Interactive visualization of information hierarchies and applications on the web

Confesor Santiago III
Rowan University

Let us know how access to this document benefits you - share your thoughts on our feedback form.

Follow this and additional works at: https://rdw.rowan.edu/etd
Part of the Electrical and Computer Engineering Commons

Recommended Citation
https://rdw.rowan.edu/etd/840

This Thesis is brought to you for free and open access by Rowan Digital Works. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Rowan Digital Works. For more information, please contact LibraryTheses@rowan.edu.
Interactive Visualization of Information Hierarchies and Applications on the Web

by

Confesor Santiago III

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Electrical and Computer Engineering
Major: Engineering (Electrical Engineering)

Approved: Members of the Committee

In Charge of Major Work

For the Major Department

For the College

Rowan University
Glassboro, New Jersey
2007
ABSTRACT

Confesor Santiago III
INTERACTIVE VISUALIZATION OF INFORMATION HIERARCHIES AND APPLICATIONS ON THE WEB
2007/05
Advisor: Dr. Adrian Rusu
Master of Science in Engineering

The visualization of information hierarchies is concerned with the presentation of abstract hierarchical information about relationships between various entities. It has many applications in diverse domains such as software engineering, information systems, biology, and chemistry. Information hierarchies are typically modeled by an abstract tree, where vertices are entities and edges represent relationships between entities. The aim of visualizing tree drawings is to automatically produce drawings of trees which clearly reflect the relationships of the information hierarchy.

This thesis is primarily concerned with problems related to the automatic generation of area-efficient grid drawings of trees, interactively visualizing information hierarchies, and applying our techniques on Web data.

The main achievements of this thesis include:

1. An experimental study on algorithms that produce planar straight-line grid drawings of binary trees,

2. An experimental study that shows the algorithm for producing planar straight-line grid drawings of degree-\(d\) trees with \(n\) nodes with optimal linear area and with user-defined arbitrary
aspect ratio, works well in practice,

3. A rings-based technique for interactively visualizing information hierarchies, in real-time,

4. A survey of Web visualization systems developed to address the "lost in cyberspace" problem,

5. A separation-based Web visualization system that we present as a viable solution to the "lost in cyberspace" problem,

6. A rings-based Web visualization system that we propose as a solution to the "lost in cyberspace" problem.
To my paternal grandparents, Confesor Sr. and Margarita,
for their care, love, and prayers, and in the everlasting memory
of my maternal grandparents, Warren and Kathryn
ACKNOWLEDGMENTS

First and foremost, I would like to thank God for this opportunity. May it be known that my works and achievements were only possible through You, and let all glory and honor be Yours in the name of Jesus Christ.

I would like to express my deep appreciation to my major advisor, Dr. Adrian Rusu, for his clear guidance, encouragement, wise counsel, and unwavering support over the past few years. His influence has been invaluable in my development as a researcher, and as a person. It is his continuous help, motivation, and insightful advice that made this thesis possible. I also want to show my respects to him for his incredible enthusiasm for research. Many thanks for supporting me as a Research Assistant. I will always be grateful for the opportunities he created for me. His hard work and dedication to me, as well as other students will never be forgotten.

Many thanks to Dr. Shreekanth Mandayam and Dr. Ganesh Baliga for being in my thesis committee. Their time and comments on my research work is greatly appreciated.

I would like to thank members of Software Engineering, Visualization, and Graphics (SEGV) Research Group including Mr. Chu Yao, Mr. Christopher Clement, and Mr. Christopher Reyes, for their help and effort. Also, I would like to thank Mr. Radu Jianu for his contributions to my research.

I would like to thank the Department of Computer Science and the Center for Innovation and Entrepreneurship for funding my Graduate Assistantship. I would like to thank Dr. Jay Kuder and Dr. Thomas Bryant, their efforts made our patent possible. Also, I would like to thank Dr.
Anthony Marchese providing funds to assist with the patent through the Rowan Undergraduate Venture Capital Fund.

Special thanks to my fiance, Leigh-Anna, for her support through every aspect of everyday life, unconditional love, and companionship, to my parents, Confesor Jr. and Barbara, for providing everything I needed and always being there for me, to my aunt, Margarita, for her prayers and cooking that cheers me up, to my grandfather, Confesor Sr., for being such a good role model, to my grandmother, Margarita, for her love and prayers, to my sister, Antonia, for being a good little sister and always being a friend growing up, to my future in-laws, for their support and treating my like family, and to my close friends, Mark, Andrew, and Ragon for being like brothers to me.
# TABLE OF CONTENTS

LIST OF FIGURES ................................................................................................. x
LIST OF TABLES ........................................................................................................ xv

1. INTRODUCTION ........................................................................................................ 1

1.1 Information Hierarchies ...................................................................................... 1

1.2 Tree Drawing ........................................................................................................ 1

1.3 Tree Drawing Conventions .................................................................................. 2

1.3.1 Grid Drawings ................................................................................................ 2

1.3.2 Planar Drawings .............................................................................................. 3

1.3.3 Straight-line Drawings .................................................................................... 4

1.3.4 Subtree Separation ......................................................................................... 4

1.4 Lost in Cyberspace .............................................................................................. 5

1.5 Contributions and Outline of This Thesis .......................................................... 5

2. GRID DRAWINGS OF BINARY TREES: AN EXPERIMENTAL STUDY ................. 7

2.1 Introduction ........................................................................................................ 7

2.2 The Drawing Algorithms Under Evaluation ....................................................... 10

2.3 Experimental Setting .......................................................................................... 15

2.3.1 Input File Format ........................................................................................... 15

2.3.2 Load/Save Routines ....................................................................................... 16

2.3.3 Test Suite ....................................................................................................... 18
6.7.3 Subjective Ratings ................................................................. 136
6.8 New Features ........................................................................ 137
  6.8.1 Variable Depth ................................................................. 138
  6.8.2 Color Coding ................................................................. 138
  6.8.3 Empty Pages ................................................................. 141
  6.8.4 Multi-level Navigation .................................................... 143
  6.8.5 Visualization Favorites .................................................. 143
7. CONCLUSION AND FUTURE WORK ............................................. 145
REFERENCES ............................................................................... 147
LIST OF FIGURES

1.1 Grid drawings of the same tree: (a) straight-line; (b) polyline; (c) non-planar. The root of the tree is shown as a shaded circle, whereas other nodes are shown as black circles. 3

2.1 Drawings of the Fibonacci tree with 88 nodes, generated by the algorithms in our study: (a) Separation, (b) Path, (c) Level, and (d) Rings. ........................................... 11

2.2 Drawings of an unbalanced-to-the-left binary tree with 100 nodes generated by the algorithms in our study: (a) Separation, (b) Path, (c) Level, and (d) Rings. For Rings, the tree is of size 25. ............................................................................. 12

2.3 Drawings of an unbalanced-to-the-right binary tree with 100 nodes generated by the algorithms in our study: (a) Separation, (b) Path, (c) Level, and (d) Rings. For Rings, the tree is of size 25. ................................................................. 13

2.4 Comparison charts of the area for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ............................................. 46

2.5 Comparison charts of the aspect ratio for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ......................................................................................... 48

2.6 Comparison charts of the size for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ......................................................................................... 50

2.7 Comparison charts of the total edge length for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ............................................. 52
2.8 Comparison charts of the maximum edge length for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ...................... 54

2.9 Comparison charts of the uniform edge length for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ...................... 56

2.10 Comparison charts of the minimum angle size for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary trees, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ...................... 58

2.11 Comparison charts of the average angle size for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary trees, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ...................... 60

2.12 Comparison charts of the angular resolution for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary trees, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ...................... 62

2.13 Comparison charts of the closest leaf for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary trees, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ...................... 64

2.14 Comparison charts of the farthest leaf for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary trees, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees. ...................... 66

3.1 Rotating a drawing Γ by 90°, followed by flipping it vertically. Note that initially node u* was located at the bottom boundary of Γ, but after the rotate operation, u* is on the right boundary of Γ. ................................................................. 70

3.2 Drawing of a randomly-generated general tree with 60 nodes constructed by Algorithm DrawGeneralTree, with A = 1 and ε = 0.2. ................................................................. 70
3.3 Drawing $T$ in all the seven subcases of Case 1 (when the separator node $u$ is not in the leftmost path of $T$): (a) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $g \neq u^*$, $0 \leq i \leq d-3$, (b) $T_d = \emptyset$, $T_C = \emptyset$, $0 \leq i \leq d-3$, (c) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $g = u^*$, $0 \leq i \leq d-3$, (d) $T_d \neq \emptyset$, $T_C = \emptyset$, $r \neq e$, $0 \leq i \leq d-3$, (e) $T_d \neq \emptyset$, $T_C = \emptyset$, $r = e$, $0 \leq i \leq d-3$, (f) $T_d = \emptyset$, $T_C \neq \emptyset$, $g \neq u^*$, $0 \leq i \leq d-3$, and (g) $T_d = \emptyset$, $T_C \neq \emptyset$, $g = u^*$, $0 \leq i \leq d-3$. For each subcase, we first show the structure of $T$ for that subcase, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. Here, $x$ is the same as $f$ if $T_B \neq \emptyset$, and is the same as the root of $T$ if $T_B = \emptyset$. In Subcases (a) and (c), for simplicity, $e$ is shown to be in the interior of $\Gamma_A$, but actually, either it is the same as $r$, or if $A < 1 (A \geq 1)$, then it is placed on the bottom (right) boundary of $\Gamma_A$. For simplicity, we have shown $\Gamma_A$, $\Gamma_B$, and $\Gamma_C$ as identically sized boxes, but in actuality, they may have different sizes.

3.4 Drawing $T$ in all the eight subcases of Case 2 (when the separator node $u$ is in the leftmost path of $T$): (a) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $v \neq u^*$, $1 \leq j \leq d-2$, (b) $T_A = \emptyset$, $T_C \neq \emptyset$, $v \neq u^*$, $j = 0$, (c) $T_A = \emptyset$, $T_C \neq \emptyset$, $v \neq u^*$, $1 \leq j \leq d-2$, (d) $T_A \neq \emptyset$, $T_C \neq \emptyset$, $v \neq u^*$, $j = 0$, (e) $T_A \neq \emptyset$, $T_B \neq \emptyset$, $v = u^*$, $1 \leq j \leq d-2$, (f) $T_A = \emptyset$, $T_B = \emptyset$, $v = u^*$, $j = 0$, (g) $T_A = \emptyset$, $T_B \neq \emptyset$, $v = u^*$, $1 \leq j \leq d-2$, (h) $T_A \neq \emptyset$, $T_B = \emptyset$, $v = u^*$, $j = 0$, (k) $T_A = \emptyset$, $T_B \neq \emptyset$, $T_C \neq \emptyset$, $r = u = u^*$, $j > 0$, (l) $T_A \neq \emptyset$, $T_B \neq \emptyset$, $T_C \neq \emptyset$, $r \neq e$, $u = u^*$, $j > 0$, and (m) $T_A \neq \emptyset$, $T_B \neq \emptyset$, $T_C \neq \emptyset$, $r = e$, $u = u^*$, $j > 0$. For each subcase, we first show the structure of $T$ for that subcase, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. In Subcases (a), (d), (e), and (h) for simplicity, $e$ is shown to be in the interior of $\Gamma_A$, but actually, either it is the same as $r$, or if $A < 1 (A \geq 1)$, then it is placed on the bottom (right) boundary of $\Gamma_A$. For simplicity, we have shown $\Gamma_A$, $\Gamma_B$, and $\Gamma_C$ as identically sized boxes, but in actuality, they may have different sizes.

3.5 Drawing $T_A$. Here, we first show the structure of $T_A$ in (a), then its drawing when $A < 1$ in (b), and then its drawing when $A \geq 1$ in (c).

3.6 Drawing $T_B$ when: (a) $T_B \neq \emptyset$, and (b) $T_B = \emptyset$. For each case, we first show the structure of $T_B$ for that case, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. In Case (a), for simplicity, $w$ is shown to be in the interior of $\Gamma_B$, but actually, it is either the same as $f$, or if $A < 1 (A \geq 1)$, then it is placed on the bottom (right) boundary of $\Gamma_B$.

3.7 Performance of the algorithm, as given by the value of $c$, for drawing a randomly-generated general tree $T$ with different values of $A$ and $\varepsilon$, where $c = \text{area of drawing}/\text{number of nodes in } T$: (a) $\varepsilon = 0.9$, (c) $\varepsilon = 0.75$, (e) $\varepsilon = 0.5$, (g) $\varepsilon = 0.25$, and (i) $\varepsilon = 0.1$. Figures (b), (d), (f), and (h) contain the projections on the $XZ$-plane of the plots shown in Figures (a), (c), (e), and (g), respectively, and show for each $\varepsilon$, the ranges of the values of $c$ for different values of $A$ for each $n$. The $XZ$-plane for $\varepsilon = 0.1$ is not shown because for nearly every value of $n, A$ is constant.

4.1 Screen shot of entire separation-based visualization system.

4.2 Example of separation-based visualization system label placements.

4.3 Zoomed in capture of separation-based visualization system.
6.4 Web crawler state diagram. Here, $n$ represents the response time and $k$ represents the number of disconnected round trips allowed. ................................................................. 126

6.5 Subjective ratings for each evaluation question. ................................................................. 136

6.6 FastRings with max depth of three and 2457 nodes total. ........................................... 139

6.7 FastRings with max depth of four and 2832 nodes total. ........................................... 140

6.8 Color coding scheme for two categories: text-based links (dark-blue) and image-based links (orange). ................................................................. 141

6.9 Our current strategy of locating empty pages in one dummy node. Here, the dummy node is in the bottom-right part of the visualization. ................................................................. 142

6.10 An example of multi-level navigation: (a) Initial visualization: a grandchild is being selected, (b) Intermediate visualization: the child of the root (i.e. the parent of the selected grandchild) of (a) is the focus, (c) Final visualization: the grandchild of (a) is the final focus. ................................................................. 144
LIST OF TABLES

6.1 Steps of first training assignment .................................................. 132
6.2 Steps of second training assignment ................................................. 132
6.3 Time results and scenario completion rate for all subjects using both *WebGPS* and a regular Web browser on all four scenarios. .................................................. 135
1. INTRODUCTION

1.1 Information Hierarchies

An information hierarchy is a collection of relational information that is arranged in a ranking organization where each entity is subject to a single other entity, except for the top (root) element. Information hierarchies are commonly referred to as tree structures, because the graph is a tree. In a tree every item can be traced to a single origin through a unique path. Information hierarchies are easier to understand than graph structures.

1.2 Tree Drawing

Tree drawing is concerned with the automatic generation of geometric representations of relational information, often for visualization purposes. The typical data structure for modeling hierarchical information is a tree whose vertices represent entities and whose edges correspond to relationships between entities. Visualizations of hierarchical structures are only useful to the degree that the associated diagrams effectively convey information to the people that use them. A good diagram helps the reader understand the system, but a poor diagram can be confusing [22]. The automatic generation of drawings of trees finds many applications, such as

- software engineering (program nesting trees, object-oriented class hierarchies),
- information systems (organization charts),
- decision support systems (activity trees),
Further applications can be found in other science and engineering disciplines, such as

- biology (evolutionary trees),
- chemistry (molecular drawings),

The usefulness of a drawing of a tree depends on its \textit{readability}, i.e. its capability of conveying the information contained in the tree quickly and clearly.

Tree drawing algorithms are methods that produce tree drawings which are easy to read. Algorithms for drawing trees are typically based on some graph-theoretic insight into the structure of the tree. The input to a tree drawing algorithm is a tree $T$ that needs to be drawn. The output is a drawing $\Gamma$ which maps each vertex of $T$ to a distinct point in the 2D space and each edge $(u,v)$ of $T$ to a simple Jordan curve with endpoints $u$ and $v$.

1.3 Tree Drawing Conventions

In this thesis we consider planar straight-line grid drawings. Now we explain the properties of these drawings and the motivation behind using them.

1.3.1 Grid Drawings

A \textit{grid drawing} is one in which each vertex is placed at integer coordinates (see Figure 1.1(a)). Grid drawings guarantee at least unit distance separation between nodes, and the integer coordinates of nodes allow such drawings to be rendered on displays, such as computer screen, without any distortions due to truncation and round-off errors. We assume that the plane is covered by horizontal
Figure 1.1: Grid drawings of the same tree: (a) straight-line; (b) polyline; (c) non-planar. The root of the tree is shown as a shaded circle, whereas other nodes are shown as black circles.

and vertical channels, with unit distance between two consecutive channels. The meeting point of a horizontal and a vertical channel is called a grid-point. The smallest rectangle with horizontal and vertical sides parallel to the $X$ and $Y$ axis, that covers the entire grid drawing, is called the enclosing rectangle. The area of a grid drawing is defined as the number of grid points contained in its enclosing rectangle. Drawings with small area can be drawn with greater resolution on a fixed-size page. The aspect ratio of a grid drawing is defined as the ratio of the length of the longest side to the length of the shortest side of its enclosing rectangle. Giving the users control over the aspect ratio of a drawing allows them to display the drawing in different kinds of displays surfaces with different aspect ratios.

The optimal use of the screen space is achieved by minimizing the area of the drawing and by providing user-controlled aspect ratio.

1.3.2 Planar Drawings

A planar drawing is a drawing in which no two edges cross (see Figure 1.1(a) and (b)). Planar drawings are normally easier to understand than non-planar drawings (see Figure 1.1(c)), i.e. drawings with edge-crossings. Planarity is also an important tree theoretic concept, which has been
widely studied. Extensive research has been done on various kinds of planar drawings. For example, [11, 13, 17, 18, 29–31, 50, 61, 70, 75, 77] provide important results.

1.3.3 Straight-line Drawings

It is natural to draw each edge of a tree as a straight line between its end-vertices. The so called straight-line tree drawings have each edge drawn as a straight line segment (see Figure 1.1(a)). Straight-line drawings are easier to understand than polyline drawings (see Figure 1.1(b) and (c)), i.e. drawings in which edges have bends (more than one line segment). The experimental study of the human perception of tree drawings has concluded that minimizing the number of edge crossings and minimizing the number of bends increases the understandability of drawings of trees [57, 59, 73]. Ideally, the drawings should have no edge crossings, i.e. they should be planar drawings, and should have no edge-bends, i.e. they should be straight-line drawings.

1.3.4 Subtree Separation

A drawing of a tree $T$ has the subtree separation property [11] if, for any two node-disjoint subtrees of $T$, the enclosing rectangles of the drawings of the two subtrees do not overlap with each other. Drawings with the subtree separation property are more aesthetically pleasing than those without the subtree separation property. The subtree separation property also allows for a focus+context style [69] rendering of the drawing, so that if the tree has too many nodes to fit in the given drawing area, then the subtrees closer to focus can be shown in detail, whereas those further away from the focus can be contracted and simply shown as filled-in rectangles.
1.4 Lost in Cyberspace

The World Wide Web (WWW) today has become an enormous source of information and more and more users have access to a steadily increasing number of Web pages, generally linked in a non-intuitive manner. Consequently, repeatedly reported problems in WWW navigation are not knowing where you are, not knowing how to get back to previously visited information, and not knowing which sites have already been visited [27, 79]. The problem of users' disorientation in the WWW, which emerges from the high complexity of the WWW environment is often referred to as the "lost in cyberspace" problem. A regular Web browser's back and forward functionality is a not a sufficient solution to this problem. A map (visualization) reduces the user's cognitive load because it abates the load on human long term and working memory, summarizing the information about the structure and organization that would otherwise have to be remembered [1, 12, 21, 38, 82]. In this thesis we present viable solutions to the "lost in cyberspace" problem.

1.5 Contributions and Outline of This Thesis

In this thesis, we study the visualization of information hierarchies by experimenting with different tree drawing algorithms and constructing area-efficient planar straight-line grid drawings of trees. Also, we develop techniques to interactively explore information hierarchies, and apply our techniques on Web data. We now outline the structure of this thesis and summarize the principal results obtained: (Note that each chapter is self-contained)

- In Chapter 1 (this Chapter), we give an overview of tree drawing and the "lost in cyberspace" problem, providing the motivation for the results presented in the remainder of this thesis.

- In Chapter 2, we show how several binary tree drawing algorithms perform based on different tree classifications and quality measures.
• In Chapter 3, we show that the planar straight-line grid drawing with optimal linear area and user-defined arbitrary aspect ratio of degree-\(d\) trees with \(n\) nodes, where \(d = O(n^{\delta})\) and \(0 \leq \delta < 1/2\) is a constant, works well in practice.

• In Chapter 4, we present an interactive visualization system, which uses the drawing algorithm of Chapter 3 as the engine, an algorithm that constructs drawings of trees in real-time, and a system that produces a real-time interactive visualization of information hierarchies.

• In Chapter 5, we present a survey of other visualization systems developed to solve the "lost in cyberspace" problem.

• Chapter 6, we present two systems that are based on the concepts covered in Chapter 4 as solutions to the "lost in cyberspace" problem.

• In Chapter 7, we summarize the main achievements of this thesis and identify future work.
2. GRID DRAWINGS OF BINARY TREES: AN EXPERIMENTAL STUDY

2.1 Introduction

A lot of research has been done on visualizing trees, which has produced a plethora of tree drawing algorithms (See for example, [11, 13, 17, 18, 29–32, 50, 61, 64, 70, 74, 75, 77]). The majority of these algorithms have been developed with the primary target of minimizing the area of the drawing, so, in addition to their practical evaluation on area, it is of interest to evaluate how these algorithms perform on other important aesthetics.

