1-3-2012

Visualizing graphs with distinguishable edges and ordered binary trees in small area

Andrew Fabian

VISUALIZING GRAPHS WITH DISTINGUISHABLE EDGES AND ORDERED BINARY TREES IN SMALL AREA

by
Andrew Fabian

A Thesis

Submitted to the
Department of Computer Science
College of Liberal Arts and Sciences
In partial fulfillment of the requirement
For the degree of
Master of Science in Computer Science
at
Rowan University
December 2011

Thesis Chair: Adrian Rusu, Ph.D.

Dedication

In loving memory of my parents, Andrew and Robin Fabian.

Acknowledgments

Many thanks to my advisor, Prof. Adrian Rusu, for providing me the opportunity to pursue this research and for all of the support and encouragement in completing this endeavor. I give many more thanks to him for presenting me with several life-changing opportunities including the co-op study with the Federal Aviation Administration that has landed me a position doing exciting and enjoyable work with them.

I thank Radu Jianu for providing me the framework he developed for creating graphs that was invaluable to the research conducted on the edge crossing problem.

For their time, input, and interest in this work, I also thank my committee members Profs. Steve Hartley, Hieu Nguyen, and Jianning Xu. I also thank them for their guidance and knowledge that they have passed on to me during the several years I have been a part of Rowan University.

Abstract

Andrew Fabian
VISUALIZING GRAPHS WITH DISTINGUISHABLE EDGES
AND ORDERED BINARY TREES IN SMALL AREA
2009-11
Adrian Rusu, Ph.D.
Master of Science in Computer Science

As graph layouts and visualizations have been at the forefront of graph drawing research for decades, it consequently led to aesthetic heuristics that not only generated better visualizations and aesthetically appealing graphs but also improved readability and understanding of the graphs. A variety of approaches examines aesthetics of nodes, edges, or graph layout, and related readability metrics. In this thesis, two solutions incorporating Gestalt principles to alleviate the effects of the edge crossing problem are presented. Alleviating this problem improves graph aesthetics and readability. Secondly, improving the known bounds on two aesthetic requirements (area and aspect ratio) for planar straight-line order-preserving grid drawings of binary trees is presented in a novel algorithm using a separations approach. The new bounds are optimal in area and aspect ratio, where the optimum values are linear and 1:1 respectively. All three topics present novel contributions to graph and tree drawing ultimately leading to a potential for improved readability and aesthetics requirements.

Table of Contents

List of Figures

Figure Page

List of Tables

Table Page

Chapter 1

Introduction

1.1 Introduction

Graphs are data structures that express pairwise relationships in a data set. They are prevalent in most areas of research. Maps of network topology, subway lines, and roads are graphs that represent how locations are interconnected. Flowcharts and process models represent how the steps in a process or algorithm are to be carried out. Graphs of social networks reveal key people in social circles and organizations. Natural language processing and automata may be formulated with graphs. A certain class of graphs, called trees, encodes hierarchies useful for such diverse applications as software design, business organization, artificial intelligence where trees may be used to encode a solution space of possible actions, decision trees in operations research and statistics, etc.

Graphs are a set of nodes (or vertices) connected by edges. The nodes are the data and the connecting edges are the pairwise relationships, e.g. in a subway map, stations and terminals are nodes in the graph and the lines connecting them are the edges. The edges can be thought of as the relations next_stop and/or previous_stop. A single pairwise relationship would be expressed next_stop(A, B), which encodes the same information as previous_stop(B, A), and is interpreted the next stop from station A is station B and the previous stop from station B is station A.

Information visualization is the discipline of researching and implementing ways to organize and display potentially large amounts of data so that useful information is quickly and effectively presented to an analyst. One area of information visualization is graph drawing, automatically generating visualizations of graphs. Graph drawing also

includes tree drawing, which often use different algorithms than the more general graph drawing algorithms.

Research into how people perceive their environment and understand visual stimuli is the focus of Gestalt psychology. This research is applied extensively within information visualization. Gestalt psychology provides more concrete reasoning for why information visualization is needed beyond this simple demonstration. This will be discussed in detail in the second chapter as well as our research into how principles from Gestalt psychology were applied to graph drawings to enhance their clarity.

