Principal Component Analyses For Tree Structured Objects

Burcu Aydın

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Statistics and Operations Research.

Chapel Hill
2009

Approved by,

Elizabeth Bullitt

Shu Lu

J.S. Marron

Andrew Nobel

Gabor Pataki

Scott Provan

Abstract

BURCU AYDIN: Principal Component Analyses For Tree Structured Objects
(Under the direction of Gabor Pataki and J.S. Marron)

This study is in the relatively new statistical area of Object Oriented Data Analysis, which considers general data objects (3D images, movies, etc) as the atoms of interest. The focus is on populations of tree-structured objects. Due to the highly non-Euclidean properties of the binary tree space, replacing classical analysis ideas with their counterparts in this new environment is a challenging task.

Ideas analogous to Principal Component Analysis (PCA) for trees have been previously developed based on *tree-lines*. In this work, numerically fast (linear time) algorithms are developed for PCA based tree-lines which enable the first large scale data analysis of trees. Our analysis of tree-line PCA has lead to the invention of improved Principal Component Analyses, based on the new concepts of *k-tree-lines* and *tree-curves*.

The tree-line analysis results give promising results. However, many tree-lines are required to explain most of the variation in the data.

The idea of tree-curves directly targets the drawback of tree-lines. However, no polynomial-time optimal algorithm to find the optimal tree-curves exists. The heuristics developed give results that explain more variation than was observed previously.

The k-tree-line study is proposed as a bridge between tree-line and tree-curve ideas. Polynomial time algorithms are sought for this group of problems.

These three different proposed PCA methods are used to conduct a study to compare the three existing data sets and measure the age effect on each subpopulation within the sets. The advantages and shortcomings of each method with respect to each other are also discussed in the context of the data analysis.

The motivating data set of this study is a collection of the brain vessel structures of 105 subjects. Due to the inaccuracies in scanning and tracking of these vessels, this data set is known to include a high amount of noise. A detailed visualization method is proposed in this work to spot the instances that require manual cleaning or need to be excluded.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Motivation: The Blood Vessels Problem

The first motivating example for this study comes from a study of Magnetic Resonance Angiography (MRA) brain images of a set of 73 human subjects of both sexes, ranging in age from 18 to 72, collected by Dr. E. Bullitt of the CASILAB (casilab.med.unc.edu). More recently, an improved version of this set had become available with an addition of 34 more subjects.

One slice of an MRA image is shown in Figure 1.1. As a distinction from more widely known MRI, this mode of imaging indicates strong blood flow as white. These white regions are tracked in 3 dimensions, then combined, to give trees of brain arteries.

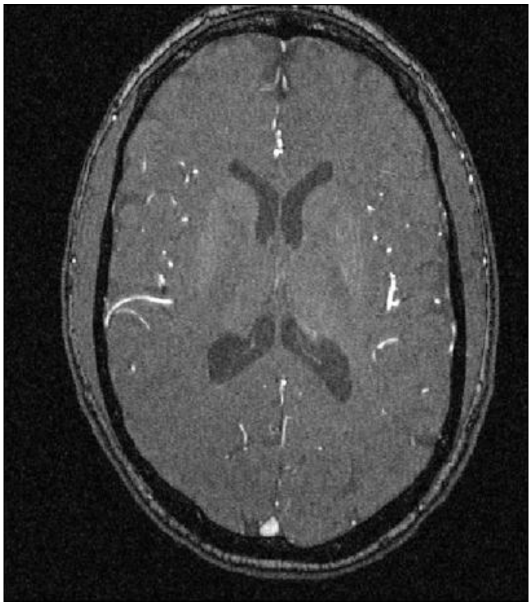

Figure 1.1: Example MRA Image

Single Slice from a Magnetic Resonance Angiography image for one subject. Bright regions indicate blood flow.

The set of trees developed from the image of which Figure 1.1 is one slice is shown in Figure 1.2. These figures can be found at (1). Trees are colored according to the regions of the brain. Each region is studied separately, where each tree is one data point in the data set of its region. The goal of the present OODA (Object Oriented Data Analysis) is to understand the population structure of 107 subjects through 4 data sets extracted from them. For every subject, the brain vessel structure mainly consists of four systems that feed the four main areas (back, left, right and back) of the brain. Each of these systems needs to be analyzed separately. Therefore the vessel data from each subject is divided into four parts, constituting the four data sets mentioned. These sets are the back data set (gold trees), left data set (cyan), right data set (blue) and front data set (red).

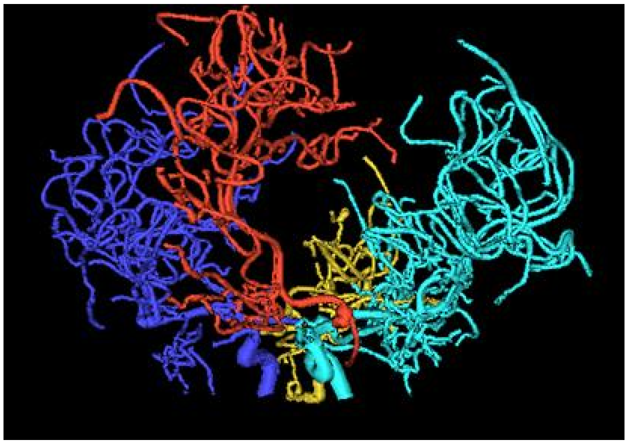

Figure 1.2: Reconstructed set of trees of brain arteries for the same subject as shown in Figure 1.1. The colors indicate regions of the brain: Gold (back), Right (blue), Front (red), Left (cyan).

The stored information for each of these trees is quite rich (enabling the detailed view shown in Figure 1.2). Each colored tree consists of a set of branch segments. Each branch segment consists of a sequence of spheres fit to the white regions in the MRA image (of which Figure 1.1 was one slice), as described in Aylward and Bullitt (2002). Each sphere has a center (with x, y, z coordinates, indicating location of a point on the center line of the artery), and a radius (indicating arterial thickness).

These reconstructed sets of trees of brain arteries for all subjects are available at Handle(2008).

1.2 Modeling The Data

One big challenge with this data is that it contains too much information to handle at once. One needs to simplify the structure for meaningful analysis results. The approach taken here is to start out with considering only variation in the *topology* of the trees, i.e. the branching structure. Other aspects of the data, such as location, thickness and curvature of each branch are initially ignored. With the removal of all information other than branching structure, the data turns into four subpopulations of *binary trees*.

The way the binary trees are extracted is as follows: For each of the instances (brain scans), the back, left, right and front regions are handled separately. Each of these subsystems usually consist of one main (root) vessel entering the brain from below, and splitting into smaller branches to feed that region of the brain. The portion of the root vessel until a branch splits off is taken as the root node, and the two vessels that take place after the split are the left and right children nodes of the root. The same procedure is applied at each juncture point. In the end, a binary tree where each vessel trunk between two split points in the original structure correspond to a node is obtained.

One question at this point is, can a vessel split into three or more branches at a single point? In this data set splitting into more than two branches is very rare, and these occurrences do not carry any important implications on structure. Therefore keeping the simple binary structure seems more important than capturing these rare occurrences. In the cases that this happens, one of the child vessels is randomly selected as the first one to split off, and the binary tree is created accordingly.

In some of the instances, there can be two root vessels feeding one region and the explained procedure yields two binary trees. In these cases, for simplicity, one phantom root node is added and the roots of these two trees are connected to the phantom root node as children, so that a single binary tree is obtained for each instance in the subpopulations.

There is one set of ambiguities in the construction of the binary trees. That is the choice, made for each adult branch, of which child branch is put on the left, and which is put on the right. As mentioned in , the following two ways of resolving this ambiguity are considered here. Using standard terminology from image analysis, the word *correspondence* is used to refer to this choice.

- **Thickness Correspondence:** Put the node that corresponds to the child with larger median radius (of the sequence of spheres fit to the MRA image) on the left. Since it is expected that the fatter child vessel will transport the most blood, this should be a reasonable notion of *dominant branch*.

- **Descendant Correspondence:** Put the node that corresponds to the child with the larger number of descendants to the left.

These correspondences are compared in Subsection 2.2 using tree-line analysis methods.

Other types of correspondence, that have not yet been studied, are also possible. An alternative approach, suggested by Marc Niethammer, is to use location information of the children in this choice. E.g. in the back tree, one could choose the child which is physically more on the left side (or perhaps the child whose descendants are more on average to the left) as the left node in this representation. This would give a representation that is physically closer to the actual data, which may be more natural for addressing certain types of anatomical issues.

It should be noted that although the size of a data tree remains the same, the inner structure may be greatly affected by the correspondence choices. As will be illustrated in Section 2.2, the analysis results may differ from one choice to another. Therefore it is important to make a careful decision when choosing correspondence, and pick the option which captures most of the important aspects of data.

Throughout this study, more that one data set of brain vessels have been available. These can be listed as follows:

- **Data Set 1:** This is the first set that had been available for analysis. It consists of 73 subjects. The back, left and right vessel subpopulations of this data set are used in analysis. This is because the source of flow for the front trees is variable, causing confusion in separating the sub-populations. Due to this problem, the front subpopulation of this set is not studied.

- **Data Set 2:** Through a careful study by Dr. Alim Ladha, many of the flow identification and starting point problems present in Data Set 1 are solved and 34 more subjects are added to the study, constituting Data Set 2. The front subpopulation is included in analyses run on this set.

- **Data Set 3:** It is known that there are still many discrepancies exist in Data Set 2. The data cleaning methods proposed in Section 5.4 aim to reduce these problems. The further cleansed set that is obtained by the methods described in Section 5.4 is labeled as Data Set 3. The final data analyses are run on this set.

The raw data (as in Figure 1.2) belonging to all subjects is publicly available at the website Handle(2008).

1.3 Background and Related Work

1.3.1 Tree Structure Studies In Literature

Phylogenetics is the study of evolutionary relations among species. To study these relations, *phylogenetic trees* are constructed which represent the evolutionary steps occurred in time. The branches on these trees are constructed according to the likeliness (common genes, etc) between species. See Li et. al. (2000) and Holmes (1999) for examples of use of trees in phylogenetics.

Cluster analysis is another important area to be listed. Here, trees are formed to again decide which objects are closer to each other and thus may be grouped together. Here, each object is a leaf node in the cluster tree, and the interior nodes are used to express the structural relations between the objects.

However, these interior nodes do not have a physical meaning. The studies in cluster analysis are usually based on forming different cluster trees through different methods and then seeking to do inference. See Everitt et. al. (2001) and Breiman (1996) for examples.

In the classification and regression tree (CART) analysis, data objects are categorized through a decision tree. The set of all objects constitute the root node. For each decision rule, (questions asked to patients, etc.) the set is partitioned into further groups (child nodes) according to the responses to that decision rule. See Breiman et. al. (1984) for a detailed study of this group of trees.

These areas listed are interested in building a particular tree to explain relations between various objects. They do not focus on the population analysis of a set of trees. Another area, in which statistical analysis of such populations are of interest is medical imaging. Shape analysis of various objects obtained through medical scanning images is a prominent area on OODA research.

An important shape representation class is *m-reps*. M-reps capture shape by dividing it into coarse or fine parts, each of them containing various information (location, radius and angles) in the form of feature vectors. M-reps provide a good shape representation for shapes that are not far from convex. For more information on the method, see Pizer et. al. (1999) and the MIDAG web-site.

When the objects to be analyzed are close to convex, carrying the differences into feature vectors and run a comparative analysis of these vectors (as in M-reps) is a powerful approach. Same methods will work when objects are further from convexity, but the members of the population have an underlying common structure. An example to this is a population of human hands (See Haonan Wang's 2003 dissertation for the details of this example). Although the shape of a hand is very far from convex, all hands have a common structure consisting of a palm and the five fingers. Each of these components can e represented by an m-rep and the results can be collected in one vector, enabling the classical statistical methods to be run on the data.

When the data present different structures from instance to instance, this approach will not be useful. In our motivating example, the brain vessel trees do not necessarily have same number of vessel trunks. It is not possible to match one vessel in one instance with another vessel in another subject's brain as the corresponding vessel, and make a comparison. Although all parts of the brain are fed with similar systems for each instance, the exact branching structure that achieves this changes from person to person. Therefore, an approach which focuses on underlying graph-theory based structures is needed instead of a shape-focused approach.

Another approach in analyzing tree structured populations is kernel methods, widely used in pattern analysis. This approach is based on mapping complex structured objects into a high dimensional feature space, which is Euclidean. Usually finding the exact corresponding point on the feature space for each data point is hard, so, the idea of *kernel functions* is used instead. This function computes the inner products of each pair of data points in the feature space instead of finding the coordinates themselves, and statistical analysis is run through these inner products. For more information on these methods, see Shawe-Taylor and Christianini (2000).

1.3.2 Principal Component Analysis

One of the most common methods used to analyze high dimensional data sets is Principal Component Analysis. In its most general sense, Principal Components are directions that explain structure in data. In this research, finding the analogues of classical Principal Component Analysis (PCA) plays an important role, so the main ideas will be explained before going into its counterparts in this research.

PC analysis is introduced by K. Pearson in his 1901 paper, in which he aimed to find lines and planes that best fit a set of points in p-dimensional space. The next important work given on the subject was Hotelling (1933), in which Hotelling aimed to find a smaller ($< p$) fundamental set of independent variables that maximize their successive contributions to the total of the variances of the original variables. He also introduced the term ”Principal Components”. To further follow the historical development of PCA ideas, see Girshick (1936,1939), Rao (1964), Gower (1966), Jeffers(1967) and Preisendorfer et. al. (1988).

PCA seeks a basis of the space, where the data reside, such that the vectors of the basis explain a maximal amount of variation in the data. As a first step, a direction vector, where the greatest variation is achieved when the data points are projected onto it, is calculated. This direction is called the First Principal Component, and it explains the maximum amount of variation possible in one dimension. An example can be seen in Figure 1.3. Both Figures 1.3 and 1.4 can be found in lecture notes web site (34) by Prof. J.S. Marron.

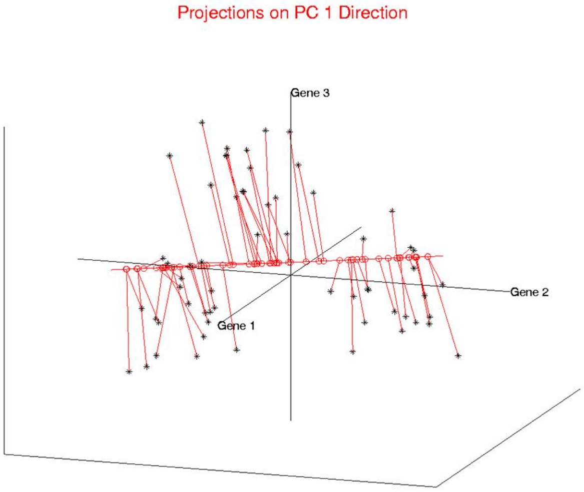

Figure 1.3: First Principal Component of a three dimensional data set, with data points projected onto it.

On second and subsequent steps, the most important directions of variation are sought again, but the vectors defining them are required to be orthogonal to those identified previously. The result at the k^{th} step is called the k^{th} Principal Component. The Second Principal Component of the previous example is illustrated in Figure 1.4.

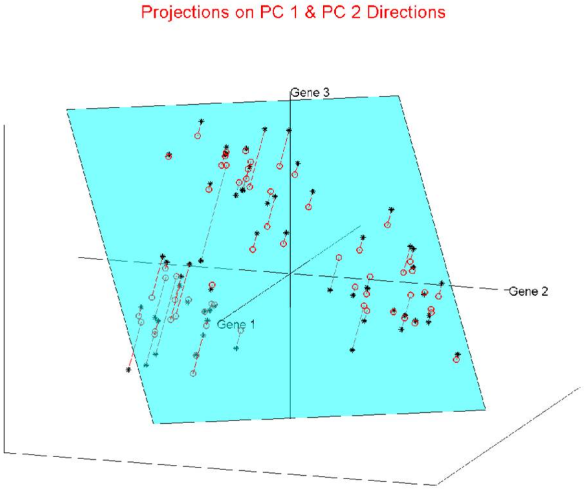

Figure 1.4: The subspace generated by the first two Principal Components of the same toy data set, with projected data points.

Let d denote the dimension of the original space the data is in. By taking the first k Principal Components ($k < d$) and projecting the data set onto the subspace defined by them, one can obtain a simplified picture of the data where the most important aspects of the variation are kept and some of the noise is eliminated. Moreover, conducting an analysis of the correlation between each mode of variation and external parameters becomes possible.

The method to find these directions for data in Euclidean space is very well-known: See Jolliffe (2002) and Muirhead (1982). The job of carrying the ideas defined in Euclidean space onto the binary tree space and coming up with an analogous method was first tackled by Wang and Marron (2007). Their work includes defining the binary tree space mentioned and an appropriate metric for it, and coming up with useful definitions of a $tree - line$, which is an analog of a line in Euclidean space. They also gave formal definitions of Principal Component lines in binary tree space, but due to the lack of a solution to the resulting optimization problems, only limited toy examples (three and four node trees, which thus allowed manual solutions) were used to illustrate the main ideas proposed. These definitions are given in Section 2.2.3.

1.4 Organization

In this research, through a detailed analysis of the underlying optimization problems of tree-lines, linear time computational methods to solve them are developed. This advancement allowed the first actual Object Oriented Data Analysis of a production scale data set of a population of tree structured objects. The results of this analysis are elaborated in Section 2.2.

The analysis results for tree-lines gave important insights into the data, however, they also led the to the understanding that each of the tree-lines found explains too little variation, and one needs to come up with richer structures to project the data sets on. Ideas related to the research for this purpose are $tree - curves$ and $k - tree - lines$, which are handled in Sections 3.3 and 4.4.2 respectively.

Finally, Section 5.4 is dedicated to developing a visualization method that will help the reduction of noise in the motivating data set, the brain vessel structures of 105 subjects. Due to the inaccuracies in scanning and tracking of these vessels, this data set is known to include a high amount of noise. A detailed visualization method is established in this work to spot the instances that require manual cleaning or need to be excluded from the data set.

CHAPTER 2

Tree-lines

2.1 Definitions And Examples

This section starts with the formal definition of a binary tree:

Definition 2.1.1. *A* ***binary tree*** *is a set of nodes that are connected by edges in a directed fashion, which starts with one node designated as* ***root****, where each node has at most two children.*

Using the notation t_i for a single tree, let

$$T = \{t_1, ..., t_n\} \tag{2.1.1}$$

denote a data set of n such trees. A toy example of a set of 3 trees is given in Figure 2.1.

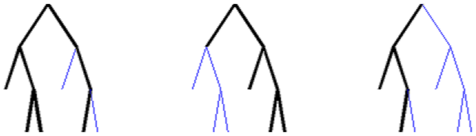

Figure 2.1: Toy example of a data set of trees, T, with $n = 3$. This will be used to illustrate several issues below.

To identify the nodes within each tree more easily, the level-order indexing method from Wang and Marron (2007) is used. The root node has index 1. For the remaining nodes, if a node has index ω, then the index of its left child is 2ω and of its right child is $2\omega + 1$.