Several experimental studies for drawing graphs are available (See for example, [5–7, 10, 37, 43, 44, 78]). However, we are not aware of any experimental study done to evaluate the practical performance of tree drawing algorithms. Given the importance of trees, and the large amount of research that has been done on developing techniques to visualize them, we believe that this is a big omission. As a first step, in this chapter, we present an experimental study of some well-known algorithms for drawing binary trees. A binary tree is one where each node has at most two children. These algorithms represent some of the distinct approaches that have been used to draw binary trees without distorting or occluding the information.

The issue of resolution of a drawing has been extensively studied, motivated by the finite resolution of physical rendering devices. The resolution of a drawing is defined as the minimum distance between two vertices. The quality measures Area, Aspect Ratio, Size, Total Edge Length, Average
Edge Length, Maximum Edge Length, Uniform Edge Length, Minimum Angle Size, Average Angle Size, Angular Resolution, Closest Leaf, and Farthest Leaf of a drawing depend on its resolution, hence two drawings can be compared for these measures only if they have the same resolution.

All algorithms in our experimental study produce planar straight-line grid drawings and exhibit the subtree separation property.

This chapter is comprised of an experimental study, which originally appeared in an abbreviated form in [66]. The work of this chapter was performed in collaboration with Radu Jianu and Christopher Clement. The contributions of this chapter can be summarized as follows:

- We have developed a general experimental setting for comparing the practical performance of drawing algorithms for binary trees. Our setting consists of (i) a new, simpler format for storing binary trees in files; (ii) save/load routines for generating binary trees to files and for uploading binary trees from files, respectively; (iii) a large suite of randomly-generated, unbalanced, complete, AVL, Fibonacci, and molecular combinatory binary trees of various sizes; (iv) twelve quality measures: area, aspect ratio, size, total edge length, average edge length, maximum edge length, uniform edge length, minimum angle size, average angle size, angular resolution, closest leaf, and farthest leaf.

- Within our experimental setting, we have performed a comparative study of four representative algorithms for planar straight-line grid drawing algorithms for binary trees, one for each of the following distinct approaches: separation-based approach [32], path-based approach [11], level-based approach [61], and ringed circular layout approach [74].

- Our comparison highlights how more than twenty years of research in this field have produced increasingly better algorithms. Our investigations include some interesting findings:
A contradiction to the popular belief [45] that, in practice, the algorithm of [61] should be generally accepted as the method of choice for drawing binary trees. Even though this algorithm achieves some important aesthetics, it scores worse in comparison to the other chosen algorithms for almost all twelve aesthetics considered in our study.

The performance of a drawing algorithm on a tree-type is not a good predictor of the performance of the same algorithm on other tree-types: some of the algorithms perform best on one tree-type, and worst on other tree-types.

Not all algorithms studied perform best on complete binary trees.

For three of the seven types of trees considered, the algorithm with the best theoretical worst-case bound produces worse area in practice than algorithms with worse theoretical worst-case bounds, or algorithms for which no theoretical bounds are available.

Of the four algorithms studied, three perform best on different types of trees, in regards to area.

Level-based algorithms produce much worse aspect ratios than algorithms designed using other approaches.

Of the four algorithms studied, three perform well on trees of different types and sizes, in regards to aspect ratio.

Path-based algorithms tend to construct drawings with better area at the expense of worse aspect ratio.

The intuition that low average edge length and area go together is contradicted in only one case.

The intuitions that average edge length and maximum edge length, uniform edge length
and total edge length, and short maximum edge length and close farthest leaf go together are contradicted for unbalanced binary trees.

The rest of the chapter is organized as follows. The four algorithms being compared are described in Section 2.2. Details on the experimental setting are given in Section 2.3. In Section 2.4, we summarize our experimental results in 84 charts, and perform a comparative analysis on each chosen aesthetic of the performance of the four algorithms. In Section 2.5, we present the charts of our experimental results.

2.2 The Drawing Algorithms Under Evaluation

We have tested four different algorithms for producing planar straight-line grid drawings of binary trees. The four algorithms can be classified into four categories on the basis of their approach to constructing drawings. Figures 2.1, 2.2, and 2.3 show drawings of Fibonacci, unbalanced-to-the-left, and unbalanced-to-the-right trees, constructed by the algorithms used in our study.

- **Separation-Based**: In the *Separation-Based Approach*, a divide-and-conquer strategy is used to recursively construct a drawing of the tree, by performing the following actions at each recursive step:
  
  - *Find a Separator Edge or a Separator Node*: A separator edge (node) of a tree $T$ with $\text{degree}(T) = d$ is an edge (node), which, if removed, divides $T$ into at most $d$ smaller, partial trees. Every tree contains such an edge or a node [31, 77], such that each partial tree contains $xn$ nodes (separator edge), where $1/3 \leq x \leq 2/3$, or contains no more than $n/2$ nodes (separator node). In the first step, these algorithms find a separator edge or a separator node.
Figure 2.1: Drawings of the Fibonacci tree with 88 nodes, generated by the algorithms in our study: (a) Separation, (b) Path, (c) Level, and (d) Rings.

- Divide Tree: Divide the tree into several partial trees by removing at most two nodes and their incident edges from it (including the separator edge or the separator node).

- Assign Aspect Ratios: Pre-assign a desirable aspect ratio to each partial tree.

- Draw Partial Trees: Recursively construct a drawing of each partial tree using its pre-assigned aspect ratio.
Figure 2.2: Drawings of an unbalanced-to-the-left binary tree with 100 nodes generated by the algorithms in our study: (a) Separation, (b) Path, (c) Level, and (d) Rings. For Rings, the tree is of size 25.

- Compose Drawings: Arrange the drawings of the partial trees, and draw the nodes and edges, that were removed from the tree to divide it, such that the drawing of the tree thus obtained is a planar straight-line grid drawing.
Several separation-based algorithms have been designed [29, 31, 32]. Even though both the algorithms of [29] and [32] achieve the worst-case theoretical bound of $O(n)$ area, the algorithm of [29], being a top-down algorithm, always constructs the worst-case drawing. We have therefore chosen to evaluate the $O(n)$-area bottom-up algorithm of [32] (we called it *Separation*). For our study, we have used the same implementation as the one used in [32].
• **Path-Based**: The *Path-Based Approach* uses a recursive winding paradigm as follows: first lay down a small chain of nodes from left to right until near a distinguished node \( v \), and then recursively lay out the subtrees rooted at the children of \( v \) in the opposite direction.

Several path-based algorithms have been designed [11, 30, 70].

For our study, we have implemented the \( O(n \log \log n) \)-area algorithm described in [11] (we call it *Path*).

• **Level-Based**: The *Level-Based Approach* is characterized by the fact that in the drawings produced, the nodes at the same distance from the root are horizontally aligned. For our study, we have implemented the recursive algorithm described in [61] (we call it *Level*). This algorithm uses the following steps: draw the subtree rooted at the left child, draw the subtree rooted at the right child, place the drawings of the subtrees at horizontal distance 2, and place the root one level above and halfway between the children. If there is only one child, place the root at horizontal distance 1 from the child.

• **Ringed Circular Layout**: The algorithms based on the *Ringed Circular Layout Approach* place a node and all its children in a circle [13, 50, 64, 74]. For our study, we have implemented the algorithm described in [74] (we call it *Rings*). Note that this algorithm was designed for general trees. In this study, we have implemented and studied its performance for the particular case of binary trees. In this algorithm, equal-sized circles corresponding to children are placed in concentric rings inside of the parent circle, around its center, thus trying to minimize the space wasted inside of the interior of the parent circle. The center of the circle is used as the position to place each node in the grid drawing.
2.3 Experimental Setting

Our experimental setting consists of (i) a new, simpler format for storing binary trees in files; (ii) save/load routines for generating binary trees to files and for uploading binary trees from files, respectively; (iii) a large suite of randomly-generated, unbalanced, complete, AVL, Fibonacci, and molecular combinatorial binary trees of various sizes; (iv) twelve quality measures: area, aspect ratio, size, total edge length, average edge length, maximum edge length, uniform edge length, minimum angle size, maximum angle size, average angle size, closest leaf, and farthest leaf.

2.3.1 Input File Format

Since trees have a simpler structure than graphs, we introduce a new, simpler format for storing binary trees in files. Each line in the input file represents a node, its left, and its right children, in this order, separated from each other by one space: node leftChild rightChild, where node is the key that uniquely identifies the node in the tree, leftChild is the key for the left child of node, or # if node has no left child, and rightChild is the key for the right child of node, or # if node has no right child. The following restriction applies to all the nodes, except the root of the tree: a node must occur as a child for another node before being itself defined.

For example, assume we have a binary tree defined using a preorder traversal as follows: 0, 1, 3, 4, 2, 5. This tree may be represented in its corresponding file in any of the following two ways:

```
0 1 2
1 3 4
2 5 #
3 # #
4 # #
```

or

```
0 1 2
1 3 4
2 5 #
3 # #
4 # #
```

Since the node with key 3 has not occurred as a child of any node before it was defined as having no children of its own, the following is not a proper representation of the tree:

```
0 1 2
2 5 #
3 ##
4 ##
1 3 4
5 ##
```

2.3.2 Load/Save Routines

We have developed simple load and save routines for transferring binary trees between the computer memory and files, using the format described in Subsection 2.3.1.

We give below pseudocode for Routines `LoadTree` and `SaveTree`.

**Algorithm LoadTree**

**Input:** An input file $F$ containing a binary tree $T$ stored in the format described in Subsection 2.3.1.

**Output:** The binary tree $T$ stored in the computer's memory.

for each line $L_i$ of $F$ {

  Let $T_i$ be the partial tree created before $L_i$ is considered;

  Parse $L_i$ into three fields: $K_1$, $K_2$, and $K_3$;

  if root($T_i$) = NULL (i.e., $L_i$ is the first line in $F$) then {

16
Let root($T_i$) have key $K_1$;
Let leftChild($K_1$) have key $K_2$;
Let rightChild($K_1$) have key $K_3$;
}
else {
    Search for key $K_1$ in $T_i$;
    if $K_2 \neq \#$ then {
        Create a left child for the node with key $K_1$;
        Assign $K_2$ as key to the left child of the node with key $K_1$;
    }
    if $K_3 \neq \#$ then {
        Create a right child for the node with key $K_1$;
        Assign $K_3$ as key to the right child of the node with key $K_1$;
    }
}
end Algorithm.

Algorithm SaveTree

*Input*: A binary tree $T$ stored in the computer's memory.

*Output*: A file $F$ containing the tree $T$ in the format described in Subsection 2.3.1.

if $T$ is the empty tree then Return;
else {
    Let \( K \) be the key corresponding to the root \( R \) of \( T \);
    Let \( L \) be a line of characters, initially empty;
    Let \( L = K \);
    if \( \text{leftChild}(R) \neq \text{NULL} \) then \( L = L \) concatenated with the key of the \( \text{leftChild}(R) \);
    else \( L = L \) concatenated with \#;
    if \( \text{rightChild}(R) \neq \text{NULL} \) then \( L = L \) concatenated with the key of the \( \text{rightChild}(R) \);
    else \( L = L \) concatenated with \#;
    Write \( L \) into \( F \);
    Let \( T_L \) be the left subtree of \( T \);
    Let \( T_R \) be the right subtree of \( T \);
    \( \text{SaveTree}(T_L) \);
    \( \text{SaveTree}(T_R) \);
}

end Algorithm.

2.3.3 Test Suite

We have generated a large test suite consisting of binary trees of various types and sizes. We then performed our experimental study on this test suite.

Our test suite consists of five binary trees for each of the following types and sizes:

- *Randomly-generated binary trees:*

  Each randomly-generated binary tree \( T_n \) with \( n \) nodes was generated by generating a sequence
To, T₁, ..., Tₙ of binary trees, where T₀ is the empty tree, and Tᵢ was generated from Tᵢ₋₁ by inserting a new leaf vᵢ into it. The position where vᵢ is inserted in Tᵢ₋₁ is determined by traversing a path p = u₀u₁...uₘ of Tᵢ₋₁, where u₀ is the root of Tᵢ₋₁, and uₘ has at most one child. More precisely, we start at the root u₀, and in the general step, assuming that we have already traversed the sub-path u₀u₁...uⱼ₋₁, we flip a coin. If “head” comes up, then if uⱼ₋₁ has a left child c, then we set uⱼ = c, and move to uⱼ, otherwise we make vᵢ the left child of uⱼ₋₁, and stop. If “tail” comes up, then if uⱼ₋₁ has a right child c, then we set uⱼ = c, and move to uⱼ, otherwise we make vᵢ the right child of uⱼ₋₁, and stop.

- **Unbalanced binary trees:**

  We consider a binary tree Tₙ with n nodes as unbalanced if its height is greater than n/log n. A binary tree Tₙ with n nodes is unbalanced-to-the-left (unbalanced-to-the-right) if it is unbalanced, and, in addition, the number of left (right) children in Tₙ is greater than its number of right (left) children.

  Each unbalanced-to-the-left (unbalanced-to-the-right) binary tree Tₙ with n nodes was generated in a similar way to the randomly-generated binary trees. The only difference occurs at the time of coin flipping: the probability of the coin coming “head” is set to be higher (lower) than the probability of the coin coming “tail”.

- **Complete binary trees:**

  A binary tree Tₙ with n nodes is complete if every non-leaf node of Tₙ has exactly two children.

- **AVL trees:**

  An AVL tree is a balanced binary tree where the height of the two subtrees of a node differs by at most one.
Each AVL tree was generated by using a generic method to maintain the tree's AVL property and by randomly inserting nodes in the tree.

- **Fibonacci trees:**

A *Fibonacci tree* $T_n$ is defined inductively as follows: $T_0$ is the empty tree, $T_1$ is the tree with one node, and $T_n$ has as left subtree $T_{n-1}$, and as right subtree $T_{n-2}$. Note that a Fibonacci tree is the most unbalanced AVL tree allowed.

- **Molecular combinatory binary trees:**

These binary trees have a strong connection to "real-life" applications. The data was obtained from the study in [47] by Dr. Bruce MacLennan at the University of Tennessee. Central to Dr. MacLennan's approach was the identification of a small set of molecular building blocks that are provably sufficient for controlling the nanoscale synthesis and behavior of materials. Within this research, Dr. MacLennan used combinatory logic [20], a mathematical formalism based on network substitution operations suggestive of supramolecular interactions. Binary trees derive from the networking conventions of combinatory logic and visualization of these binary trees could improve the investigator's ability in interpreting the substitution operations involved in combinatory logic.

We have generated random, unbalanced, and AVL binary trees with sizes 100, 200, ..., 1000, 2000, ..., 10000, 20000, ..., 50000, complete binary trees with sizes $2^7 - 1, 2^8 - 1, \ldots, 2^{16} - 1$, Fibonacci trees with every size between 143 and 46367 nodes, and molecular combinatory binary trees with sizes 133, 181, ..., 1813, 2005, ..., 9973, 11989, ..., 40021, 50005.

We ran the experiments and produced the statistics by grouping the binary trees by their
type (i.e., randomly-generated, unbalanced-to-the-left, unbalanced-to-the-right, complete, AVL, Fibonacci, and molecular combinatory), and by their number of nodes.

2.3.4 Quality Measures

The following eight well-known quality measures have been considered:

- **Area**: the number of grid points contained within the smallest rectangle with horizontal and vertical sides covering the drawing.

- **Aspect Ratio**: the ratio of the smaller and the longer sides of the smallest rectangle with horizontal and vertical sides covering the drawing.

- **Size**: the maximum between the height and width of the drawing.

- **Total Edge Length**: the sum of the lengths of the edges in the drawing.

- **Average Edge Length**: the average of the lengths of the edges in the drawing.

- **Maximum Edge Length**: the maximum among the lengths of the edges in the drawing.

- **Uniform Edge Length**: the variance of the edge lengths in the drawing.

- **Angular Resolution**: the minimum among the sizes of the angles between any two edges in the drawing.

It is widely accepted [4, 22, 58, 60] that small values of the area, size, total edge length, average edge length, maximum edge length, and uniform edge length are related to the perceived aesthetic appeal and visual effectiveness of the drawing. In addition, an aspect ratio is considered *optimal* if it is equal to 1.

We have also considered four new quality measures, specially designed for trees:
- **Minimum Angle Size**: the minimum among the sizes of the angles between the edges connecting each node to its children in the drawing.

- **Average Angle Size**: the average of the sizes of the angles between the edges connecting each node to its children in the drawing.

- **Closest Leaf**: the smallest Euclidean distance between the root of the tree and a leaf in the drawing.

- **Farthest Leaf**: the largest Euclidean distance between the root of the tree and a leaf in the drawing.

The *angular resolution* \( \rho \) of a straight-line drawing is the smallest angle formed by two edges incident on the same node. High angular resolution is desirable in visualization applications and in the design of optical communication networks. For binary trees, the degree of a node is at most 3, hence a trivial upper bound on the angular resolution is \( \rho \leq 120^\circ \). Therefore, for trees, it is more important to analyze the angle between the edges connecting a node to its children. The new aesthetics we propose: Minimum Angle Size and Average Angle Size help determine whether the drawing contains two children which are connected to their parent with edges drawn too close to each other, or, on average, if the nodes of the drawing are connected to their parents with edges too close to each other. As with Angular Resolution, higher values for these two aesthetics help in visually distinguishing the right child from the left child. Note that, for binary trees, these measures are calculated only for nodes with exactly two children.

The aesthetics Closest Leaf and Farthest Leaf help determine whether the algorithm places leaves close or far from the root. It is important to minimize the distance between the root and the leaves of the tree, especially in the case when the user wants to visually analyze binary search
2.4 Experimental Analysis

Let \( T_n \) be a binary tree with \( n \) nodes that is provided as input to the algorithms being evaluated.

Two of the algorithms chosen in this study, namely Separation and Path, allow user-controlled aspect ratio, i.e. the user may change the aspect ratio by providing some parameters as input to the algorithms. The other two algorithms, (Level and Rings), generate unique drawings for each value of \( n \). In order to find the parameters for which Separation and Path perform the best on each of the aesthetics considered in our study, we used the studies in [32] and [65], respectively.

2.4.1 Comparison Analysis

In order to compare the algorithms, we varied \( n \) up to 50,000 for randomly-generated, unbalanced-to-the-left, unbalanced-to-the-right, and AVL binary trees, up to 46,367 for Fibonacci trees, and up to 65,535 for complete binary trees. We compared the performance of the algorithms for each tree-type separately.

Since, for any tree, when the desirable aspect ratio is set to 1, we can always find the actual aspect ratio of the drawing produced by Separation close to 1 [32], we decided not to evaluate the performance of Separation for this quality measure.

In the case of unbalanced and molecular combinatory binary trees, Rings quickly becomes prohibitive to use. For example, for an unbalanced tree with 1,000 nodes, Rings produces a drawing with area of \( 2^{100} \). The area very rapidly increases to \( 2 \cdot 10^{226} \) for a tree with 10,000 nodes. Also, for molecular combinatory binary trees with 469 nodes, Rings produces a drawing with area of \( 2 \cdot 10^9 \). For this reason, we have decided to not consider Rings in our comparisons for unbalanced and molecular combinatory binary trees.
A binary tree with \( n \) nodes has \( n - 1 \) edges. Thus, the average edge length is always equal to the total edge length divided by \( n - 1 \). Therefore, we do not create separate charts for the quality measure Average Edge Length, we rather use the charts for the quality measure Total Edge Length to analyze the behavior of the algorithms for this aesthetic.

Since both Path and Rings produce orthogonal drawings (i.e. drawings in which each edge is drawn as a chain of alternating horizontal and vertical segments), the angles between the edges connecting a parent to its children will always be either 90° or 180°. For this reason, we do not consider Path and Rings in our analysis of quality measures Minimum Angle Size, Average Angle Size, and Angular Resolution.

Figures 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, and 2.14 (See Section 2.5) display the performance of Level, Path, Rings, and Separation, as given by the quality measures Area, Aspect Ratio, Size, Total Edge Length, Maximum Edge Length, Uniform Edge Length, Minimum Angle Size, Average Angle Size, Angular Resolution, Closest Leaf, and Farthest Leaf, respectively. The x-axis of each chart shows the number of nodes.

The analysis of the performance of the four algorithms for each quality measure, and for each tree-type is summarized below:

- **Area:** (See Figure 2.4)
  
  - Complete binary trees: (See Figure 2.4(a)) Order of performance: Rings, Separation, Path, Level. While the difference in the areas produced by Rings and Separation grows slowly, the difference in the areas produced by Separation and Path grows much faster. The same behavior is exhibited in Level and Path. For \( n = 65,535 \), Level produces a drawing having an area almost four times more than the drawing produced by Path.
- **AVL trees**: (See Figure 2.4(b)) Order of performance: Rings, Separation, Path, Level. Rings and Separation exhibit similar behavior, with Rings being slightly better. The differences in the areas produced grow slowly.

- **Randomly-generated binary trees**: (See Figure 2.4(c)) Order of performance: Separation, Path, Level, Rings. The performances of all the algorithms are worse than their respective performances on complete trees. In comparison to its behavior on complete trees, where it was the best, Rings exhibits the most dramatic change: its behavior is now the worst of all four algorithms. The area produced by Level grows rapidly in comparison to the area produced by Path, being already three times more for \( n = 50,000 \).

- **Fibonacci trees**: (See Figure 2.4(d)) Order of performance: Separation, Path, Level, Rings. Rings quickly becomes prohibitive, with area ten times more than the area of Separation, for 10,000 nodes. The difference between the areas produced by Separation and Path grows slowly, while the difference between the areas produced by Path and Level grows much faster.

- **Unbalanced-to-the-left binary trees**: (See Figure 2.4(e)) Order of performance: Path, Separation, Level, Rings. Level quickly becomes prohibitive, and it has been only partially plotted. Path slightly outperforms Separation.

