Automatic Branch Decomposition for Tubular Structures

Release 0.1

Guanglei Xiong[1], Lei Xing[2] and Charles Taylor[3]

Apr 20, 2009

[1]Biomedical Informatics Program, Stanford University
[2]Department of Radiation Oncology, Stanford University
[3]Department of Bioengineering, Stanford University

Abstract

Branches of tubular structures (vasculature, trachea, neuron, etc.) in medical images are critical for the topology of these structures. In many applications, It is very helpful to be able to decompose tubular structures and identify every individual branch. For example, quantification of geometric vascular features, registration of trachea movement due to respiration, tracing of neuron path. However, manual decomposition can be tedious, time-consuming, and subject to operator bias. In this paper, we propose a novel method to decompose tubular structures automatically and describe how to implement it in ITK framework. The input is a 2D/3D binary image that can be obtained from any segmentation techniques, as well as the junctions, which can be generated automatically from our previously contributed ITK class: `itk::JunctionDetectionFilter`. The output will be branches with their labels and their connection. There are only two parameters which need to be set by the user. We provide here the implementation as a ITK class: `itk::BranchDecompositionFilter`.

Contents

1 Introduction

Branches of tubular structures in medical images are critical for their topology. These structures include vasculature, trachea, neuron, etc. Decomposition of them and identification of every individual branch is helpful in many applications. For example, the branches of vessels can be used to define vascular trees and quantify features of vessel segments, such as length and diameter. Tracking of branches in trachea could enable more robust registration of movement due to respiration. Branch enables the tracing of the paths of neurons that can be useful to study their growth. However, manual identification of branches can be tedious, time-consuming, subject to operator bias, and intractable when many are present. To our knowledge, there is no free and open-source software that are readily available for this task. In this paper, we propose a novel method to decompose tubular structures them automatically and construct branch connections based on a list of junctions. The junctions can be generated automatically using our previously contribution [1]. We describe how to implement it in ITK framework [2]. In the end, we introduce how to use the implementation and demonstrate our results.

2 Design

Decomposition of tubular structures are not easy for several reasons. First, the patterns of branches in tubular structures can be very diverse. They can be straight, curved, or even loopy. Second, the branches are connected to each other through junctions, which have topologically different from the former. Furthermore, it is usually necessary to identify branches of different sizes simultaneously. We designed our method to take these challenges into account.

The basic idea of our approach is simple. We use user-provided junctions shown as dark color in Fig. 1 to split the tubular structure into individual branches. we mark voxels within all the junctions. Then, we treat each connected component as individual branch for the rest of voxels in the image. Clearly, if the provided junctions are appropriate, the branch should be connected to one or two junctions. For example in Fig. 1, branches 1 and 2 connects only one junction: J1. Similarly, branches 4 and 5 has one junction J2 as their neighbor. Branch 3 connects to both J1 and J2. As you can imagine, branch decomposition will be incomplete if the user does not provide junction J2. In this case, the branches 3, 4, and 5 are considered as one single "branch". In contrast, branch decomposition will be over-complete if the user provides another junction J3, e.g. in the middle of branch 3, to divide this branch. Notice that our current method labels voxels to be within either junction or branches. That is to say, we do not divide junctions based on their connected branches.

We further develop the basic idea by describing the details of our method that are designed to better handle the difficulties mentioned before. First, the user provides a input binary image and a list of junctions with their centers c_{jc} and sizes r_{jc}. We define $r_{\mathrm{in}} = \alpha r_{\mathrm{jc}}$ and $r_{\mathrm{out}} = \beta r_{\mathrm{jc}}$, where $\alpha < \beta$ are user-defined parameters. The variable r_{in} is used as the distance to c_{jc}, within which the voxels are considered as junction voxels. We use the geodesic distance as opposed to the Euclidean distance because the former is measured within the tubular structure. Second, for each junction, we label voxels between r_{in} and r_{out} away from the center as branch voxels, one branch label for each connected components. For example in Fig. 1, The junction J1 will lead to three labels, similar to junction J2 but with three different labels. Then, the labels for branch voxels are propagate to their neighboring unlabeled voxels until all voxels have been labeled. Clearly, we will have to merge the labels if one voxel has a neighboring voxel with a different branch label. For instance, this issue will happen in branch 3 in Fig. 1. The branch label, which has the

number of voxels fewer than the minimal number of voxels $n_{\min}$, with its original junction label. Finally, we will resort the branch labels to be consecutive and find their connected junction labels.