Several useful metrics have been devised for measuring the effectiveness of graph drawings. These metrics are called aesthetic requirements and are indicators of a graph drawing's readability, understandability, and overall aesthetic value. Given two drawings, one can be deemed 'better' if it more successfully meets one or more of these aesthetic requirements. The following is a list of the most important aesthetics of graph drawings:

- Area: 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. Note that one cannot discuss the area of non-grid drawings (i.e. drawings that have the nodes placed at real coordinates), since, by placing the nodes closer or farther, such a drawing can be scaled down or up by any value.
- Aspect Ratio: The aspect ratio of a grid drawing is defined as the ratio of the width to the height of its enclosing rectangle. An aspect ratio is considered optimal if it is equal to 1:1. 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.

- Size: the longest side of the smallest rectangle with horizontal and vertical sides covering 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 smallest angle formed by two edges incident on the same node.
- Symmetry: visual identification of symmetries in the drawing.

It is generally accepted [16, 44-46] that small values of the 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. 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 three; hence a trivial upper bound on the angular resolution is 120 degrees. Given a symmetric drawing, a conceptual understanding of the entire tree can be built up from that of a smaller subtree, replicated a number of times. The presentation of a new binary tree drawing algorithm that considers both the aesthetics of area and aspect ratio optimization is the focus of the third chapter.

1.2 Contributions and Outline

The major contributions of this thesis are two novel approaches to applying Gestalt psychology to alleviate the edge crossing problem in graph drawings as well as a planar non-upward straight-line order-preserving grid drawing of binary trees with optimal area and aspect ratio.

The thesis will follow this outline:

- Chapter 1 gives an introduction and outline to the thesis.
- Chapter 2 presents two methods of applying Gestalt psychology to alleviating the edge crossing problem in graph drawings, related work on Gestalt psychology in computer visualization, and future work.
- Chapter 3 presents an algorithm to create a planar non-upward straight-line order-preserving grid drawing of binary trees with optimal area and aspect ratio, related work on binary trees, and future work.
- Chapter 4 summarizes the results of the thesis.

Chapter 2

Alleviating the Edge Crossing Problem

A natural visualization for graphs is the embedding of the graph into a plane, referred to as a geometric graph. A classic problem in graph drawing is how to embed a graph into the plane so that it meets certain aesthetic requirements, such as reducing edge crossings, maximizing angles between edges, minimizing the graph's area, etc., to produce readable and aesthetically pleasing geometric graphs. Aesthetic requirements are indicators of a graph drawing's readability, understanding, and overall aesthetic value. Therefore, given two drawings of a graph structure the 'better' drawing is more successful in meeting one or more of these aesthetic requirements than the other.

Most graph drawing algorithms focus on placement of the nodes constrained by the aesthetic requirements, yet further refinement of the drawing methods after embedding are not as well studied. This chapter presents two algorithms arising from application of principles from Gestalt psychology. These principles derive from the law of prägnanz, which argues that people tend to order experience in a matter that is simple, orderly, symmetric, and regular [26]. The principles of grouping suggest that predictions can be made on how the mind will interpret visual stimuli since it naturally seeks order and patterns based on certain rules. The principles are ordered into the following seven categories [Fig. 1]:

1. Proximity - elements that are close together are perceived collectively, whereas elements that are far apart are perceived separately.
2. Similarity - like elements are perceived collectively, whereas differing elements are perceived separately.

3. Closure - incomplete or partially obscured elements tend to be completed by the mind.
4. Symmetry - elements displaying symmetry are perceived collectively in spite of distance.
5. Continuation - when elements intersect or overlap, each is perceived separately and as uninterrupted.
6. Common fate - when elements are moving at the same velocity, they are perceived collectively.
7. Figure-ground - elements can be distinguished from their surroundings by dividing a scene into foreground and background.

