then, the key idea underlying the heuristics is to construct the parsing tree by cutting the smaller, which are also the last, sub-branches first. Therefore, a simple analysis of the dendritic whole branch of Figure 2 indicates that the correct order of application of rule 2 is that presented in Figure 4. The parsing algorithm must decide whether or not to make the substitution $EbE \Rightarrow E$ once an "*EbE*" sub-string is found during the parsing. This decision, which is based on a comparison between the arc length of segments of the current sub-branch and the arc length of segments of its neighbor sub-branches, is taken as follows. First, once an "*EbE*" sub-string is found, which will be called *the current* "*EbE*" and denoted by "$\underline{EbE}$" , the algorithm tests whether there is another "*EbE*" sub-string to the right of "$\underline{EbE}$", where the second "*E*" of the first sub-string is the first "*E*" of the second, i.e. the union of the two sub-strings actually forms a sub-string "$\underline{EbE}bE$". In this case, the arc length between the first "$\underline{b}$"[1] and the middle "*E*" (indicated as "arc $\underline{b}^\frown\underline{E}$" in Figure 2) is compared to the arc length between this same middle "*E*" and the second "*b*" (indicated as "arc $\underline{E}^\frown b$" in Figure 2). In a similar manner, the algorithm checks whether there is another "*EbE*" sub-string to the left of "$\underline{EbE}$", where the union of these two sub-strings forms the sub-string "$Eb\underline{EbE}$". In this case the arc length of $b^\frown\underline{E}$ is compared to the arc length of $\underline{E}^\frown\underline{b}$; if (length($\underline{b}^\frown\underline{E}$) < length($\underline{E}^\frown b$)) and (length($\underline{E}^\frown\underline{b}$) < length($b^\frown\underline{E}$)), then the sub-string "$\underline{EbE}$" is replaced by "*E*". Otherwise, the algorithm skips the first sub-string and tries to substitute the second "*EbE*" sub-string, repeating the same test. These two comparisons are made assuming the existence of "$\underline{EbE}bE$" and "$Eb\underline{EbE}$". In the case when one (or both) of them are not present, the respective(s) test is just ignored, which occurs when the sub-branch does not have neighboring sub-branches to the right or to the left (or both). This heuristic, based on an information-crossing between symbolic and geometrical representations, induces the algorithm to attempt making substitutions corresponding to the smaller sub-branches first, as desired. Therefore, **algorithm 2** should be updated to version 2, which will be called **algorithm 3**:

1.**Algorithm 3:** Parsing (Version 2)

[1] The contour branch point corresponding to the "*b*", or any other symbol in a string, will be henceforth referred as "first "*b*" point" and so on, for simplicity's sake.

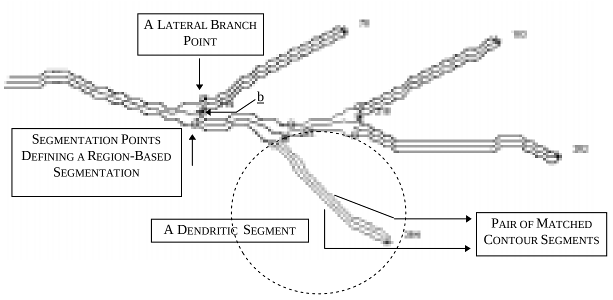

A Lateral Branch Point
b
Segmentation Points Defining a Region-Based Segmentation
A Dendritic Segment
Pair of Matched Contour Segments

Figure (5): Parts of the dendritic whole branch.

```
  2. Change all the occurrences of "e" by "E", through the application of rule 3;
  3. While there are remaining "EbE" sub-strings do:
     4. If a "$\underline{EbE}bE$" sub-string is found then
        5. If (length($\underline{b}\cap\underline{E}$) < length($\underline{E}\cap b$)) then
           6. Substitution_flag = true;
        7. Else
           8. Substitution_flag = false;
     9. Else
        10. Substitution_flag = true;
     11. If a "$Eb\underline{EbE}$" sub-string is found then
        12. If (length($\underline{E}\cap\underline{b}$) < length($b\cap\underline{E}$)) then
           13. Substitution_flag = true;
        14. Else
           15. Substitution_flag = false;
     16. Else
        17. Substitution_flag = true;
     18. If (Substitution_flag = true) then
         19. Change "$\underline{EbE}$" by "E", through the application of rule 2;
  20. Change the occurrence of "$e_0\ E\ e_f$" by "E0", through the application of rule 1;
21.End.
```