These indices enable the identification of a binary tree by only listing the indices of its nodes.

The basis of the analysis is an appropriate metric, i.e. distance, on tree space. The common notion of Hamming distance is used for this purpose:

Definition 2.1.2. *Given two trees t_1 and t_2, their* ***distance*** *is*

$$d\,(t_1, t_2) = |t_1 \backslash t_2| + |t_2 \backslash t_1|,$$

where $\backslash$ denotes set difference.

It should be noted that the trees are treated as sets of nodes in this definition. This distance function is shown to be *pseudo-metric* in Wang (2003).

In literature, there are various functions defined to measure the distance between two trees. It is possible to go over all subtrees within two trees and take the total count of un-common subtrees as a distance measure. (See Shawe-Taylor and Cristianini (2000).) For other examples and further references on tree distance metrics, see Müller-Molina et. al. (2009).

One possible shortfall of the distance function defined above is that, all the nodes in data trees have the same impact on total distance. One can imagine situations where certain nodes (the ones closer to the root node, or thicker ones, etc.) carry more importance. One possibility is to assign non-negative weights to each node on the data trees that signify their relative importance. In this case, the distance function would sum the weights of the nodes that are not in the intersection of the two data trees instead of simply counting them. Although this possibility was not deeply investigated within the scope of this work, it was established that this function also is a metric, and Theorem 2.1.1 will continue to hold with this metric.

Two more basic concepts are defined below; the notion of support tree has already been shown in Figure 2.1 (as the thin blue lines).

Definition 2.1.3. *For a data set T, given as in (2.1.1), the support tree, and the intersection tree are defined as*

$$\begin{aligned} \mathrm{Supp}(T) &= \cup_{i=1}^{n} t_i \\ \mathrm{Int}(T) &= \cap_{i=1}^{n} t_i. \end{aligned}$$

Figure 2.4 shows the support trees of the first data set used in this study. Figure 2.5 includes the corresponding intersection trees.

The main idea of a tree-line (notion used for one dimensional representation in tree space) is that it is constructed by adding a sequence of single nodes, where each new node is a child of the most recent child:

Definition 2.1.4. *A* ***tree-line****,* $L = \{\ell_0, \cdots, \ell_m\}$*, is a sequence of trees where* ℓ_0 *is called the starting tree, and* ℓ_i *comes from* ℓ_{i-1} *by the addition of a single node, labeled* v_i*. In addition each* v_{i+1} *is a child of* v_i*.*

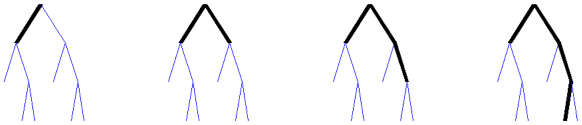

Figure 2.2: Toy example of a tree-line. Each point is obtained by adding one node to the previous point. Starting point (ℓ_0) is the intersection tree of the toy data set of Figure 2.1.

An example of a tree-line is given in Figure 2.2. Insight as to how well a given tree-line fits a data set is based upon the concept of projection:

Definition 2.1.5. *Given a data tree* t*, its* ***projection*** *onto the tree-line* L *is*

$$P_L(t) = \underset{\ell \in L}{\arg\min}\{d(t, \ell)\}.$$

Wang and Marron (2007) show that this projection is always unique. This will also follow from Claim A.1.1 in Section A.1, whose characterization of the projection will be the key in computing the principal component tree-lines, defined shortly.

The above toy examples provide an illustration. Let t_1 be the first tree shown in Figure 2.1. Name the trees in the tree-line, L, shown in Figure 2.2, as ℓ_0,ℓ_1,ℓ_2,ℓ_3. The set of distances from t_1 to each each tree in L is tabulated as

j	0	1	2	3
$d(t_1, \ell_j)$	7	6	5	4

The minimum distance is 4, achieved at $j = 3$, so the projection of t_1 onto the tree-line L is ℓ_3.

Next, an analog of the first principal component ($PC1$) is developed, by finding the tree-line that best fits the data.

Definition 2.1.6. *For a data set T and a starting point ℓ_0, the **first principal component tree-line** starting from this point, i.e. $PC1$, is*

$$L_1^* = \arg\min_{L} \sum_{t_i \in T} d(t_i, P_L(t_i))$$

The Principal Components we defined in binary tree space start from a fixed starting point tree. Unlike the Euclidean space, the nature of binary tree space requires the first point not to be completely arbitrary, but a sufficiently small tree that the line growing from it can cover enough variation. The first natural approach may be using the root node as the starting point, and this is the first approach taken in this study. However, it was observed that in this case the first principal component follows the path of leftmost nodes, which are common to all data trees, and thus does not provide any information on variation. The second approach is to take the *intersection tree* as the starting point. The nodes in the intersection tree exist in all of the data trees, and therefore does not contain any variation, and thus seem to be an ideal starting point. Indeed when this approach was taken, as will be seen in Section 2.2, the 1^{st} PC explains the age correlation the best in most cases, as expected.

Another suggestion on this issue is to *not* assume the starting point as a fixed tree, but to incorporate it into the optimization problem. This approach is not taken in this study, but it may lead to some interesting results. Although a detailed study is not done for this new problem, it is conjectured that the optimal solution in this case will use the *median tree* as the starting point. See Wang(2003) for a detailed discussion of the median tree.

In this problem the last point on the first PC does not have any restrictions as the first point does. However, extending it beyond the nodes in support tree will not result in any gain in the objective function, but it will not cause any loss either. Therefore, in theory, there are infinitely many optimal tree-lines for this problem that extend beyond the support tree, but these are useless in practice, so we take the smallest of these (the one contained in support tree) and use it for our analyses.

In conventional Euclidean PCA, additional components are restricted to lie in the subspace orthogonal to existing components, and subject to that restriction, to fit the data as well as possible. For an analogous notion in tree space, first the concepts of the union of tree-lines and a projection onto it need to be defined.

Definition 2.1.7. *Given tree-lines* $L_1 = \{\ell_{1,0}, \ell_{1,1}, \ldots, \ell_{1,p_1}\}, \ldots, L_q = \{\ell_{q,0}, \ell_{q,1}, \ldots, \ell_{q,p_q}\}$*, their* ***union*** *is the set of all possible unions of members of* L_1 *through* L_q*:*

$$L_1 \cup \cdots \cup L_q \quad = \quad \{\ell_{1,i_1} \cup \cdots \cup \ell_{q,i_q} \,|\, i_1 \in \{0, \ldots, p_1\}, \ldots, i_q \in \{0, \ldots, p_q\}.\}$$

Given a data tree t*, the projection of* t *onto* $L_1 \cup \cdots \cup L_q$ *is*

$$P_{L_1 \cup \cdots \cup L_q}(t) = \underset{\ell \in L_1 \cup \cdots \cup L_q}{\arg\min} \{d(t, \ell)\}. \tag{2.1.2}$$

In the non-Euclidean tree space, at the present time there is no readily available notion of orthogonality is present, so instead the 2nd tree-line is just required that it fits as much of data as possible when used in combination with the first, and so on.

Definition 2.1.8. *For* $j \geq 1$ *the* j*th principal component tree-line is defined recursively as*

$$L_j^* = \underset{L}{\arg\min} \sum_{t_i \in T} d(t_i, P_{L_1^* \cup \cdots \cup L_{j-1}^* \cup L}(t_i)), \tag{2.1.3}$$

and it is abbreviated as PCj*.*

For the concept of PC tree-lines to be useful, it is of crucial importance to be able to compute them efficiently. Another notion to explain the computations will be needed:

Definition 2.1.9. *Given a tree-line*

$$L = \{\ell_0, \ell_1, \cdots, \ell_m\}$$

Define the path of L *as*

$$V_L = \ell_m \setminus \ell_0.$$

Intuitively, a tree-line that well fits the data "should grow in the direction that captures the most information". Furthermore, the jth PC tree-line should only aim to capture information that has not been explained by the first $j-1$ PC tree-lines. This intuition is made precise in the following theorem, which is the main theoretical result of the section:

Theorem 2.1.1. *Let u_0 be an exogenously defined starting point, $j \geq 1$, and $L_1^*, \ldots, L_{j-1}^*$ be the first $j-1$ PC tree-lines. For $v \in \mathrm{Supp}(T)$ define*

$$w_j(v) = \begin{cases} 0, & \text{if } v \in V_{L_1^*} \cup \cdots \cup V_{L_{j-1}^*}, \\ \sum_{i=1..n} \delta(v, t_i), & \text{otherwise} \end{cases} \tag{2.1.4}$$

Then the jth PC tree-line L_j^* is the tree-line whose path maximizes the sum of w_j weights in the support tree, i.e. $\sum_{v \in V_{L_j^*}} w_j(v)$.

Here, the *delta* function $\delta(v, t_i)$ is equal to 1 if v is a node that exists in tree t_i, and 0 otherwise. The proof of Theorem 2.1.1 is given in Appendix A.1.

This theorem also points out that, even when we are considering the tree-lines contained in support tree only, the optimal tree-lines may not be unique. There may be more than one paths extending from the starting point tree and have the same sum of weights. In this case the selection is made arbitrarily.

Figure 2.3 is an illustration: the weight of a node is the number of times the node appears in the trees of Figure 2.1. The black edge is the intersection tree of the same data set. The maximum weight path attached to $\mathrm{Int}(T)$ is the red path, which gives rise to the tree-line of Figure 2.2, which is thus the first principal component of the data set of Figure 2.1.

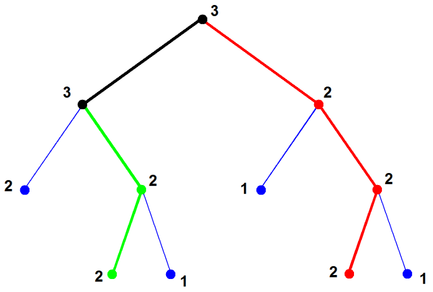

Figure 2.3: Weighted support tree illustrating Theorem 2.1.1

After setting the weights of the nodes on the red path to zero, the maximum weight path attached to Int(T) becomes the green path, which by Theorem 2.1.1 gives rise to $PC2$. The usefulness of these tools is demonstrated with actual data analysis of the full tree data set.

2.2 Data Analysis Results

This section describes an exploratory data analysis of the three data sets discussed above using the tree-line ideas. The principal component tree-lines are computed as defined in Theorem 2.1.1. For Data Set 1 only, both correspondence types, defined in Section 1.2 are considered and compared. The data analysis methodology is explained in detail on Data Set 1 in Section 2.2.1. Next, the comparative results of all data sets are presented in Section 2.2.2. Finally, data analysis conclusions are summarized in Section 2.2.3.

2.2.1 Data Set 1

The different brain location types (shown as different colors in Figure 1.2) are analyzed as separate populations (i.e. the blue trees are first considered to be a population, then the gold trees, etc.), called *brain location sub-populations*. Partitioning the vessel system in this way reveals some interesting differences between the brain location types in terms of symmetry.

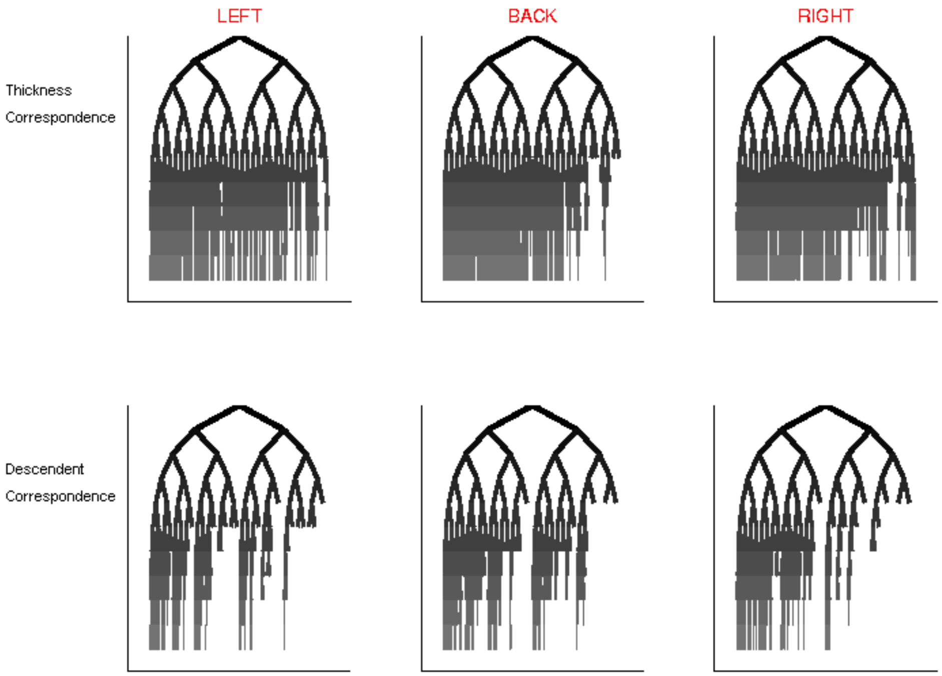

Figure 2.4: Support trees of Data Set 1, for both types of correspondence (shown in the rows), and for three brain location tree types (shown in columns, corresponding to the colors in Figure 1.2). Shows that the descendant correspondence gives a population with more compact variation than the thickness correspondence.

First, the two types of correspondence defined in Section 1.2 are compared using the concept of the support tree. This is done by displaying the support trees for each type of correspondence, and for each of the three tree location types (shown with different colors in Figure 1.2), in Figure 2.4. Note that all of the support trees for the descendant correspondence (bottom) are much smaller than for the thickness correspondence (top), indicating that the descendant correspondence results in a much more compact population. This seems likely to make it easier for the PCA method to find an effective representation of the descendant based population.

Figure 2.4 already reveals an aspect of the population that was previously unknown: there is not a very strong correlation between median tree thickness of a branch, and the number of children.

Figure 2.5 shows the first 3 PC tree-lines in Data Set 1, for the three sub-populations (shown as rows), with the intersection tree as the starting tree, for the descendant correspondence.

In the human brain, the back circulation (gold) arises from a single vessel (the basilar artery) and immediately splits into two main trunks, supplying the back sides of the left and right hemispheres. These two parts of the back circulation are expected to be approximately mirror-image symmetrical with both sides containing one main vessel and other branches stemming from that. Consequently, for each tree on the back data set, if a vertical axis that goes through the root node is imagined, the subtrees on both sides of the axis are expected to be symmetrical with each other.

The results of this model for the back subpopulation are consistent with this expectation. The main vessel of one of the hemispheres can be seen in the starting point (intersection tree) as the leftmost set of nodes, while the other main vessel becomes the first principal component.

As for the left and right circulations (cyan and blue trees) of the brain, they are expected to be close to mirror images of each other. Unlike the case of the back subpopulation, in each of these circulations there is a single trunk from which smaller branches stem. For this reason the bilateral symmetry observed within the back trees is not expected to be found here.

The fact that $PC1$'s for left and right subpopulations are at later splits suggest that the earlier splits tend to have relatively few descendants. The remaining $PC2$ and $PC3$ tree-lines do not contain much additional information by themselves. However, when PC's 1,2 and 3 are considered together and the left and right subpopulations are compared, i.e. the second and third rows of Figure 2.5, the structural likeliness is quite visible.

It should also be noted that for both of the subpopulations all PC's are on the left side of the root-axis, indicating a strong bilateral asymmetry, as expected.

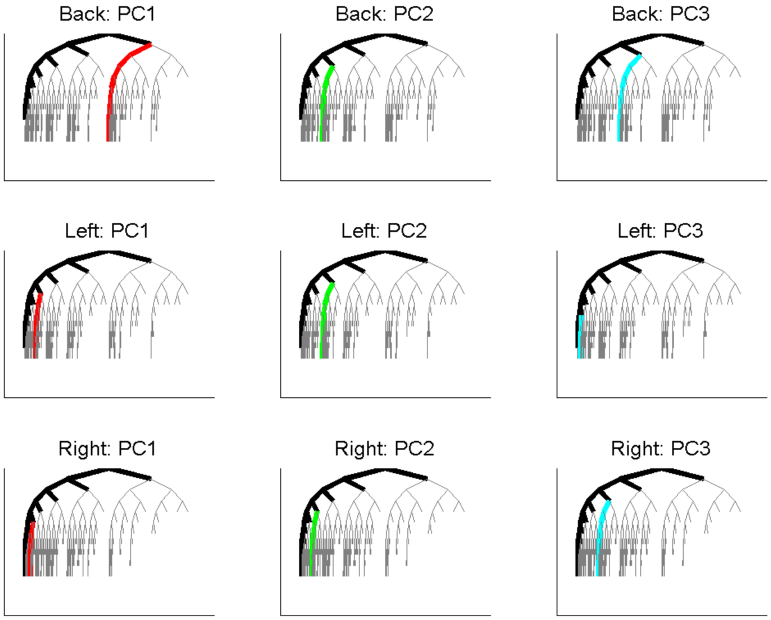

Figure 2.5: Best fitting tree-lines for Data Set 1, for different sub-populations (rows), and PC number (columns). Intersection trees are shown in black.

The tree-lines, and insights obtained from them, were essentially similar for the thickness correspondence, so those graphics are not shown here.

Next comes is the study of the tree-line analog of the familiar *scores plot* from conventional PCA (a commonly used high dimensional visualization device, sometimes called a *draftsman's plot* or a *scatterplot matrix*). In that case, the scores are the projection coefficients, which indicate the size of the component of each data point in the given eigen-direction. Pairwise scatterplots of these often give a set of useful two dimensional views of the data. In the present case, given a data point and a tree-line, the corresponding *score* is just the length (i.e. the number of nodes) of the projection. Unlike conventional PC scores, these are all integer valued.

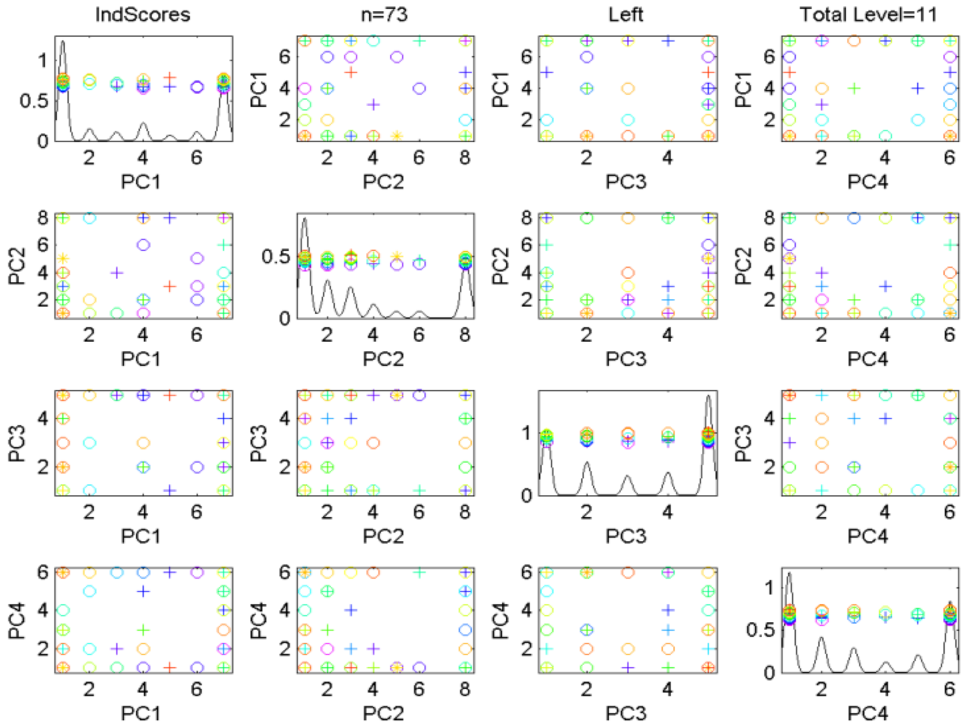

Figure 2.6: Scores Scatterplot for the Descendant Correspondence, Left Side sub-population of Data Set 1. Colors show age, symbols gender. No clear visual patterns are apparent.