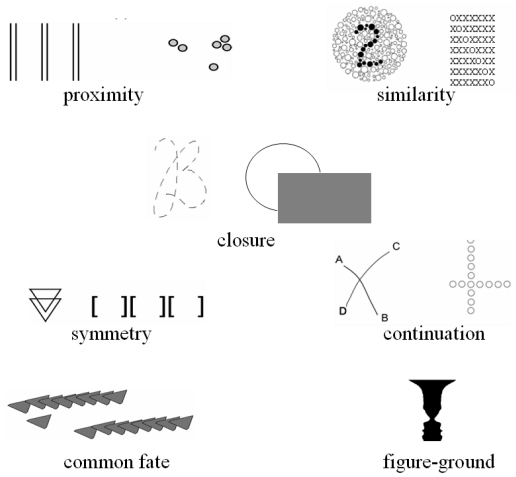


Figure 1. Gestalt principles.

Such experimental studies as [16] provide evidence that edge crossings are the primary inhibitor to readability. However, not all graphs are planar so it is impossible to guarantee that a drawing is free of edge crossings in the general case. The two

algorithms[1] presented in this chapter provide original methods to alleviate this problem. The first of these two algorithms proceeds from node placement by subsequently coloring the edges of the graph where close edges are painted with perceptually opposing colors in order to maximize the viewer's ability to distinguish them. The second focuses on utilizing the principle of closure to improve readability inhibited by edge crossings by allowing for edge breaks around crossing points.

There are several algorithms available for embedding geometric graphs. Among the most popular are FD-FR by Fruchterman and Reingold, [9] based on an original idea by Eades [5]; FD-K by Kamada and Kawai, [12] also based on FDP; POGB by Tamassia [21]; PG by Woods [25]; PGS by de Fraysseix et al. [8]; SEIS by Seisenberger [20]; and Tu by Tunkelang [23]. FD-FR and FD-K are force-directed algorithms which equate graph drawing to minimizing energy in a physical system, POGB is a planar orthogonal grid algorithm, PG is a planar grid algorithm with many sloped edges, PGS uses all straight lines, SEIS uses PGS followed by compression to reduce the overall area, and Tu is an incremental algorithm whose drawing is similar to a force-directed algorithm.

In [16], some of the previous algorithms were compared for aesthetics. A user study was conducted to determine the users' performance in reading graphs, and the criteria that made the graphs easier or more difficult to read. The empirical evidence from this study suggests that edge crossings are one of the primary culprits that make graphs difficult to read.

[1] The first solution was published in the Proceedings of the 13th International Conference on Information Visualization, 2009 in collaboration with Adrian Rusu, Radu Jianu, and David Laidlaw [47].
The second solution was published in the Proceedings of the 15th International Conference on Information Visualization, 2011 in collaboration with Amalia Rusu, Radu Jianu, and Adrian Rusu [48].

An embedding algorithm with few edge crossings that is interactive, easy to implement, and adaptable is required as the first step to both of our algorithms. The Force Directed Placement (FDP) algorithm FD-FR [9] meets these requirements. In FDP, the nodes can be imagined as point masses connected by springs. The total energy of the system is then minimized iteratively by adjusting the lengths of the springs. The final embedding is the minimum energy state found. The original algorithm has a costly time complexity of $O(n^3)$, though [2] and [22] present optimizations to produce algorithms with a time complexity of $O(n^2)$ and [15] further improves it with a hybrid algorithm utilizing approximations by both sampling and interpolating, achieving an algorithm with time complexity of $O(n^{5/4})$.

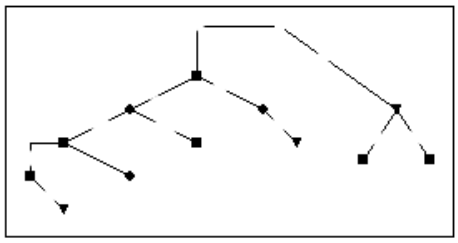
While geometric embedding algorithms have been the primary method of addressing edge crossings, graph drawing coloring algorithms have received little attention. The traditional graph coloring problem of assigning adjacent nodes different colors does exist, but it only deals with colors in a theoretical and abstract fashion. However, there has been some research in traditional coloring of geometric graphs. Examples of some of these investigations include a study by Bern, Eppstein, and Hutchings on an algorithm for coloring quadtrees [1], colorings of geometric intersection graphs by Eppstein [6], and colorings of arrangement graphs by Felsner, Hurtado, Noy, and Streinu [7]. There have also been studies by Levkowitz and Herman [14], Robertson [18], and Ware [24] on effectively building color maps corresponding to data values in data visualization or images. Tracing a path through color space to construct a color scale has been studied by Rheingans and Tebbs [17].