- **Unbalanced-to-the-right binary trees**: (See Figure 2.4(f)) Order of performance: Path, Separation, Level, Rings. Path produces excellent results for this type of tree. This is the best case for Path and the worst case for Separation. Level rapidly becomes prohibitive, and it has been only partially plotted.

- **Molecular combinatorial binary trees**: (See Figure 2.4(g)) Order of performance: Path, Separation, Level, Rings. Even though Path is the best performing algorithm on both
unbalanced and molecular combinatory binary trees, its behavior on molecular combinatory binary trees is much better: for \( n = 50,000 \), the area of molecular combinatory binary trees is 78,360, as opposed to unbalanced-to-the-left binary trees, with an area of 258,355. Level rapidly becomes prohibitive to use, producing an area over 1,000,000 for \( n = 5,989 \). This is the best case for Path and the worst case for Separation.

• **Aspect Ratio:** (See Figure 2.5)

  – **Complete binary trees:** (See Figure 2.5(a)) Order of performance: Separation, Path, Rings, Level. Quite interestingly, the behavior of Path and Rings is very similar. Neither algorithm always produces drawings with aspect ratios close to optimal. For example, if \( n = 2^{14} \), the best aspect ratio Path produces is around 0.5. Rings produces optimal aspect ratios when \( n = 2^i \), with \( i \) an odd number, and aspect ratios close to 0.5, with \( i \) an even number. The aspect ratios of the drawings produced by Level are very low (the highest value is close to 0.06), decreasing rapidly as \( n \) increases.

  – **AVL trees:** (See Figure 2.5(b)) Order of performance: Separation, Rings, Path, Level. Rings exhibits a very interesting pattern: its aspect ratios are either 0.5 or optimal. The performances of Path and Level decrease dramatically, with Level quickly producing very small aspect ratios, and Path producing aspect ratios less than 0.01 for 50,000 nodes.

  – **Randomly-generated binary trees:** (See Figure 2.5(c)) Order of performance: Separation, Path, Rings, Level. Level produces drawings with better aspect ratios for trees with smaller number of nodes (the highest value is close to 0.1). Still, its behavior is unsatisfactory, as the value of aspect ratio decreases rapidly as \( n \) increases. Path and Rings
have uneven behaviors. Most of their aspect ratios are over 0.8, and none are under 0.5.

- **Fibonacci trees**: (See Figure 2.5(d)) Order of performance: *Separation, Rings, Path, Level*. Interestingly, *Rings* exhibits exactly the same behavior as in the case of complete binary trees: optimal aspect ratios when \( n = 2^i \), with \( i \) an odd number, and aspect ratios close to 0.5, with \( i \) an even number. The behavior of *Level* is only significant for trees with small number of nodes.

- **Unbalanced-to-the-left binary trees**: (See Figure 2.5(e)) Order of performance: *Separation, Path, Level, Rings*. Quite surprisingly, *Level* produces drawings with better aspect ratios than before and its performance decreases very slowly as \( n \) increases. Quite interestingly, the performance of *Path* is almost identical with the one for randomly-generated binary trees, with most of the aspect ratios over or close to 0.8, and no aspect ratio under 0.6.

- **Unbalanced-to-the-right binary trees**: (See Figure 2.5(f)) Order of performance: *Separation, Path, Level, Rings*. *Level* exhibits similar behavior as in the case of unbalanced-to-the-left binary trees. Interestingly, while for small values of \( n \) the performance of *Path* is close to optimal, it decreases rapidly as \( n \) increases. For example, for 50,000 nodes, *Level* produces better aspect ratios than *Path*.

- **Molecular combinatory binary trees**: (See Figure 2.5(g)) Order of performance: *Separation, Path, Level, Rings*. *Path* produces close to optimal values until \( n = 5,989 \). After this point, the values plummet, decreasing to 0.1 for \( n = 50,005 \). Very interestingly, *Level* always produces values close to 0.1. In our analysis, it was discovered that *Level* always produces drawings of width equal to \( n \). Hence, for molecular combinatory binary trees, the height is almost always one-tenth of the width.

27
- **Complete binary trees:** (See Figure 2.6(a)) Order of performance: Rings, Separation, Path, Level. Level grows at an exponential rate for all categories of binary trees and will not be mentioned. Path, Rings, and Separation grow fast and then reach an individual point where rate of growth is comparable.

- **AVL trees:** (See Figure 2.6(b)) Order of performance: Separation, Rings, Path, Level. Strangely, Rings and Separation go back and forth between best performance. Separation grows at a constant rate becoming 409 for 50,000 nodes, while Rings values are scattered about the performance of Separation. The performance of Path is at its worst, becoming 2.5 times larger than Separation at \( n = 50,000 \).

- **Randomly-generated binary trees:** (See Figure 2.6(c)) Order of performance: Separation, Path, Rings, Level. Path grows faster than Rings and Separation. Separation grows at a constant rate, resulting in a size of 298 for \( n = 20,000 \). Also, Path grows almost two times faster than Separation.

- **Fibonacci trees:** (See Figure 2.6(d)) Order of performance: Separation, Path, Rings, Level. Ironically, Path starts as the best, but as \( n \) increases, so too does its rate of growth. Ultimately, the faster rate of Path enables Separation to outperform all in the end. Rings performs at its worst compared to all other categories of binary trees.

- **Unbalanced-to-the-left binary trees:** (See Figure 2.6(e)) Order of performance: Separation, Path, Level, Rings. The performance of Rings and Path begin very similar, but as \( n \) gets larger Path becomes worse. For example, for \( n = 50,000 \), Separation is 649 and Path is 818. Level again exhibits unsatisfactory performance.
- Unbalanced-to-the-right binary trees: (See Figure 2.6(f)) Order of performance: Path, Separation, Level, Rings. The performance of Path on this tree type is very identical to that of unbalanced-to-the-left binary trees. Unlike Path, Separation is not similar to the performance on unbalanced-to-the-left binary trees resulting in size of 945.4 for \( n = 50,000 \), as opposed to 649 on unbalanced-to-the-left binary trees.

- Molecular combinatorial binary trees: (See Figure 2.6(g)) Order of performance: Separation, Path, Level, Rings. The performance of Path starts out as the best algorithm, but is overtaken by Separation at \( n = 11,989 \). The performance of Path is near linear, becoming two times larger than the results of Separation at \( n = 50,005 \).

- Total Edge Length and Average Edge Length: (See Figure 2.7)

- Complete binary trees: (See Figure 2.7(a)) Order of performance: Rings, Separation, Path, Level. The similarity between the plots for total edge length and area fits the intuitive notion that low total edge length and area generally go together. All of the observations made for Area apply in this case, except when \( n = 50,000 \), Path performs better than Separation, and is very close to the performance of Rings.

- AVL trees: (See Figure 2.7(b)) Order of performance: Rings, Separation, Path, Level. The results for Rings and Separation are very similar. Level performs poorly, producing results nearly four times greater than Rings. The intuition is confirmed in this case.

- Randomly-generated binary trees: (See Figure 2.7(c)) Order of performance: Separation, Rings, Path, Level. The interesting finding here is that Rings and Path exhibit almost identical behavior. Separation performs a little better than these two, and the gap between the performance of Level and the performance of the other algorithms grows
fast. It is also interesting to see that, with lower total edge length than Path and Level, Rings constructs larger area than the two, thus contradicting the intuition.

- **Fibonacci trees**: (See Figure 2.7(d)) Order of performance: Separation, Path, Rings, Level. The intuition is confirmed for the two best-performing algorithms (Separation and Path), but contradicted for the two worst performing algorithms (Rings and Level). The performances of Path and Rings are similar.

- **Unbalanced-to-the-left binary trees**: (See Figure 2.7(e)) Order of performance: Path, Separation, Level, Rings. A very interesting finding is discovered in this case. In this situation, the intuition that low total edge length and area go together is confirmed. For example, Path exhibits better behavior than Separation for total edge length, which complements the results from area. Also, Level and Separation exhibit almost identical behavior for total edge length, while Level produces an area exponentially higher than Separation.

- **Unbalanced-to-the-right binary trees**: (See Figure 2.7(f)) Order of performance: Path, Separation, Level, Rings. Again, the intuition is confirmed. Within their order of performance, the behavior of the algorithms individually increase at a constant rate with respect to the number of nodes.

- **Molecular combinatory binary trees**: (See Figure 2.7(g)) Order of performance: Path, Separation, Level, Rings. Again, the intuition is confirmed. Within their order of performance, the behaviors of Path and Separation individually increase at a constant rate with respect to the number of nodes, and the performance of Level grows exponentially.

- **Maximum Edge Length**: (See Figure 2.8)
- **Complete binary trees**: (See Figure 2.8(a)) Order of performance: Rings, Separation, Path, Level. Level's maximum edge length grows very quickly in comparison to the other algorithms. The difference between the maximum edge lengths produced by Rings and those produced by Separation grows slowly, and the difference between Separation and Path grows fast for small trees but narrows rapidly for large trees. For \( n = 65,535 \), maximum edge length for Rings is 128, for Separation is 255, and for Path is 320.

- **AVL trees**: (See Figure 2.8(b)) Order of performance: Rings, Separation, Path, Level. The behavior of Path behaves in a non-linear fashion. Also, Level exhibits behavior much worse than the others. Separation and Rings produce very good results and the difference between their performances seem to grow very slowly.

- **Randomly-generated binary trees**: (See Figure 2.8(c)) Order of performance: Separation, Path, Rings, Level. Again, Level produces unsatisfactory results, and quickly becomes prohibitive to use. The performance of Rings is better than the performance of Path for all \( n \), except \( n = 40,000 \) and 50,000. The behavior of Separation grows slowly, and is 240.6 for \( n = 50,000 \).

- **Fibonacci trees**: (See Figure 2.8(d)) Order of performance: Separation, Path, Rings, Level. Separation and Path have almost identical behavior and Rings grows much faster. Also, again, Level exhibits behavior much worse than the others.

- **Unbalanced-to-the-left binary trees**: (See Figure 2.8(e)) Order of performance: Level, Path, Separation, Rings. Quite surprisingly, Level produces almost constant maximum edge length. Moreover, it is very low. Path also produces steady results. Separation exhibits a much faster growing behavior.

- **Unbalanced-to-the-right binary trees**: (See Figure 2.8(f)) Order of performance: Level,
Path, Separation, Rings. Very surprisingly, it seems that both Path and Level produce similar, low, almost constant maximum edge lengths. Separation grows fast as it did for unbalanced-to-the-left binary trees.

- Molecular combinatorial binary trees: (See Figure 2.8(g)) Order of performance: Separation, Path, Level, Rings. To this point, the performances on molecular combinatorial binary trees and unbalanced trees have been similar. Very interestingly, Level produces the worst maximum edge length, but for unbalanced binary trees Level produces the best results. Initially, Path produces the best results, until \( n = 5,989 \), at which point Separation overtakes Path while its own rate of growth decreases.

- Uniform Edge Length: (See Figure 2.9)

  - Complete binary trees: (See Figure 2.9(a)) Order of performance: Rings, Separation, Path, Level. The performances agree with the intuition that the performance for total edge length and uniform edge length are similar. Rings and Separation produce very low, almost constant values. Path exhibits a very interesting non-linear behavior.

  - AVL trees: (See Figure 2.9(b)) Order of performance: Rings, Separation, Path, Level. The performances of Rings and Separation are very good: they almost mimic their performances on complete binary trees. The results for Path do not have a consistent rate of change.

  - Randomly-generated binary trees: (See Figure 2.9(c)) Order of performance: Rings, Separation, Path, Level. The performances of Rings and Separation are very similar to those on complete binary trees. With respect to the intuition, Path exhibits a contradicting behavior.
- **Fibonacci trees**: (See Figure 2.9(d)) Order of performance: Separation, Path, Rings, Level. Separation outperforms all other algorithms, while maintaining a nearly constant value. The distance between the performance of Path and Rings almost remains at an approximately constant 30 as \( n \) increases.

- **Unbalanced-to-the-left binary trees**: (See Figure 2.9(e)) Order of performance: Path, Level, Separation, Rings. Path has gone from one of the poorer performing algorithms for uniform edge length to performing the best. Level and Separation almost perform exactly the same. Rings again exhibits unsatisfactory performance.

- **Unbalanced-to-the-right binary trees**: (See Figure 2.9(f)) Order of performance: Path, Level, Separation, Rings. Surprisingly, Path and Level seemed to have the same performance as in unbalanced-to-the-left binary trees, where Separation became noticeably worse.

- **Molecular combinatorial binary trees**: (See Figure 2.9(g)) Order of Performance: Separation, Path, Level, Rings. Separation continues to perform similar to unbalanced-to-the-left binary trees. Interestingly, as it was the case for maximum edge length, Level performs very poorly, which again is different from unbalanced binary trees. Path quickly becomes prohibitive with \( n = 9,973 \) resulting in a uniform edge length of 60. Interestingly, the intuition is not confirmed in this case: while the total edge length for Separation is larger than the one for Path, the uniform edge length for Path grows much faster than the one for Separation.

- **Minimum Angle Size**: (See Figure 2.10) Order of performance: Rings, Level, Path, Separation.
In general, all algorithms produce drawings with satisfactory angle sizes between the edges connecting parent-nodes to their children. *Rings* always places the children on opposite sides of their parents, thus making an 180° angle between them. By always placing the root one unit above and in between its children, *Level* guarantees a minimum angle of 90°. *Separation* produces drawings with minimum angle of 90° for complete, AVL, and Fibonacci trees, and 45° for randomly-generated, unbalanced-to-the-left, and molecular combinatory binary trees. In some cases, *Separation* does not produce satisfactory results. For example, for some unbalanced-to-the-right trees, *Separation* produces angles of sizes 4° or 5°.

- **Average Angle Size**: (See Figure 2.11) Order of performance: *Rings, Level, Path, Separation*.

All algorithms produce drawings with the average angles between parent-nodes and their children above 80°, which represents that none of them produce many small angles. The average of angle sizes produced by *Separation* is between 80° and 90°. In the case of complete trees, *Separation* produces orthogonal drawings. For *Level*, the average is between 115° and 125° for complete, randomly-generated, and AVL trees, and between 135° and 170° for unbalanced trees. The average angle size for *Rings* is 180° for all tree-types, which represents that *Rings* always places its children separated by a 180° angle. In the case of *Path*, the average angle size is between 90° and 100°, which represents that *Path* produces very few 180° angles. For molecular combinatory binary trees, the average angle among the algorithms varies between 83° and 129°.

- **Angular Resolution**: (See Figure 2.12)

  - Complete binary trees: (See Figure 2.12(a)) Order of performance: *Rings, Path, Separation, Level*. *Path* has not been considered in the analysis of angular resolution, because
Path only draws binary trees using 90° and 180° angles. Rings and Separation provide good angular resolution, because of constant angles of 90°. Level begins very poorly and as \( n \) increases Level angular resolution becomes more severe.

- **AVL trees**: (See Figure 2.12(b)) Order of performance: Rings, Path, Separation, Level. Similarly, the results resemble closely to complete binary trees. Once again, Rings and Separation reach the optimal angular resolution value.

- **Randomly-generated binary trees**: (See Figure 2.12(c)) Order of performance: Rings, Path, Separation, Level. Interestingly, Rings provide the optimal results. Furthermore, Separation slightly falls behind Rings with angular resolution of no less than 45°. Again, Level is increasingly prohibitive.

- **Fibonacci trees**: (See Figure 2.12(d)) Order of performance: Rings, Path, Separation, Level. Rings for the fourth time has all values of 90°. The consistency of Level being the worst is confirmed.

- **Unbalanced-to-the-left binary trees**: (See Figure 2.12(e)) Order of performance: Path, Separation, Level, Rings. Overall, unbalanced-to-the-left binary trees result in very poor angular resolution values for all algorithms. This negative becomes greater as the tree size increases. Level again exhibits unsatisfactory performance.

- **Unbalanced-to-the-right binary trees**: (See Figure 2.12(f)) Order of performance: Path, Separation, Level, Rings. Results mimic that of its opposite tree-type unbalanced-to-the-left.

- **Molecular combinatory binary trees**: (See Figure 2.12(g)) Order of Performance: Path, Separation, Level, Rings. Similar to unbalanced binary trees, Level produces poor angular resolution. Also, Separation produces satisfactory angular resolution for \( n = 133 \)
and $n = 181$, but quickly becomes prohibitive at $n = 805$.

- **Closest Leaf:** *(See Figure 2.13)*

  - **Complete binary trees:** *(See Figure 2.13(a))* Order of performance: *Rings, Separation, Path, Level*. In general, all algorithms place at least one leaf close to the root. For example, for $n = 65,535$, the longest distance between the root and the closest leaf is $15.03$ in the case of *Level*. Interestingly, *Rings* always places a leaf very close to the root at a constant distance of $1.41$. *Separation, Path,* and *Level* place the closest leaf increasingly farther away from the root, growing at a very slow rate, with the distance between the root and the closest leaf for *Level* growing faster than the one for *Separation* and *Path*. *Separation* and *Path* exhibit almost identical behavior.

  - **AVL trees:** *(See Figure 2.13(b))* Order of performance: *Rings, Path, Level, Separation*. The distance to the closest leaf for *Level* and *Path* slowly increases as $n$ increases. Quite interestingly, *Separation* has an uneven behavior. Also, *Level* and *Path* exhibit similar behaviors as in the case of complete and random binary trees. Again, *Rings* always places a leaf at distance $1.41$ from the root all of the time.

  - **Randomly-generated binary trees:** *(See Figure 2.13(c))* Order of performance: *Path, Level, Separation, Rings*. The performances of *Path* and *Level* have similar rates of growth, with *Path* producing almost two times better results. *Level* and *Separation* provide almost identical results, except at $n = 30,000$, where *Level* is $14$ and *Separation* is $20.2$. In contrast, *Rings* performs poorly, becoming seven times larger than *Path* at $n = 30,000$.

  - **Fibonacci trees:** *(See Figure 2.13(d))* Order of performance: *Path, Separation, Level,*
Rings. Rings places leaf nodes far from the root compared to the other algorithms, and the performance grows very quickly. For instance, while the distance is 4.22 for \( n = 88 \), it increases to 383.2 for \( n = 46,367 \). Again, Separation, Level, and Path produce almost constant results, with Separation and Level exhibiting almost identical behavior.

- **Unbalanced-to-the-left binary trees**: (See Figure 2.13(e)) The order of performance alternates for different tree sizes, but in general is: Path, Separation, Level, Rings. Quite surprisingly, Path, Level, and Separation perform best on unbalanced-to-the-left trees. The highest distance for Path is 6.19 when \( n = 50,000 \), for Level is 28.26 when \( n = 50,000 \), and for Separation is 10.82 when \( n = 30,000 \). Interestingly, in the case of Path and Separation, the distance between the root and the closest leaf seems to be growing slowly with the number of nodes.

- **Unbalanced-to-the-right binary trees**: (See Figure 2.13(f)) Order of performance: Path, Level, Separation, Rings. The relationship between the performances of Level, Path, and Separation maintain the same behavior (good and bad) as \( n \) increases. For larger values of \( n \), all algorithms experience a decrease in rate change.

- **Molecular combinatorial binary trees**: (See Figure 2.13(g)) Order of performance: Path, Separation, Level, Rings. Level grows very fast. Very interestingly, Path always places a leaf at a distance of 2.24 from the root. Furthermore, Separation initially fluctuates about the distance of 20 and then levels at \( n = 20,005 \).

- **Farthest Leaf**: (See Figure 2.14)

  - **Complete binary trees**: (See Figure 2.14(a)) Order of performance: Rings, Separation, Path, Level. While the performances of Path, Rings and Separation are very good, with
a very slow growth rate, the performance of Level is unsatisfactory. For example, for 
\( n = 8,191 \), the distance to the farthest leaf for Rings is 89.1, for Separation is 219.1, for 
Path is 354.8, and for Level is 4,095. For \( n = 65,535 \), farthest leaf for Rings is 284.8, 
for Separation is 626, for Path is 811.3, and for Level is 32,767.

- **AVL trees**: (See Figure 2.14(b)) Order of performance: Rings, Separation, Path, Level. 
The performances of the algorithms on this measure are almost identical to their respective performances on complete trees, except Path is slightly worse.

- **Randomly-generated binary trees**: (See Figure 2.14(c)) Order of performance: Separation, Path, Rings, Level. Surprisingly, both Separation and Rings perform just slightly worse on randomly-generated binary trees compared to complete trees. Performances of Path and Rings are almost identical. Again, the distance to the farthest leaf grows much faster for Level than for Path, Separation and Rings.

- **Fibonacci trees**: (See Figure 2.14(d)) Order of performance: Separation, Path, Rings, Level. For Separation and Path the same pattern remains. On the other hand, Rings still remains better than Level, but the rate of growth in Rings has increased. Level exhibits the same unsatisfactory behavior.

- **Unbalanced-to-the-left binary trees**: (See Figure 2.14(e)) Order of performance: Path, Separation, Level, Rings. Interestingly, the performances of the algorithms closely resemble that of the previous two tree-types.

- **Unbalanced-to-the-right binary trees**: (See Figure 2.14(f)) Order of performance: Path, Separation, Level, Rings. A pattern has formed in the performances of the algorithms. The results in unbalanced-to-the-left trees closely mimic that of unbalanced-to-the-right trees.
- Molecular combinatory binary trees: (See Figure 2.14(g)) Order of performance: Separation, Path, Level, Rings. As the case in all categories of binary trees, Level rapidly becomes prohibitive. Separation produces satisfactory results, placing its farthest leaf at a distance of 764.6 for \( n = 50,005 \). The behavior of Path is approximately three times worse than Separation. Very interestingly, the intuition that short maximum edge length and close farthest leaf go together is verified in the case of molecular combinatory binary trees, but not verified in the case of unbalanced trees.