Figure 2.6 shows the scores scatterplot for the set of left trees, based on the descendant correspondence. The data points have been colored in Figure 2.6, to indicate age, which is an important covariate, as discussed in Bullitt et al (2008). The color scheme starts with purple for the youngest person (age 20) and extends through a rainbow type spectrum (blue-cyan-green-yellow-orange) to red for the oldest (age 72). An additional covariate, of possible interest, is sex, with females shown as circles, males as plus signs, and two transgender cases indicated using asterisks.

It was hoped that this visualization would reveal some interesting structure with respect to age (color), but it is not easy to see any such connection in Figure 2.6. One reason for this is that the tree-lines only allow the very limited range of scores (projection lengths) as integers, where the score is bounded by the depth of each tree. A simple way to generate a wider range of scores is to project not just onto simple tree-lines, but instead onto their union, as defined in (2.1.2). Figure 2.7 shows a scatterplot matrix, of several union PC scores, in particular $PC1$ vs. $PC1 \cup 2$ (shorthand for $PC1 \cup PC2$) vs. $PC1 \cup 2 \cup 3$ vs. $PC1 \cup 2 \cup 3 \cup 4$. This combined plot, called the *cumulative scores scatterplot*, shows a better separation of the data than is available in Figure 2.6. The PC unions show a banded structure, which again is an artifact that follows from each PC score individually having a very limited range of possible values. This seems to be a serious limitation of the tree-line approach to analyzing population structure.

As with Figure 2.6, there is unfortunately no readily apparent visual connection between age and the visible population structure. However, visual impression of this type can be tricky, and in particular it can be hard to see some subtle effects.

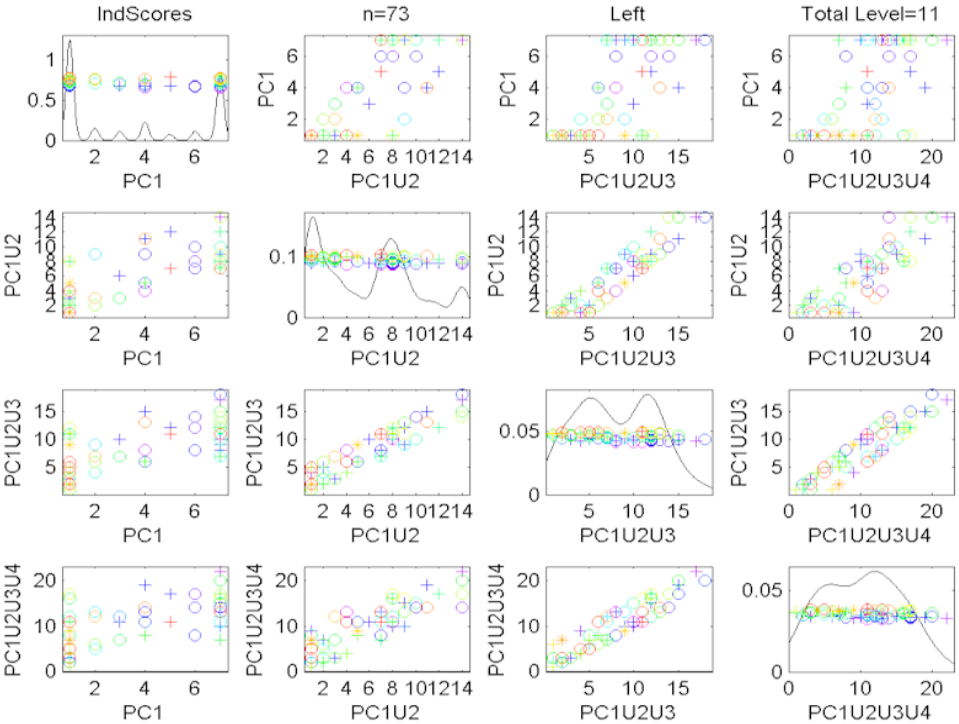

Figure 2.7: Cumulative Scores Scatterplot for the Descendant Correspondence, Left Side sub-population of Data Set 1.

Figure 2.8 shows a view that more deeply scrutinizes the dependence of the $PC1$ score on age, using a scatterplot, overlaid with the least squares regression fit line. Note that most of the lines slope downwards, suggesting that older people tend to have a smaller $PC1$ projection than younger people. Statistical significance of this downward slope is tested by calculating the standard linear regression p-value for the null hypothesis of 0 slope, which from this on will be called *slope p-value*. For the left tree, using the descendant correspondence, the slope p-value is 0.0025. This result is strongly significant, indicating that this component is connected with age. This is consistent with the results of Bullitt et al (2008), who noted a decreasing trend with age in the total number of branchings. This study's result is the first location specific version of this.

Similar score versus age plots have been made, and hypothesis tests have been run, for other PC components, and the resulting slope p-values, for the left tree using the descendent correspondence are summarized in this table:

PC1	0.003
PC2	0.169
PC3	0.980
PC4	0.298
PC1 $\cup 2$	0.003
PC1 $\cup 2 \cup 3$	0.004
PC1 $\cup 2 \cup 3 \cup 4$	0.007

Table 2.1: The slope p-values associated with each PC and their unions for Left subpopulation, Data Set 1.

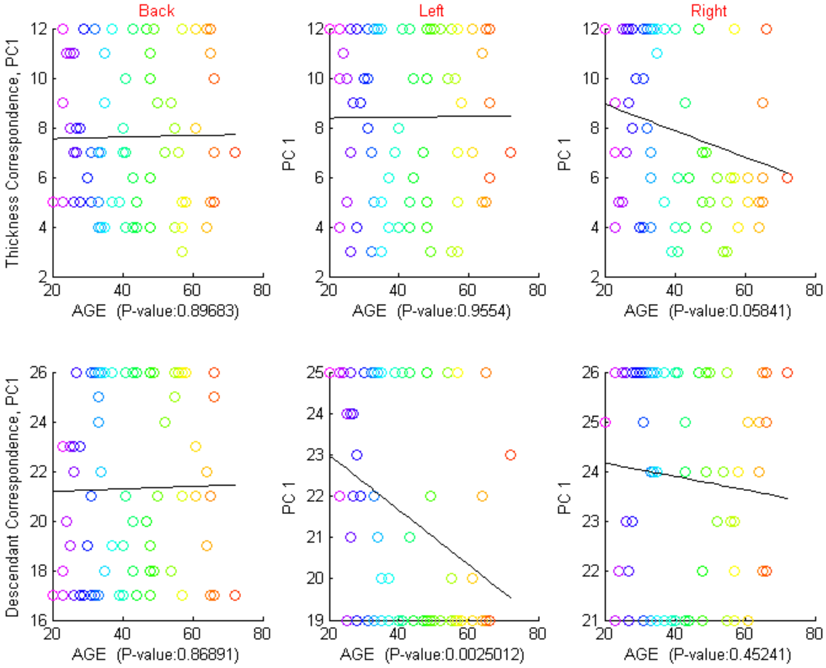

Figure 2.8: Scatterplot of $PC1$ score versus age in Data Set 1. Least squares fit regression line suggests a downward trend in age. Trend is confirmed by the slope p-value of 0.0025 for Left subpopulation, descendant correspondence.

Note that for the individual PCs, only $PC1$ gives a statistically significant result. For the cumulative PCs, all are significant, but the significance diminishes as more components are added. This suggests that it is really $PC1$ which is the driver of all of these results.

To interpret these results, recall from Figure 2.5, that for the left trees, $PC1$ chooses the left child for the first 3 splits, and the right child at the 4th split. This suggests that there is not a significant difference between the ages in the tree levels closer to the root, however, the difference does show up when one looks at the deeper tree structure, in particular after the 4th split. This is consistent with the above remark, that for the left brain sub-population, the first few splits did not seem to contain relevant population information. Instead the effects of age only appear on splits after level 4.

A similar analysis of the back and right brain location sub-populations is done, but none of these found significant results, so they are not shown here. However, these can be found at the web site (33).

Parallel results for the thickness correspondence are also considered, which again did not yield significant results (but these are on the web site (33)). The fact that descendant correspondence gave some significant results, while thickness never did, is one more indication that descendant correspondence is preferred. The analyses of the newer data sets (2 and 3) use descendant correspondence only.

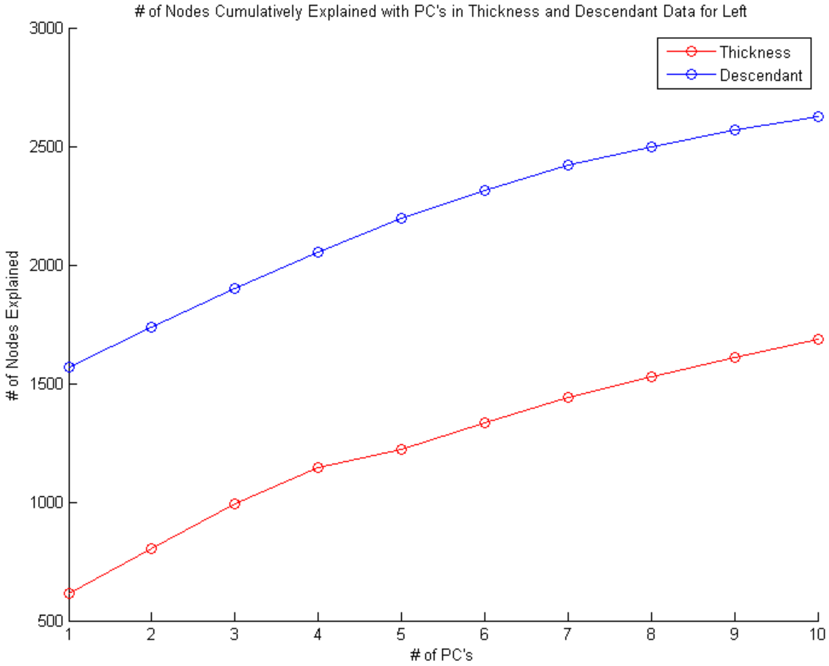

Figure 2.9: Total number of nodes explained, as a function of Cumulative PC Number. Shows that the descendant correspondence allows PCA to explain a much higher proportion of the variation in the population than the thickness correspondence.

One more approach to the issue of correspondence choice is shown in Figure 2.9. This shows the amount of variation explained, as a function of the order of the Cumulative Union PC, for both the thickness and the descendant correspondences, for the left brain location sub-population. The *amount of variation explained* is defined to be the sum, over all trees in the sub-population of the lengths of the projections. There are 5023 nodes in total for both correspondences. (The correspondence difference affects the locations of nodes, total count remains the same.)

It is not surprising that these curves are concave, since the first PC is designed to explain the most variation, which each succeeding component explaining a little bit less. But the important lesson from Figure 2.9 is that the descendant correspondence allows PCA to explain much more population structure, at each step, than the thickness correspondence.

2.2.2 Comparisons Among Data Sets

Up to this point, using Data Set 1, the difference between the two correspondence types are investigated, and descendant correspondence is found to be a more compact representation of the data. The visual structures of the first three principal components are examined and important results on symmetry are observed. The scatterplots of principal component's projection lengths are studied, but visual patterns could not be found.

Similar properties also can be found in Data Sets 2 and 3 as shown in this section.

The first two principal components of all subpopulations of Data Set 3 are shown in Figure 2.10. The maximum depth of data trees in this set (37) is much larger than those in Data Set 1 (11), rendering the visualization technique used in Section 2.2.1 impossible for this set. That technique requires the locations of each possible node for the maximum depth to be allocated on the graph area. For Data Set 1, 2^{11} points can be allocated beforehand, but for Data Sets 2 and 3 the requirement goes up to 2^{37} which is unmanageable. A new visualization, more appropriate for trees of greater depth, is developed for these sets. On these graphs the coordinates of each node displays its number of descendants (vertical axis) and its level on the binary tree (horizontal axis). For a compact summarization, the logarithm with base 2 of the descendant count is the y-axis. Some fine details and refinements of this visualization will be given in Chapter 5.4.

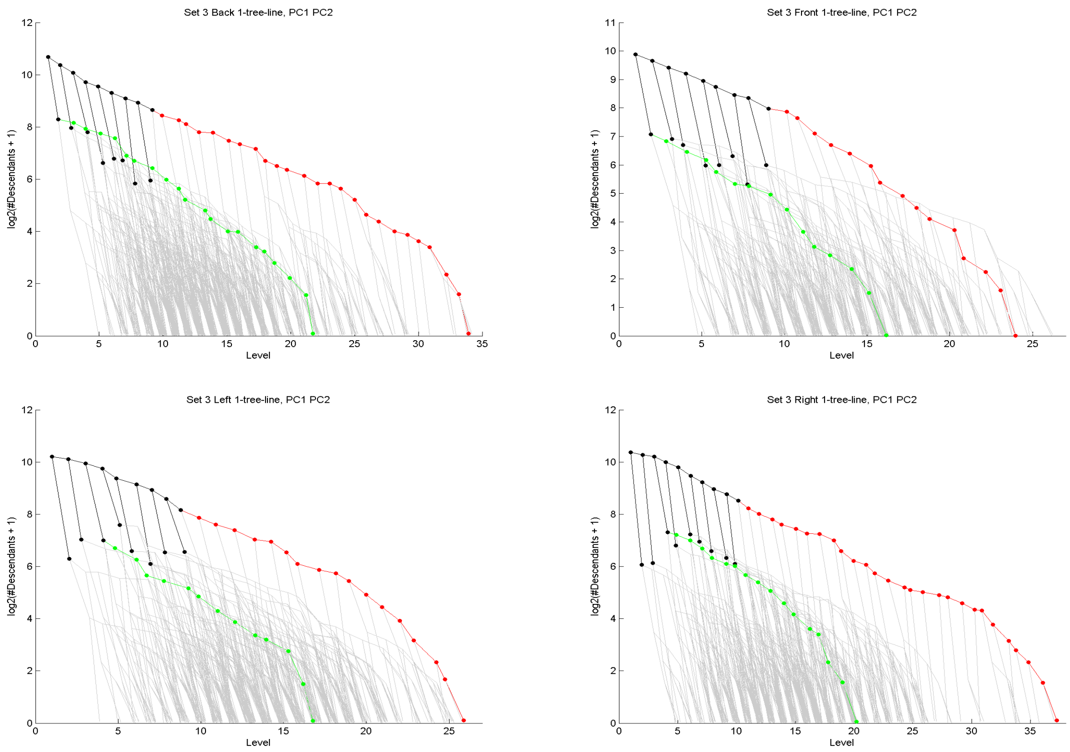

Figure 2.10: The first two principal components in Data Set 3. Grey nodes are support tree, black nodes the intersection tree, red nodes are PC1's path and green nodes are PC2's path.

The lines belonging to Data Set 2 are very similar to these so they are not given here.

A careful comparison of Figures 2.5 and 2.10 reveals that the same structural properties observed in Data Set 1, such as higher symmetry in the Back sub-population, also exist in Data Set 3. An additional insight obtained from the Data Set 3 plots is that the same symmetry property observed in the Back sub-population also exists in the Front. This result is be expected since the Front sub-population consists of vessels feeding both the left and right hemispheres of the brain. However this was not be observed in Data Set 1 because the Front sub-population data was too unreliable in that set.

The Right and Left sub-populations in Figure 2.10 lack axial symmetry, but they have the same first and second principal component structure. These systems are expected to be roughly mirror images of each other. The observed structure found by principal components is consistent with that expectation.

Next, the score scatter plots, similar to Figures 2.6 and 2.7, for the Left sub-population of Data Set 3 are given. The scatterplot representations of the results given in Figures 2.11 and 2.12 do not yield a visual pattern as was the case with Data Set 1. The corresponding plots for Data Set 2 are very similar so they are not shown here.

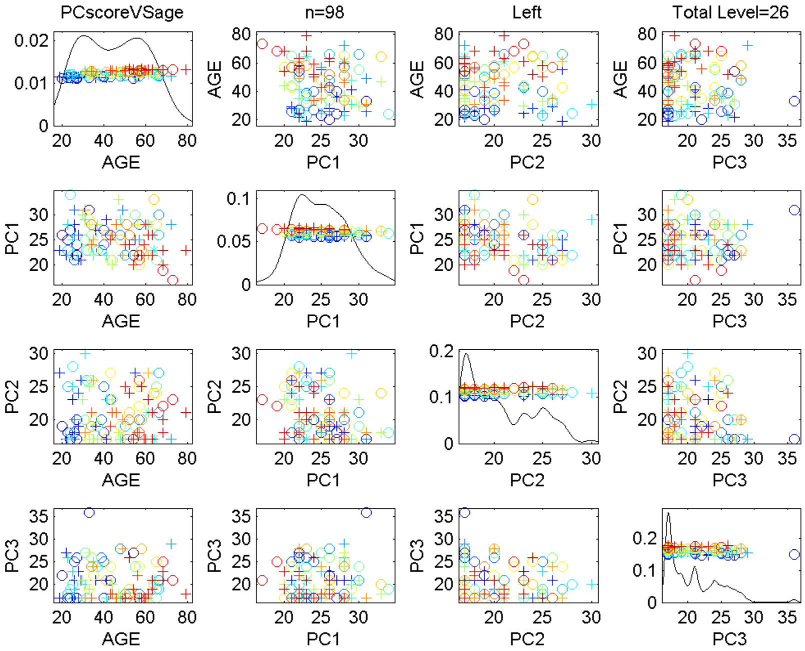

Figure 2.11: Scores Scatterplot for Left sub-population of Data Set 3. Colors show age, symbols gender. No clear visual patterns are apparent.