From a methodology standpoint, [4] comes closest to the first algorithm of this chapter by introducing an algorithm for coloring neighboring nodes in a geometric graph using perceptually different colors. We are unaware of similar work for graph edges.

Choosing appropriate colors to represent data values or using perceptually similar colors to indicate similarity between objects have received more attention. The studies in [10], [14] and [24] perform empirical studies to address the problem of generating perceptually effective color-maps. [3] uses perceptually uniform colors to underline similarities in DTI fiber tracts.

The second algorithm of this chapter only affects the areas immediately around the edge crossings and can still be employed when restricted to black-on-white drawings. We are unaware of any other research employing edge breaks in the graph drawing to enhance perception. However, researchers in [27-29, 31] provide evidence of the cognitive ability of the human mind to correctly interpret highly distorted images or shapes, useful in applications such as human interactive proof systems and CAPTCHAs [Fig. 2]. Using the high success rate as evidence that humans use Gestalt principles, such as closure, in order to interpret distorted images, it is predicted that the minimal effects incorporated by the second algorithm will not hinder readability; in fact, it is expected to enhance readability by breaking other unintentional gestalts.

a) English words (city names) as CAPTCHA to distinguish humans from automatic programs (bots) in online services.



b) Tree structure as CAPTCHA.

Figure 2. Visualizations transformed according to Gestalt principles which pose no problem to human recognition.

2.1. Edge Coloring Method

2.1.1 Introduction

The edge coloring method can be reduced to producing a 2D or 3D embedding of a distance metric (measuring perceived and actual distance between edges) and immersing it into a color space. Representing a distance metric in a viewable space has mostly been researched in the context of multi-dimensional data visualization. So-called multi-dimensional scaling (MDS) techniques map points in multiple dimensions to a visualizable 2D or 3D space, while preserving the distance relations among them.

The MDS techniques fall into two categories: linear and non-linear methods. The linear methods perform linear combinations of the multiple dimensions to approximate them in lower dimensions. Two such methods are Principal Component Analysis (PCA) [11] and Star coordinates [13]. The drawbacks to these linear methods are: requiring an

explicit vector representation of the points, needing to recompute the result for every change to the dataset, and a lack of interaction.

Non-linear methods solve the drawbacks of the linear methods. They can usually take the distance function directly as input and define an embedding error measure to convey how well the embedding preserves the original distances. Gradient descent or force simulation can then be used to find an embedding that corresponds to a local minimum of the embedding error measure. Two such methods are Sammon's Mapping [19] and Force Directed Placement (FDP) [9], which is an adaptation of the previously mentioned FDP algorithm for graph embedding to this problem of distance embeddings. The choice of FDP makes sense here since it is already being used for the graph embedding, and it accepts distances directly as input, tends to yield better embeddings than the linear methods, is iterative and thus interactive, and is easy to implement and adapt.

As for color spaces, the most popular color space for display devices is RGB where the intensities of red, green, and blue are varied to produce colors. The problem with RGB is that perceived distance in color does not correlate to Euclidean distance between color points making it impractical for translating a spatial embedding into a corresponding color embedding. CIE L*a*b* (Lab) overcomes this drawback. It defines colors as unique combinations of luminosity (L), degree of magenta to green (a) and degree of yellow to blue (b). The property that makes Lab suited for color mappings is that Euclidean distances between color points in the 3D Lab space correspond to a similar perceived distance in color. Thus, preserving the inverse of the edge distance metric in

this space will yield edge colors that are perceptually far for edges that are geometrically close.

2.1.2 Edge Distance

Assuming a geometric embedding of a graph is provided, a distance metric of the set of edges can be defined. Intuitively, the distance between two edges is small if they interact visually [Fig. 3]. Thus the distance metric is defined as a function of line segment distance, crossing, and angle. Specifically, equation 1 is used to derive values for this distance function (this is not a true metric since the triangle inequality fails, but the importance is on the behavior between edges, not the behavior across pairs of edges). The inverse of this function is then embedded into a color space to compute edge colors. As a result, perceptually close edges will be assigned colors that are far apart.