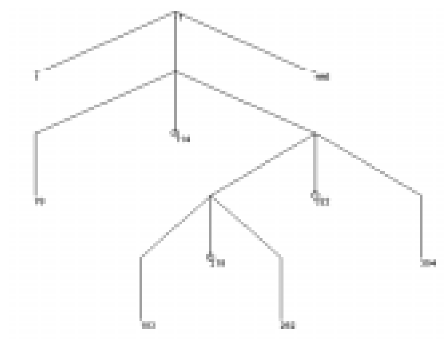
It is important to note that during its bottom-up execution, the algorithm should bookkeep, e.g. by using a string (or an equivalent data structure) to store the "upper" tree nodes, i.e. those nodes which do not have any parent node. This string, which is called *control string*, plays a central role in the parsing process, as well as in the determination of the lateral branch points, as it will be seen.

An additional important feature of the parsing procedure is that it allows the region-based segmentation of the neural cell in terms of its *parts*, i.e. each of its dendritic segments. Figure 5 presents a schematic illustration of the concepts involved in this segmentation process. First, note that each *dendritic segment* is defined in terms of its *branch point*, which is located by the (contour) segmentation algorithm introduced in Sections 10 and 11., and an other point, located "on the other side" of the dendritic segment, which will be called henceforth *lateral branch point*. The lateral branch points of the experiments of this work are denoted by "°", as illustrated in Figure 5. Each branch point is associated to two lateral branch points, since two dendritic segments appear from a branch point, and there is always an extremity or a dendritic sub-tree between the branch point and its respective lateral branch point. However, lateral branch points may not be easy to locate, since usually they do not present special features (such as being a curvature maxima). Nevertheless, the control string provides a straightforward heuristics that can be applied to define and locate the lateral branch point. Recall that the dominant points found by the segmentation algorithm define a contour partitioning in terms of segments between each pair "*eb*" (extremity-branch point) or "*be*" (branch point-extremity), and that the initial control string of the dendritic whole branch of Figure 2 and 5 is $e_0ebebebee_f$. Therefore, the left lateral branch point of the first branch point of this dendritic whole branch (i.e. $e_0e\underline{b}ebebee_f$) is defined as the nearest point belonging to the segment comprised between the next two symbols to the left of $\underline{b}$, i.e. between e_0 and *e*. Both the branch point $\underline{b}$ and its left lateral branch point are indicated in Figure 5. All other lateral branch points can be located during the parsing procedure in a similar manner, by searching for the nearest point (to the current branch point) belonging to the segment defined between the next two symbols to the left (or to the right) of the control string.

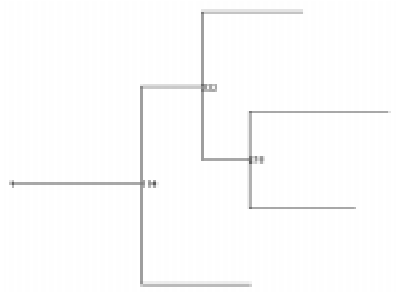
Finally, it is important to observe that each dendritic segment is defined by a *pair of matched contour segments* or *arcs*, each of them defined in terms of a branch point and a lateral branch point, which establish the beginning of the dendritic segment, and an extremity or a dendritic sub-tree, which identifies the end of the dendritic segment. An example of such construction can be found in Figure 5.

6 The Dendrogram

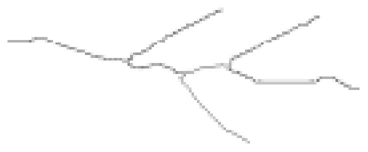
The dendrogram is obtained from the parsing tree in a straightforward manner. Recall that the dendrogram describes the distance between the successive branch points and extremities. It is also fortunate that the parsing procedure explained in the last section generates a parsing tree that respects the smaller-to-larger branches in a dendritic whole branch. Furthermore, the branch points and the extremities can be retrieved in an ordered manner from the dendritic whole branch, once each node of a symbol "*E*" is the father of either a node



(a)