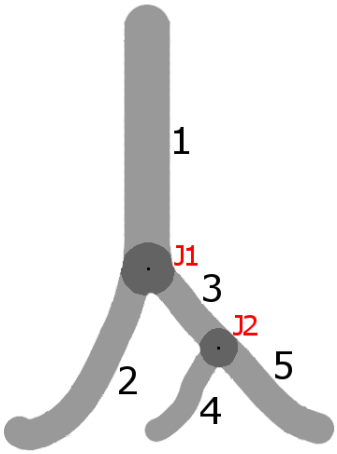

Figure 1 The basic idea of our branch decomposition method is to use the junctions (J1 and J2) to split branches (1, 2, 3, 4, and 5).

The flowchart of our method is summarized in Fig. 2.

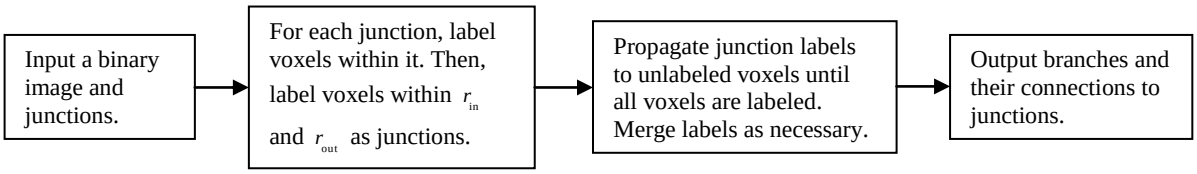

Figure 2 The flowchart of our branch decomposition method.

3 Implementation

We implemented the method as a ITK class: `itk::BranchDecompositionFilter`, which is subclass of `itk::ImageToImageFilter`. It is templated by the input image type and controlled by four functions:

- **[Set/Get]InnerRadius**: define α.
- **[Set/Get]InnerRadius**: define β.
- **[Set/Get]MinNumberOfPixel**: define $n_{\min}$.
- **SetJCLabelMap**: input junction information.

The junction information is provided by a std::map: JCLabelMapType, defined as:

```
typedef std::pair< IndexType, float >          JCLabelPairType;
typedef std::map< long, JCLabelPairType >      JCLabelMapType;
```

The input image should be a binary image with zero background and nonzero foreground. The output image is an image with branches and junctions marked by their labels. The information of branches and their connections to junctions can also be queried by a std::map: BRJCConnectMapType, defined as:

typedef std::set< long > BRJCSetType;
typedef std::map< long, BRJCSetType > BRJCConnectMapType;

Each element in the map is a branch with the branch label as its key and the set of connected junctions.

The core of our method is in the function of `GenerateData()`.

4 Usage and Examples

It is very easy to use the class. It behaves like a common filter which has one input and one output.

First, a 3D image of the Circle of Willis with identified branches (47 in total) is shown in Fig. 3.

The input junctions (slightly modified from the output of Ref. [1]) are

jcLabel	jcIndex[0]	jcIndex[1]	jcIndex[2]	jcRadius
1	218	129	193	1.1225
2	196	111	213	1.4156
3	140	149	260	0.6708
4	269	159	264	0.6708
5	148	50	277	0.8485
6	190	53	279	0.9950
7	213	38	283	0.7348
8	225	39	285	0.5196
9	292	61	285	0.9000
10	242	80	285	1.1225
11	214	120	288	1.2416
12	237	98	286	0.6708
13	177	65	291	1.1225
14	122	27	292	0.6708
15	193	110	299	0.6708
16	333	66	304	0.9000
17	343	66	315	0.6708
18	103	43	334	0.5843
19	73	124	343	0.8485
20	97	33	348	0.5843
21	223	29	349	0.6000
22	66	108	352	2.9699
23	22	200	361	0.5196
24	232	14	362	0.6000
25	82	191	365	0.5843

The code snippet is

```
DecomposerType::Pointer decomposer = DecomposerType::New();
```

```
decomposer->SetInnerRadius(2.5);
decomposer->SetOuterRadius(3.0);
decomposer->SetMinNumberOfPixel(16);
decomposer->SetJCLabelMap(&jcLabelMap);
decomposer->SetInput(inputImage);
decomposer->Update();
```