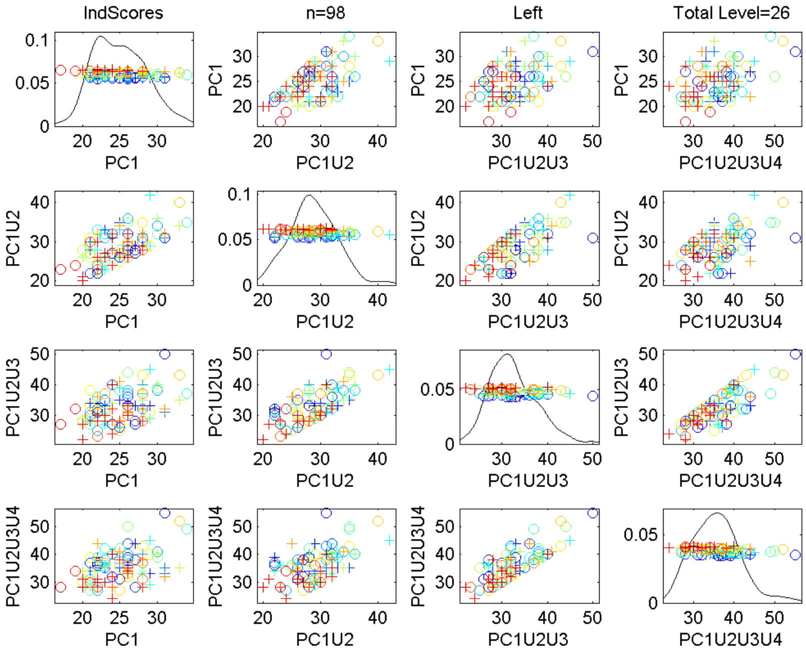

Figure 2.12: Cumulative Scores Scatterplot for Left sub-population of Data Set 3.

These three data sets come from the same ongoing study, where the last two are improved versions of Data Set 1. Therefore it should not be surprising that basic structural characteristics of the first set do not change as improvements are made. However, the improvement in one key characteristic is worth mentioning in detail. The age effect that was observed only in the Left sub-population now becomes visible in all four sub-populations.

Figure 2.13 gives an example scatter plot of projection size versus age for the Back sub-population of Data Set 3, PC1. The significant slope p-value $(0.016 < 0.05)$ is a new result that was not observed for Data Set 1.

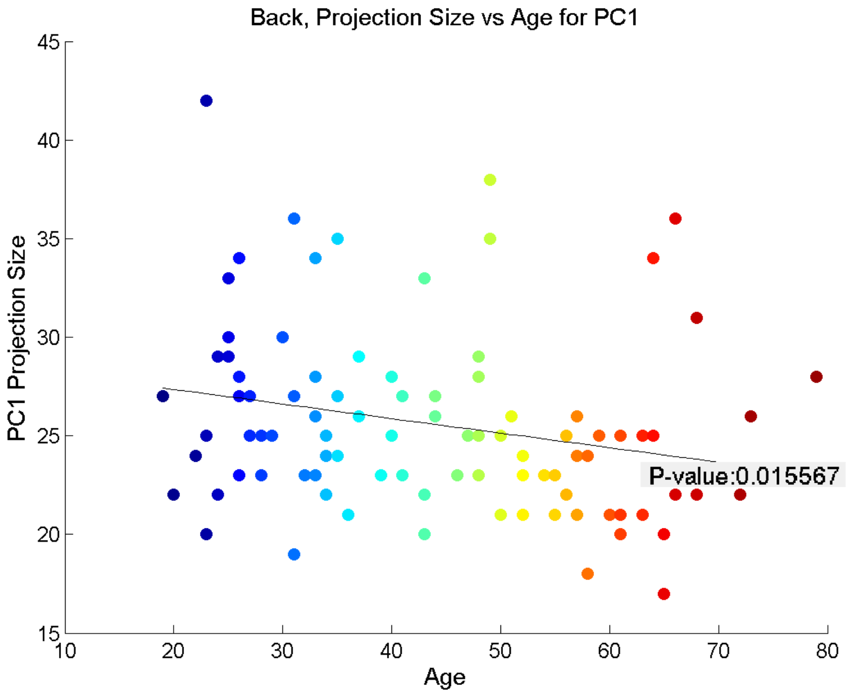

Figure 2.13: Projection size on PC1 versus age for Back sub-population, Data Set 3. Low slope p-value indicates strong dependence.

In fact, the slope p-value summary given in Table 2.2 demonstrates the major improvement achieved by Data Sets 2 and 3.

		Set 1	Set 3	Set 2
Back	PC1	*	0.0156	0.0441
	PC2	*	*	*
	PC3	*	*	*
	PC4	*	*	*
	PC1U2	*	*	*
	PC1U2U3	*	*	0.0283
	PC1U2U3U4	*	0.0005	0.0110
Front	PC1	-	*	*
	PC2	-	*	*
	PC3	-	*	*
	PC4	-	0.0232	0.0129
	PC1U2	-	*	*
	PC1U2U3	-	*	*
	PC1U2U3U4	-	0.0282	0.0202
Right	PC1	*	0.0186	0.0178
	PC2	*	0.0164	0.0221
	PC3	*	*	*
	PC4	*	*	*
	PC1U2	*	0.0002	0.0002
	PC1U2U3	*	0.0004	0.0001
	PC1U2U3U4	*	0.0002	0.0002
Left	PC1	0.003	*	*
	PC2	*	*	0.0490
	PC3	*	*	*
	PC4	*	*	0.0422
	PC1U2	0.003	*	0.0056
	PC1U2U3	0.004	*	*
	PC1U2U3U4	0.007	0.0063	0.0035

Table 2.2: The slope p-values obtained using 1-tree-lines. Columns compare all three data sets. The slope p-values above the 0.05 significance level are marked with (*). The front sub-population does not exist in Data Set 1, so those entries are marked with (-).

2.2.3 Data Analysis Summary

In summary, there are several important conclusions of this chapter:

- In real data sets with branching structure, tree PCA can reveal interesting insights, such as symmetry.

- The descendant correspondence is clearly superior to the thickness correspondence, and is recommended as the default choice.

- As expected, the back sub-population is seen to have a more symmetric structure.

- The improved Data Sets 2 and 3 seem to contain more information because for all sub-populations there is a statistically significant structural age effect. In particular, the projection size tends to decrease with age which is consistent with the results of Bullitt (2008).

- There seems to be room for improvement of the tree-line idea for doing PCA on populations of trees. Other data analytic approaches which allow a richer branching structure within each component are developed in Sections 4.4.2 and 3.3.

CHAPTER 3

Tree-curves

3.1 Definitions

A tree-curve is a sequence of trees, such that, given a tree in the tree-curve, the next tree in the sequence is obtained by adding one node. This node has to be a child of existing nodes in the previous tree to satisfy connectivity requirement. The tree-curve idea is a generalization of the tree-line concept: the constraint on the location of the next added node is removed from the tree-line definition to obtain the tree-curve definition. In Euclidean space, all points on a line are required to lie on a single direction. The constraint on the location of the next added node is considered to emulate this property in tree-lines. By removing it, a structure considered to be the counter part of a curve in Euclidean space is obtained.

Definition 3.1.1. *A* ***tree-curve****,* $C = \{c_0, \cdots, c_m\}$*, is a sequence of trees where* c_0 *is called the starting tree, and* c_i *comes from* c_{i-1} *by the addition of a single node, labeled* v_i*.*

An example tree-curve can be seen in Figure 3.1. Note that it starts from an initial tree of two nodes, and ends at the support tree.

The projection of a data tree onto a tree-curve is the point on the tree-curve with smallest distance to the data tree:

Definition 3.1.2. *Given a data tree* t*, its* ***projection*** *onto the tree-curve* C *is*

$$P_C\left(t\right) = \underset{c \in C}{\arg\min}\{d\left(t, c\right)\}.$$

The principal components are also defined similarly:

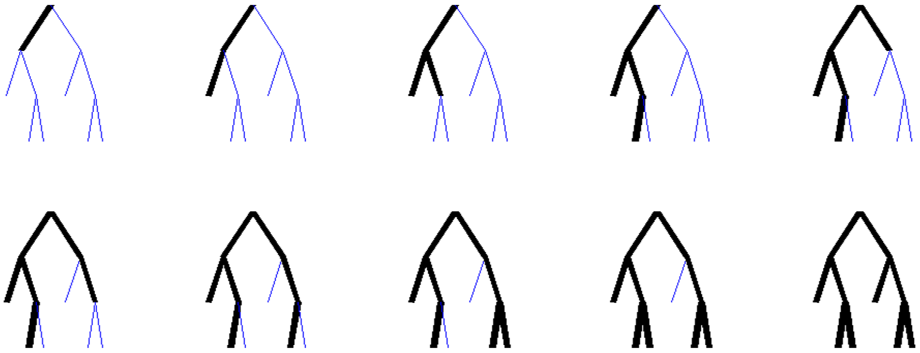

Figure 3.1: A toy example curve consisting of 10 points.

Definition 3.1.3. *For a data set T, the* ***first principal component tree-curve*** *is*

$$C_1^* \;=\; \arg\min_{C} \sum_{t_i \in T} d(t_i, P_C(t_i))$$

Definition 3.1.4. *For $j \geq 1$ the j****th principal component tree-curve*** *is defined recursively as:*

$$C_j^* = \arg\min_{C} \sum_{t_i \in T} d(t_i, P_{C_1^* \cup \cdots \cup C_{j-1}^* \cup C}(t_i)) \tag{3.1.1}$$

3.2 Heuristics

Unlike the case with tree-lines, the sequence of nodes added to a starting point that define a tree-curve can be a member of a data tree in any order, as long as connectivity requirement of the points on the tree-curve is satisfied. So far, this prevented the development of an easy characterization of the projection of a data tree onto a tree-curve. Moreover, the set of all possible tree-curves on a given support tree has an order of $O(n!)$, where n is the number of nodes in the support tree.

As mentioned before, the problem of finding the optimal first principal component have not yet been has been intractable so far. However, some heuristic methods that give promising results exist. All heuristics mentioned below are known to give non-optimal results in some cases.

To test their effectiveness, a simulation with 30 randomly generated data sets, each containing 4 trees with 3 levels, is run. This size of a data set seems to be the largest where the optimal best fitting tree-curve can be quickly found using an exhaustive search. The performance of each heuristic is measured by comparing their resulting tree-curve, C, with the optimal tree-curve C^* that is found through exhaustive search. In particular, the performance of a tree-curve C on a data set T is measured with the following function:

$$Performance(C, T) = \frac{F(C^*, T)}{F(C, T)} * 100$$

,

Where $F(C, T)$ is the objective function value that needs to be minimized to reach the optimal tree-curve:

$$F(C, T) = \sum_{t_i \in T} d(t_i, P_C(t_i))$$

So far following algorithms have been considered:

3.2.1 Weight Order Algorithm (WO)

This algorithm starts from a given starting tree, and adds the nodes from the support tree in the order of their weights (their number of occurrences in the data set). Ties are broken according to parent-child relationship when possible: parents are added before their children. This algorithm achieved a performance measure of 98.82.

3.2.2 Greedy Algorithm (G)

Starting from an initial point, at each step the children nodes of the existing nodes in current step are gone over, and the improvement in objective function if that node is selected to be added is calculated. The candidate with best contribution is appended to the current tree to obtain the next tree in the curve. This algorithm gave a performance of 89.76.

3.2.3 Switching Algorithm (S)

This method starts from an arbitrary tree-curve, and couples of nodes that bring improvement in the objective function when their locations on the sequence defining the curve are switched are identified.

The method is terminated when no such couples of nodes remain. This algorithm performed at 94.02.

3.2.4 Weight Order + Switching Algorithm (WO+S)

This method combined two heuristics mentioned above, by running the Weight Order algorithm first and feeding its result to Switching algorithm, to see if any improvement can be achieved over WO result by simple switching. This has proved to be the best performing method in the simulation with a measure of 99.91, and is used to conduct the data analysis.

3.3 Data Analysis Results

The data analysis has been conducted by running the (WO+S) method, since this one seems to consistently give the best results. Each data point is projected onto the resulting best fitting tree-curve. Figure 3.2 shows an example of the relation between the size of this projection with the age of each subject. The red line is fitted to the data using linear regression. This plot is created for all of the sub-populations available, but none of them seem to present a visual pattern that is different from the others. Set 2, Back sub-population is randomly selected to be shown in Figure 3.2, and the rest is omitted from this text.

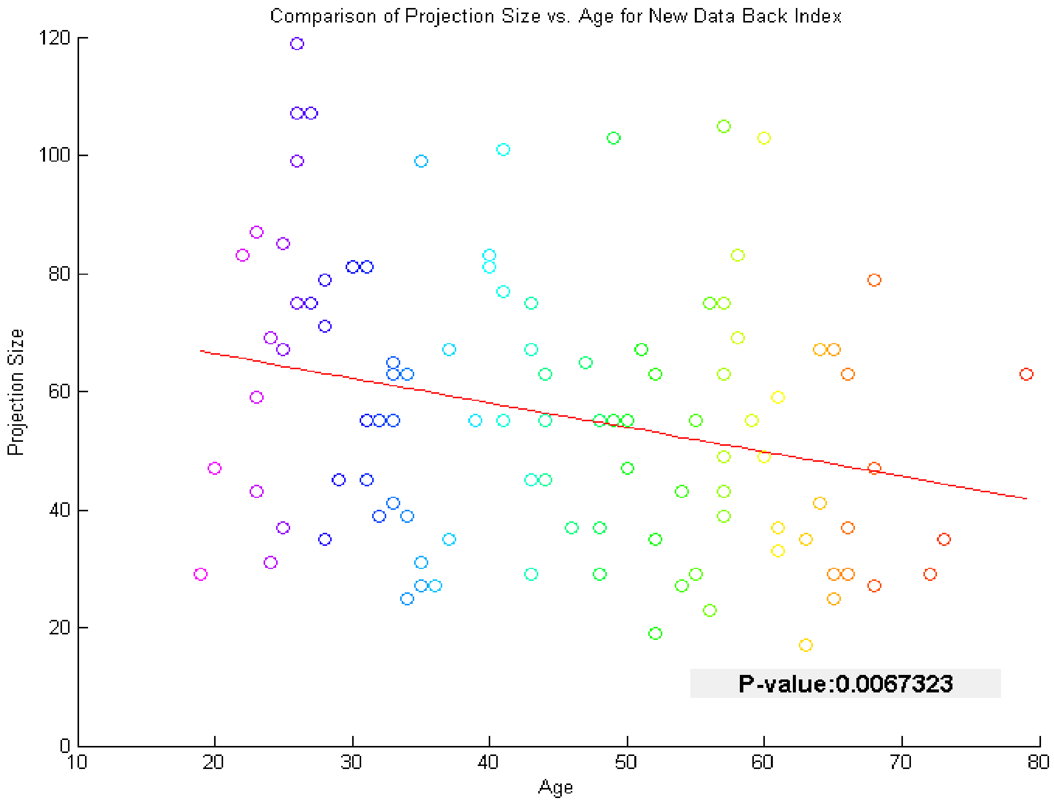

Figure 3.2: Size of projection onto tree-curve compared with age for back sub-population. Significant slope p-value proves correlation.

The tree-curve tool yields significant slope p-values for most of the sub-populations available. These are summarized in Table 3.1.

Set 3	B	0.0285
	L	0.0118
	R	0.0246
	F	0.0500
Set 2	B	0.0067
	L	0.0059
	R	0.0169
	F	*
Set 1	B	0.0058
	L	0.0184
	R	*

Table 3.1: The slope p-values obtained by the first principal tree-curve for all data sets. Values more than 0.05 significance level are shown with *.

The strong significant results obtained using tree-curves prove that this mode of analysis is a powerful tool to explain variation in binary trees. In addition, as shown in Figure 3.2, the very high projection sizes obtained renders this tool of analysis an attractive option. Tree-curves seem to capture 60% of the nodes that exist in the data sets. This ratio well exceeds what was obtained in Section 2.2 by combining the first 4 tree-lines. However, note that for this tool, the length of the projection is not exactly equal the number of nodes covered by the principal component, as was the case for tree-lines. Due to the structural nature of tree-curves, some nodes that do not exist in a data tree may appear in its projection.

A major drawback of tree-curves is that, it is hard to visually express the resulting tree-curve of an analysis run. Each tree-curve contains all the nodes that exist in the support tree, and what differentiates one curve from another is the sequence of its nodes. Although it is possible to visually express a sequence in some ways (one can use changing rainbow colors, etc.), visual inspection of those and trying to infer a structural trend from them is extremely hard. For example, the structure properties observed using tree-lines (symmetry) is very challenging to infer by using the tree-curve tool.

Another drawback of this method is that, we were unable to solve the resulting optimization problems to optimality. Although the heuristic results are quite accurate in smaller data sets where their accuracy can be determined by comparing them to brute force method calculation results, the performance in larger data sets has not been measured.

CHAPTER 4

K-tree-lines

So far, the tree-line and tree-curve ideas have been examined in Chapters 2.2.3 and 2.2.3, respectively. The tree-line problem can be solved efficiently. However, the number of nodes that can be explained by any tree-line of nodes of a data tree is limited to the depth of the data set as pointed out in Section 2.2. Therefore every tree-line explains only a limited amount of variation. Tree-curves seem to overcome this limitation. The tree-curve considered in Chapter 3.3 includes 60 percent of the nodes in Data Set 1. A major drawback of the tree-curve approach is that, a method to find an optimal tree-curve has not yet been discovered. The data analysis is currently run using heuristic methods. Although these heuristics are known to give accurate results on small data sets, their performance in large data sets cannot be accurately measured.

The idea of *k-tree-lines* is developed as a bridge between the tree-line and tree-curve concepts. When constructing a tree-line, each tree is obtained by adding a child to the last node of the previous tree. This last node, whose children are candidates for addition, is called an *active* node. In tree-curves, a child of any node in the previous tree can be added to get the next element of the sequence. Therefore all the nodes in the previous tree are active, provided they have un-added children. In a k-tree-line, at each step, only k nodes that were added last are active.

The organization of this Chapter is follows: In Section 4.1, the definitions of main concepts are explained. The complexity of the k-tree-line problem is examined in Section 4.2. Section 4.3 is dedicated to the 2-tree-line Branch&Bound Algorithm, which is used to obtain the numerical results elaborated in Section 4.4.

4.1 Definitions

The formal definition of a k-tree-line is:

Definition 4.1.1. *A **k-tree-line**, $K = \{\ell_0, \cdots, \ell_m\}$, is a sequence of trees where ℓ_0 is called the starting tree, and ℓ_i comes from ℓ_{i-1} by the addition of a single node, labeled v_i. In addition each v_{i+1} is a child of one of the nodes in $\{v_{i-k+1}, \cdots, v_i\}$, or in the case where $k > i$, it is a child of one of the members of $\{\ell_0, v_1, \cdots, v_i\}$. A k-tree-line of which the last k nodes are leaves of the support tree, that is, a k-tree-line that cannot be further extended is called a **maximal** k-tree-line. All other lines are called **partial** k-tree-lines.*

It can be seen that the k-tree-line is a generalization of both problems. When $k = 1$, the tree-line problem is obtained, when $k = \infty$, the structure becomes a tree-curve.

The limitation of tree-lines, that they cover too little for each line, can be overcome when larger k's are used. The advantage over the tree-curve idea is that k-tree-lines can be easier to compute while still yielding a relatively rich structure. It is conjectured that a 4 or 5-tree-line will behave very much like a tree-curve. While it does not seem to be possible to find an optimal solution to the tree-curve problem in polynomial time, it may be possible to do so for a k-tree-line problem. The complexity issue of k-tree-lines will be discussed in Section 4.2.

As before, the projection of a data tree onto a k-tree-line is the point on the k-tree-line with smallest distance to the data tree:

Definition 4.1.2. *Given a data tree t, its **projection** onto the k-tree-line K is*

$$P_K(t) = \arg\min_{\ell \in K} \{d(t, \ell)\}.$$

Unlike the tree-line case, the projection of a data point does not have to be unique.