2.4.2 Conclusions

After evaluating the performances of Level, Path, Rings, and Separation, we have reached the following conclusions:

- Quality measure Area:

  - Rings produces excellent results for complete binary trees and AVL trees. Its performance degrades dramatically for other types to trees. For randomly-generated binary trees, its behavior is the worst of the four algorithms. For unbalanced binary trees, the area produced was so large that it could not be used for comparison.

  - Path produces excellent results for unbalanced and molecular combinatory binary trees. These results come at the expense of a very low aspect ratio. Path may produce drawings of these types of trees, with aspect ratio close to optimal, but in this case, its performance is comparable to that of Separation.

  - Level always produces results worse than Path and Separation.

  - Separation produces best results for randomly-generated and Fibonacci trees. The drawings that Separation constructs in these cases, also achieve optimal aspect ratios.
• Quality measure **Aspect Ratio:**

  – Since the aspect ratio of the drawings produced by *Separation* is user-controlled, this algorithm always achieves an aspect ratio close to optimal. This may come at the expense of a slightly larger area.

  – *Level* always produces unsatisfactory results.

  – Generally, *Path* produces aspect ratios close to optimal. This comes at the expense of a larger area. For unbalanced-to-the-right binary trees, AVL trees, Fibonacci trees, and molecular combinatory trees, the aspect ratios produced degrade quickly and become unsatisfactory.

  – For the cases in which can be plotted (complete, randomly-generated, AVL, and Fibonacci trees), *Rings* always produces aspect ratios over 0.5, and many times it achieves optimality.

• Quality measure **Size:**

  – *Level* produces unsatisfactory results for all categories of trees.

  – *Path* performs best on unbalanced-to-the-right, well on Fibonacci trees, and worst on randomly-generated and molecular combinatory binary trees.

  – *Rings* performs best on complete and unbalanced-to-the-left binary trees, and worst on randomly-generated and Fibonacci trees.

  – *Separation* performs good on all categories of trees and produces the best results on randomly-generated, unbalanced-to-the-left, AVL, Fibonacci, and molecular combinatory binary trees.
• **Quality measures** **Total Edge Length** and **Average Edge Length**:

  - *Rings* produces best results for complete and AVL binary trees.
  
  - *Separation* produces best results for randomly-generated and Fibonacci trees.
  
  - *Path* produces best results for unbalanced and molecular combinatorial binary trees.
  
  - *Level* produces worst results in all categories except for unbalanced-to-the-right binary trees.

• **Quality measure** **Maximum Edge Length**:

  - *Level* produces very good results for unbalanced binary trees, and very bad results for all other categories of trees.
  
  - *Separation* performs best on randomly-generated, Fibonacci, and molecular combinatorial binary trees, and worst on unbalanced-to-the-right binary trees.
  
  - *Path* performs worst on randomly-generated binary trees, best on unbalanced binary trees, and well on the other categories of trees.
  
  - *Rings* performs best on complete and AVL binary trees, and well on the other types of trees (with the exception of unbalanced and molecular combinatorial trees where we could not have a plot).

• **Quality measures** **Uniform Edge Length**:

  - *Level* performs best on unbalanced binary trees, and very poorly on all other categories of trees.
  
  - *Path* performs best on unbalanced binary trees, well on random binary trees, and bad on Complete and AVL binary trees.
- *Rings* produces very good results for all categories of trees.

- *Separation* produces good results for all categories of trees, except unbalanced-to-the-right binary trees with the best results on Fibonacci and molecular combinatorial binary trees.

• Quality measure *Minimum Angle Size*:

  - *Rings* produces best results in all categories, because it always places children on opposite sides of their corresponding parents.

  - *Level* produces the same minimum angle size in all categories of trees. The smallest angle size produced is $90^\circ$, in the case when the subtrees rooted at the left and right children of a node have the same structure.

  - *Path* produces orthogonal drawings. Therefore, in all cases, the drawings produced by *Path* have at least a $90^\circ$ separation between the edges connecting a parent-node to its children.

  - *Separation* performs best on complete, AVL, and Fibonacci trees, satisfactory on randomly-generated and unbalanced-to-the-left trees, and sometimes unsatisfactory on unbalanced-to-the-right binary trees.

• Quality measure *Average Angle Size*:

  - *Rings* always produces $180^\circ$ angles.

  - *Level* consistently produces high angles between children. Highest angles are produced for unbalanced trees.
- \textit{Path} performs best on unbalanced binary trees. In general, \textit{Path} produces very few 180° angles. In the case of complete, AVL, and Fibonacci trees, \textit{Path} produces only 90° angles.

- \textit{Separation} is slightly less than optimal on randomly-generated binary trees and it has almost the same behavior on the rest of the trees.

* Quality measures \textbf{Angular Resolution}:

- \textit{Level} produces bad results for all categories of trees.

- \textit{Path} produces orthogonal drawings. Therefore, in all cases, the drawings produced by \textit{Path} have at least a 90° separation between the edges connecting a parent-node to its children.

- \textit{Rings} produces optimal results for all trees except unbalanced binary trees.

- \textit{Separation} performs the best on complete, AVL, Fibonacci, and molecular combinatorial binary trees, well on randomly-generated binary trees, and unsatisfactory on unbalanced binary trees.

* Quality measure \textbf{Closest Leaf}:

- \textit{Rings} produces excellent results on complete and AVL trees and worse, unsatisfactory results on Fibonacci and unbalanced binary trees.

- \textit{Level} performs best on unbalanced and Fibonacci trees and worst on AVL, complete, and randomly-generated binary trees.

- \textit{Path} produces excellent results on all types of trees, with its best performance on Fibonacci trees, and its worse performance on complete binary trees.
- *Separation* produces best results for molecular combinatory binary trees and very good results for all other types of trees, except for unbalanced-to-the-right binary trees, on which it produces its worse results.

- **Quality measure Farthest Leaf:**

  - *Rings* performs best on complete and AVL trees, and worst on unbalanced and Fibonacci trees.
  
  - *Level* performs worse than the other algorithms for all categories of trees. The performance of *Level* is not satisfactorily on any category of trees.

  - *Path* produces good results for tree-types, with its best being for unbalanced binary trees.

  - *Separation* produces very good results on all categories of trees. Its best performance is on complete, AVL, and molecular combinatory binary trees, and its worse performance is on unbalanced-to-the-right binary trees.

### 2.5 Charts of Experimental Results

The following are charts of the results derived from our experimental study. Please note all charts span two pages.
Figure 2.4: Comparison charts of the area for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
Figure 2.5: Comparison charts of the aspect ratio for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
Figure 2.6: Comparison charts of the size for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
Figure 2.7: Comparison charts of the total edge length for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
Figure 2.8: Comparison charts of the maximum edge length for \textit{Level}, \textit{Path}, \textit{Rings}, and \textit{Separation}, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
Figure 2.9: Comparison charts of the uniform edge length for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary tree, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
Figure 2.10: Comparison charts of the minimum angle size for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary trees, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
Figure 2.11: Comparison charts of the average angle size for *Level*, *Path*, *Rings*, and *Separation*, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary trees, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
Figure 2.12: Comparison charts of the angular resolution for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary trees, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
Figure 2.13: Comparison charts of the closest leaf for \textit{Level, Path, Rings,} and \textit{Separation}, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary trees, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
Figure 2.14: Comparison charts of the farthest leaf for Level, Path, Rings, and Separation, for each tree-type: (a) Complete binary trees, (b) AVL trees, (c) Randomly-generated binary trees, (d) Fibonacci trees, (e) Unbalanced-to-the-left binary trees, (f) Unbalanced-to-the-right binary trees, (g) Molecular Combinatory binary trees.
3. A PRACTICAL ALGORITHM FOR PLANAR STRAIGHT-LINE GRID DRAWINGS OF
GENERAL TREES WITH LINEAR AREA AND ARBITRARY ASPECT RATIO

3.1 Introduction

In this chapter, we investigate the problem of constructing planar straight-line grid drawings of
general trees with small area. Any planar grid drawing of a tree with \( n \) nodes requires \( \Omega(n) \) area.
A long-standing fundamental question, therefore, has been that whether this is a tight bound also,
i.e., given a tree \( T \) with \( n \) nodes, can we construct a planar straight-line grid drawing of \( T \) with area
\( O(n) \)?

This question was answered in affirmative in [31], which presented an algorithm for constructing
a planar straight-line grid drawing of an \( n \)-node degree-\( d \) tree \( T \), where \( d = O(n^\delta) \) is a positive
integer and \( 0 \leq \delta < 1/2 \) is a constant, with area \( O(n) \) and with any pre-specified aspect ratio \( A \) in
the range \( [n^{-\alpha}, n^\alpha] \), where \( 0 \leq \alpha \leq 1 \) is any constant, in \( O(n \log n) \) time. The algorithm also achieves
the subtree separation property.

While the algorithm of [31] was significant from a theoretical point of view, it suffered from the
following drawbacks, that made it unsuitable for practical use:

- The constant \( c \), where \( c \) is the ratio between area of drawing and number of nodes, hidden in
  the "Oh" notation for area can be quite large (for binary trees was 3900). One might argue
  that \( c \) is really the worst-case bound, and the algorithm might perform better in practice.
However, the problem is that given a tree $T$ with $n$ nodes, the algorithm will always prere-allocate a rectangle $R$ with size exactly equal to $cn$, and draw $T$ within $R$. Thus, the area of $R$ is always equal to the worst-case area, and correspondingly, the drawing also has a large area. This is the major drawback of the algorithm.

- Also, it uses another algorithm, called $Algorithm u^* - HV - Draw$, as a subroutine. This increases the complexity of implementing the algorithm.

In this chapter, we have made several improvements to the algorithm of [31], which make it more suitable for practical use (note that the properties of the drawing constructed by this newer version of the algorithm are preserved: linear area, planarity, straight-line, subtree separation property):

- We have developed a newer version of the algorithm of [31] that does not require the pre-assignment of a rectangle with the worst-case area to draw a tree. Instead, it only pre-assigns an aspect ratio to the tree, which is used to draw the tree recursively in a bottom-up fashion. This makes it possible for the algorithm to construct a more area-efficient drawing in practice.

- This newer version does not require $Algorithm u^* - HV - Draw$ as a subroutine, which makes it easier to implement.

- We have also implemented this newer version, and experimentally evaluated its performance for randomly-generated general trees with up to 50,000 nodes. Our experiments show that it constructs area-efficient drawings in practice, with area at most 19 times the number of nodes, which represents an area more than 200 times less than the one generated by the algorithm of [31].

Our algorithm is a generalization of the algorithm of [32], which draws binary trees, and originally appeared in [67].
3.2 Preliminaries

Let $\Gamma$ be a drawing of $T$. Let $T$ be a degree-$d$ tree, with one distinguished node $v$, which has at most $d - 1$ children. $v$ is called the link node of $T$. Let $n$ be the number of nodes in $T$. $T$ is an ordered tree if the children of each node are assigned a left-to-right order. The first child of $T$ is allowed to be empty, and can be followed by 0 to $d - 2$ children. A partial tree of $T$ is a connected subgraph of $T$. If $T$ is an ordered tree, then the leftmost path $p$ of $T$ is the maximal path consisting of nodes that are leftmost children, except the first one, which is the root of $T$. The last node of $p$ is called the leftmost node of $T$. Two nodes of $T$ are siblings if they have the same parent in $T$. $T$ is an empty tree, i.e., $T = \emptyset$, if it has zero nodes in it.

Let $R$ be a rectangle, such that $\Gamma$ is entirely contained within $R$. $R$ has a good aspect ratio, if its aspect ratio is in the range $[n^{-\alpha}, n^\alpha]$, where $0 \leq \alpha < 1$ is a constant.

Let $r$ be the root of $T$. Let $u^*$ be the link node of $T$. $\Gamma$ is a feasible drawing of $T$, if it has the following three properties:

- **Property 1:** The root $r$ is placed at the top-left corner of $\Gamma$.

- **Property 2:** If $u^* \neq r$, then $u^*$ is placed at the bottom boundary of $\Gamma$. Moreover, we can move $u^*$ downwards in its vertical channel by any distance without causing any edge-crossings in $\Gamma$.

- **Property 3:** If $u^* = r$, then no other node or edge of $T$ is placed on, or crosses the vertical and horizontal channels occupied by $r$.

Let $v$ be a node of tree $T$ located at grid point $(i,j)$ in $\Gamma$. Let $\Gamma$ be a drawing of $T$. Assume that the root $r$ of $T$ is located at the grid point $(0,0)$ in $\Gamma$. We define the following operations on $\Gamma$ (see Figure 3.1):
Figure 3.1: Rotating a drawing $\Gamma$ by $90^\circ$, followed by flipping it vertically. Note that initially node $u^*$ was located at the bottom boundary of $\Gamma$, but after the rotate operation, $u^*$ is on the right boundary of $\Gamma$.

Figure 3.2: Drawing of a randomly-generated general tree with 60 nodes constructed by Algorithm DrawGeneralTree, with $A = 1$ and $\varepsilon = 0.2$.

- **rotate operation**: rotate $\Gamma$ counterclockwise by $\delta$ degrees around the z-axis passing through $r$. After a rotation by $\delta$ degrees of $\Gamma$, node $v$ will get relocated to the point $(i \cos \delta - j \sin \delta, i \sin \delta + j \cos \delta)$. In particular, after rotating $\Gamma$ by $90^\circ$, node $v$ will get relocated to the grid point $(-j, i)$ (See first part of Figure 3.1).

- **flip operation**: flip $\Gamma$ vertically or horizontally. After a horizontal flip of $\Gamma$, node $v$ will be located at grid point $(-i, j)$. After a vertical flip of $\Gamma$, node $v$ will be located at grid point $(i, -j)$ (See second part of Figure 3.1).
3.3 Practical General Tree Drawing Algorithm

Let $T$ be a degree-$d$ tree with a link node $u^*$, where $d = O(n^3)$ is a positive integer, $0 \leq \delta < 1/2$ is a constant, and $n$ is the number of nodes in $T$. Let $A$ and $\varepsilon$ be two numbers such that $\delta/(1-\delta) < \varepsilon < 1$, and $A$ is in the range $[n^{-\varepsilon}, n^{\varepsilon}]$. $A$ is called the \textit{desirable aspect ratio} for $T$.

Our tree drawing algorithm, called \textit{DrawGeneralTree}, takes $\varepsilon$, $A$, and $T$ as input, and uses a simple divide-and-conquer strategy to recursively construct a feasible drawing $\Gamma$ of $T$.

Figure 3.2 shows a drawing of a randomly-generated general tree with 60 nodes constructed by Algorithm \textit{DrawGeneralTree}, with $A = 1$ and $\varepsilon = 0.2$.

We now give the details of each action performed by Algorithm \textit{DrawGeneralTree}:

3.3.1 Split Tree

The splitting of tree $T$ into partial trees is done as follows:

- Order the children of each node such that $u^*$ becomes the leftmost node of $T$.
- Find a separator node $u$ of $T$.
- Based on whether, or not, $u$ is in the leftmost path of $T$, we get two cases:

  - \textit{Case 1: The separator node $u$ is not in the leftmost path of $T$}. In the general case, $T$ has the form as shown in Figure 3.3(a). In this figure:

    * $r$ is the root of $T$,
    * $c_1, \ldots, c_j$ are the children of $u$, $0 \leq j \leq d-1$,
    * $T_1, \ldots, T_j$ are the trees rooted at $c_1, \ldots, c_j$ respectively, $0 \leq j \leq d-1$,
    * $T_u$ is the subtree rooted at $u$,
    * $w$ is the parent of $u$,
* $a$ is the last common node of the path $r \to v$ and the leftmost path of $T$,

* $f$ is the child of $a$ that is contained in the path $r \to v$.

* $T_b$ is the maximal tree rooted at $f$ that contains $w$ but not $u$.

* $T_B$ is the tree consisting of the trees $T_a$ and $T_b$, and the edge $(w, u)$.

* $e$ is the parent of $a$.

* $g$ is the leftmost child of $a$.

* $T_A$ is the maximal tree rooted at $r$ that contains $e$ but not $a$.

* $T_C$ is the tree rooted at $g$.

* $b_1, \ldots, b_i$ are the siblings of $f$ and $g$.

* $T'_1, \ldots, T'_l$ are the trees rooted at $b_1, \ldots, b_l$ respectively, $0 \leq i \leq d-3$, and

* $g \neq u^*$. 

In addition to the general case, we get six special cases: (b) $T_A = 0, T_C = 0, 0 \leq i \leq d-3$,

(c) $T_A \neq 0, T_C \neq 0, g = u^*, 0 \leq i \leq d-3$, (d) $T_A \neq 0, T_C = 0, r \neq e, 0 \leq i \leq d-3$, (e) $T_A \neq 0, T_C = 0, r = e, 0 \leq i \leq d-3$, (f) $T_A = 0, T_C \neq 0, g \neq u^*, 0 \leq i \leq d-3$, and (g) $T_A = 0, T_C \neq 0, g = u^*, 0 \leq i \leq d-3$. (The reason we get these seven subcases is as follows: $T_a$ has at least $n/2$ nodes in it. Hence $T_a \neq \phi$, and so, $T_B \neq \phi$. Based on whether $T_A = \phi$ or not, $T_C = \phi$ or not, $g = u^*$ or not, and $r = e$ or not, we get totally sixteen cases. 

From these sixteen cases, we obtain the above seven subcases, by grouping some of these cases together. For example, the cases $T_A = \phi, T_C = \phi, g \neq u^*, r = u^*, \text{and } T_A = \phi, T_C = \phi, g \neq u^*$ are grouped together to give Case (a), i.e., $T_A = \phi, T_C = \phi, g \neq u^*$. 

So, Case (a) corresponds to 2 cases. Similarly, Cases (c), (d), (e), (f), and (g) correspond to 2 cases each, and Case (b) corresponds to 4 cases.) In each case, we remove nodes $a$ and $u$, and their incident edges, to split $T$ into at most $2d-1$ partial trees $T_A, T_C, T_B$. 

72
We also designate $e$ as the link node of $T_A$, $w$ as the link node of $T_B$, and $u^*$ as the link node of $T_C$. We arbitrarily select a leaf $e_i$ of $T_i'$, $0 \leq i \leq d - 3$, and a leaf $e_j$ of $T_j$, $0 \leq j \leq d - 1$, and designate them as the link nodes of $T_i'$ and $T_j$, respectively.

- **Case 2**: The separator node $u$ is in the leftmost path of $T$. In the general case, $T$ has the form as shown in Figure 3.4(a). In this figure,

  * $r$ is the root of $T$,
  * $v$ is the leftmost child of $u$,
  * $c_1, \ldots, c_j$ are the siblings of $v$, $1 \leq j \leq d - 2$,
  * $T_1, \ldots, T_j$ are the trees rooted at $c_1, \ldots, c_j$ respectively, $1 \leq j \leq d - 2$,
  * $e$ is the parent of $u$,
  * $T_A$ is the maximal tree rooted at $r$ that contains $e$ but not $u$,
  * $T_C$ is the subtree of $T$ rooted at $v$,
  * $F_B$ is the forest composed by trees $T_1, \ldots, T_j$, $1 \leq j \leq d - 2$, and
  * $v \neq u^*$.

In addition to the general case, we get the following ten special cases: (b) $T_A = 0$, $j = 0$, $v \neq u^*$, (c) $T_A = 0$, $1 \leq j \leq d - 2$, $v \neq u^*$, (d) $T_A \neq 0$, $j = 0$, $v \neq u^*$, (e) $T_A \neq 0$, $1 \leq j \leq d - 2$, $v = u^*$, (f) $T_A = 0$, $j = 0$, $v = u^*$, (g) $T_A = 0$, $1 \leq j \leq d - 2$, $v = u^*$, (h) $T_A \neq 0$, $j = 0$, $v = u^*$, (k) $T_A = 0$, $T_B \neq 0$, $T_C \neq 0$, $r = u = u^*$, $j > 0$, (l) $T_A \neq 0$, $T_B \neq 0$, $T_C \neq 0$, $r \neq e$, $u = u^*$, and (m) $T_A \neq 0$, $T_B \neq 0$, $T_C \neq 0$, $r = e$, $u = u^*$, $j > 0$.

(The reason we get these eleven subcases is as follows: $T_C$ has at least $n/2$ nodes in it. Hence, $T_C \neq \emptyset$. Based on whether $T_A = \emptyset$ or not, $F_B = \emptyset$ or not, $v = u^*$ or not, and $u = u^*$

73
or not we get the eleven subcases given above.) In each case, we remove node \( u \), and its incident edges, to split \( T \) into at most \( d \) partial trees \( T_A, T_C, \) and \( T_1, \ldots, T_j, \) \( 0 \leq j \leq d-2 \).

We also designate \( e \) as the link node of \( T_A \), and \( u^* \) as the link node of \( T_C \). We randomly select a leaf \( e_j \) of \( T_j \) and designate it as the link node of \( T_j, 0 \leq j \leq d-2 \).

### 3.3.2 Assign Aspect Ratios

Let \( T_k \) be a partial tree of \( T \), where for Case 1, \( T_k \) is either \( T_A, T_C, T_{i_1}, T_{i_2}, \ldots, T_{i_l}, \) \( 0 \leq i \leq d-3 \), or \( T_1, \ldots, T_j, \) \( 0 \leq j \leq d-1 \), and for Case 2, \( T_k \) is either \( T_A, T_C, \) or \( T_1, \ldots, T_j, \) \( 0 \leq j \leq d-2 \). Let \( n_k \) be number of nodes in \( T_k \).