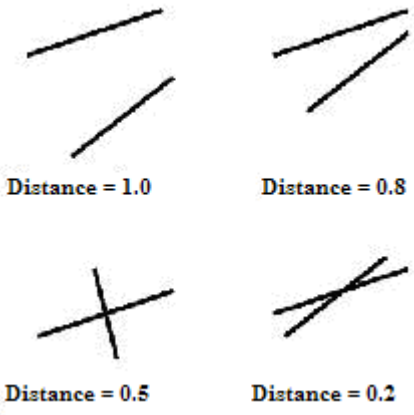


Figure 3. Examples of edge pairs and their approximate, normalized distance values.

The distance (closeness) D of two edges, e1 and e2, is computed as follows:

$$D(\text{e1},\text{e2}) = \begin{cases} \theta, & dist = 0 \\ dist + 1, & otherwise \end{cases} \qquad (1)$$

where $dist$ is the minimum distance between e1 and e2

and $\theta = \dfrac{\text{minimum angle between e1 and e2}}{\pi / 2}$

It was found that a metric that varies linearly with edge distance and crossing angle performs better than polynomial functions of the two because it is less affected by the wide range of angles and distances common to many graph drawings.

2.1.3 Color embedding

Similar to [4], the colors are computed by embedding the inverse metric described in the previous section into the Lab color space using a force directed method. However, a different method of constraining the points to the visible Lab space is used here.

In our algorithm a mesh approximation of the Lab gamut [Fig. 4] is made and used as a 3D container in which embedding can occur. Points corresponding to edges are created and placed within the Lab gamut container, and forces are computed on them iteratively. While in traditional spring based methods, forces are translated directly into positional displacement, the algorithm here uses a physically correct simulation by using the forces to compute velocities, which are then translated into displacements. The Lab gamut boundaries, enforced by the mesh, will act as a closed container that keeps the points inside by performing collision detection.

Points outside the container mesh are beyond the visible color spectrum and would be impossible to render. While [4] uses a combination of scaling and truncating to contain the points within the visible region of Lab, the physically based simulation described above already avoids this problem. This method was chosen because of two particularities of the Lab gamut: irregular shape and saturated colors close to the boundaries. Similarly to [4], it is noted that although it would be possible to simulate a repulsive force at the container boundaries, it would need to have a steep gradient to

allow points to come close to the saturated areas near the boundaries. Such a force would be hard to control in simulations. The physically based simulation allows points to bounce off the container and even slide across the container faces to positions that minimize the system's energy. Overall, good embeddings are achieved using this simulation strategy. A graphical result of the final distance that is embedded and its corresponding embedding error is shown in Figure 5.

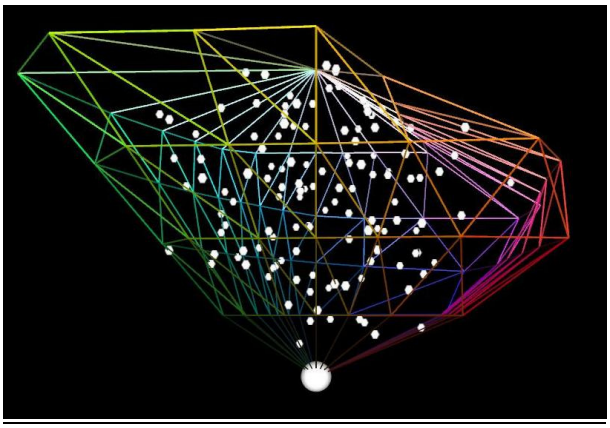


Figure 4. Mesh approximation of the Lab gamut and embedded points corresponding to edges of a graph. The white sphere at the bottom exerts a repulsive force on the points so that edge colors are picked from the lighter spectrum.