The principal components are also defined similarly:

Definition 4.1.3. *For a data set T, the **first principal component k-tree-line** is*

$$K_1^* = \arg\min_K \sum_{t_i \in T} d(t_i, P_K(t_i))$$

Definition 4.1.4. *For $j \geq 1$ the j**th principal component k-tree-line** is defined recursively as:*

$$K_j^* = \arg\min_K \sum_{t_i \in T} d(t_i, P_{K_1^* \cup \cdots \cup K_{j-1}^* \cup K}(t_i)) \tag{4.1.1}$$

4.2 Complexity Of The 2-Tree-Line Problem

The first step in solving the k-tree-line problem is to determine if a polynomial-time solution exists. For simplicity, the 2-tree-line problem will be handled only.

The approach taken here is to count all possible 2-tree-lines on a given support tree. It should be noted that for a given data set, the number of possible k-tree-lines depends on the size of its support tree only, and not to the number of data trees in it. In this section, it will be assumed that the support tree is a full tree, i.e. all levels of the support tree include all the nodes on those levels. Another simplification is that we will assume the starting tree is the root node. This approach will give an upper bound on the 2-tree-line count, since arranging the same number of nodes in a full tree and starting from the root node will give the highest number of possible 2-tree-lines. These two assumptions will enable us to disregard the structure of an arbitrary starting tree and the support tree. The exact count of all 2-tree-lines would depend on the structure of the support tree, which will vary from data set to data set, and also it is not necessary. The aim is to check the problem's tractability only, and a polynomial upper bound on the number of possible solutions is sufficient for our purpose.

Lemma 4.2.1. *For a data set with a full support tree of m nodes, the number of all 2-tree-lines has an order of $O(m^{2.9})$.*

The proof of Lemma 4.2.1 is in Appendix A.2. This result is used to obtain the following theorem:

Theorem 4.2.1. *For a data set with a full support tree of m nodes, the run time of the brute force method of listing all possible 2-tree-lines has an order of $O(m^{2.9} \log m)$.*

Lemma 4.2.1 already establishes the order for the total count of 2-tree-lines. To prove Theorem 4.2.1, we also need the maximum length of these 2-tree-lines.

The full support tree with m nodes has a depth $\log_2(m+1)$. And it is easy to see that the 2-tree-line with maximum number of nodes in it that can be defined on this support tree has $1 + 2 * (\log_2(m+1))$ nodes. We obtain this number by using the observation that a 2-tree-line starting from the root can contain at most 2 nodes from each level, except the root level. Therefore, the maximum number of nodes contained in each 2-tree-line has an order of $O(\log m)$. Combine this with Lemma 4.2.1, and we see that the order of all nodes contained in the list of all 2-tree-lines is $O(m^{2.9} \log m)$.

The last step needed to prove Theorem 4.2.1 is to show that the brute force method needs to account every node on the 2-tree-line list only once to form the list. For this, we refer to the Section 4.3.2, where the mechanism of brute force method is described on a small example.

Theorem 4.2.1 establishes that we have a polynomial time problem with order less than 4^{th}. However, despite this initial optimistic result, a deeper look at our data sets shows that even the current order is very hard to compute with the current input size being used in this research.

The smallest data set, Data Set 1, has around 500 nodes on its support tree. This means that 500^3, or 125 billion 2-tree-lines need to be stored before searching for the optimal 2-line, and this seems to exceed the current computation capacity of regular personal computers. For that reason, a Branch & Bound based algorithm is developed to find the optimal 2-tree-lines. This algorithm will be elaborated in the following section.

4.3 A Branch And Bound Algorithm For 2-Tree-Lines

4.3.1 Background

The basic idea of this family of algorithms is as follows:

Consider the general optimization problem $\mathcal{OP}$:

$$\text{Minimize } \; G = g(x) \; \text{ Subject to: } \; x \in \mathcal{F}$$

Where $\mathcal{F}$ represents the set of all feasible solutions to the problem $\mathcal{OP}$, and G^* is the optimal solution value being sought. Notice that the definition of $\mathcal{OP}$ is generic enough so that almost any optimization problem can be written in this form.

Now suppose that the set $\mathcal{F}$ can be divided into n partitions, where each partition is called $\mathcal{F}^i$ with $i = 1 \ldots n$. Therefore we have: $\mathcal{F} = \mathcal{F}^1 \cup \ldots \cup \mathcal{F}^n$. Over these n pieces of the feasible region, we can define the problems $\mathcal{OP}^i$ as:

$$\text{Minimize } \; G_i = g(x) \; \text{ Subject to: } \; x \in \mathcal{F}^i$$

Where G_i^* are the optimal solution values of the respective problems. Clearly, we can write:

$$G^* = \min_{i=1\ldots n} \{G_i^*\}$$

The idea of partitioning a bigger problem into smaller portions is called *Branching*.

At this stage, we have n smaller problems to handle instead of the original problem, $\mathcal{OP}$. If the original problem is hard to solve to optimality, so will be the partition problems $\mathcal{OP}^i$. However, suppose that we have methods to:

1. Recognize if problem $\mathcal{OP}^i$ has no feasible solution and thus $\mathcal{F}^i = \emptyset$.
2. Find at least one feasible point $x^i_{feas} \in \mathcal{F}^i$ if $\mathcal{F}^i$ is nonempty.
3. Solve a *relaxed* version of problem $\mathcal{OP}^i$, and obtain a point x^i_{rel}. This point may or may not be in $\mathcal{F}^i$.

For each of the subproblems with a nonempty feasible region, the x^i_{feas} and x^i_{rel} are found. Since x^i_{feas} is any point in $\mathcal{F}^i$, we know that $g(x^i_{feas}) \geq g(x^{i*})$. Also it is clear that relaxation of a problem always achieves a smaller or equal objective function value than actual problem: $g(x^{i*}) \geq g(x^i_{rel})$. Therefore, through these two methods we construct an interval in which the optimal objective function value resides: $g(x^{i*}) \in [g(x^i_{rel}), g(x^i_{feas})]$.

Each iteration of the Branch And Bound Algorithm (B&B) starts with finding x^i_{feas} and x^i_{rel} for each subproblem, or identifying the subproblem to be infeasible. Such cases are excluded from further branching. For each of the feasible partitions, the intervals $[g(x^i_{rel}), g(x^i_{feas})]$ are compared. If one of the intervals is dominated by another, say, for subproblems i and j, $[g(x^i_{rel}), g(x^i_{feas})] \cap [g(x^j_{rel}), g(x^j_{feas})] = \emptyset$ and $g(x^i_{feas}) < g(x^j_{rel})$, then subproblem j and its feasible region $\mathcal{F}^j$ is removed from consideration. This step of finding bounds for each subproblem is called *bounding*, and the removed infeasible or dominated subproblems (or branches) are called to be *cut* or *pruned*.

At any point during the progress of the algorithm, the smallest lower bound of any of the partitions is the overall lower bound for the general problem, and the lowest upper bound of the partitions is the overall upper bound. This bracket gives the interval for the overall optimal objective function value at any time point.

After all possible cuts are performed, the remaining subproblems are further partitioned according to a pre-defined method. The B&B terminates when a point in $\mathcal{F}$ gets singled out as the optimal solution, or when a sufficiently small interval for the optimal solution is reached.

4.3.2 2-Tree-Lines Adaptation

Recall the objective function of the first 2-tree-line problem:

$$K_1^* = \arg\min_K \sum_{t_i \in T} d(t_i, P_K(t_i))$$

That is, we are looking for a set of ordered nodes to be added to a starting point tree where each node is a child of one of the two most recently added nodes, or, for the first two added nodes, child of the starting tree.

Throughout this section, it will be assumed that the starting tree is the root node. The algorithm we describe is easily adapted to the case where the starting tree is arbitrary, and the adaptation consists of technical details rather than any enhancement in the theory.

Let us consider the process of listing all possible 2-tree-lines. For simplicity, assume that the support tree ST is a full tree with 4 levels:

$$ST = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$$

The counting method developed in Appendix A.2 implies that there are 208 possible maximal 2-tree-lines on this support tree. To be able to list all of these 2-tree-lines, we need to start from the root node and run an iterative procedure.

Let the list of current partial 2-tree-lines at step i be called L_i. Clearly, $L_0 = \{(1)\}$, the root node. At the next step, we look for the nodes that can be used to extend the elements of current list by one node. For L_1, either node 2 or node 3 can be appended to the root node to reach a next level 2-tree-line, so $L_1 = \{(1, 2), (1, 3)\}$. Applying this logic iteratively, we get:

$$\begin{aligned}
L_2 &= \{(1,2,3),(1,2,4),(1,2,5),(1,3,2),(1,3,6),(1,3,7)\} \\
L_3 &= \{(1,2,3,4),(1,2,3,5),(1,2,3,6),(1,2,3,7),(1,2,4,5),(1,2,4,8),(1,2,4,9),(1,2,5,4), \\
&\quad (1,2,5,10),(1,2,5,11),(1,3,2,6),(1,3,2,7),(1,3,2,4),(1,3,2,5),(1,3,6,7),(1,3,6,12), \\
&\quad (1,3,6,13),(1,3,7,6),(1,3,7,14),(1,3,7,15)\} \\
&\vdots
\end{aligned}$$

Notice that the partial 2-tree-line $(1,2,3)$ in list L_2 gives rise to 4 partials in list L_3, since there are 4 possible nodes that can be appended to it.

Now, the question is that, can we develop a method to determine if a partial in a list L_i is dominated by another partial in the same list? That is, for example, if all the 2-tree-lines that can be obtained by extending partial line $(1,3,6)$ have better objective function values than the ones obtained by extending $(1,2,3)$, can we detect that without actually extending them? If that can be done, then the partial $(1,2,3)$ can be excluded (pruned) from the list and all the 2-tree-lines that start with nodes $(1,2,3)$ can be removed from consideration at an early stage.

The 2-tree-line Branch and Bound Algorithm goes over the elements of each list L_i at every iteration, calculates an upper and lower bound for the best objective function value that can be obtained starting with that particular node sequence, and removes the partial lines that are dominated by others within that list.

If we go back to the terminology of the previous subsection, the feasible region of the main problem, $\mathcal{F}$ corresponds to all the 2-tree-lines within the support tree. With the first iteration, the feasible region is divided into two partitions: $\mathcal{F}^1$ is the set of all 2-tree-lines in the support tree that start with sequence $(1,2)$ and $\mathcal{F}^2$ is the set of lines starting with $(1,3)$. A dominance check is performed, and in the next iteration, remaining un-dominated partitions are further divided according the next possible nodes that can be appended to them. The procedure terminates when a set of maximal 2-tree-lines with optimal objective function is reached. The optimal solution does not have to be unique for this problem, so alternative optima are possible.

An observation that will be used later is that, given a data set, the total distance to the data set as we add nodes to a 2-tree-line sequence is non-increasing. Each next point on the line is obtained by adding a single node to the previous, and for each data tree, if this single node adds to the distance, then the projection remains at a previous point on the line. Therefore distance of the data tree to the projection remains the same. This results in the fact that the sum of distances of the trees in a data set to a partial 2-tree-line either improves or stays the same as more nodes are added to it.

When a 2-tree-line reaches to its maximum length and no more nodes can be added to it, then partitioning for that branch stops, and the upper and lower bounds are both set to the sum of distances of the data set to the line.

Clearly, the effectiveness of this algorithm depends on finding tight upper and lower bounds on the objective function values of each branch. Loose bounds, in other words, large brackets will result in not being able to eliminate enough number of branches.

An intuitive method to calculate both a lower and an upper bound for each branch is proposed in the following Subsection.

4.3.3 The 2-Path Problem: A Bound Determination Method For 2-Tree-Line B&B

Remember that in the 1-tree-line problem, the notion of a path with maximum sum of weights is used. In this case the maximum path is equivalent to the first principal component if the root node is taken as the starting point, and the sum of weights belonging to this line exactly equals the sum of lengths of projections of data trees onto this line. This property follows from the fact that once a 1-tree-line steps out of a data tree, none of the following added nodes will be members of that data tree, so the projection can be determined by this unique cut off point. Therefore the set of nodes of a 1-tree-line that are common with a given data tree gives the size of projection.

In the 2-tree-line case, this property that enabled a linear time algorithm to find optimal 1-tree-lines is lost. The 2-tree-line can step in and out of a data tree.

However, if we examine the projection of any data tree onto a 2-tree-line closer, we can see that there is still some structure. Notice that if two consecutive nodes on a 2-tree-line are not members of a given data tree, then none of the nodes added afterwards will be members. By the definition of 2-tree-lines, all consecutive nodes after these two will be descendants of them, and if parent nodes are not members of a tree then neither are their children. Therefore we can say that the 2-line sequence can only ”step out” for one node at a time if it will step back in. To illustrate this idea on an example, consider the data tree t:

$$t = (1, 2, 3, 6, 12, 24, 48, 49)$$

And a 2-tree-line K starting from the root node that is defined by the sequence:

$$(1, 2, 3, 4, 6, 12, 13, 24, 26, 52, 53)$$

Along this sequence, denote the nodes that are members of t with ”$+$” and non-members with ”$-$”, to get the structure:

$$(+ + + - + + - + - - -)$$

The next question is that, does this structure lead to a characterization of the projection of a data tree onto a line? On the sequence, the projection corresponds to first ”m” nodes that will yield the smallest distance to the data tree. Each ”$+$” in the previous representation is a node common to the line and the data tree, so including it in the projection will reduce the distance by 1 unit. On the other hand, each ”$-$”, that is, a node existing in the line but not in the data tree, will increase the distance by 1 unit if included in the projection. It can be seen that in the 2-tree-line case each ”$-$” node that is followed by a common (”$+$”) node is compensated by this follower, so including both in the projection does not increase the distance, and possible ”$+$”'s that will follow may reduce it. However, two consecutive negatives will indicate that the line has stepped out of the tree forever and thus these nodes and their descendants cannot be in the projection. Therefore, the last ”$+$” on this sequence determines the cut off point for the projection, or at least one of the projections since these do not have to be unique.

For the previous example, the projection of the data tree t onto K is $(1, 2, 3, 4, 6, 12, 13, 24)$, or if we use the ”$+-$” representation, $(+ + + - + + -+)$. The distance of the projection to t can be given as:

$$d(t, P_K(t)) = |t| - |t \cap P_K(t)| + |P_K(t) \backslash t|$$

Where $|t \cap P_K(t)|$ is in fact the count of ”$+$”s and $|P_K(t) \backslash t|$ is count of ”$-$”s on the projection.

Going back to the objective function of the minimization problem, the sum of distances of the data trees to their projections was:

$$\begin{aligned}
\min_K \sum_{t_i \in T} d(t_i, P_K(t_i)) &= \min_K \sum_{t_i \in T} (|t_i| - |t_i \cap P_K(t_i)| + |P_K(t_i) \backslash t_i|) \\
&= \min_K (\sum_{t_i \in T} |t_i| - \sum_{t_i \in T} |t_i \cap P_K(t_i)| + \sum_{t_i \in T} |P_K(t_i) \backslash t_i|) \\
&= \sum_{t_i \in T} |t_i| - \max_K (\sum_{t_i \in T} |t_i \cap P_K(t_i)| - \sum_{t_i \in T} |P_K(t_i) \backslash t_i|) \\
&\geq \sum_{t_i \in T} |t_i| - \max_K (\sum_{t_i \in T} |t_i \cap P_K(t_i)|)
\end{aligned}$$

Combine this with the fact that all nodes that are both in the sequence of the 2-tree-line and a given data tree (the ”$+$”s) are included in the projection, we see that the count of these common nodes, i.e., the maximum sum of weights of nodes of a 2-tree-line give us a lower bound on the best objective function. Note that what was exactly the optimal objective function value in 1-tree-lines now becomes a lower bound.

To find this lower bound, the order of nodes within the 2-tree-line can be ignored. Therefore, given a support tree ST, calculating the **maximum** **2-path** is sufficient to obtain the bound we are looking for, of which definition is as follows:

Definition 4.3.1. *A* ***2-path*** *in any given tree t is a sub-tree of t which starts from the root node and includes at most 2 nodes on each level of t. The* ***maximum 2-path*** *of a support tree ST is a 2-path in the support tree with maximum sum of weights.*

A dynamic programming approach is used to find the maximum 2-path given a support tree. This same approach also can be used to find the maximum 2-path in a support tree when the first m nodes of the path are fixed. This means we will be searching for the 2-path that starts with the given m-node sequence and extends in such a way that it covers most weight under that condition.

To find the upper bound for the subproblem, we will need the maximum 2-path that gave the lower bound. In the generic B&B algorithm, the objective function value of any point within the feasible region can be used as the upper bound. In our case, this feasible point (a feasible 2-tree-line) is chosen as a line that goes through the nodes of the maximum 2-path that gave the lower bound.

Although this line does not maximize the total length of projections, and thus is not necessarily the optimal line, in practice, it provides a tight upper bound. Recall that, in the optimization problem that defines the best 2-tree-line, we have subtracted the term $\sum_{t_i \in T} |P_K(t_i) \backslash t_i|$ from the objective function, and maximized the total number of ”$+$”'s in the projections. Using this upper bound means we are now looking at the same set of nodes that made up the maximum 2-path, but now we are calculating the actual projection lengths. That is, the ”$-$”'s in the projection sequences are now also being considered. In fact, the count of ”$-$”'s (which is equivalent to $\sum_{t_i \in T} |P_K(t_i) \backslash t_i|$) exactly gives the gap between the upper and lower bounds for the subproblem.

The numerical results at the end of the section verify that these lines indeed provide very close if not exact approximations of the objective function value.

4.3.4 Pseudo Code

Let $T = \{t_1, t_2, ..., t_n\}$ be the binary tree data set, SP the starting tree, and ST the support tree with weights attached. The objective is to find K^*, the 2-tree-line that starts from SP and minimizes the sum of the distances of the data trees to their projections in it. Also define L_i as the set of partial 2-tree-lines at iteration i with $L_0 = \{SP\}$.

For each i:

- Go over the members of partial line list $L_i = \{l_i^1, l_i^2, ..., l_i^k\}$ and determine the set of nodes that are feasible to be added to each partial, that is, the set of nodes that are children of the active nodes and not yet added.

- Form the partial list L_{i+1} by extending each of the partial lines with all possible next nodes. Note that each partial line in L_i can give rise to at most 4 partial lines at L_{i+1}.

- For each partial line l_{i+1}^k in L_{i+1}:
  - Determine the 2-path that goes through it.
  - Calculate the upper and lower bounds for the best objective function value that can be obtained by extending the partial l_{i+1}^k. Call this bracket $[LB_{i+1}^k, UB_{i+1}^k]$.

- For any partial line pair $\{l_{i+1}^k, l_{i+1}^j\}$, if $UB_{i+1}^j < LB_{i+1}^k$, then partial l_{i+1}^k is dominated by partial l_{i+1}^j, so delete l_{i+1}^k from the list.

Stop when a set of optimal maximal 2-lines are reached. The output list of 2-tree-lines obtained have the same upper and lower bounds since they are maximal lines, and thus they have the same objective function values.