**Definition:** \( T_k \) is a *large* partial tree of \( T \) if:

- \( A \geq 1 \) and \( n_k \geq (n/A)^{1/(1+\varepsilon)} \), or
- \( A < 1 \) and \( n_k \geq (An)^{1/(1+\varepsilon)} \),

and is a *small* partial tree of \( T \) otherwise.

In Step Assign Aspect Ratios, we assign a desirable aspect ratio \( A_k \) to each nonempty \( T_k \) as follows: Let \( x_k = n_k/n \).

- If \( A \geq 1 \): If \( T_k \) is a large partial tree of \( T \), then \( A_k = x_k A \), otherwise (i.e., if \( T_k \) is a small partial tree of \( T \)) \( A_k = n_k^{-\varepsilon} \).
- If \( A < 1 \): If \( T_k \) is a large partial tree of \( T \), then \( A_k = A/x_k \), otherwise (i.e., if \( T_k \) is a small partial tree of \( T \)) \( A_k = n_k^\varepsilon \).

Intuitively, this assignment strategy ensures that each partial tree gets a desirable aspect ratio,
Figure 3.3: Drawing $T$ in all the seven subcases of Case 1 (when the separator node $u$ is not in the leftmost path of $T$): (a) $T_A \neq 0$, $T_C \neq 0$, $g \neq u^*$, $0 \leq i \leq d-3$, (b) $T_A = 0$, $T_C = 0$, $0 \leq i \leq d-3$, (c) $T_A \neq 0$, $T_C \neq 0$, $g = u^*$, $0 \leq i \leq d-3$, (d) $T_A \neq 0$, $T_C = 0$, $r \neq e$, $0 \leq i \leq d-3$, (e) $T_A \neq 0$, $T_C = 0$, $r = e$, $0 \leq i \leq d-3$, (f) $T_A = 0$, $T_C \neq 0$, $g \neq u^*$, $0 \leq i \leq d-3$, and (g) $T_A = 0$, $T_C \neq 0$, $g = u^*$, $0 \leq i \leq d-3$. For each subcase, we first show the structure of $T$ for that subcase, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. Here, $x$ is the same as $f$ if $T_B \neq \phi$, and is the same as the root of $T_B$ if $T_B = \phi$. In Subcases (a) and (c), for simplicity, $e$ is shown to be in the interior of $\Gamma_A$, but actually, either it is the same as $r$, or if $A < 1$ ($A \geq 1$), then it is placed on the bottom (right) boundary of $\Gamma_A$. For simplicity, we have shown $\Gamma_A$, $\Gamma_B$, and $\Gamma_C$ as identically sized boxes, but in actuality, they may have different sizes.

and therefore, the drawing of each partial tree constructed recursively by Algorithm $\text{DrawGeneralTree}$ will fit inside a rectangle with linear area and good aspect ratio.
Figure 3.4: Drawing $T$ in all the eight subcases of Case 2 (when the separator node $u$ is in the leftmost path of $T$): (a) $T_A \neq 0, T_C \neq 0, v \neq u^*, 1 \leq j \leq d - 2$, (b) $T_A = 0, T_C \neq 0, v \neq u^*, j = 0$, (c) $T_A = 0, T_C \neq 0, v \neq u^*, 1 \leq j \leq d - 2$, (d) $T_A \neq 0, T_C \neq 0, v \neq u^*, j = 0$, (e) $T_A \neq 0, T_B \neq 0, v = u^*$, $1 \leq j \leq d - 2$, (f) $T_A = 0, T_B = 0, v = u^*, j = 0$, (g) $T_A = 0, T_B \neq 0, v = u^*, 1 \leq j \leq d - 2$, (h) $T_A \neq 0, T_B = 0, v = u^*, j = 0$, (k) $T_A = 0, T_B \neq 0, T_C = 0, r = u = u^*, j > 0$, (l) $T_A \neq 0, T_B = 0, T_C = 0, r \neq e, u = u^*, j > 0$, and (m) $T_A \neq 0, T_B = 0, T_C = 0, r = e, u = u^*, j > 0$. For each subcase, we first show the structure of $T$ for that subcase, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. In Subcases (a), (d), (e), and (h) for simplicity, $e$ is shown to be in the interior of $\Gamma_A$, but actually, either it is same as $r$, or if $A < 1$ ($A \geq 1$), then it is placed on the bottom (right) boundary of $\Gamma_A$. For simplicity, we have shown $\Gamma_A$, $\Gamma_B$, and $\Gamma_C$ as identically sized boxes, but in actuality, they may have different sizes.
3.3.3 Draw Partial Trees

First, we change the desirable aspect ratios assigned to $T_A$ and $T_B$ in some cases as follows: Suppose $T_A$ and $T_B$ get assigned desirable aspect ratios equal to $m$ and $p$, respectively, where $m$ and $p$ are some positive numbers. In Subcase (d) of Case 1, and if $\alpha \geq 1$, then in Subcases (a) and (c) of Case 1, and Subcases (a), (d), (e), and (h) of Case 2, we change the value of the desirable aspect ratio of $T_A$ to $1/m$. In Case 1, if $\alpha \geq 1$, we change the value of the desirable aspect ratio of $T_B$ to $1/p$. We make these changes because, as explained later in Section 3.3.4, in these cases, we need to rotate the drawings of $T_A$ and $T_B$ by $90^\circ$ during the Compose Drawings step. Drawing $T_A$ and $T_B$ with desirable aspect ratios $1/m$ and $1/p$, respectively, compensates for this rotation, and ensures that the drawings of $T_A$ and $T_B$ used to draw $T$ have the desirable aspect ratios, $m$ and $p$, respectively.

Next we draw recursively each nonempty partial tree $T_k$ with $\alpha_k$ as its desirable aspect ratio, where the value of $\alpha_k$ is the one computed in the previous step. The base case for the recursion happens when $T_k$ contains exactly one node, in which case, the drawing of $T_k$ is simply the one consisting of exactly one node.

3.3.4 Compose Drawings

Let $\Gamma_k$ denote the drawing of a partial tree $T_k$ constructed in Step Draw Partial Trees. We now describe the construction of a feasible drawing $\Gamma$ of $T$ from the drawings of its partial trees in both Cases 1 and 2.

In Case 1, we first construct a feasible drawing $\Gamma_\alpha$ of the partial tree $T_\alpha$ by composing $\Gamma_1, \ldots, \Gamma_j$, $0 \leq j \leq d-1$, as shown in Figure 3.5, then construct a feasible drawing $\Gamma_B$ of $T_B$ by composing $\Gamma_\alpha$ and $\Gamma_B$ as shown in Figure 3.6, and finally construct $\Gamma$ by composing $\Gamma_\alpha, \Gamma_B, \Gamma_\gamma, \Gamma'_1, \ldots, \Gamma'_i$, $0 \leq i \leq d-3$, as shown in Figure 3.3.
Figure 3.5: Drawing $T_a$. Here, we first show the structure of $T_a$ in (a), then its drawing when $A < 1$ in (b), and then its drawing when $A \geq 1$ in (c).

Figure 3.6: Drawing $T_B$ when: (a) $T_B \neq \emptyset$, and (b) $T_B = \emptyset$. For each case, we first show the structure of $T_B$ for that case, then its drawing when $A < 1$, and then its drawing when $A \geq 1$. In Case (a), for simplicity, $w$ is shown to be in the interior of $\Gamma_B$, but actually, it is either same as $f$, or if $A < 1$ ($A \geq 1$), then is placed on the bottom (right) boundary of $\Gamma_B$.

$\Gamma_a$ is constructed as follows (see Figure 3.5): If $A < 1$, place $\Gamma_j, \ldots, \Gamma_2, \Gamma_1$, $1 \leq j \leq d - 1$, one above the other, in this order, separated by unit vertical distance, such that the left boundaries of $\Gamma_j, \ldots, \Gamma_2$ are aligned, and one unit to the right of the left boundary of $\Gamma_1$. Place $u$ in the same vertical channel as $c_1$ and in the same horizontal channel as $c_j$. If $A \geq 1$, place $\Gamma_1, \Gamma_2, \ldots, \Gamma_j$, $1 \leq j \leq d - 1$ in a left-to-right order, separated by unit horizontal distance, such that the top boundaries of $\Gamma_1, \Gamma_2, \ldots, \Gamma_{j-1}$ are aligned, and one unit below the top boundary of $\Gamma_j$. Place $u$ in the same vertical channel as $c_1$ and in the same horizontal channel as $c_j$.

$\Gamma_B$ is constructed as follows (see Figure 3.6):

- if $T_B \neq \emptyset$ (see Figure 3.6(a)) then, if $A < 1$, then place $\Gamma_B$ one unit above $\Gamma_a$ such that the left boundaries of $\Gamma_B$ and $\Gamma_a$ are aligned; otherwise (i.e., if $A \geq 1$), first rotate $\Gamma_B$ by 90° and then flip it vertically, then place $\Gamma_B$ one unit to the left of $\Gamma_a$ such that the top boundaries of $\Gamma_B$ and $\Gamma_a$ are aligned. Draw edge $(w, y)$. 78
• Otherwise (i.e., if \( T_B = \emptyset \)), \( \Gamma_B \) is same as \( \Gamma_A \) (see Figure 3.6(b)).

\( \Gamma \) is constructed from \( \Gamma_A, \Gamma_B, \Gamma_C, \Gamma_1', \ldots, \Gamma_p', 0 \leq i \leq d - 3 \), as follows (see Figure 3.3): Let \( x \) be the root of \( T_B \). Note that \( x = f \) if \( T_B \neq \emptyset \), and \( x = u \) otherwise.

• In Subcase (a), as shown in Figure 3.3(a), if \( A < 1 \), stack \( \Gamma_A, \Gamma_1', \ldots, \Gamma_p', \Gamma_B, \Gamma_C \) one above the other, in this order, such that they are separated by unit vertical distance from each other, and the left boundaries of \( \Gamma_1', \ldots, \Gamma_p', \Gamma_B \) are aligned with each other and are placed at unit horizontal distance to the right of the left boundaries of \( \Gamma_A \) and \( \Gamma_C \). Place node \( a \) in the same vertical channel as \( r \) and \( g \) and in the same horizontal channel as \( b_i \). If \( A \geq 1 \), then first rotate \( \Gamma_A \) by 90°, and then flip it vertically. Then, place \( \Gamma_A, \Gamma_C, \Gamma_1', \ldots, \Gamma_p', \Gamma_B \) from left-to-right in this order, separated by unit horizontal distances, such that the top boundaries of \( \Gamma_C, \Gamma_1', \ldots, \Gamma_p' \) are aligned, and are at unit vertical distance below the top boundaries of \( \Gamma_A \) and \( \Gamma_B \). Then, move \( \Gamma_C \) down until \( u^* \) becomes the lowest node of \( \Gamma \). Place node \( a \) in the same vertical channel as \( g \) and in the same horizontal channel as \( r \) and \( x \). Draw edges \((a, e), (a, x), (a, g), (a, b_1), \ldots, (a, b_i)\).

• In Subcase (b), as shown in Figure 3.3(b), if \( A < 1 \), stack \( \Gamma_1', \ldots, \Gamma_p', \Gamma_B \) one above the other, such that they are separated by unit vertical distance from each other, and their left boundaries are aligned. Place node \( r \) one unit above and left of the top boundary of \( \Gamma_1' \). If \( A \geq 1 \), place \( \Gamma_1', \ldots, \Gamma_p', \Gamma_B \) in a left-to-right order such that they are separated by unit horizontal distance from each other, and their top boundaries are aligned. Place node \( r \) one unit above and left of the top boundary of \( \Gamma_1' \). Draw edges \((r, b_1), \ldots, (r, b_i), (r, x)\).

• The drawing procedure for Subcase (c) is similar to the one in Subcase (a), except that we also flip \( \Gamma_C \) vertically (see Figure 3.3(c)).
• In Subcase (d), as shown in Figure 3.3(d), if \( A < 1 \), flip \( \Gamma_i', \ldots, \Gamma_1', \Gamma_B \) first vertically, and then horizontally, so that their roots get placed at their lower-right corners. Then, first rotate \( \Gamma_A \) by \( 90^\circ \), and then flip it vertically. Next, place \( \Gamma_A, \Gamma_i', \ldots, \Gamma_1', \Gamma_B \) one above the other, in this order, with unit vertical separation, such that their left boundaries are aligned, next move node \( e \) (which is the link node of \( \Gamma_A \)) to the right until it is either to the right of, or aligned with the rightmost boundary among \( \Gamma_i', \ldots, \Gamma_1', \Gamma_B \) (since \( \Gamma_A \) is a feasible drawing, by Property 2, as given in Section 3.2, moving \( e \) will not create any edge-crossings), and then place \( u^* \) in the same horizontal channel as \( x \) and one unit to the right of \( e \). If \( A \geq 1 \), first rotate \( \Gamma_A \) by \( 90^\circ \), and then flip it vertically. Then flip \( \Gamma_i', \ldots, \Gamma_1', \Gamma_B \) vertically. Then, place \( \Gamma_A, u^*, \Gamma_i', \ldots, \Gamma_1', \Gamma_B \) left-to-right in this order, separated by unit horizontal distances, such that the bottom boundaries of \( \Gamma_i', \ldots, \Gamma_1', \Gamma_B \) are aligned, and are at unit vertical distance above the bottom boundary of \( \Gamma_B \). Move \( \Gamma_B \) down until its bottom boundary is either aligned with or below the bottom boundary of \( \Gamma_A \). Also, \( u^* \) is in the same horizontal channel with \( x \). Draw edges \((u^*, e), (u^*, b_1), \ldots, (u^*, b_i), (u^*, x)\).

• In Subcase (e), as shown in Figure 3.3(e), if \( A < 1 \), first flip \( \Gamma_i', \ldots, \Gamma_1', \Gamma_B \), vertically, then place \( \Gamma_A, \Gamma_i', \ldots, \Gamma_1', \Gamma_B \) one above the other, in this order, with unit vertical separation, such that the left boundaries of \( \Gamma_i', \ldots, \Gamma_1', \Gamma_B \) are aligned, and the left boundary of \( \Gamma_A \) is at unit horizontal distance to the left of the left boundary of \( \Gamma_B \). Place \( u^* \) in the same vertical channel with \( r \) and in the same horizontal channel with \( x \). If \( A \geq 1 \), then first flip \( \Gamma_1', \ldots, \Gamma_B \) vertically, next place \( \Gamma_A, \Gamma_1', \ldots, \Gamma_B \) in a left-to-right order at unit horizontal distance, such that the top boundaries \( \Gamma_A, \Gamma_i', \ldots, \Gamma_1' \) are aligned, and the bottom boundary of \( \Gamma_B \) is one unit below the bottom boundary of the drawing among \( \Gamma_A, \Gamma_1', \ldots, \Gamma_B \) with greater height. Then, place \( u^* \) in the same vertical channel as \( r \) and in the same horizontal channel as \( x \). Draw edges
(u*, r), (u*, b1), ..., (u*, b1), (u*, x). Note that, since ΓA is a feasible drawing, by Property 3 (see Section 3.2), drawing (u*, r) will not create any edge-crossings.

- The drawing procedure in Subcase (f) is similar to the one in Subcase (a), except that we do not have ΓA here (see Figure 3.3(f)).

- The drawing procedure in Subcase (g) is similar to the one in Subcase (f), except that we also flip ΓC vertically (see Figure 3.3(g)).

In Case 2, we construct Γ by composing ΓA, Γ1, ..., Γj, ΓC (see Figure 3.4).

- In Subcase (a), as shown in Figure 3.5(a), if A < 1, stack ΓA, u, Γj, ..., Γj, ΓC one above the other, in this order, such that they are separated by unit vertical distance from each other, and the left boundaries of Γj, ..., Γj are aligned with each other and are placed at unit horizontal distance to the right of the left boundaries of ΓA. Place node u in the same vertical channel as r. If A ≥ 1, then first rotate ΓA by 90°, and then flip it vertically. Then, place ΓA, ΓC, Γj, ..., Γj from left-to-right in this order, separated by unit horizontal distances, such that the top boundaries of ΓC, Γj, ..., Γj-1, are aligned, and are at unit vertical distance below the top boundaries of ΓA. Then, move ΓC down until u* becomes the lowest node of Γ. Place node u in the same vertical channel as v and in the same horizontal channel as r and c_j. Draw edges (e, u), (u, v), (u, c1), ..., (u, c_j).

- In Subcase (b) as shown in Figure 3.5(b), if A < 1, place u in the same horizontal channel and at one unit to the left of v; otherwise (i.e. A ≥ 1), place u in the same vertical channel and at one unit above v. Draw edge (r, v).

- The drawing procedure in Subcase (c) is similar to the one in Subcase (a), except that we do not have ΓA here (see Figure 3.5(c)).

81
• In Subcase (d), as shown in Figure 3.4(d), if $A < 1$, we place $\Gamma_D$ above $\Gamma_C$, separated by unit vertical distance such that the left boundary of $\Gamma_C$ is one unit to the right of the left boundary of $\Gamma_D$. Place $u$ in the same vertical channel as $r$ and in the same horizontal channel as $v$. If $A \geq 1$, then first rotate $\Gamma_D$ by $90^\circ$, and then flip it vertically. Then, place $\Gamma_D$ to the left of $\Gamma_C$, separated by unit horizontal distance, such that the top boundary of $\Gamma_C$ is one unit below the top boundary of $\Gamma_D$. Then, move $\Gamma_C$ down until $u^*$ becomes the lowest node of $\Gamma$. Place $u$ in the same vertical channel as $v$ and in the same horizontal channel as $r$. Draw edges $(u, v)$ and $(u, e)$.

• The drawing procedures in Subcases (e), (f), (g), and (h) are similar to those in Subcases (a), (b), (c), and (d), respectively, (see Figures 3.4(e,f,g,h)), except that we also flip $\Gamma_C$ vertically.

• In Subcase (k), as shown in Figure 3.4(k), if $A < 1$, stack $\Gamma_1, \Gamma_2, \ldots, \Gamma_j$, one above the other, such that they are separated by unit distance from each other, and their left boundaries are aligned. Place node $r$ one unit above and left of the top boundary of $\Gamma_1$. If $A \geq 1$, place $\Gamma_1, \Gamma_2, \ldots, \Gamma_j$ in a left-to-right order such that they are separated by unit horizontal distance from each other, and their top boundaries are aligned. Place node $r$ one unit above and left of the top boundary of $\Gamma_1$. Draw edges $(r, c_1), (r, c_2), \ldots, (r, c_j)$.

• In Subcase (l), as shown in Figure 3.4(l), if $A < 1$, flip $\Gamma_j, \ldots, \Gamma_2, \Gamma_1$, first vertically, and then horizontally, so that their roots get placed at their lower-right corners. Then, first rotate $\Gamma_D$ by $90^\circ$, and then flip it vertically. Next, place $\Gamma_D, \Gamma_j, \ldots, \Gamma_2, \Gamma_1$ one above the other, in this order, with unit vertical separation, such that their left boundaries are aligned, next move node $e$ (which is the link node of $T_A$) to the right until it is either to the right of, or aligned with the rightmost boundary among $\Gamma_j, \ldots, \Gamma_2, \Gamma_1$ (since $\Gamma_A$ is a feasible drawing, by Property 2,
as given in Section 3.2, moving \( e \) will not create any edge-crossings), and then place \( u^* \) in the same horizontal channel as \( c_1 \) and one unit to the right of \( e \). If \( A \geq 1 \), first rotate \( \Gamma_A \) by \( 90^\circ \), and then flip it vertically. Then flip \( \Gamma_1, \Gamma_2, \ldots, \Gamma_j \) vertically. Then, place \( \Gamma_A, u^*, \Gamma_1, \Gamma_2, \ldots, \Gamma_j \) left-to-right in this order, separated by unit horizontal distances, such that the bottom boundaries of \( \Gamma_1, \ldots, \Gamma_{j-1} \), are aligned, and are at unit vertical distance above the bottom boundary of \( \Gamma_j \). Move \( \Gamma_j \) down until its bottom boundary is either aligned with or below the bottom boundary of \( \Gamma_A \). Also, \( u^* \) is in the same horizontal channel with \( c_j \). Draw edges \((e, u), (u, c_1), (u, c_2), \ldots, (u, c_j)\).

- In Subcase (m), as shown in Figure 3.4(m), if \( A < 1 \), first flip \( \Gamma_j, \ldots, \Gamma_2, \Gamma_1 \), vertically, then place \( \Gamma_A, \Gamma_j, \ldots, \Gamma_2, \Gamma_1 \) one above the other, in this order, with unit vertical separation, such that the left boundaries of \( \Gamma_j, \ldots, \Gamma_2, \Gamma_1 \) are aligned, and the left boundary of \( \Gamma_A \) is at unit horizontal distance to the left of the left boundary of \( \Gamma_1 \). Place \( u^* \) in the same vertical channel with \( r \) and in the same horizontal channel with \( c_1 \). If \( A \geq 1 \), then first flip \( \Gamma_1, \Gamma_2, \ldots, \Gamma_j \) vertically, next place \( \Gamma_A, \Gamma_1, \Gamma_2, \ldots, \Gamma_j \) in a left-to-right order at unit horizontal distance, such that the top boundaries \( \Gamma_A, \Gamma_1, \ldots, \Gamma_{j-1} \) are aligned, and the bottom boundary of \( \Gamma_j \) is one unit below the bottom boundary of the drawing among \( \Gamma_A, \Gamma_1, \ldots, \Gamma_{j-1} \) with greater height. Then, place \( u^* \) in the same vertical channel as \( r \) and in the same horizontal channel as \( c_j \). Draw edges \((r, u), (u, c_1), (u, c_2), \ldots, (u, c_j)\). Note that, since \( \Gamma_A \) is a feasible drawing, by Property 3 (see Section 3.2), drawing \((u^*, r)\) will not create any edge-crossings.