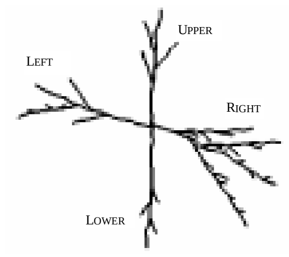
(b)

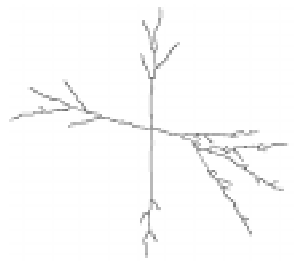


(c)

Figure (6): (a) Parsing tree; (b) Dendrogram; (c) Skeleton. All for the whole branch of Figure 2.

“*b*” (i.e. a branch point) or of a node “*e*”. Therefore, the dendrogram can be obtained by traversing the parsing tree, starting from the initial parent node, and calculating the arc length distance between the current node and its children, i.e. the length of each branch. It is equally important to note that each branch is composed by a pair of matched contour segments (see last section). Hence, the length of each branch is defined as the mean between the arc lengths of the pair of matched segments. In order to understand this process, refer to Figures 6(a) and (b), which present, respectively, the parsing tree automatically generated by the algorithm and the corresponding dendrogram of the dendritic main branch of Figure 2. The first dendritic segment is that beginning at the first and the last points of the dendritic whole branch and finishing at the lateral branch points associated to the branch point for *t=114* (see Figure 2). The algorithm then draws a horizontal straight line with length proportional to the length of this first branch, as it is shown in Figure 6(b). Two new branches appear at this first branch point and the dendrogram is ramified. The branch to the left of the





(a) (b)

Figure (7): A more complex artificial neural cell.

parsing tree finishes at the extremity for *t=78*, defining a new pair of matched segments, and the algorithm a draws a new horizontal straight line with length proportional to the length of this branch. The algorithm continues recursively until all branches have been analyzed.

7 Obtaining the Skeleton

A welcomed byproduct of the parsing scheme defined in the present work is the skeleton [Ballard & Brown, 1982] of the neural cell, which can be obtained in a straightforward manner. As it was highlighted, each branch is composed by a pair of matched contour segments. Therefore, the skeleton of each branch may be defined as the curve whose points are equidistant to each segment that composes the branch. In other words, let $u_l(t)$ and $u_r(t)$ be the pair of (reparametrized) matched segments of a generic branch, where “u_l” and “u_r” stands for the left and the right segment, respectively. Then the skeleton $\xi(t)$ of this branch is:

$$\xi(t) = \frac{u_l(t) + u_r(t)}{2}$$

The respective skeleton of the dendritic whole branch is also presented in Figure 2 or 6(c).

8 Experimental Results

Figure 7 and 8 present additional experimental results obtained for a more complex cell, whose contours are depicted in Figure 7(b). This cell has 4 major dendritic whole branches, which are referred to as *left tree*, *upper tree*, *right tree* and *lower tree*, as indicated in Figure 7(a). The dendrograms of each of these trees are shown in Figure 8. As it can be readily inferred from the complexity of this cell, the problem of manually dendrogram generation may become a tiresome and lingering task, which can be thankfully speeded up by the semi-automated method introduced in this work.

9 Conclusions

This work has presented a new approach to shape analysis of neural cells based on the semi-automated generation of dendrograms. The proposed framework, which relies on a multiscale curvature-based segmentation of the neural contour followed by syntactic analysis of the partitioned shape, has proved to be robust and to greatly improve the dendrogram generation process. Further refinements on the method will include the introduction of more intelligent mechanisms to help the operator in procedure 4, as well as automatically checking for valid strings generated by the segmentation algorithm. Other applications, such as characterization of plants and trees, are also being considered.

References

[Ballard & Brown, 1982] D.H. Ballard and C.M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Cesar & Costa (1996)] R.M. Cesar Jr. e L. da F. Costa, Towards Effective Planar Shape Representation with Multiscale Digital Curvature Analysis based on Signal Processing Techniques, *Pattern Recognition*, 29 (1996), 1559-1569.

[Cesar & Costa (1997)] R.M. Cesar Jr. and L. da F. Costa, The Application and Assessment of Multiscale Bending Energy for Morphometric Characterization of Neural Cells, *Review of Scientific Instruments*, 68 (1997), 2177-2186.