4.4 Numerical Results

This section starts with a performance analysis of the B&B algorithm developed in Section 4.3. Next, the results obtained regarding the 3 available data sets are examined in Section 4.4.2. In particular, a comparison of the variation explained by the first 2-tree-line to that of $PC1$ and $PC1 \cup 2$ of the 1-tree-lines is drawn, and the age effect on projection sizes are examined.

4.4.1 Performance Analysis Of The 2-Tree-Line B&B

In terms of performance analysis, there are several measures that can be used to determine the contribution of an algorithm to computation power. One can investigate the size of the largest problem instance that can be solved with previous methods and compare it with the possible size that the new method can deal with. Another possibility is to compare computation times of previous and new methods for the same instances.

The 2-tree-line B&B is developed with the current data sets in mind, so these are used to measure performance. The size of the largest problem that can be handled by the brute force method has not been measured, but experience revealed that the optimal 2-tree-lines for current data sets cannot be found using a regular Personal Computer. The memory requirement for the number of 2-tree-lines that need to be stored for these data sets seem to exceed the current capacity. As seen in Figure 4.1, the largest number of partial lines that needs to be stored by the B&B algorithm at once is 131, therefore the memory limitation is overcome.

As pointed out previously, the B&B algorithm terminates in $O(\log n)$ steps for a data set with a full support tree. Combined with the fact that the size of variables kept in memory at each step is relatively low, the 2-tree-line B&B enables us to find the optimal solutions in a short time interval. In fact for the largest data set at hand (Set 2), the optimal result was obtained in a matter of minutes whereas no results at all could be reached for this data set when the brute force method was used.

Figure 4.1 summarizes the progress of B&B for each of the 4 subpopulations of Data Set 3. In each of the 4 charts the x axis signifies the number of iterations run before the optimal value is reached. The y axis is the number of branches (or nodes) created in the B&B tree at that iteration, which is equivalent to the size of the partial line list L_i for that iteration. The blue bars indicate the number of partial lines created at that iteration, while the red bars give the number of remaining partial lines at that iteration after the pruning step is executed.

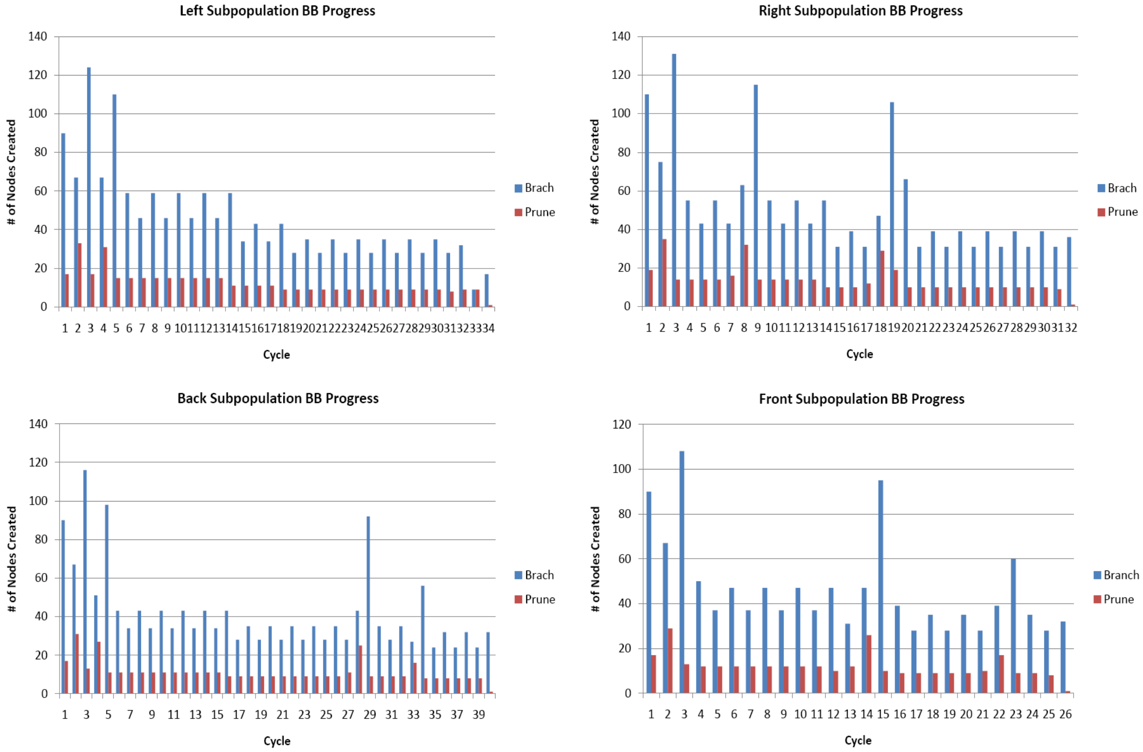

Figure 4.1: The B&B algorithm progress for 4 subpopulations of Data Set 3. Blue bars indicate the number of partial lines created at the beginning of each step. Red bars give the number of partial lines remained after the pruning step for each iteration.

As seen in Figure 4.1, the number of partial lines stored at any time point never gets large due to the cuts performed at each step. This property lets the B&B algorithm reach the final results very quickly.

4.4.2 Data Analysis

One interesting question regarding the 2-tree-lines is, how much of the existing variation in the data sets can be explained by them compared to the $PC1$ and $PC1 \cup 2$ of 1-tree-lines? It is reasonable to expect the coverage of $PC1 \cup 2$ of 1-tree-lines to be close to the coverage of the 2-tree-lines, but it should also be noted that it is possible to build artificial examples where that intuition can be countered. Moreover the structure revealed by the 2-tree-lines will surely be different than the shape of the $PC1 \cup 2$ of 1-tree-lines.

		$PC_1 1$	$PC_1 1 \cup 2$	$PC_2 1$
Set 3	B	2501	3039	3336
	L	2449	2817	3232
	R	2633	3008	3404
	F	2336	2749	3006
Set 2	B	2648	3213	3545
	L	2570	2977	3389
	R	2779	3169	3601
	F	2465	2885	3179
Set 1	B	1555	1803	1884
	L	1570	1741	1784
	R	1742	1939	1937

Table 4.1: The number of nodes explained by $PC_1 1$, $PC_1 1 \cup 2$ and $PC_2 1$ for all data sets.

Table 4.1 shows the number of the nodes explained by first 1-tree-line PC ($PC_1 1$), the combination of the first and second PC's of the 1-tree-lines ($PC_1 1 \cup 2$) and the first 2-tree-line PC ($PC_2 1$), for all sub-populations of the 3 data sets. The score of $PC_2 1$ is consistently slightly higher than that obtained by $PC_1 1 \cup 2$. This tells us that the first 2-tree-line explains more than the first two 1-tree-lines combined.

The second question is: What information can we infer about the underlying structure of our data sets using 2-tree-lines? Recall that the first principal components of 1-tree-lines provided valuable insight on symmetry issues. Now we will investigate if the same observations are available using the 2-tree-line analysis and if any more insights can be obtained.

Figures 4.2, 4.3, 4.4 and 4.5 depict the 2-tree-lines and the first two 1-tree-lines drawn on each subpopulation's support trees. The black nodes indicate the starting trees in all plots, while red nodes constitute the first principal components ($PC_1 1$ on left hand side and $PC_1 2$ on right hand side plots) and green nodes are second principal components ($PC_2 1$ on left hand side).

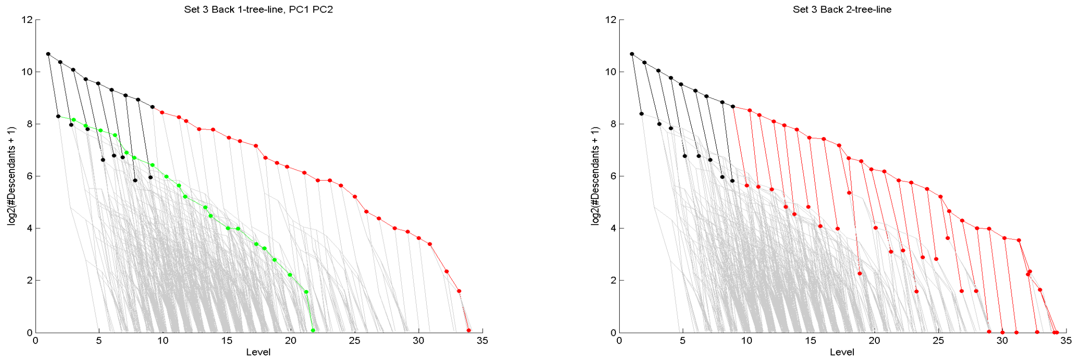

Figure 4.2: Comparison of 1-tree-lines and 2-tree-lines for Data Set 3, Back sub-population.

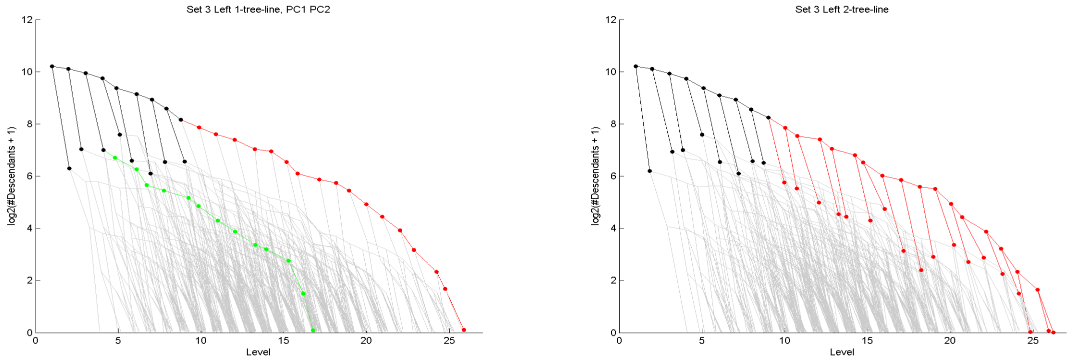

Figure 4.3: Comparison of 1-tree-lines and 2-tree-lines for Data Set 3, Left sub-population.

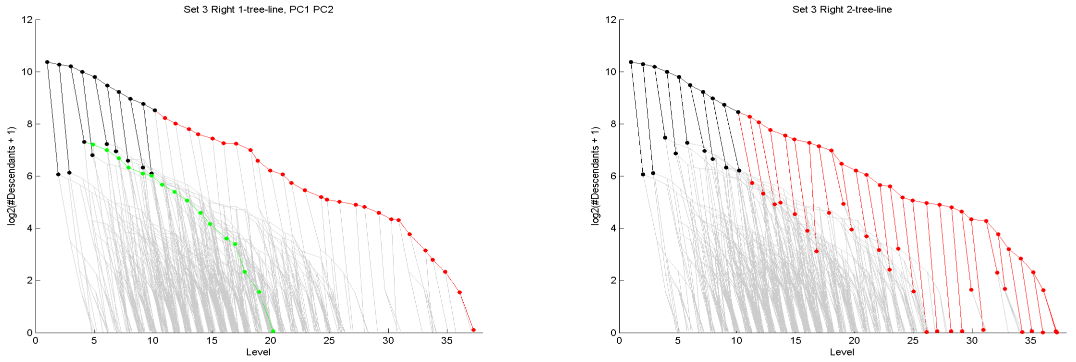

Figure 4.4: Comparison of 1-tree-lines and 2-tree-lines for Data Set 3, Right sub-population.

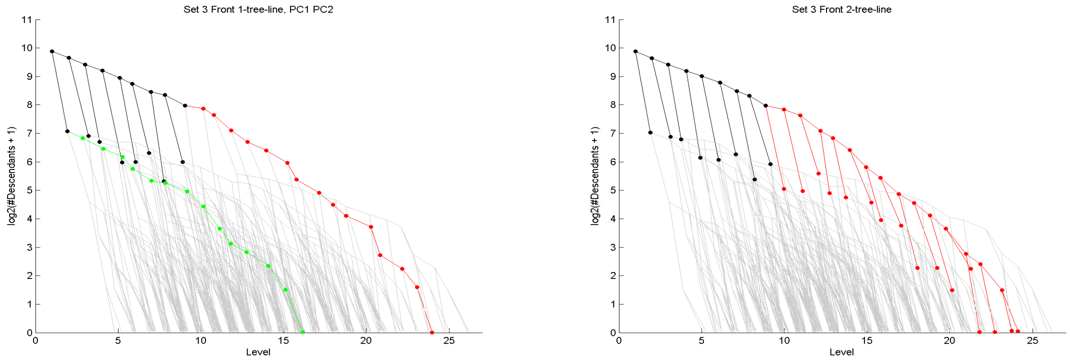

Figure 4.5: Comparison of 1-tree-lines and 2-tree-lines for Data Set 3, Front sub-population.

As can be easily seen from the figures, the first principal components of the 2-tree-lines invariably follow the path of $PC1$ of the 1-tree-lines, with the exception that siblings of the same nodes now appear on the line. This is a consequence of our original data extraction. Recall from Section 1.2 that, whenever a vessel split onto two smaller vessels, two nodes are added to the corresponding binary tree, and thus two sibling nodes on the binary tree represent the trunks of two vessels that split from one parent vessel trunk. Therefore the binary trees in the data sets have nodes with either zero or two children. In other words, if a node exists in one of the binary trees, then so does its sibling.

The 1-tree-lines can only follow a 1-path in the support tree, therefore the double-node nature of the data sets is lost. The first PC follows the path determined by the sibling which is the parent of the rest of the nodes on the line. Although each of the nodes on the line has a sibling with the exact same weight, they cannot appear on the first PC due to the structural limitation, and the second PC simply follows another path instead of covering these sibling nodes since its nodes have to be connected. The 2-tree-lines seem to remedy this shortcoming. Although the second, third, etc. principal components of the 2-tree-lines are not calculated within the context of this text, it is reasonable to expect them to follow the structure of the second, third, etc. principal components of 1-lines.

This reasoning also explains why the $PC_2 1$ explains more nodes than $PC_1 1 \cup 2$. The $PC_1 1$ goes through the path with maximum sum of weights, and $PC_1 2$ through a path that has a slightly smaller sum. The siblings of the nodes on the $PC_1 1$ path also have the same exact weight count, so being able to include them into the $PC_2 1$ results in a better coverage than $PC_1 1 \cup 2$. Note that the score of $PC_2 1$ is not the double of $PC_1 1$ in Table 4.1 since the starting tree also contributes to the scores.

Finally, the age effect on the 2-tree-line scores is investigated. The slope p-values for all 3 data sets are listed in Table 4.2, along with the slope p-values obtained from the 1-tree-line principal components.

		$PC_1 1$	$PC_1 1 \cup 2$	$PC_2 1$
Set 3	B	0.0156	*	0.0159
	L	*	*	*
	R	0.0186	0.0002	*
	F	*	*	*
Set 2	B	0.0441	*	0.0101
	L	*	0.0056	*
	R	0.0178	0.0002	*
	F	*	0.0056	*
Set 1	B	*	*	*
	L	0.003	0.003	0.003
	R	*	*	*

Table 4.2: The slope p-values obtained by $PC_1 1$, $PC_1 1 \cup 2$ and $PC_2 1$ for all data sets. The slope p-values above the 0.05 significance limit are marked with (*).

In each case the $PC_2 1$ found a significant slope at least one of the 1-tree-lines found as well. The table shows that the use of the 2-tree-lines do not give an age-dependence proof that could not be found by the 1-tree-lines. However, the ability of 2-tree-lines to capture the two-split nature in the data sets is a clear advantage over 1-tree-lines, and the computational ease of solving this problem presents this option as a valuable tool in structure searching in tree shaped data sets.

CHAPTER 5

Visualization And Data Cleaning

5.1 Introduction

As in many real life examples, the data set collected to use in this study is known to include discrepancies and noise. Elimination of these may result in sharper results, or even relationships of tree structure to covariates that have not been visible before may become discoverable. The improvement of results, in terms of finding a strong age effect in Data Set 2 over Data Set 1, is a remarkable example.

Graphically representing each data tree can be a powerful method to both understand the data and discover any problems in it. Carefully designing a visual representation of the data provides efficient visual inspection of each of these instances and may enable the discovery of certain problems without any further diagnostics. The challenge is to clearly display the important aspects of the data and eliminate the details that will not help with the diagnostics. In this section, an efficient method of visualization is proposed. An example can be seen in Figure 5.1.

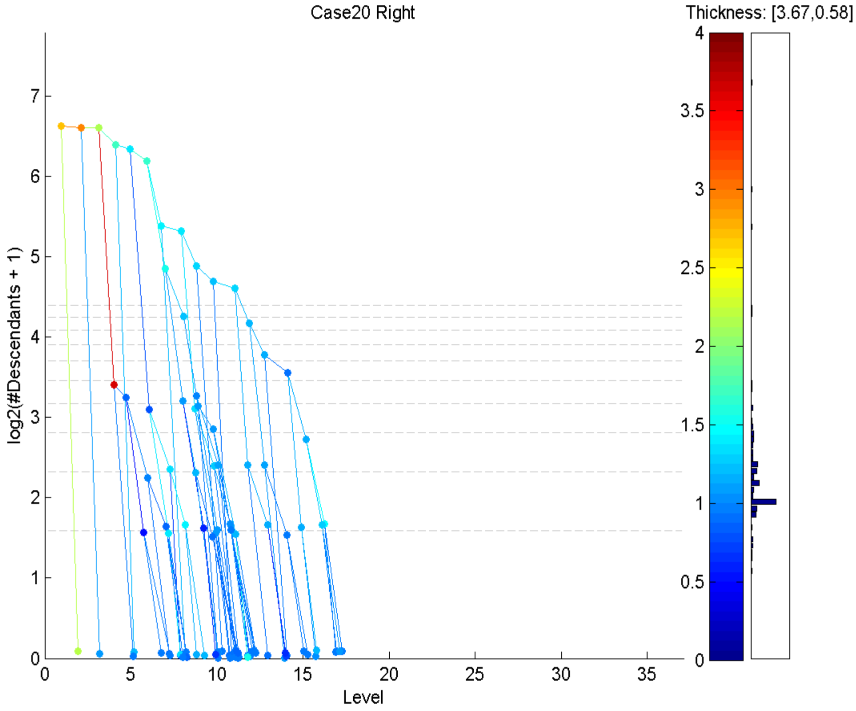

Figure 5.1: A visual display of the right tree of subject 60, Data Set 2.

In Figure 5.1, each node is represented with a colored dot and each parent-child couple is connected with a line segment, revealing the connectivity. The coordinates of each node displays its number of descendants (vertical axis) and its level on the binary tree (horizontal axis). For a compact summarization, the logarithm with base 2 of the descendant count is the y-axis.

In the lower levels, many nodes have same coordinates (number of descendants and level). To avoid the over-plotting of these nodes, some jittering (adding small amounts of random displacements for improved visual separation) is applied at lower levels.