### 3.4 Experimental Results

We implemented the algorithm in C++. The implementation consists of about 5,000 lines of code. We have also experimentally evaluated the algorithm on randomly-generated general trees,
Each randomly-generated general tree $T_n$ with $n$ nodes was generated by generating a sequence $T_0, T_1, \ldots, T_n$ of general trees, where $T_0$ is the empty tree, and $T_i$ was generated from $T_{i-1}$ by inserting a new leaf $v_i$ into it. To prevent a subtree from having a greater degree than $n^{1/2}$, each $T_i$ has a maximum degree $d_{\text{max}}$, where $d_{\text{max}}$ is any integer value between $[1, n^{1/2} - 1]$. The position where $v_i$ is inserted in $T_{i-1}$ is determined by traversing a path $p = u_0 u_1 \ldots u_m$ of $T_{i-1}$, where $u_0$ is the root of $T_{i-1}$, and $u_m$ has at most $d_{\text{max}} - 1$ children. More precisely, we start at the root $u_0$, and in the general step, assuming that we have already traversed the subpath $u_0 u_1 \ldots u_{j-1}$, we randomly select a number $r$ such that $0 \leq r \leq \text{degree}(u_{j-1})$. If a child $c$ existed at position $r$ in $u_{j-1}$, then we set $u_j = c$, and move to $u_j$, otherwise we make $v_i$ the $r^{th}$ child of $u_{j-1}$, set the $d_{\text{max}}$ of $v_i$ to a randomly-generated integer between $[1, n^{1/2}]$, and stop.

Recall that the algorithm takes three values as input: a general tree $T$ with $n$ nodes, a number $\varepsilon$, where $0 < \varepsilon < 1$, and a number $A$ in the range $[n^{-\varepsilon}, n^\varepsilon]$.

The performance criteria we have used to evaluate the algorithm is the ratio $c$ of the area of the drawing constructed of a tree $T$, and the number of nodes in $T$.

To evaluate the algorithm, we varied $n$ up to 50,000. For each $n$, we used five different values for $\varepsilon$, namely, 0.1, 0.25, 0.5, 0.75, and 0.9. For each $(n, \varepsilon)$ pair, we used 20 different values of $A$ uniformly distributed in the range $[1, n^\varepsilon]$. The performance of the algorithm is symmetrical for $A < 1$ and $A > 1$. Hence, we varied $A$ only from 1 through $n^\varepsilon$, not from $n^{-\varepsilon}$ through $n^\varepsilon$ (the only difference between $A < 1$ and $A > 1$ is that for $A < 1$ the algorithm constructs drawings with longer height than width, whereas for $A > 1$, it constructs drawing with longer width than height). Hence, in the rest of this section, we will assume that $A \geq 1$. For each tree, and for each triplet $(n, A, \varepsilon)$, we generated five trees. We constructed a drawing of each tree using the algorithm, and computed the value of $c$. 
Next, we averaged the values of \( c \) obtained for the five trees to get a single value for each triplet \((n, A, \varepsilon)\) for each tree.

Our experiments show that the value of \( c \) is generally small, and is at the most 23, but generally is between 7 and 14. Figure 3.7 shows how \( c \) varies with \( n, A, \) and \( \varepsilon \). We discovered that \( c \) is constant for all \( n \) when \( \varepsilon \) is 0.1 and for \( n \) equal to 30,000 or 40,000 when \( \varepsilon \) is 0.25. We also discovered that \( c \) increases with \( A \) for all \( n \) and \( \varepsilon = 0.5 \), \( c \) starts at a low point for the first distribution of \( A \), then increases to a high point for the second distribution, then levels off for the rest of the distribution for all \( n \) and \( \varepsilon = 0.75 \), and \( c \) decreases for the first values of \( A \), then levels off for a given \( n \) when \( \varepsilon = 0.9 \).

Consequently, for a given \( n \) and \( \varepsilon \) equal to 0.9, 0.75, 0.5, or 0.25, the range for \( c \) over all the values of \( A \) is small. For example, for \( n = 10,000 \) and \( \varepsilon = 0.9 \), the range for \( c \) is \([6.6, 8.2]\). Also, for every \( n \) and \( \varepsilon \) equal to 0.1, the values of \( A \) are constant.

Finally, we would like to comment that the aspect ratio of the drawing constructed is, in general, different from the input aspect ratio \( A \). We computed the ratio \( r \) of the aspect ratio of the drawing constructed by the algorithm and input aspect ratio \( A \). We discovered that \( r \) is close to 1 for \( A=1 \), and generally decreases as we increase \( A \). However, we also discovered that for a large range of values for \( A \), \( r \) is far from 1. Even in applications that require the drawing to have exactly the same aspect ratio as \( A \), we can obtain a drawing with small area and aspect ratio exactly equal to \( A \) by adding "white space" to the drawing constructed by our drawing algorithm.
Figure 3.7: Performance of the algorithm, as given by the value of $c$, for drawing a randomly-generated general tree $T$ with different values of $A$ and $\varepsilon$, where $c = \text{area of drawing/number of nodes } n \text{ in } T$: (a) $\varepsilon = 0.9$, (c) $\varepsilon = 0.75$, (e) $\varepsilon = 0.5$, (g) $\varepsilon = 0.25$, and (i) $\varepsilon = 0.1$. Figures (b), (d), (f), and (h) contain the projections on the $XZ$-plane of the plots shown in Figures (a), (c), (e), and (g), respectively, and show for each $\varepsilon$, the ranges of the values of $c$ for different values of $A$ for each $n$. The $XZ$-plane for $\varepsilon = 0.1$ is not shown because for nearly every value of $n$, $A$ is constant.
4. INTERACTIVE VISUALIZATION OF INFORMATION HIERARCHIES

4.1 Introduction

Information hierarchies are commonly used in a variety of areas such as file systems, hierarchies of object-oriented programs, social networks, bibliographies, and structures of the World Wide Web (WWW). Since each node in an information hierarchy has a certain level of detail, it is important for the user to interact with the information and be able to investigate further inside the hierarchy, while at the same time be guaranteed that the information is up to date. In some cases a node may have too much information to display within the visualization of the hierarchy, thus a separate detail browsing window to display information may be required.

Extensive effort has been put in developing methods to visually represent information hierarchies. Treemaps [42], Cone Trees [63], Hyperbolic Tree [46], 3D Hyperbolic Space [53], Information Pyramids [3], SpaceTree [56], and Zoomology [39] are some of the methods specifically designed to represent information hierarchies.

The rest of the chapter is organized as follows. In Section 4.2, we present the techniques of our separation-based information hierarchy visualization system. In Section 4.3, we present the techniques of our rings-based information hierarchy visualization system.

The work of Section 4.2 originally appeared in [2] and the work of Section 4.3 originally appeared in [68]. The work of this chapter was performed in collaboration with Radu Jianu.
4.2 Separation-Based Visualization System

The interface of separation-based visualization system is presented in Figure 4.1. Each node in the visualization represents one entity of the information hierarchy. A node's children are nodes radiated outward, and connected by edges. A node with no deriving edges is called a leaf node and is considered to contain no further information. The user can interact with the information within the visualization, as described in Section 4.2.2.

4.2.1 Separation-Based Drawing Algorithm

We use the drawing algorithm presented in Chapter 3 as the engine to our separation-based visualization system. The algorithm uses a divide and conquer methodology, in which it recursively separates the tree, draws the subtrees, and arranges the partial drawings to create the complete tree drawing, hence we call it Separation. Separation draws trees with optimal linear area and arbitrary aspect ratio, therefore it makes efficient use of space. In addition, Separation achieves the subtree separation property, which is an important aesthetic in generating interactive visualizations.
4.2.2 Interaction Strategy

Each node consists of a label that is used as the identifier in the visualization (i.e. Web address of Web page or name of file). Initially, no labels appear in the visualization, but the user has the ability to select specific nodes for labeling. The user places label to investigated further into the information hierarchy without overwhelming the user’s cognitive load. The user reveals a label of a node by clicking it, and hides a label by re-clicking it. Upon clicking the node, the label appears above the node it identifies. An example of labeled nodes are shown in Figure 4.2. Also, when the user hovers over a node, its labels appears in the text area at the top of the interface.

It is important when labeling nodes not to occlude other information. We have devised a method to automatically redraw portions of the tree affected by label placements. When labeling nodes, we use a fixed height and width font size to easily calculate the pixel dimension of each label. Once
a label is shown, the affected portions of the tree are re-spaced horizontally by the amount of the label’s width, and vertically by the amount of the label’s height, thus assuring no other nodes are occluded. Moreover, the subtree separation property of Separation guarantees the re-spacing will not create edge crossings.

The user has the capability to re-scale the size of the visualization to get a better view of the information hierarchy, and closely investigate different portions of the information hierarchy. The user re-scales the visualization using the Zoom In and Zoom Out buttons located at the top of the interface (See Figure 4.1). The zooming functionality affects the size of the nodes, edges, and labels. An example of the zooming functionality is shown in Figure 4.3.
Figure 4.4: Screen shot of entire rings-based visualization system: (a) Detail browsing window, (b) Visualization window.

4.3 Rings-Based Visualization System

The interface of the rings-based visualization system is presented in Figure 4.4. This interface is composed by two separate windows: detail browsing window, located in the left part, and visualization window, located in the right part. Each circle in the visualization window represents one node of the information hierarchy. A node’s children are located inside of itself. In the detail browsing window the information contained in the root node of the hierarchy is presented in detail. The user can interact with information either from the detail browsing window or from the visualization window, as described in Section 4.3.5.
Radial graph visualizations locate the focus node at the center of the layout and nodes connected to the focus node radiate outward on uniformly separated rings [26]. We adapted the radial layout rings-based tree drawing algorithm of [74] to use as the engine of rings-based visualization system.

In the original algorithm a tree is drawn as a circle with the root placed in the center, hence we call it *Rings*. The subtrees rooted at the children of the root are drawn recursively as circles placed in concentric rings around the center of the circle to ensure efficient use of space.

In the original *Rings* algorithm, the children of the root are divided into N categories according to their size. One ring is assigned to each category, so the outer rings consists of the “largest” trees while the inner circles consists of the “smallest” ones. This way a tree containing more information is allocated more space. Because of this ordering the algorithm needs to know the entire tree before it can start generating the drawing (bottom-up).

In order to increase the speed of displaying the information and hence to meet the constraints of a real-time system, we modified *Rings* by not organizing the subtrees based on their sizes, thus allowing the algorithm to start drawing the tree much sooner (top-down). The tradeoff is that our visualization is slightly less efficient than original version of *Rings* in terms of use of space, but allows for a much faster generation of the drawing. We call our adaptation of the original *Rings* algorithm, *FastRings*. Another modification to the original *Rings* was in the arrangement of nodes inside the tree. Since the size of a tree is not considered, all the nodes of the tree are considered to be equivalent. In our visualization, the best method of arranging the nodes is to draw them as close to the same size as possible for every depth in the tree, hence making the nodes appear visually equal.

Our layout is different than 2-D Cone trees [63], since, in our case, child nodes are completely contained by their parent node, and in 2-D Cone trees they are arranged outside, thus requiring
more space. A new arrangement algorithm was developed to capture the requirements of FastRings.

As mentioned above, Rings places circles corresponding to children in concentric rings around the center of the parent node [74], as shown in Figure 4.5. Connecting the centers of \( n \) equal circles placed in a ring makes an \( n \)-sided regular polygon. In Figure 4.5, \( \theta = \pi / n \), where \( n \) is the number of circles in a ring, and \( \theta \) is in radians. A simple relationship can be derived between the number of children circles in the outermost ring and the percentage of area taken up by the ring. Next, \( f(n) \), the fraction of the area left after \( n \) circles have been placed in the ring is given by [74]:

\[
f(n) = \frac{(R_2)^2}{(R_1)^2} = \frac{(1 - \sin(\theta))^2}{(1 + \sin(\theta))^2} = \frac{(1 - \sin(\frac{\pi}{n}))^2}{(1 + \sin(\frac{\pi}{n}))^2}
\]

Using the relationship in Equation (4.1), the number of children to be placed in each concentric ring can be determined. The new arrangement algorithm starts by calculating \( N \), the total number of circles to be placed in the entire circle (all ring levels). Next, the algorithm finds the value \( k \), the number of nodes to be placed in the outermost ring. We find the value \( k \) by calculating which value of \( f(k) \), where \( k \) is an integer between \( \{1, 2, \ldots, N\} \), has the smallest fractional difference to percentage of children used. For example, we want the percentage of space used in the ring to
be as close as possible to the percentage of the total number of children used in that space (i.e. \( k/N \approx f(k) \)). Therefore, \( k \) circles are placed in the outermost ring. The rest of the children are placed in the same way in the inner rings. We give pseudocode for determining \( k \):

**Algorithm** Find \( k \)

*Input:* The total number of nodes in the tree, \( N \).

*Output:* The number of circles to place in the outermost ring, \( k \).

\[
\text{minDifference} = \text{INFINITY};
\]

//Find the arrangement that results in the percent of nodes and area left after placement being the closest.

for each number of node \( i \) in \( N \) {
    areaLeft = \( f(i) \);
    nodesLeft = 1 - (i/N);
    if ( areaLeft < nodesLeft ) then
        difference = nodesLeft - areaLeft;
    else
        difference = areaLeft - nodesLeft;
    if ( difference < minDifference ) {
        minDifference = difference ;
        kToReturn = i;
    }
}
//Do not want one node left, so add to previous

if \((N-kToReturn) = 1\) then return \((kToReturn+1)\);

return \(kToReturn\);

end Algorithm.

Only one level in advance is needed to produce a first drawing of a tree. This drawing can be refined later by filling up the circles drawn in the previous step once new information becomes available. This new information allows the user to "browse into the future" because these detail pages, which appear deeper in the hierarchy, can be evaluated without having to load them in the detail browsing window. FastRings constructs a straight-line grid drawing, using a small amount of space with optimal aspect ratio. In addition, there are no edge crossings inside child nodes, the only crossings are with the edges that connect a parent to its children.

Our experiments show that FastRings increases the speed of constructing entire drawings by 51%, and is 12 times faster in producing first drawings in comparison to Rings.

4.3.2 Navigation Strategy

We provide the user with the ability to navigate the information hierarchy through the visualization. We developed a novel navigation strategy to transition the focus of the visualization. Initially, the focus is the tree with the root placed in the center of the main circle since that tree has the largest drawing area allocated. Some illustrative screen shots of how this animation is implemented in rings-based visualization system are presented in Figure 4.6. The procedures of our navigation strategy are:
Figure 4.6: Procedures of the navigation strategy.

- **Step 1 - Node Selection:** The user selects a node for focus change. The selected node is shaded gray (see Figure 4.6(a)).

- **Step 2 - Move Selected Node to Outermost Ring:** The closest positioned (in terms of distance) node in the next outer ring (next radial layout layer, Figure 4.6 shows three rings) is found and its position is swapped with the selected node. This is repeated until the selected node is on the outermost ring.

- **Step 3 - Scale Up, Scale Down:** The selected node is extracted from the parent node and magnified (scaled up) to the size of the parent. At the same time, the size of the parent is reduced (scaled down) to the size that will appropriately fit inside the selected node as a child. In addition, the link between the two nodes is thickened to display the history trail (see Figure 4.6(b)).
• **Step 4 - Make Space for Parent:** After the selected node and its parent are scaled to their suitable sizes (see Figure 4.6(c)), space is made available inside the selected node to place the parent node. If space is available inside the selected node, the parent is moved to that location. If space does not exist, the nodes inside the innermost ring are scaled to a specific smaller size and bunched together in a fashion that allows space for one more node inside the ring. After space is made available inside the ring, the closest node inside the next outer ring is moved to the newly created space. This is repeated until space for the parent node is made available inside the outermost ring (see Figure 4.6(d)).

• **Step 5 - Place Parent in Newly Allocated Space:** Now that space is available for the parent node, we calculate the position of the free space. Next, we move the parent node to the free space (see Figure 4.6(e)).

• **Step 6 - Move Selected Node To Center:** The transition within the visualization is complete. Finally, the selected node is moved to the center of the visualization window, and it becomes the new focus. The data retrieval starts at the new child nodes that are in focus and the user can continue browsing and navigating.

In support of the navigational functionality, we color-code the parent node during the animation, in order to avoid the user loosing familiarity with the visualization during the navigation process. We have also employed a two-group gray-shading method to distinguish chronological browsing order. The most recent visits appear as black thickened edges, and older visits appear as gray thickened edges. Also, vectored edges are used to reveal the direction in which the user has browsed from one node to another.
4.3.3 Labeling Method

A labeling technique is used to identify each children of the root node. Each child node in the visualization has a certain identifying string (i.e. Web address of Web page or name of file), which can be found in the detail browsing window. A fixed height and width font is used to easily calculate the pixel dimension of the labels. The width of each label is restricted to a maximum width, in our case we choose three times the radius of the children. If the label is too lengthy, then it is shortened, and an etcetera is appended to the end, denoting that the label was shortened. Once the dimensions for each label are calculated, by default the labels are placed in the center "slot" of the upper half of each node. An example of a node’s label slots is shown in Figure 4.7.

Sometimes labels overlap in the default locations; therefore an algorithm was developed to avoid overlaps. The algorithm loops through every child of the root and checks for label overlapping with all the other children that appear within a specific distance apart. This distance is set up using the label maximum width. In our case, only children that are within four times the radius of the node are compared. If overlap exists, we iterate through the available labels slots for that node and choose the first slot that does not overlap. We give pseudocode for the overlapping avoidance algorithm:
Algorithm Overlapping avoidance label placement

Input: The total number of children, N.

Output: The non-overlapping positions of the labels.

Calculate the different label slots for each node;

Place all labels at their default slots;

for each child $i$ in $N$ {

    for each child $j$ in $N$ with processed label {

        if ( child($i$) is within $maxWidth+radius(child(j))$ of child($j$) ) {

            if ( position of child($i$) and child($j$) labels intersect ) {

                Place child($i$) label at next available slot

                Mark child($i$) as processed

            }

        }

    }

}
shortened to less than 25% of their length, they are temporarily removed.

Another way we advertise information about a certain node is through the labels in a text area located at the top of the visualization window (see Figure 4.4). When the user hovers over a node, detailed information (i.e. Web address, Web link name, description, and keywords) about the node appears in this text area.

4.3.4 Real-Time

With \textit{FastRings}, only one level of the tree is needed in order to generate a first representation, which is continuously refined while the user is reading the information in the detail browsing window. Our system uses a small amount of graphics, therefore rendering the information to the screen does not slow down our real-time system.

4.3.5 Synchronization

In order to decrease the user's cognitive load, we integrate the browsing and visualization together synchronized in the same interface. When a link in the detail browsing window is hovered over, the location of the link in the visualization window is advertised to the user. Detail pages are also loaded in the detail browsing window when selected from the visualization window.

The user can also interact with the information in the visualization window by selecting links in the detail browsing window. This way, a node selected via its corresponding link in the detail browsing window is transitioned to focus as described in Section 4.3.2. After the refocusing is complete, the detail page that the new focus node represents is loaded by our detail browsing window. This creates a complete two-way synchronization between the detail browsing window and the visualization window, which is unique to our rings-based visualization system.
5. SURVEY OF WEB VISUALIZATIONS

5.1 Introduction

A wide-range of approaches have been developed to visually represent Web data, in an attempt to solve the "lost in cyberspace" problem. We have reviewed technical aspects and established some drawbacks of other approaches. The result of these investigations is found below.

5.2 Drawbacks of Other Approaches

*Pad++* [8] (See Figure 5.1) is a zoomable Web browser that visualizes the structure of the WWW. The nodes of the tree-based visualization are thumbnails of the Web page. Pad++ institutes focus+context by allowing other Web pages to be in view while there is a specific Web page zoomed in as the focus. Pad++ lacks the ability to show which Web pages have already been visited and Web pages that will arise in the future. In addition, Pad++ does not make efficient usage of the screen space.

*Hy+* [35] (See Figure 5.2) is a method used to visualize the portion of the WWW explored during a browsing session. Hy+ uses a graph to represent the relationships between Web pages. Hy+ does not make efficient usage of the screen space. Another drawback is when a user clicks the "Back" and "Forward" button in the Web browser, the edge in the visualization representing this action is omitted. Omitting this action fails to answer the "where have I been?" question.
Navigational View Builder [52] (See Figure 5.3) is a tool which allows the user to interactively create useful visualizations of the information space. It uses binding, clustering, filtering, and hierarchization to form effective views of the WWW. Navigational View Builder uses a database-oriented hypermedia system, which over time becomes out-of-date. Also, it does not make efficient use of space.

HyperSpace [81] (See Figure 5.4) is a prototype WWW visualizer that can display the organization of areas of the Web. HyperSpace structures the information not according to geographical location, but according to user-defined structure. Each Web page is represented as a sphere, and the links from one page to another are represented as links between the spheres. The spheres are
located in a 3D virtual reality environment system. HyperSpace uses an adapted browser and separate program to extract links from visited pages. Other drawbacks of HyperSpace are that the links and sphere nodes are heavily occluded, browsing history is not tracked, and the system is not synchronized with a Web browser.
Natto [71] (See Figure 5.5) visualizes a number Web pages in a graph structure. Natto's initial node/link graph is distributed on a flat horizontal surface. The placement of the nodes is dictated by attributes of the Web page (e.g. its size, title, number of images) which are mapped to the two axes of the space. According to the technical survey by Benford et al. [9], Natto has two shortcomings: limits numbers of nodes that may comfortably occupy the flat plane (occlusion issue) and the range of pages is fixed.