[Costa (1995)] L. da F. Costa, Computer Vision based Morphometric Characterization of Neural Cells, *Review of Scientific Instruments*, 66 (1995), 3770-3773.

[Costa (1997)] L. da F. Costa, *Novas perspectivas em neuromorfometria e neuromodelagem*. IFSC, Universidade de São Paulo, jan. 1997.

[Poznanski (1992)] Roman R. Poznanski, Modelling the Electronic Structure of Starburst Amacrine Cells in the Rabbit Retina: Functional Interpretation of Dendritic Morphology, *Bulletin of Mathematical Biology*, 54 (1992), 905-928.

[Schutter & Bower (1994)] E. De Schutter and J. M. Bower, An Active Membrane Model of the Cerebellar Purkinje Cell I. Simulation of Current Clamps in Slice, *Journal of Neurophysiology*, 71 (1994), 375-400.

[Turner *et al.* (1995)] D.A. Turner, X.-G. Li, G.K. Pyapali, A. Ylinen and G. Buzsaki, Morphometric and Electrical Properties of Reconstructed Hippocampal CA3 Neurons Recorded In Vivo, *The Journal of Comparative Neurology*, 356 (1995), 556-580.

[Velte & Miller (1995)] T.J. Velte and R.F. Miller, Dendritic Integration in Ganglion Cells of the Mudpuppy Retina, *Visual Neuroscience*, 12 (1995), 165-175.

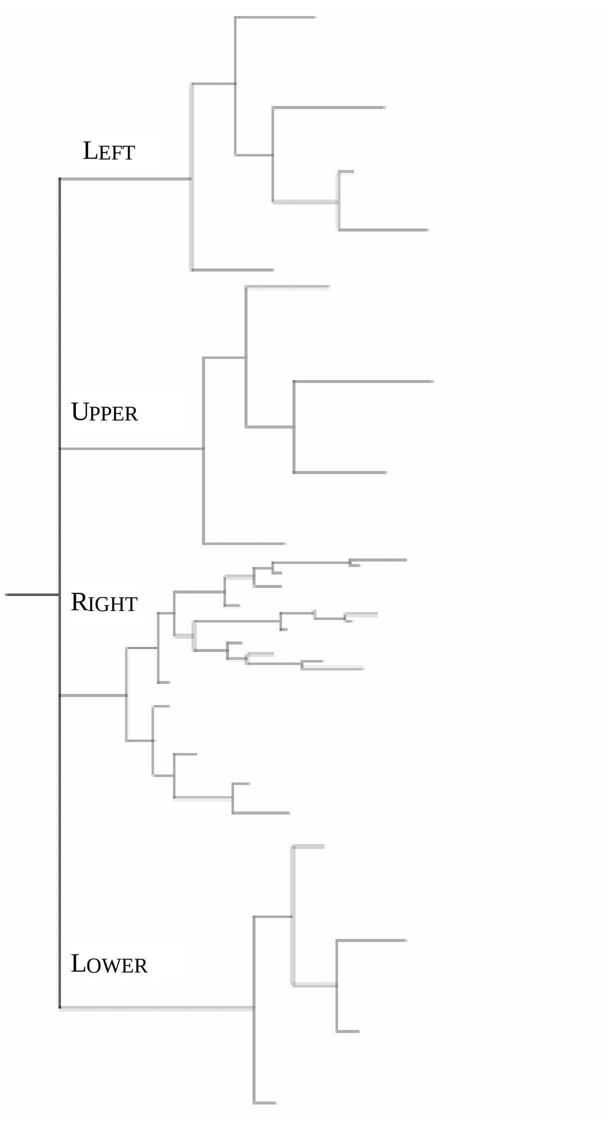


Figure (8): Dendrogram of the neuron in Figure 7.

Acknowledgements: Luciano da F. Costa is grateful to FAPESP (94/3536-6 and 94/4691-5) and CNPq (301422/92-3) for financial help. The authors are indebted to Regina C. Coelho for supplying of the images of artificial neurons. Roberto M. Cesar Jr. in grateful to J.-P. Antoine (FYMA-UCL-Belgium) for the partial computational support and to CAPES.