The nodes are colored to indicate the median thickness of the vessel segment they represent. In the brain vessel data set, this thickness rarely exceeds 4 millimeters, so the assumed range of thickness is $[0, 4]$. This range is linearly projected onto a color map which consists of 100 shades, or bins, shown as the color bar on the right. The thicknesses close to $4mm$ correspond to dark red shades. As vessels thin, the color follows the usual spectrum through yellow, green, to a dark shade of blue. A bar chart indicating the counts of the nodes that fall into the range of each bin is also displayed on the right hand side of the figure. The numbers displayed on the right top corner show the thickness range of the nodes for that particular data tree, i.e. in Figure 5.1 the thinnest node has a thickness of $0.58mm$.

Through an intensive collaboration with Dr. E. Bullitt, several types of discrepancies that may exist in the data have been identified. Note that some natural noise exists in this data along with tracking and labeling errors listed below. The aim of this section is to design a clear representation of the data such that these errors can be told apart from the natural noise. In Figure 5.1, on level 4, a node thicker than its parent exists, which is not a regular occurrence. However, the thickness difference of that node with its parent is within measurement error range, so this instance is not flagged for checking.

The Data Set 2 is a result of a different, more anatomically based, clean up process performed on Data Set 1 by Dr. Alim Ladha and the addition of 34 more cases. That clean-up process aimed to correct the cut off point problems of the vessel systems, by manually going through each instance and examining them. This study is meant to develop a visual method to diagnose a wider range of problems as explained in Section 5.2 without scrutinizing the raw data form as shown in Figure 1.2, which is loaded with information and thus is hard to examine. The methods explained here are applied to Data Set 2, as presented in Section 5.3, to obtain Data Set 3.

5.2 Targeted Discrepancies

The major kinds of discrepancies that can be corrected through inspection can be listed as follows:

- **Misconnections:** In the process of building the representation seen in Figure 1.2, due to lack of accuracy in MRA image slices, sometimes vessels that come too close at a certain point are identified as connections. This error may result in misinterpretation of the blood flow of direction, and one of these vessels is seen as an extension of the other in the model. Normally, the vessel trunks that are connected to each other are expected to have similar thicknesses, and this thickness should gradually decrease as one goes from root nodes to the leaves. In a misconnection situation, the thickness may make a sudden jump, i.e. a thinner parent node may have a thicker sub-tree descending from it. Figure 5.2 is a possible candidate for this situation. Notice that the red/yellow subtree starting from level 6 is much thicker than its parent vessel.

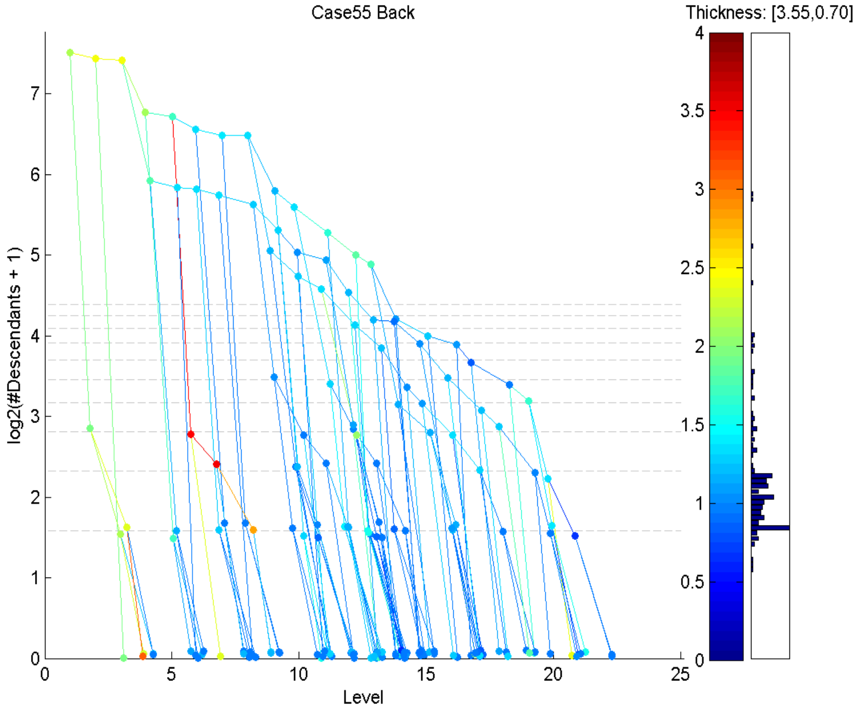

Figure 5.2: A visual display of the back tree of subject 55. Red/yellow subtree starting from level 6 is a possible misconnection.

- **Starting Point Problems:** Determining the point where a vessel enters the brain highly depends on the cutoff point of the MRA images. In some cases, this cutoff point is mistakenly taken at a too low or too high level. It is also possible to mark a child of a root as the root node. Figure 5.3 shows a tree with a possible starting point problem. The initial series of thick red nodes suggest that the MRA starting point was taken too low.

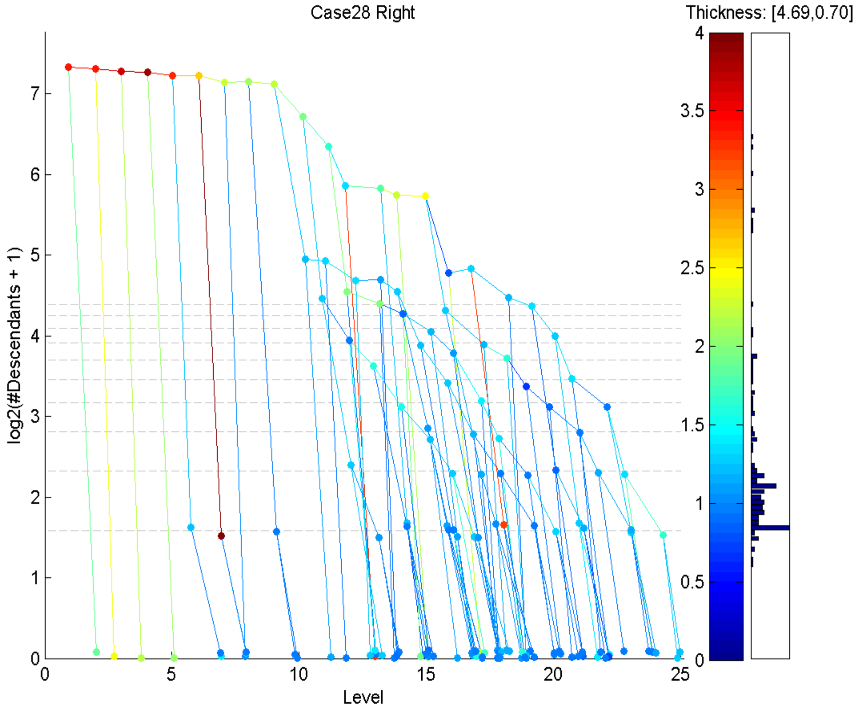

Figure 5.3: A visual display of the right tree of subject 28. The irregularities close to the root node indicate a possible starting point problem.

- **Veins:** The tracking system used to obtain the data trees is intended to record only arteries in the brain. However, in some cases, veins that run very close to an artery are mistakenly identified as an extension of that artery. These veins are usually thicker than the parent artery, and show up as red leaf nodes on the visualization. A possible example of this kind of problem is displayed in Figure 5.4.

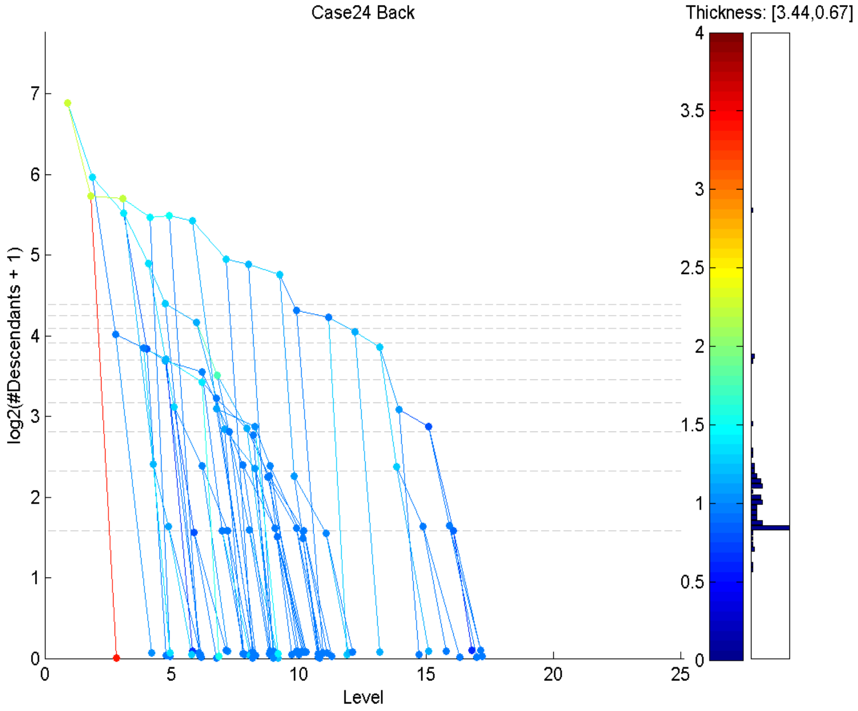

Figure 5.4: A visual display of the back tree of subject 24. The red leaf on the third level is identified as a possible vein.

5.3 Corrections

Through a careful inspection of the visual displays of all the data points in the data set, problematic instances are marked to be reviewed again. The number of instances marked for review for each sub-population is given in Table 5.1.

Misconnection				Starting Point				Vein			
B	L	R	F	B	L	R	F	B	L	R	F
7	3	4	2	3	6	4	3	18	16	23	9

Table 5.1: The number of instances marked for each kind of problem for reviewing in Data Set 2.

Note that the 4 sub-populations contain a total of 420 data trees. The process resulted in spotting 98 of them (24%) as possibly problematic instances that require reviewing.

The visualization method is not claimed to detect all possible problems that exist in the data sets. It relies on the irregular thickness differences of seemingly adjacent nodes. It is possible that there are still instances in the data set that contain for example mis-connected vessels, but if those vessels wrongly connected have similar thicknesses, they will show up as normal branches on the visualization. However, there is no way of spotting those problems without using anatomical knowledge and the raw data, which would be an enormously time consuming task. The aim here is to provide a simple and quick new tool for finding problem instances.

Another point is that, the data set contains a great deal of noise. Slight movements of subjects during MRA process also may blur the images as can be seen in Figure 1.1. Pre-set resolution settings may slightly change the outcome of the scanning process. Therefore the thicknesses obtained by tracking them inherently include a large noise level. It is inevitable that some of the natural noise will also be flagged by our visual diagnostic. The task of trying to determine correctly what is within the levels of natural noise and what instances should be manually checked is the aim of this Section.

As a result of a careful study of the instances flagged, 7 cases are deemed to be severely problematic, and were excluded from the data set. For the remaining cases, Table 5.2 shows the number of instances that were manually modified out of each problem group.

Misconnection				Starting Point				Vein			
B	L	R	F	B	L	R	F	B	L	R	F
4	2	3	0	2	6	4	3	4	1	5	1

Table 5.2: The number of instances manually modified for each kind of problem in Data Set 2 after raw data inspection.

In total, 35 cases out of the 98 marked were selected for manual modification through inspection of the raw data. 7 were excluded from data set. The artifacts in the remaining 56 instances were manually determined to be consistent with the natural noise.

It should be noted that a few of the cases that are flagged for inspection turned out to have only noise at the locations that got them flagged, but other problems are discovered during the raw data examination and these got fixed. Although there are very few cases of this type, this is an indication that other normal-looking instances might contain discrepancies that cannot be detected by our diagnostic visualization, but require the raw data examination to be discovered. However, as stated before, this inspection would take a great deal of time and and attention to complete, so is infeasible.

Next, each of the example cases with discrepancies displayed before will be re-drawn, this time after the cleaning process.

Figure 5.5 shows the corrected version of Case 55, Back subtree (Figure 5.2). Notice that the red/yellow subtree that was incorrectly attached to this tree has been removed.

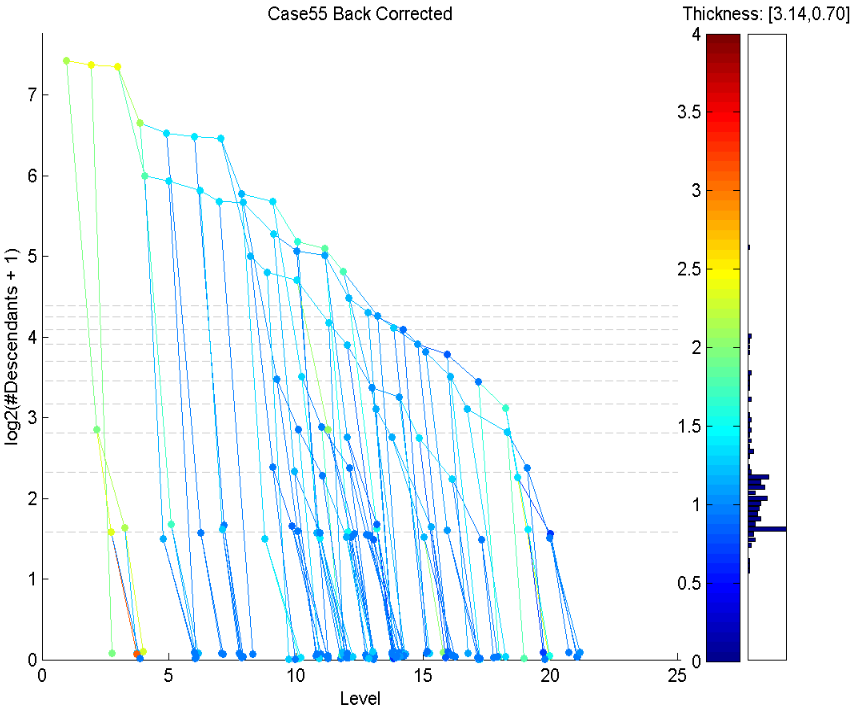

Figure 5.5: A visual display of the Back subtree of subject 55 (Figure 5.2) after correction. The misconnected subtree is deleted.

After Figure 5.3 revealed that there might be a possible starting point problem in Case 28 Right subtree, a revision done by Dr. E. Bullitt resulted in Figure 5.6. Notice that the irregularity close to the root node (a series of dark red nodes) is corrected by changing the cut-off level to a higher point.

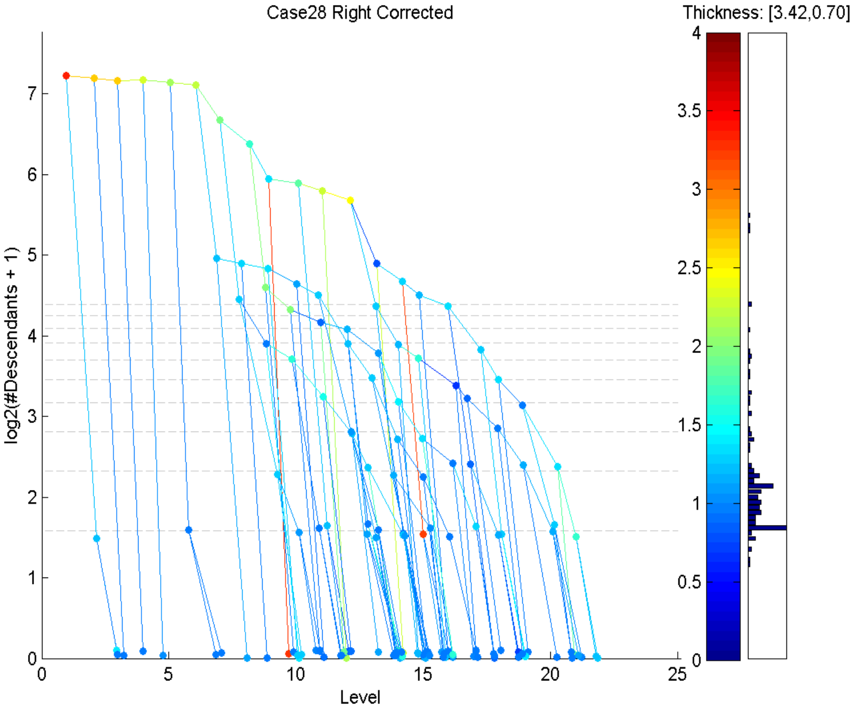

Figure 5.6: A visual display of the Right subtree of subject 28 (Figure 5.3), corrected by changing the cut-off point of the root vessel.

With the revision of subject 24's Back subtree, the red leaf node seen in Figure 5.4 was indeed identified as a vein and was removed from the data tree. The resulting corrected tree can be seen in Figure 5.7.

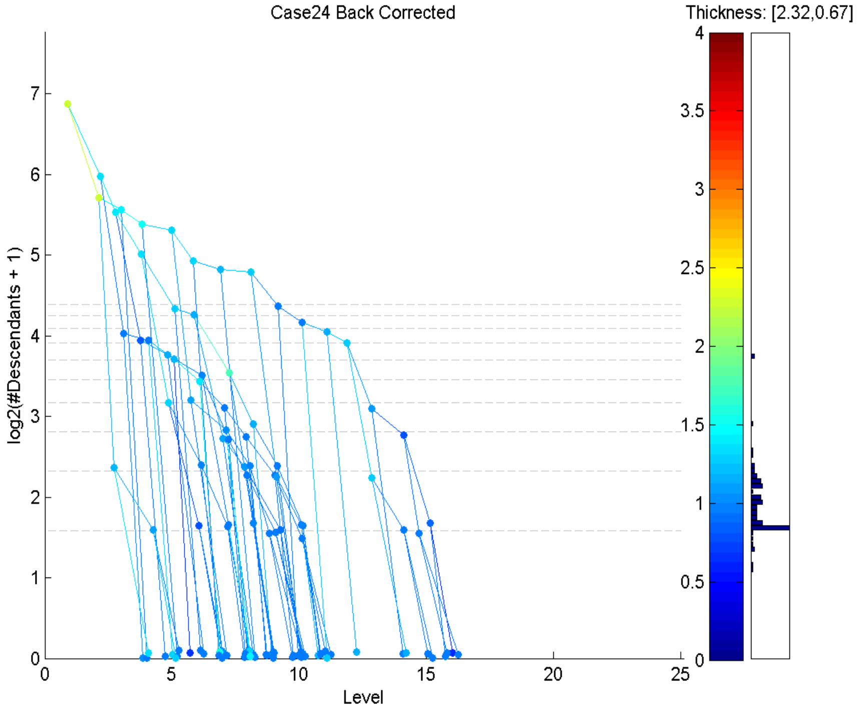

Figure 5.7: A visual display of the Back subtree of subject 24 (Figure 5.4), corrected by the deletion of a vein.

5.4 Results

After the 7 high-problem instances are removed from Data Set 2 and the 35 corrected cases are replaced, the Data Set 3 is obtained.

An important question regarding the clean up study is that: does the elimination of several problems improve the data set in terms of age effect? In other words, is the age effect more pronounced now or not? Table 5.3 summarizes the slope p-value comparisons of two data sets (2&3) using the 1-tree-line tool.