Ptolomaeus [23] (See Figure 5.6) helps users deal with the complexity of Web sites with visualizing Web maps. Ptolomaeus representation of the WWW looks similar to a tree structure, but is graph-based. The user has the ability to assign how many Web pages and levels in the WWW appear in the visualization. Ptolomaeus shows only the Web pages that appear in the visualization after the Web crawler completes the Web page retrieval process. Also, another drawback of Ptolomaeus is in its inefficient use of space.

MAPA [25] (See Figure 5.7) extracts a hierarchical structure from an arbitrary Website, with some minimal user assistance and creates an interactive map of that Website that can be used for

Figure 5.4: Screenshot of HyperSpace.
Figure 5.5: Screenshot of Natto.

Figure 5.6: Screenshot of Ptolomaeus.
orientation and navigation MAPA uses cards to represent Web pages. The cards are spatially arranged and color is used to represent different levels in the hierarchy. The cards representing Web pages deriving out of a "parent" Web page are located in a single file line behind the parent card. MAPA uses labels and cards to represent the WWW and the information quickly becomes occluded. Also, MAPA is not dually synchronized with a Web browser (does not affect visualization) and all the information is stored in a database, therefore is limited.

Disk Trees [15] (See Figure 5.8) is used to represent a discrete time slice of the Web ecology. Disk Trees uses a circular layout and each successive circle denotes levels in the tree. The layout algorithm makes two passes through the tree to guarantee enough space is allocated per node. Disk Trees uses many overlaying linking edges that occlude information. Another drawback of Disk Trees is that it is a bottom-up algorithm; the whole tree needs to be processed before displaying to the user.
VISVIP [19] (See Figure 5.9) visualizes the hierarchical structure of a Website and the paths the users take through that Website. Pages are represented by boxes, and the connections between pages are directed edges. Color is used to identify different types of information. VISVIP uses dotted columns over top of each box to represent how much time the user spent at a specific page. VISVIP makes poor use of space, contains many edge crossings, and label boxes occlude information.

Dome Trees [14] is similar to Disk Trees, but uses only 3/4 of the disk and the disk is extruded along the Z dimension, hence the shortcoming of being a bottom-up algorithm is still an issue.

BrowsingGraph/BrowsingIcons [48] (See Figure 5.10) provides nine differently motivated contexts of Web pages. Each supports orientation and navigation according to the context. The user is provided information about already visited Web pages, related Web pages, and defines "neighborhoods" of Web pages in terms of hyperlinks. Moreover, the system has the ability to interact with the Web browser. BrowsingGraph/BrowsingIcons uses a Web browser that is not completely integrated within the system. The algorithm used to draw the graph is not space-efficient; much
whitespace in the drawing area is unused.

XML3D [62] (See Figure 5.11) uses a 3D hyperbolic space to represent the WWW. XML3D is designed in a typical graph fashion; nodes represent Web pages and the links between them depict their relations. Within the interface of XML3D there are text lists of additional information about the visualization. The lists are selectable and contain information such as search history, parents, children, and siblings of the focused node. XML3D contains node/label occlusion and the distant features within the 3D space are distorted. Furthermore, long connecting edges in a graph are harder to follow than shorter edges [22]. Hence, another drawback of XML3D is its use of lengthy intersecting connecting edges.

**Figure 5.9:** Screenshot of VISVIP.
Figure 5.10: Screenshot of BrowsingGraph/BrowsingIcons.

Figure 5.11: Screenshot of XML3D.
HotSauce [34] (See Figure 5.12) is fly-through interface for navigating the WWW. HotSauce organizes the information spatially in order to improve the regular Internet users navigational abilities. Web pages are represented by rectangular blocks, which are labeled with the Web page's title. It uses color and depth within the 3D layout to depict different viewing levels into the WWW. HotSauce supports the selection of labels, and uses a smooth zooming mechanism to refocus the visualization. According to the Atlas of Cyberspace [24], drawbacks of HotSauce are its difficulties in finding pages, and once immersed in the space and surrounded by blocks, it is easy to become disoriented. Another drawback with HotSauce is the frequent occlusion of labels.

MemoSpace [79] (See Figure 5.13) is a 3D visualization of browsing histories that can be interactively explored in a separate navigation window besides the Web browser. MemoSpace uses color to mark Web pages that have been already visited. Users of MemoSpace can assign labels containing the Web page's address to specific nodes. Complementary to the 3D functionality, MemoSpace
has 2D capabilities as well. A 2D tree-like visualization can be selected, when desired by user, to emphasize the order in which Web pages appear in the WWW. MemoSpace does not make efficient usage of the screen space, and labels denoting a Web page’s address are large in size and occlusive.

*Grokker [33]* (See Figure 5.14), developed by Groxis Inc., is a web-based tool used to visualize Web data. Grokker allows user to enter federated searches and organizes the results in two ways: outline view and map view. The map view uses a radial layout algorithm similar to the one used in our method. One key difference resides in how the Web data is organized; Grokker organizes based on content relationships, our approach creates a hierarchy of Web pages based on their location in the WWW. Another difference is Grokker visualizes a broad range of pages stemming from the user’s query, in contrast, our method visualizes a particular area in the WWW starting from a user-specified Web page.

*WebTracer [54]* (See Figure 5.15) uses two separate programs the ‘Spider’ and the ‘Visualizer’. The Spider visits Websites and produces map files containing information about their internal hypertext links. The Visualizer allows these maps to be viewed in 3D and represents the pages and
links as an abstract molecular structure. WebTracer uses a system in which Web crawling and visualization are separate processes. Other drawbacks are: the user can click on an atom (Web page) and the Web page appears in the computer’s default Web browser, it does not make efficient usage of the screen space, and it contains many edge intersections, which makes it harder to understand the Web pages’ relationships. A comparison between our solution and WebTracer was performed. Both our solution and WebTracer were used on the same computer, using the same Internet speed, and starting from the same Web page. The results of our solution compared to WebTracer were as follows:

- **Computer Memory (RAM)** - 27% more efficient.

- **Computer processing (CPU)** - 50% more efficient.

- **Web crawling speed** - 63% faster.

The reason why none of the following systems have been accepted as the solution to the "lost in cyberspace" problem is because cyberspace is not easily measurable in space-time geometry [24].
The majority of these systems are not real-time, therefore can not deal with the mutability and dynamism of the WWW. Effective visualizations, especially for navigation and search, must cope with such changes, automatically, and if they are to be used as modes of navigation and query, need to be interactive, so that territory and map become one [24].
6. REAL-TIME SPACE-EFFICIENT SYNCHRONIZED TREE-BASED WEB VISUALIZATION AND DESIGN

6.1 Introduction

The World Wide Web (WWW) today has become an enormous source of information and more and more users have access to a steadily increasing number of Web pages, generally linked in a non-intuitive manner. Consequently, repeatedly reported problems in WWW navigation are not knowing where you are, not knowing how to get back to previously visited information, and not knowing which sites have already been visited [27, 79]. The problem of users’ disorientation in the WWW, which emerges from the high complexity of the WWW environment, is often referred to as the "lost in cyberspace" problem. A regular Web browser's back and forward functionality is not a sufficient solution to this problem. A map (visualization) reduces the user's cognitive load because it abates the load on human long term and working memory, summarizing the information about the structure and organization that would otherwise have to be remembered [1, 12, 21, 38, 82]. Therefore, extensive effort has been put in developing methods to visually represent Web data. HyperSpace [81], Pad++ [8], HotSauce [34], Natto [71], Ptolomaeus [23], MAPA [25], Disk Trees [15], VISVIP [19], WebTracer [54], Dome Trees [14], XML3D [62], and MemoSpace [79] are some of the methods specifically designed to represent Web data, in an effort to improve navigation through the WWW, reduce orientation problems within the WWW, and increase the ease and speed of exploring and
retrieving pages of interest. Other methods such as Treemaps [42], Hyperbolic Tree [46], 3D Hyperbolic Space [53], Navigational View Builder [52], Hy+ [35], Space Tree [56], and MoireGraphs [41] which were initially designed to visualize hierarchical data, have been adapted by simply applying them to Web data. However, very few of these methods have been adopted and are currently being used as viable solutions to the "lost in cyberspace" problem.

In this chapter, we present a novel Web visualization method called Separation WebVis as a viable solution to the "lost in cyberspace" problem. with the following innovative combination of features:

- **Tree-based engine**: Humans perceive relational information easier if it is modeled as a graph [22]. The structure of the World Wide Web can be modeled as a graph: the nodes are HTML pages, and a hyperlink from one page to another is represented as a directed edge. However, trees have much simpler structures than graphs, which makes it faster to generate aesthetically pleasing displays of trees. Similarly to several previous methods such as Pad++, Space Tree, Dome Trees, Disk Trees, and Hyperbolic Tree, our method uses a tree-based visualization engine.

- **Space-efficient**: Providing space-efficient visual representations of Web data is of utmost importance given the large amount of information and the limited space available on a computer monitor [2]. Previous methods which use graph-based engines for visualization do not make good use of the screen space by not filling the available white space with information properly. Providing more information in a visualization (and thus minimizing the white space) is recommended, as the human brain is capable of filtering the information [76]. Our method uses a tree-based rings engine to generate the visualization, which maximizes the data-ink ratio and uses smaller area than previous systems by having high data density, thus allowing
the user to either see more information in the same area, or to display the same amount of information in a smaller portion of the screen.

We initially focused on *Separation WebVis* in our research of the "lost in cyberspace" problem. After developing the prototype of *Separation WebVis*, we decided to concentrate on the real-time aspect of solving the "lost in cyberspace" problem. The focal point of this chapter is a novel Web browsing and visualization method, called *Web Global Positioning System (WebGPS)*, as a viable solution to the "lost in cyberspace" problem, with the innovative combination of the two previously mentioned features, as well as the following:

- **Real-time**: According to the study performed by Fetterly et al. [28], 40% of all web pages in their set changed within a week, hence pre-recorded Web data becomes outdated quickly. Our method retrieves and displays Web data in real-time.

- **Synchronization**: Our method integrates Web browsing and visualization together, synchronized in the same interface. Integration and synchronization of the interfaces provides a solution to the cognitive overload discovered in the study of supporting tools used when retrieving information from the WWW [36].

A user study on *WebGPS* confirmed that we found the right combination of features to provide a practical solution to the "lost in cyberspace" problem.

Regular Internet users take less than 1 second to evaluate a Website and decide whether or not to stay and browse [49], hence the quality of a Website’s design is an important factor in how desirable the Website is to regular Internet users. Much research exist in the development of effective Web design practices, systems that analyze regular Internet users’ browsing patterns, and evaluate Website usability [40, 80]. *WebGPS* also helps Website designers to better design and present the
information in their Website, by visually analyzing their design; hence, improving Website "stickiness" (i.e. increasing the noticeability and minimizing the time spent in locating information on their Website.)

The rest of the paper is organized as follows. In Section 6.2, we present the development of Separation WebVis. In Section 6.3, we present WebGPS in detail. We discuss the real-time and synchronization qualities of WebGPS in Section 6.4 and 6.5, respectively. Next, we distinguish the different users of WebGPS in Section 6.6. We present the outcome of our user study performed on WebGPS in Section 6.7. Finally, we present additional features implemented resulting from the user study in Section 6.8.

The work of Sections 6.3 and 6.7 originally appeared in [68]. The work of this chapter was performed in collaboration with Radu Jianu.

6.2 Separation WebVis

Separation WebVis is the application of the techniques described in Section 4.2 on the Web. The concepts of Separation WebVis were established in [2], and the prototype was developed to prove these concepts. WebGPS is divided into two main parts: the crawler and the visualization, which are described in detail in Sections 6.2.1 and 6.2.2, respectively.

6.2.1 Crawler

The Web crawler we use in this application was developed by us and tailored to meet Separation WebVis's requirements. Our Web crawler executes via a separate Java application. The user runs the Web crawler using a configuration file as input. The configuration file sets up the starting Web address to crawl, filename in which to store the output, maximum depth to crawl, and maximum number of links to extract per Web page. The Web crawler generates a formatted file describing the
Figure 6.1: Application example of *Separation WebVis* (old interface).

results of the Web data extraction process. The file represents the hierarchy of Web data retrieved, and is similar to the format described in Section 2.3.1. In order to meet the specifications of general trees used in Chapter 3, we have created a new input file format. Similar to the file format of Section 2.3.1, each line in the file represents a node in the information hierarchy. Each entry in a line is separated by one space. The first entry in a line is either 0, denoting the node has no first child, or is 1, denoting the node has a first child. In our Web application, the first entry is always 1. The second entry in a line corresponds to the Web address of the Web page the node represents. The third entry in a line corresponds to the node’s unique integer key identifier. Each of the remaining entries represent each child of the node, with the entry corresponding to the child’s unique integer key identifier. The general format for each line as follows:

```
1 address nodeKey childKey1 childKey2 ... childKeyN
```

6.2.2 Visualization

We visualize the particular area of interest of the WWW using a tree structure (see Figure 6.1). Each node is the information hierarchy represents a Web page in the WWW. A node’s children are Web pages that can be accessed via links in the Web page for which that node represents. The user
can interact with the visualization, as described in Section 4.2.

Once Separation WebVis is executed, the user must select the Open Tree File button to select an input tree file to visualize. The user is presented a standard file selection dialog, where the user selects a tree file previously created by our Web crawler. Next, the tree file is loaded by Separation WebVis, and the algorithm, DrawGeneralTree, of Section 3.3 is used to create the drawing of the tree. The user interface of the visualization is also created using Java Swing. We use Java2D for all the graphics generated within the visualization. In addition, we use the Java AWT Event Toolkit to handle the user interaction with the visualization.

Separation WebVis's interactive capabilities, which were described in Section 4.2.2, allow the user to investigate how certain Web pages are related to each other, as well as understand and evaluate the design of a Website. For example, Figure 6.1 presents a partial visualization of the Frontiers In Education (FIE) 2005 Conference Website using an older version of Separation WebVis's interface. In this example, the root node, represented by the FIE 2005 homepage, has two children that are the same Web page. Furthermore, it was found that within the FIE 2005 homepage the links for these two Web pages were identified by different titles (Conference Proceedings and Final Program). This error in the Website design was reported, and was later fixed.

6.2.3 Label Method Extension

The amount of screen space available on a computer monitor is limited, hence methods to visually represent information in a small amount of space is important. Sometimes a node's label, represented by a Web address, can be lengthy, thus we have extended the labeling technique of Section 4.2.2 to organize the text of a label in a fashion that shortens the horizontal space the label uses. The algorithm used to shorten the length of the visible labels is specific to systems that use the '/'
character as a delimiter, hence can only be applied to Web and file system information hierarchies. In this section, we focus on shortening labels in Web information hierarchies. Our algorithm is applied to every label in the information hierarchy. The algorithm begins by separating the label into different parts. The first part of every label begins with the "http" prefix, and ends with the Web page’s initial extension (i.e. .com, .edu, .org). The next parts are all the strings derived by delimiting the rest of the label by '/'. After separating all the parts, we find the part with the maximum length, and assign that value as the maximum width of the label. Next, we iterate through every part, and if the concatenation of the current part with the next part is less than or equal to the maximum width value, we concatenate the two strings together, otherwise we accept this part and move to the next part. If the strings were concatenated together, we calculate the length of the string if concatenated with the next part, and if the string is still less than or equal to the maximum width value, we concatenate. This process is continued until we processed all of the label’s parts. For example, using the label http://www.animals.com/birds/white/doves.html, the algorithm would initially break the label into four parts: http://www.animals.com, /birds, /white, and /doves.html. The first part has the largest length, therefore 22 is set as the label’s maximum width. Hence, the final label assignment would contained three parts: http://www.animals.com, /birds/white, and /doves.html. These three parts would be placed on top of each other, which reduces the width of the label from 45 to 22, thus allowing more space for label placement. Examples of shortened labels are shown in Figure 4.2. This algorithm could be adapted for other types of information hierarchies; the algorithm would be contingent on the prospect of delimiters.

6.3 Web Global Positioning System

*WebGPS* has been designed using free floating windows, therefore the user can move each window in a desired location, based on free space available, and minimize, maximize, or close windows.
Also, WebGPS is the application of the techniques described in Section 4.3 on the Web. The browser is used to view Web pages in the WWW, and is located in the left part of Figure 6.2. The visualization of the particular area of interest in the WWW is located in the right part of Figure 6.2. A statistics window is also available for advanced users, and is located in the bottom right window of Figure 6.2. The parameters window offers advanced users the ability to modify WebGPS settings. It can be accessed from the Options menu which is located at the top of the browser, and is shown in Figure 6.3.

We allow the advanced users to modify parameters which affect the Web crawler and the visualization generator. There are three categories of parameters: control parameters, aesthetic parameters, and the bias parameter. The control parameters influence the Web crawler of the system. The aesthetic parameters affect the appearance of the visualization. Lastly, the bias parameter has an effect on what type of Web pages to add to the visualization. The details of each parameter were
discussed in Section 6.3.2.

The statistics that describe WebGPS during run-time are defined as follows:

- **Number of links**: The number of links located in the crawler both processed and not.

- **Number of processed links**: The number of processed links in the crawler.

- **Number of Idle Connections**: The number of established connections in the idle state (see Figure 6.4).

- **Number of Active Connections**: The number of established connections which are active or in the running state (see Figure 6.4).

- **Number of Nodes**: The current total number of nodes available in the visualization.

- **Links per Second**: The rate at which links are added to the crawler for processing.

- **Updating Tree State**: Indicates whether the visualization (tree drawing) is updating with new nodes. Indicator is 'y' for in updating tree state and 'n' for not in the state.

- **Elapsed Time**: The time in seconds since the application was launched.

6.3.1 Browser (Detailed Visualization Window)

Since our focus was on the visualization part, we have created a Web browser with only basic functionality. More functionality could easily be added on top of the existing functionality. The user can enter a Web address in the provided textbox. Upon entering an address, if valid, the page loads and shows as in any regular Web browser. Forward and back buttons are included to allow for traversal through already viewed pages. The user interface of the Web browser is created using the built-in Java Swing Toolkit. The displaying of Web pages is handled by the HTMLDocument class,
and the interaction within the links in the Web page is controlled by the \textit{HTMLFrameHyperlinkEvent} class. Back and forward functionality is controlled by a queuing structure that maintains a record of browsing history.

6.3.2 Crawler

The Web crawler we use in this application was developed by us and tailored to meet \textit{WebGPS}'s requirements. Similarly to SPHINX [51] Web crawler, we use a Java, multi-threaded approach in which Web pages are accessed by different threads. For each page the Web crawler encounters, the description, keywords, link addresses, and their label within the page is extracted. An advanced user can assign the maximum number of threads via \textit{max threads} in the parameter window (see Figure 6.3). The default value of this parameter is 25. We limit the number of threads that can
coexist in the system to not overcharge the processor. Furthermore, the number of Web links extracted from a Web page is controlled by the *children limit* parameter. Link extraction is performed sequentially, and crawling is terminated when the *children limit* parameter is met, or the end of the Web page’s HTML source is reached. To eliminate Web pages that may not be useful, the *page type ban* parameter allows the crawler to neglect specific Web page types. In addition, if the *allow duplicates* parameter is true, the crawler extracts all links found, otherwise the user can manipulate the *duplicate release number* parameter, which allows duplicate Web pages to be extracted, but with one constraint. For example, if the *allow duplicates* parameter is false, the *duplicate release number* parameter is 2, and *Page XYZ* is found at depth 1, then *Page XYZ* can be re-extracted at depth 3 or below. If *Page XYZ* appears again at depth 3, it can not be re-extracted until depth 5. This procedure is recursive as farther depths are encountered.

Under the restriction of a real-time system, the information from the Web crawler needs to be extracted as fast as possible. In order to deal with slow and non-responsive servers, the user can assign a wait time limit through the combination of *response time* and *disconnect round trip* parameters. If the server does not respond, it is placed in a queue with other non-responsive and unprocessed links and served when a thread becomes available. The visualization is updated with the Web crawler’s findings at the rate denoted by the *update time* parameter. Links with slower connections are simply added to the visualization at this rate while the user is analyzing the new information. In Figure 6.4, we show the states and transitions between processing Web pages in our Web crawler.

The performance of our Web crawler is dependent on the characteristics of the computer on which the system is being run, on the Internet connection, and on the domains that are visited. While testing *WebGPS*, the Web crawler’s performance averaged approximately 5 links processed/second.
Figure 6.4: Web crawler state diagram. Here, \( n \) represents the response time and \( k \) represents the number of disconnected round trips allowed.

The results were obtained on a Pentium 4 computer running at 2.80GHz, with 512 MB RAM, and an Internet cable connection of 1Mb/sec.

6.3.3 Visualization

We visualize the particular area of interest of the WWW using a tree structure. The Web page that appears in the browser is represented by the root node in the tree. A node’s children are Web pages that can be accessed via links in the Web page for which that node represents. Hence, we model a WWW boundary as a hierarchical structure. Each circle in the visualization represents one node of the tree, which denotes a particular Web page in the WWW. A node’s children are located inside of itself. We provide the minimum radius parameter to control the size of the smallest nodes rendered in the visualization (See Figure 6.3). The user interface of the visualization is also created using Java Swing. We use Java2D for all the graphics generated within the visualization. In addition, we use the Java AWT Event Toolkit to handle the user interaction with the visualization.

The structure of the World Wide Web can be modeled as a graph: the nodes are HTML documents, and a hyperlink from one document to another is represented as a directed edge. However,
trees have much simpler structures than graphs, which makes it easier to display trees in an aesthetically pleasing manner. Although graphs may be most appropriate to represent the data on the Web, tree-based visualization makes Web surfing, Web searching, Web design, and tracking user navigation patterns simpler, and thus, less intimidating to regular Internet users.