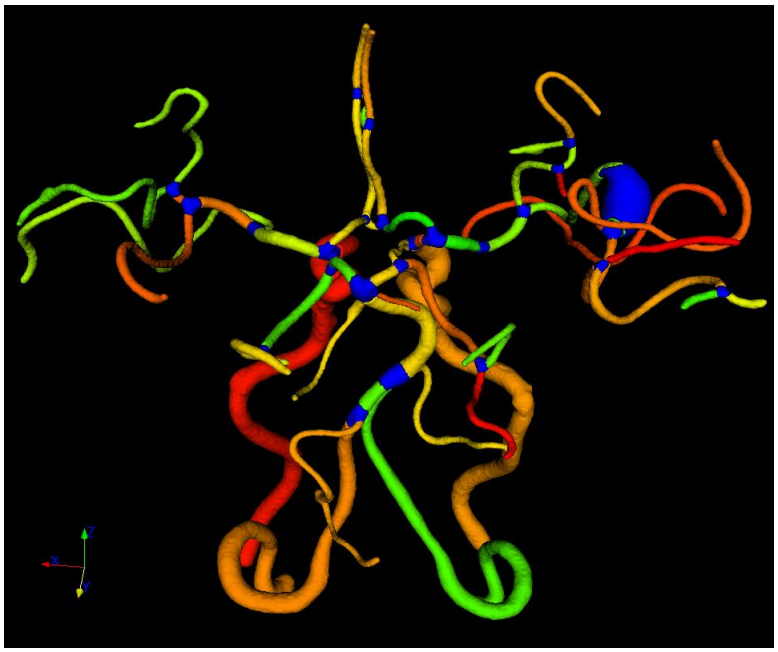

Figure 3 The Circle of Willis with its junctions (in blue) and branches (other than blue).

Second, a 3D image of trachea with identified branches (20 in total) is shown in Fig. 4.

The input junction file can be downloaded online in the Data section.

The branches were identified using the same code and parameters as in the Circle of Willis.

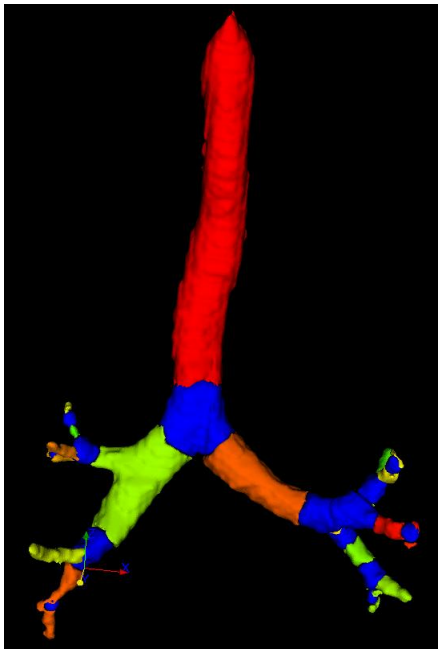

Figure 4 The trachea with its junctions (in blue) and branches (other than blue).

Finally, we tested the method on a 2D neuron image as seen in Fig. 5.

Total number of identified branches: 96.

The branches were identified using the same code (in 2D) and parameters as previous.

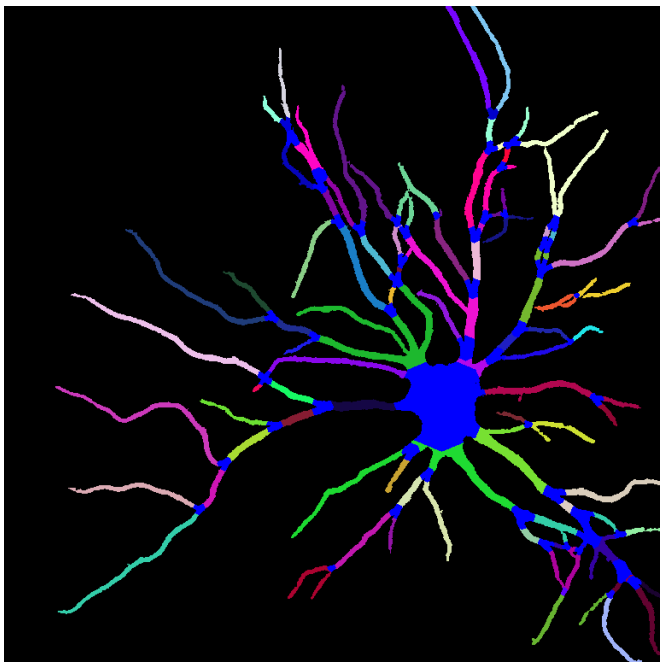

Figure 5 The neuron with its junctions (in blue) and branches (other than blue).

5 Software Requirements

This paper has described a method of automatic branch decomposition. The method requires a minimal number of parameters and works both in 2D and 3D. The implementation is ready to be used within the ITK framework. In the future work, we hope to speed up the method by parallelism. In addition, the junction could be divided further by assignment of its voxels to neighboring identified branches. For suggestions or bugs, feel free to contact us[1].

6 Software Requirements

You need to have the following software installed:

- Insight Toolkit 3.8.0
- CMake 2.4

Acknowledgement

We would like to thank Nan Xiao for providing the segmentation of the Circle of Willis. Guanglei Xiong was supported by a Stanford Graduate Fellowship.

Reference

[1] G. Xiong, L. Xing, and C. Taylor. Automatic Junction Detection for Tubular Structures. *The Insight Journal*, http://www.insight-journal.org/browse/publication/324, Jan-Jun, 2009.

[2] L. Ibanez and W. Schroeder. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, 2003.

[1] Corresponding author: Guanglei Xiong: guangleixiong at gmail.com