		Set 3	Set 2
Back	PC1	0.0156	0.0441
	PC2	0.6912	0.6256
	PC3	0.1275	0.0924
	PC4	0.1109	0.0775
	PC1U2	0.1370	0.2672
	PC1U2U3	0.1311	0.0283
	PC1U2U3U4	0.0005	0.0110
Front	PC1	0.3532	0.3818
	PC2	0.9519	0.8489
	PC3	0.5659	0.7105
	PC4	0.0232	0.0129
	PC1U2	0.4710	0.3683
	PC1U2U3	0.3277	0.3010
	PC1U2U3U4	0.0282	0.0202
Right	PC1	0.0186	0.0178
	PC2	0.0164	0.0221
	PC3	0.5757	0.3825
	PC4	0.2692	0.2752
	PC1U2	0.0002	0.0002
	PC1U2U3	0.0004	0.0001
	PC1U2U3U4	0.0002	0.0002
Left	PC1	0.0977	0.5796
	PC2	0.7684	0.0490
	PC3	0.2597	0.1948
	PC4	0.0644	0.0422
	PC1U2	0.2503	0.0056
	PC1U2U3	0.0765	0.0599
	PC1U2U3U4	0.0063	0.0035

Table 5.3: The slope p-values obtained using the 1-tree-line tool. Columns compare the Data Sets 2 and 3.

Table 5.3 shows us that the significance levels of slope p-values obtained from Data Set 3 usually corresponds closely to those of obtained from Data Set 2. If we assume a 0.02 significance level, we see that in two cases (Right sub-population, PC2 and Back sub-population, PC1) the relation between age and projection size becomes significant in the new data set, while in two cases (Left sub-population, PC1U2 and Front sub-population, PC4) the existing significance is lost. The other commonly used significance level for evaluating p-values is 0.05. According to this level, in two sub-populations (Back, PC1U2U3 and Left, PC2, PC4, PC1U2), a significance level loss is observed.

The other two available tools (2-tree-lines and tree-curves) yield very similar results, so they are not shown here.

The main conclusion is that the corrections performed on the data set did not dramatically affect the age-projection size relation. It is clear that the visualization process resulted in the correction of many instances and thus provided a more reliable data set. However, most of these corrections are local compared to the sizes of whole trees within the sub-populations, and they did not change the general trend of *branchiness* within those trees: data trees with many branches still have a lot branches and vice versa. Another point is that, although fixed, the variation created by these mistakes is relatively less compared to the inherent natural noise, therefore removing it did not create a big jump in significance levels.

Despite this result, the contribution of the visualization method is clear. 43% of the instances that are marked for inspection revealed mistakes that it was appropriate to correct. The method provides an effective diagnostic for data clean up in similar populations of binary trees.

Appendix

A.1 Proof of Theorem 2.1.1

This section is devoted to the proof of Theorem 2.1.1 with some accompanying claims.

Claim A.1.1. *Let $L = \{\ell_0, \dots, \ell_m\}$ be a tree-line, and t a data tree. Then*

$$P_L(t) = \ell_0 \cup (t \cap V_L). \tag{A.1.1}$$

Proof: Since $\ell_i = \ell_{i-1} \cup v_i,$ we have

$$d(t, \ell_i) \quad = \quad \begin{cases} d(t, \ell_{i-1}) - 1 & \text{if } v_i \in t; \\ d(t, \ell_{i-1}) + 1 & \text{otherwise.} \end{cases} \tag{A.1.2}$$

In other words, the distance of the tree to the line decreases as we keep adding nodes of V_L that are in t, and when we step out of t, the distance begins to increase, so Claim (A.1.1) follows.

□

Claim A.1.2. *Let $L_1, \dots, L_q$ be tree-lines with a common starting point, and t a data tree. Then*

$$P_{L_1 \cup \cdots \cup L_q}(t) = P_{L_1}(t) \cup \cdots \cup P_{L_q}(t).$$

Proof: For simplicity, we only prove the statement for $q = 2$. Assume that

$$\begin{aligned} L_1 &= \{\ell_{1,0}, \ell_{1,1}, \dots, \ell_{1,p_1}\} \\ L_2 &= \{\ell_{2,0}, \ell_{2,1}, \dots, \ell_{2,p_2}\} \end{aligned}$$

with $\ell_0 = \ell_{1,0} = \ell_{2,0}$, and

$$V_{L_1} = \{v_{1,1}, \dots, v_{1,p_1}\},\ V_{L_2} = \{v_{2,1}, \dots, v_{2,p_2}\}. \tag{A.1.3}$$

Also assume

$$\begin{aligned} P_{L_1}(t) &= \ell_{1,r_1}, \quad (A.1.4)\\ P_{L_2}(t) &= \ell_{2,r_2}. \quad (A.1.5)\end{aligned}$$

For brevity, let us define

$$f(i,j) \quad = \quad d(t, \ell_{1,i} \cup \ell_{2,j}) \text{ for } 1 \leq i \leq p_1,\ 1 \leq j \leq p_2. \quad (A.1.6)$$

Using Claim A.1.1, (A.1.4) means

$$v_{1,i} \in t, \text{ if } i \leq r_1, \text{ and } v_{1,i} \notin t, \text{ if } i > r_1, \quad (A.1.7)$$

hence

$$\begin{aligned} f(i,j) &\leq f(i-1,j) \text{ if } i \leq r_1;\\ f(i,j) &\geq f(i-1,j) \text{ if } i > r_1. \end{aligned} \quad (A.1.8)$$

By symmetry, we have

$$\begin{aligned} f(i,j) &\leq f(i,j-1) \text{ if } j \leq r_2;\\ f(i,j) &\geq f(i,j-1) \text{ if } j > r_2. \end{aligned} \quad (A.1.9)$$

Overall, (A.1.8) and (A.1.9) imply that the function f attains its minimum at $i = r_1,\ j = r_2,$ which is what we had to prove. □

Claim A.1.3. *Let S be a subset of $\mathrm{Supp}(T)$ which contains ℓ_0. For $v \in \mathrm{Supp}(T)$ define*

$$w_S(v) \quad = \quad \begin{cases} 0, & \text{if } v \in S,\\ \sum_{v \in t_i} 1, & \text{otherwise} \end{cases} \quad (A.1.10)$$

Then among the treelines with starting tree ℓ_0 the one which maximizes

$$\sum_{t_i \in T} |(V_L \cup S) \cap t_i|$$

is the one whose path V_L maximizes the sum of the w_S weights: $\sum_{v \in V_L} w_S(v)$.

Proof: For $v \in \text{Supp}(T),$ and a subtree t of $\text{Supp}(T),$ let us define

$$\delta(v,t) \quad = \quad \begin{cases} 1, & \text{if } v \in t, \\ 0, & \text{otherwise} \end{cases} \tag{A.1.11}$$

Then

$$\begin{aligned} \arg\max_{\ell \in L} \textstyle\sum_{t_i \in T} |(V_L \cup S) \cap t_i| \quad &= \quad \arg\max_{\ell \in L} \textstyle\sum_{t_i \in T} \sum_{v \in V_L \cup S} \delta(v, t_i) \\ &= \quad \arg\max_{\ell \in L} \textstyle\sum_{v \in V_L \cup S} \sum_{t_i \in T} \delta(v, t_i) \\ &= \quad \arg\max_{\ell \in L} \textstyle\sum_{v \in V_L \cup S} w_{\emptyset}(v) \\ &= \quad \arg\max_{\ell \in L} \textstyle\sum_{v \in V_L} w_S(v). \end{aligned}$$

□

Finally, we prove our main result:

Proof of Theorem 2.1.1: For better intuition, we first give a proof when $k = 1$. Using Claim A.1.1 in Definition 2.1.6, we get

$$L_1^* \quad = \quad \arg\min_{L} \sum_{t_i \in T} d(t_i, \ell_0 \cup (t_i \cap V_L)).$$

Since V_L is disjoint from ℓ_0,

$$L_1^* \quad = \quad \arg\max_{L} \sum_{t_i \in T} |V_L \cap t_i|,$$

the statement follows from Claim A.1.3 with $S = \emptyset$.

We now prove the statement for general k. For an arbitrary data tree t, and tree-line L, we have

$$\begin{aligned} P_{L_1^* \cup \cdots \cup L_{k-1}^* \cup L}(t) \quad &= \quad P_{L_1^*}(t) \cup \cdots \cup P_{L_{k-1}^*}(t) \cup P_L(t) \\ &= \quad \ell_0 \cup (V_{L_1^*} \cap t) \cup \cdots \cup (V_{L_{k-1}^*} \cap t) \cup (V_L \cap t) \\ &= \quad \ell_0 \cup [(V_{L_1^*} \cup \cdots \cup V_{L_{k-1}^*} \cup V_L) \cap t], \end{aligned} \tag{A.1.12}$$

with the first equation from Claim A.1.2, the second from Claim A.1.1, and the third straightforward.

Combining (A.1.12) with (2.1.3) we get

$$L_k^* = \arg\min_L \sum_{t_i \in T} d(t_i, \ell_0 \cup [((V_{L_1^*} \cup \cdots \cup V_{L_{k-1}^*} \cup V_L) \cap t_i]). \tag{A.1.13}$$

Again, the paths of $L_1^*, \ldots, L_{k-1}^*$ and L are disjoint from ℓ_0, so (A.1.13) becomes

$$L_k^* = \arg\max_L \sum_{t_i \in T} |(V_{L_1^*} \cup \cdots \cup V_{L_{k-1}^*} \cup V_L) \cap t_i|, \tag{A.1.14}$$

so the statement follows from Claim A.1.3 with $S = V_{L_1^*} \cup \cdots \cup V_{L_{k-1}^*}$. □

A.2 Proof Of Theorem 4.2.1

Definition A.2.1. *Let:*

$f(n) =$ *Number of* 2*-tree-lines of which last added node is on* n^{th} *level on a full support tree.*

Also define:

$f_1(n) =$ *Number of* 2*-tree-lines in* $f(n)$ *with only one node on* n^{th} *level*

$f_2(n) =$ *Number of* 2*-tree-lines in* $f(n)$ *with two nodes on* n^{th} *level*

We know that:

$$f(n) = f_1(n) + f_2(n) \quad \forall n \geq 0$$

We will write a recursive formula for $f(n)$. If we consider the most trivial case where our tree is only the root node, we obtain the initial condition for the recursion:

$$f_1(1) = 1$$

$$f_2(1) = 0$$

To get the recursive formula, assume that we know the values of $f_1(n)$ and $f_2(n)$, and we are looking for $f_1(n+1)$ and $f_2(n+1)$. First let us count the 2-tree-lines that end at $(n+1)^{st}$ level with a single node. This single node can be either one of the two children of a 2-tree-line ending at level n with a single node, or it can be one of the four children of a 2-tree-line ending at level n with two nodes.

Therefore:

$$f_1(n+1) = 2f_1(n) + 4f_2(n)$$

For $f_2(n+1)$, first consider $f_1(n)$. These lines end with a single node at n^{th} level, and have two children, where both of of them need to be added. Since the order of the addition matters, each such line gives us two options for extension. For $f_2(n)$, we need to choose two nodes out of the four children of n^{th} level nodes. However, not all of the 2-combinations of these are available. Now let us name the nodes on n^{th} level as a and b, b being the last added node. Let us name their children as a_1, a_2 and b_1, b_2 respectively. Now the possible choices for addition are (a_1, a_2), (a_2, a_1), (b_1, a_1), (b_1, a_2), (b_2, a_1), (b_2, a_2). Summing all the choices up, we get:

$$f_2(n+1) = 2f_1(n) + 6f_2(n)$$

These two formulas are valid for all n greater than 1. Now let us write these two in matrix form:

$$\begin{bmatrix} f_1(n+1) \\ f_2(n+1) \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} f_1(n) \\ f_2(n) \end{bmatrix}$$

Using this formula, we can write:

$$\begin{bmatrix} f_1(n+1) \\ f_2(n+1) \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 2 & 6 \end{bmatrix}^n \begin{bmatrix} f_1(1) \\ f_2(1) \end{bmatrix}$$

To further simplify this, we can re-write the coefficient matrix using spectral decomposition:

$$\begin{bmatrix} 2 & 4 \\ 2 & 6 \end{bmatrix} = \begin{bmatrix} \sqrt{3}-1 & 1 \\ 1 & \frac{1-\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \sqrt{3}-1 & 1 \\ 1 & \frac{1-\sqrt{3}}{2} \end{bmatrix}^{-1}$$

Where $\lambda_1 = 4 + 2\sqrt{3}$ and $\lambda_2 = 4 - 2\sqrt{3}$, the eigenvalues of coefficient matrix. Now we can get the n^{th} multiple of this easily:

$$\begin{bmatrix} 2 & 4 \\ 2 & 6 \end{bmatrix}^n = \begin{bmatrix} \sqrt{3}-1 & 1 \\ 1 & \frac{1-\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}^n \begin{bmatrix} \sqrt{3}-1 & 1 \\ 1 & \frac{1-\sqrt{3}}{2} \end{bmatrix}^{-1}$$

Insert this into $f(n+1)$ formula along with the initial conditions, and do the necessary simplifications:

$$f_1(n+1) = \frac{\sqrt{3}-1}{2\sqrt{3}}(\lambda_1)^n + \frac{\sqrt{3}+1}{2\sqrt{3}}(\lambda_2)^n$$

$$f_2(n+1) = \frac{1}{2\sqrt{3}}(\lambda_1)^n - \frac{1}{2\sqrt{3}}(\lambda_2)^n$$

Summing these up, we get the desired quantity:

$$f(n+1) = f_1(n+1) + f_2(n+1) = \frac{(\lambda_1)^n + (\lambda_2)^n}{2}$$

This result enables us to find the order of the 2-tree-line problem. We know that a full support tree with n levels has $\sim 2^n$ nodes. If we call the total number of nodes in the support tree m, we have $n = \log_2(m)$. So for a problem with full support tree size m, the total number of 2-tree-lines is:

$$\frac{\lambda_1^{(\log_2 m)-1} + \lambda_2^{(\log_2 m)-1}}{2}$$

So the order of the problem of finding all 2-tree-lines is:

$$O(\frac{1}{2\lambda_1} m^{\log_2 \lambda_1}) = O(m^{2.9})$$

Bibliography

[1] Aydin, B., Pataki, G., Wang, H., Bullitt, E., Marron, S. (2009) A Principal Component Analysis For Trees, submitted to *Annals of Applied Statistics*.

[2] Aylward, S. and Bullitt, E. (2002) Initialization, noise, singularities and scale in height ridge traversal for tubular object centerline extraction, *IEEE Transactions on Medical Imaging*, 21, 61-75

[3] Bullitt, E., Zeng, D., Gerig, G., Aylward, S., Joshi, S., Smith, J.K., Lin, W., Ewend, M.G. (2005) Vessel tortuosity and brain tumor malignancy: A blinded study, *Academic Radiology*, 12:1232-1240.

[4] Bullitt, E., Zeng, D., Ghosh, A., Aylward, S. R., Lin, W., Marks, B. L., Smith, K. (2008) The effects of healthy aging on intracerebral blood vessels visualized by magnetic resonance angiography, submitted to *Neurobiology of Aging*.

[5] Breiman, L., Friedman, J. H., Olshen, J. A., Stone, C. J. (1984), *Classification and Regression Trees*. Belmont, CA: Wadsworth.

[6] Breiman, L., (1996), Bagging Predictors, *Machine Learning*, vol 24, Number 2, 123-140.

[7] Everitt, B. S., Landau, S., Leese, M. (2001), *Cluster Analysis (4th edition)*, Oxford University Press, New York.

[8] Ferraty, F. and Vieu, P. (2006) *Nonparametric functional data analysis: theory and practice*, Berlin, Springer.

[9] Fletcher, P.T., Joshi, S., Lu, C., Pizer, S.M. (2004) Principal geodesic analysis for the study of nonlinear statistics of shape, *IEEE Transactions on Medical Imaging*, 23, 995-1005.

[10] Girshick, M.A. (1936). Principal components. *J. Amer. Statist. Assoc.*, 31, 519528.

[11] Girshick, M.A. (1939). On the sampling theory of roots of determinantal equations. *Ann. Math. Statist.*, 10, 203224.

[12] Gower, J.C. (1966). Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53, 325338.

[13] Handle (2008) http://hdl.handle.net/1926/594

[14] Holmes, S. (1999). Phylogenies: An Overview, IMA series, vol 112, on Statistics and Genetics, (ed. Halloran and Geisser), 81-119 Springer Verlag, New York.

[15] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *J. Educ. Psychol.*, 24, 417441, 498520.

[16] Hotelling, H. (1936). Simplified calculation of principal components. *Psychometrika*, 1, 2735.

[17] Jeffers, J.N.R. (1967). Two case studies in the application of principal component analysis. *Appl. Statist.*, 16, 225236.

[18] Jolliffe, I.T. (2002) *Principal Component Analysis*, Berlin, Springer.

[19] Li, S., Pearl, D. K., Doss, H. (2000) Phylogenetic Tree Constructure Using Markov Chain Monte Carlo. *Journal of the American Statistical Association*, 95:493- 508.

[20] Midag Web Site: http://midag.cs.unc.edu

[21] Muirhead, R.J. (1982) *Aspects of Multivariate Statistical Theory*, New York : Wiley.

[22] Müller-Molina, A. J., Hirata, K. and Shinohara, T. (2009) A Tree Distance Function Based on Multi-sets, *New Frontiers in Applied Data Mining: PAKDD 2008 International Workshops, Osaka, Japan, May 20-23, 2008. Revised Selected Papers*, Berlin, Heidenberg: Springer-Verlag.

[23] Pearson, K. (1901) On Lines and Planes of Closest Fit to Systems of Points in Space, *Philosophical Magazine*, 2(6), 559-572

[24] Pizer, S. M., Thall, A., Chen, D. (1999), M-Reps: A New Object Representation for Graphics, Submitted to *ACM TOG*.

[25] Preisendorfer, R.W. and Mobley, C.D. (1988). *Principal Component Analysis in Meteorology and Oceanography.* Amsterdam: Elsevier.

[26] Ramsay, J. O. and Silverman, B. W. (2002) *Applied Functional Data Analysis*, New York: Springer-Verlag.

[27] Ramsay, J. O. and Silverman, B. W. (2005) *Functional Data Analysis*, New York: Springer-Verlag (2nd edition).

[28] Rao, C.R. (1964). The use and interpretation of principal component analysis in applied research. *Sankhya A*, 26, 329358.

[29] Shawe-Taylor, J. and Christianini, N. (2000) *Kernel Methods For Pattern Analysis*, Cambridge University Press.

[30] Siddiqi, K. and Pizer, S.M. (2007) *Medial Representations Mathematics Algorithms and Applications*, Springer, New York.

[31] Wang, H. (2003). Functional data analysis of populations of tree-structured objects. Ph.D. dissertation, *Dept. Statistics, Univ. North Carolina at Chapel Hill.*

[32] Wang, H. and Marron, J. S. (2007) Object oriented data analysis: Sets of trees, *The Annals of Statistics*, 35, 1849-1873.

[33] Wang, H. (2008) http://www.stat.colostate.edu/∼wanghn/tree.htm

[34] Marron, S. (2007) http://www.stat-or.unc.edu/webspace/courses/marron/UNCstor891OODA-2007/Stor891-07Home.html