In WebGPS, we model the WWW as a general tree. Websites structured as broad shallow general trees offer many choices at each level. In contrast, Websites structured as narrow deep general trees require many clicks to get to the bottom level. Users prefer broad shallow general trees over narrow ones [49], thus most Websites are designed in this fashion.

Radial graph visualizations, introduced by [22, 26], locate the focus node at the center of the layout and nodes connected to the focus node radiate outward on uniformly separated rings. Further implementations of radial layout are presented in [41, 72, 83]. In Chapter 4, we presented our adaptation of Rings [74] called FastRings. We use FastRings as the engine of WebGPS.

Moreover, balanced binary trees resemble shallow general trees, hence intuitively the performance of an algorithm on balanced binary trees is a good indication of their performance on shallow general trees. As established in Chapter 2, for balanced binary trees, Rings has very good performance on important aesthetics such as area, maximum and uniform edge length, angular resolution, distance of closest and farthest leaves, in comparison with other tree drawing algorithms.

6.4 Real-Time

Several studies have been conducted to try to establish average change rates of different types of Web pages. The results show that more than 40% of Web pages on commercial servers (.com) change on a daily basis [16]. Other domains, such as educational or governmental Websites, are less dynamic but still change over time. Effective visualization, especially for navigation and search, must be able to cope with such changes [24]. Hence, with this evolution, a method is needed to
handle visualizing the constantly changing data in real-time (i.e. not using pre-recorded data).

The majority of previous Web visualization systems concentrate exclusively on the visual part of the problem using pre-recorded or human generated data sets as input. Those systems are, without doubt, valuable in showing how data can be visualized in an efficient way. However, displaying a data structure that is being generated in real-time implies serious constraints that most of the previous systems cannot meet.

WebGPS utilizes the real-time feature of FastRings (See Section 4.3.4), and performs drawing and Web crawling simultaneously, offering partial results to the user as soon as possible. Also, we utilize the time the user spends reading the Web page for performing the Web crawling and tree generation. WebGPS uses a small amount of graphics, therefore rendering the information to the screen does not slow down our real-time system. In addition, FastRings creates a top-down approach that allows Web data to be generated and displayed in real-time. Only one level of the tree is needed in order to generate a first representation. This representation can be continuously refined by adding detail once new information becomes available. The response time and disconnect round trip parameters established in Section 6.3.2, although necessary in order to make the system real-time, may have a negative effect on the accuracy of the data displayed, depending on how fast the user is analyzing the new information. If the user is moving too quickly, some pages might be displayed as having no links deriving from them, while, in fact, they are located on slow servers. WebGPS enables users to adjust these time limits and decide whether they want a slower more accurate system or a faster less accurate one.

6.5 Synchronization

In order to decrease the user’s cognitive load, we integrate the browsing and visualization together synchronized in the same interface. When a link in a Web page is hovered over, if the link
exists in the visualization, then the location of the link is advertised to the user. Web pages are also
loaded in the browser when selected from the visualization.

As established in Section 4.3.5, the user can change focus by selecting a subtree rooted in one
of the children of the main root or by selecting a Web link, which exist in the visualization, via
the browser. If the child was selected via the browser and does not exist in the visualization, the
visualization and crawling will completely restart at this Web page, otherwise that child will be
transitioned to the focus as described in Section 4.3.2. This is performed as a smooth animation in
order to preserve the user's mental map. After the refocusing is complete, the Web page that the new
focus node represents is loaded by our browser. This creates the synchronization characteristic of
WebGPS. The Web browser and visualization object each have access to certain functionality within
each other. In order to maintain synchronization, when a node in the visualization is selected, our
system makes a call to the method that loads a new page in the Web browser with the selected Web
page's address as a parameter. Furthermore, when a link is selected in the Web browser, our system
makes a call to its associating method that changes the focus of the visualization with the link's
address as a parameter. The \texttt{HTMLFrameHyperlinkEvent} allows the system to realize when a link
in the Web browser is hovered, and one property of the event is the address of the link. Hence, we
have designed \textit{WebGPS}, that whenever this event occurs, to find the Web page in the visualization,
and advertise its location to the user.

6.6 Users

The users of \textit{WebGPS} are the regular Internet users and Web designers. We characterize a
regular Internet user as one who uses the WWW as a resource for obtaining information by browsing
different Web pages. We characterize Web designers as those who create Web pages for regular
Internet users. In this section we will explain how \textit{WebGPS} serves these two individual needs.
It is easy for regular Internet users to browse the WWW, traversing from link to link, to become disoriented within the area of the WWW they are viewing. Frequently, the regular Internet user is unaware of how Web pages of interest are related, and what Web pages could appear in the future. Moreover, the regular Internet user is mostly interested in viewing the contents of a Web page. 

*WebGPS* presents the user with a visualization of the area of the WWW they are viewing, which helps the user find interesting Web links faster. The regular Internet users represent the majority users of *WebGPS*.

To the Web designer, *WebGPS* serves a different purpose. The Web designer is interested in knowing how regular Internet user traverses their Web pages. In addition, the Web designer is concerned with how her Web pages are structured. Using *WebGPS*, the Web designer is better equipped in developing a Website that a regular Internet user can follow. For example, the system can be used to illustrate poor Website design, such as Web cycles or duplicate links. Web cycles occur when a link on a particular Web page is directed to itself or its parent. Duplicate links exist when a Web page has more than one link which point to the same Web address. By not allowing duplicate links (checking the *allow duplicates* parameter), Web cycles can be visually detected through the empty pages: an empty page represents either a dead link or a duplicate (hence a cycle). Another major problem while browsing the WWW is dead link encounters [55]. A Web designer could use *WebGPS* to find dead links in seconds. Non-responsive links can be reported to the designer via the Web server responsiveness functionality of the Web crawler. Dead links are Web links which take the regular Internet user to a Web page with no data. This scenario would become evident when a link to a node (Web page) in the visualization existed, but as an empty circle. The Web designer could analyze the placement of Web pages within the visualization, and gain information about which Web pages appear first, and are the most responsive, because they
will appear in the outermost ring. Furthermore, the Web designer can put the most important Web pages in the outermost ring by placing their links in the beginning of the HTML source code. Also, it is important that links within the same Web page are related. Within the visualization, the Web designer can use the advertising of each pages' description to verify that adjacent links are indeed associated.

6.7 User Study

Twenty subjects voluntarily participated in our study to evaluate the effectiveness of WebGPS in exchange for a chance at a monetary prize drawing. We wanted to capture results from two technical levels, therefore we recruited ten non-technical subjects and ten technical subjects. We consider a non-technical subject as a person who has no prior background in software development and technical as those who had background in software development. Both types of subjects were experienced in browsing the WWW. In agreement with our university, we chose to implement the user study with subject anonymity. The only information of the subject recorded was user type and sex. For a general idea, the majority of subjects were undergraduate students, but a few graduate students and non-student subjects existed.

6.7.1 Procedure

Each subject participated in our study on a one-on-one basis with the user study proctor. The user study proctor was one of the authors of this paper. First, the subject was given a half page description of what he/she will do in our study. After reading the description, if the subject had questions then they were discussed, otherwise the proctor explained the "lost in cyberspace" problem, how to use the system, and how we propose our system to be a solution. This explanation lasted approximately 15 minutes. After the explanation and background, the subject took part in two training assignments.
Each training assignment lasted approximately 15 minutes. During the first training assignment, the subject was asked to find specific Web pages in the visualization, select them, and later asked to return to some previously visited location, therefore simulating procedures performed while casually browsing. The tasks of training assignment 1 are detailed in Table 1. The second training assignment was focused on revealing how the Web browser affected the Web visualization. In the second training task the subject was asked to hover over links in a specific Web page within the Web browser to demonstrate how the node that represents the page is advertised in the visualization. In addition, the subject was asked to click specific links, which exposed the subject to the focus change procedure of the visualization, when a link is selected, as well as provide a visual representation of
Two different methods of browsing were used in our study: a regular Web browser and WebGPS. While using our solution, subjects could use the associated Web browser to obtain more detail, but selections were only allowed through the visualization. The scenarios were selected to cover different real-world situations. In Scenario 1, the user navigates through unfamiliar Web pages with unfamiliar structures, containing a moderate amount of information to search. In Scenario 2, the user navigates through unfamiliar Web pages with familiar structures, for some, containing a large amount of information to search. In Scenario 3, the user navigates through familiar Web pages with familiar structures, containing a large amount of information to search. In Scenario 4, the user navigates through unfamiliar Web pages with unfamiliar structures, containing a small amount of information to search. Each scenario was timed, and each subject had a maximum time of 10 minutes to complete each scenario. Each subject used a regular Web browser for two scenarios and our solution for the other two. Also, the starting scenario for each subject switched each time. For example, if Subject $N$ started with scenario 1 using the regular Web browser, our solution would be used for scenario 2, regular Web browser would be used for scenario 3, and our solution would be used for scenario 4, then Subject $N+1$ would start with scenario 2 using the regular Web browser, our solution would be used for scenario 1, regular Web browser would be used for scenario 4, and our solution would be used for scenario 3. The purpose for implementing the user study in this manner is to keep the order of browsing type consistent, and because one subject can not duplicate a scenario using both browsing types; the subject already knows the answer. The starting point and target destination of each scenario are as follows:

- **Scenario 1**: Starting from the International World Wide Web Conference's Call for Papers Web page, find the (ACM) Web page where women are the main topic.
- **Scenario 2**: Starting from the University of Calgary home-page, find the Web page about on-campus student employment opportunities.

- **Scenario 3**: Starting from the Rowan University home-page, find the Web page that contains information about Class Schedules/Registration Schedule.

- **Scenario 4**: Starting from the Transportation Security Administration (TSA) home-page, find the Web page that contains information about bringing baby formula onto aircraft during flight.

The rationale behind choosing scenario 1 and 2 is the affiliation with WWW2007, scenario 3 for its page type similarities to scenario 2, and scenario 4 for its difference from the other scenarios.

When using the regular Web browser, the subject was not allowed to perform Web search queries, nor regular find functionality within a Web page, but allowed to use the back and forward button. The goal was to simulate the process of "casual" browsing.

After a subject completed both training tasks and the four scenarios, they were asked to complete an eight question subjective evaluation sheet. In addition, the evaluation included an area where the subject can make specific remarks such as compliments, complaints, suggestions, and concerns. The subjects were given as much time as they needed to complete the evaluation. The proctor of the user study was responsible for processing and archiving the data.

6.7.2 Timed Results

Since this paper presents our proposed solution to the "lost in cyberspace" problem, it was essential we prove this conjecture. The time results and scenario completion rate for all subjects performing each scenario using our solution and a regular Web browser are presented in Table 6.3.
Table 6.3: Time results and scenario completion rate for all subjects using both WebGPS and a regular Web browser on all four scenarios.

WebGPS was most effective when subjects were browsing through a non-trivial amount of unfamiliar information and structures. In fact, 60% of non-technical subjects and 40% of technical subjects could not complete the task in such a case, when using a regular Web browser. On the other hand, all subjects were able to complete the task using WebGPS. As expected, WebGPS was least effective when subjects were asked to locate readily available information. We also found, when browsing familiar structures, technical subjects’ performance is similar for both WebGPS and regular Web browser. The user study shows that non-technical users locate information faster using WebGPS, when browsing through non-trivial information spaces.

An overall Analysis of Variance (ANOVA) comparing which browsing method was used, which scenario was performed, and which type of subject performed the tasks was implemented on the time results. The case when non-technical subjects used the regular Web browser on Scenario 1 was omitted from the overall ANOVA, because three subjects could not complete the task, therefore only two data points remained for this case. The overall ANOVA proved to reject the null hypothesis that no difference between the means exist; d.f. error = 55, $F = 3.58$, $P < 0.05$. 

<table>
<thead>
<tr>
<th>User Type</th>
<th>Scenario</th>
<th>Time (secs)</th>
<th>Completion %</th>
<th>Time (secs)</th>
<th>Completion %</th>
<th>Improvement %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical</td>
<td>1</td>
<td>118</td>
<td>100%</td>
<td>199</td>
<td>60%</td>
<td>67%</td>
</tr>
<tr>
<td>Non-technical</td>
<td>1</td>
<td>200</td>
<td>100%</td>
<td>372</td>
<td>40%</td>
<td>86%</td>
</tr>
<tr>
<td>Technical</td>
<td>2</td>
<td>290</td>
<td>80%</td>
<td>146</td>
<td>100%</td>
<td>None</td>
</tr>
<tr>
<td>Non-technical</td>
<td>2</td>
<td>221</td>
<td>100%</td>
<td>230</td>
<td>80%</td>
<td>None</td>
</tr>
<tr>
<td>Technical</td>
<td>3</td>
<td>105</td>
<td>100%</td>
<td>83</td>
<td>100%</td>
<td>None</td>
</tr>
<tr>
<td>Non-technical</td>
<td>3</td>
<td>65</td>
<td>100%</td>
<td>114</td>
<td>100%</td>
<td>75%</td>
</tr>
<tr>
<td>Technical</td>
<td>4</td>
<td>305</td>
<td>100%</td>
<td>34</td>
<td>100%</td>
<td>None</td>
</tr>
<tr>
<td>Non-technical</td>
<td>4</td>
<td>235</td>
<td>100%</td>
<td>24</td>
<td>100%</td>
<td>None</td>
</tr>
</tbody>
</table>
6.7.3 Subjective Ratings

After completing all four scenarios, the subjects were asked to fill out an evaluation. The ratings for each question were on a 7-point scale; 1=lowest, 7=highest. With the understanding of the "lost in cyberspace" problem the subjects gained through the explanation by the proctor, the subjects were asked to rate the effectiveness of WebGPS with the capabilities of the regular Web browser in mind. Figure 6.5 shows the subjective ratings for each question.

- **Question 1**: The average rating for the effectiveness of the interactive animation within visualization to provide the subject with a better understanding of "where they are going" was 5.7.

- **Question 2**: The average rating for the effectiveness of the browsing history advertisement to provide the subject with a method of seeing "where they came from" and "how did they get there" was 6.15.
• **Question 3:** The average rating for the effectiveness of the integration and synchronization of the Web browser and visualization was 6.25.

• **Question 4:** Subjects agreed with an average rating of 5.85 that the tree-based structure of representing Web data was easy to follow.

• **Question 5:** Subjects agreed with an average rating of 4.3 that the speed of transitioning to a new Web page (navigation strategy) and the time it took to present new data was desirable.

• **Question 6:** Subjects agreed with an average rating of 5.8 that the use of labeling nodes helped them understand what each node represented.

• **Question 7:** Subjects agreed with an average rating of 5.75 that the labels did not cover nodes in a way that made it difficult to see the Web pages they represented.

• **Question 8:** The average rating of the overall effectiveness of WebGPS in solving the "lost in cyberspace" problem was 5.4.

Overall, the feedback from the subjects affirms that WebGPS was helpful in solving the "lost in cyberspace" problem. The feedback also reveals that the system was effective with minimal training, therefore the transition from understanding what WebGPS provides to actually using it is short. The speed of the transitional animation within the visualization scored moderately, therefore to meet the requests of the subject we plan on speeding up the animation by decreasing the number of frames during the transition.

6.8 New Features

During the user study we noticed the need for certain features. After completing the evaluation questionnaire, the subjects provided feedback in a free-form textbox. We reviewed the feedback
from the subjects and our own observations, and implemented additional features to the system, which are being presented next.

6.8.1 Variable Depth

Initially, our visualization provided only two levels from the main Web page (one level into the future, i.e. the links inside the Web pages available at the main Web page). We increased the scalability of WebGPS by providing the max depth parameter (See Figure 6.3), which the user can use to control the number of levels into the future by halting the crawler at a certain depth. In order to observe the effectiveness of varying this parameter, we generated visualizations with max depth of three (See Figure 6.6) and max depth of four (See Figure 6.7).

A dynamic variation of the depth can also be employed by automatically choosing the max depth based on the number of links and the size of the visualization window. For instance, if there are not too many links and/or the visualization window is large, the depth could automatically be increased, thus maximizing the amount of information being presented. On the other hand, if there are many links and/or the visualization window is small, the depth could automatically be decreased.

6.8.2 Color Coding

The diversity of the information on the WWW can be categorized in many ways by designers. We can capture different categorizations in our visualization by implementing a color coding scheme, in which each category is assigned a different color.

\[
Luminance = (0.30 \times Red) + (0.59 \times Green) + (0.11 \times Blue)
\]  

(6.1)
Figure 6.6: FastRings with max depth of three and 2457 nodes total.

Different colors are chosen based on the Luminance Equation (6.1), taking into consideration the default white background and dark-blue node color. In the Luminance Equation, $Red$, $Green$, and $Blue$ are a particular color’s RGB values, each with values between 0.0 and 1.0 (i.e. Brown equals $Red=0.59$, $Green=0.11$, and $Blue=0.08$.) For good contrast, the difference between the luminance of the background and foreground colors should be greater than 0.40. The higher the difference, the more readable and easier to distinguish the visualization will be. In general, we choose the colors
uniformly distributed between the difference of the default luminance value and 0.40.

For instance, links in a Web page can be either text-based or image-based. Certain users may be interested in distinguishing between these two types of links. Our color coding scheme differentiates the two types of links: we use dark-blue for nodes denoting text-based links and orange, which was chosen based on the Luminance Equation (2), for nodes denoting images-based links (See Figure 6.7: FastRings with max depth of four and 2832 nodes total.)
Figure 6.8: Color coding scheme for two categories: text-based links (dark-blue) and image-based links (orange).

We could also use the color coding scheme to allow the user to distinguish between intra and inter-domain links. Nodes in the same domain as the main Web page are assigned the same color and nodes that link to other domains are assigned other colors using the Luminance Equation.

6.8.3 Empty Pages

One drawback of our choice for visualization engine (FastRings) is that, in case of Web pages with dead links or duplicates, those would use an unjustified amount of space. Hence, we are
developing strategies that would free up space by dismissing or providing less space to these empty pages. One idea we are contemplating is, once empty pages are known, we group them inside a dummy node labeled "Empty Pages". This new node will be distinguished from the other nodes by not connecting it to the root through an edge (See Figure 6.9). We plan on experimenting with such techniques, within the constraints of the real-time aspect of our system. By employing such a strategy, we can provide more space to display existing information in greater detail.
6.8.4 Multi-level Navigation

Initially, the user could only select nodes from our visualization within one level of the current Web page. We implemented a multi-level navigation strategy by generating an automated sequence of the current navigation steps described in Section 4.3.2. This new strategy has three types of stages:

- **Initial visualization**: the node into the future is being selected by the user.

- **Intermediate visualizations**: step by step progress from selecting the node into the future to reaching it.

- **Final visualization**: the node into the future is now the focus, and its corresponding Web page is loaded in the browser.

For instance, in Figure 6.10(a), the user is selecting a node one level into the future. The intermediate visualization, in Figure 6.10(b), displays the child of the root (i.e. the parent of the node into the future) of Figure 6.10(a) as the focus. Finally, the node into the future is brought to focus in Figure 6.10(c).

6.8.5 Visualization Favorites

While browsing the WWW there may exist Web pages to which the users may want to return (favorites). We implemented a favorites strategy for our visualization: the users can save desirable locations for future selection. Once selected, a visualization favorite is automatically navigated to using the technique introduced in Section 6.8.4.
Figure 6.10: An example of multi-level navigation: (a) Initial visualization: a grandchild is being selected, (b) Intermediate visualization: the child of the root (i.e. the parent of the selected grandchild) of (a) is the focus, (c) Final visualization: the grandchild of (a) is the final focus.
Many years of tree drawing research has produced a diverse setting. Experimental studies are important to determine the practicality of tree drawing algorithms in real-life applications. In general, the algorithms under evaluation exhibit various trade-offs with respect to the quality measures analyzed, and, in general, none of them perform the best for all categories. We have obtained that the algorithm for producing planar straight-line grid drawings of degree-\(d\) trees with \(n\) nodes, where \(d = O(n^\delta)\) and \(0 \leq \delta < 1/2\) is a constant, with optimal linear area and with user-defined arbitrary aspect ratio works well in practice. In this thesis we also presented two novel methods to interactively visualize information hierarchies. The visualizations are not only represented in an aesthetically pleasing manner, but also makes efficient use of space. Our rings-based visualization system introduces an innovative combination of features such as real-time synchronization between browsing and visualization. We have applied our interactive visualizing techniques on Web data, and presented two Web visualization systems as solutions to the "lost in cyberspace" problem. Our study of \textit{WebGPS} shows users are able to orient themselves better in cyberspace and locate Web pages of interest faster.

We have also identified the following future work, regarding interactively visualizing information hierarchies:

- Generate other special types of binary trees, such as \(k\)-balanced trees, and extend our study of binary tree drawing algorithms to include algorithms specifically designed for them.
- Implement algorithms specifically designed for AVL and Fibonacci trees, and extend our study of binary tree drawing algorithms to include these algorithms.

- Implement a hybrid tree drawing algorithm and perform an experimental study, then compare its performance against the four tree drawing algorithms described in this thesis.

- Develop different classifications of general trees, and perform a similar experimental study on algorithms that draw general trees.

- Apply our interactive visualization techniques on other data sources (i.e. file systems, bibliographies, or other real-world applications).

- Integrate *Separation WebVis* to be synchronized with a Web browser.

- Add a more effective Web browser within *WebGPS*.

- Experiment *WebGPS* to devices with small screens, such as cell phones and PDAs, and in 3-dimensional and virtual reality realms.
REFERENCES