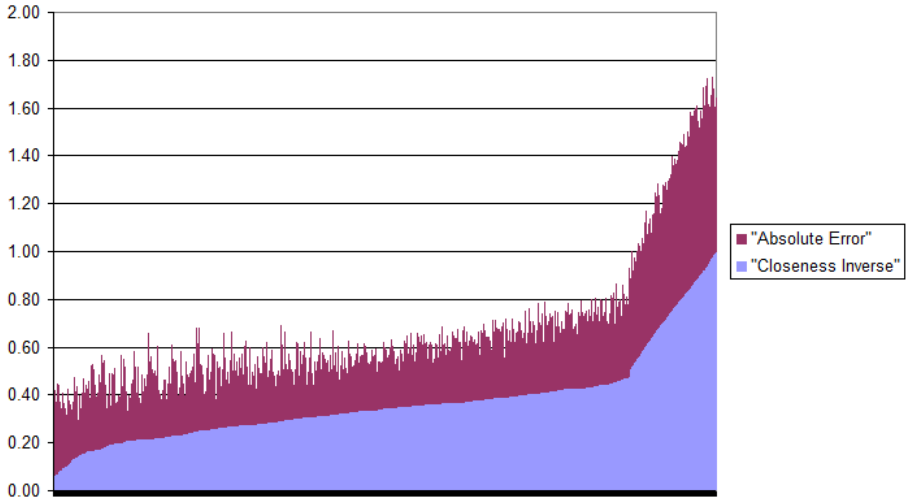


Figure 5. The inverse of the closeness metric and the absolute embedding error. The inverse of the closeness metric for each pair of edges is plotted in increasing order of magnitude with its corresponding error stacked on top.

It is also important to note that while the objective is for close edges to be colored differently, it does not matter how distant edges are colored. This enables better embeddings to be obtained since the number of constraints on the system is significantly reduced, especially for large graph drawings.

2.1.4 Implementation details

These methods were implemented in C++ using the G3D extension library for OpenGL. QT 4.3 was used for user interface support. For the actual embedding algorithm, Chalmer's method [2] was used to reduce the complexity of one simulation iteration of FDP to $O(n^2)$. The collision detection of points with the gamut also needs to be factored in. To accelerate this process, a set of successive tests based on the proximity of a point to a gamut face was employed. Interactive embeddings were achieved for moderately large graphs (1000 edges). For an example graph with 545 nodes and 880 edges, this algorithm required an average of 0.21 seconds/iteration. The number of iterations required for convergence depends on the specific graph layout.

A problem observed while testing was that some edges were colored near the graph drawing's background color, thus the ability to add a repulsive color that is "avoided" by the embedder was included. This can be used to ensure that edges are not colored with the background color of the scene and remain distinguishable.

2.1.5 Results

To the best of our knowledge the concept of assigning perceptually opposing colors to geometrically close edges in graph drawings with the purpose of making edges more distinguishable and reducing the effects of edge crossings is new. This can

complement edge crossing techniques that act on the geometric embedding side and help in cases where a crossing free drawing is not possible.

The first metric that defines edge closeness to define the amount of visual interaction between edges in a graph drawing is also provided. This has the potential to be used in further research and applications related to graph drawing readability.

Presented is a novel and efficient force-based and physically accurate algorithm that can assign perceptually opposing colors to close edges in a graph drawing. The colors cover the full visible spectrum and have low embedding errors.

More importantly, from a graph-reading standpoint, there seem to be several benefits to this method. For instance, lines rooted in the same node and having angles close to 0 or π are colored differently [Fig. 9]. This can help the user tell them apart and understand the topology of the graph better. In the case of long edges that stem from the same node and flow in the same direction, the different colors could make it easier for users to visually follow paths. It is also likely that the method will minimize the effect of edge crossings especially in areas with high edge density.

Figures 6 - 11 show some outputs of the program. Figure 6 illustrates a real-life example: a protein interaction network extracted from the HPRD (Human Protein Reference Database) is depicted with this edge coloring technique. Despite the high visual clutter and wide range of angles and distances between edges, the algorithm manages to generate a good coloring; the upper-right region especially shows many crossing edges colored with perceptually different colors.

Figure 7 illustrates the concept in a simple example and allows one to better grasp the coloring metric. Figure 8 depicts a graph of three sparsely connected clusters with an

especially good coloring. It must be noted, however, that the algorithm performs better for clustered graphs, since edges in different clusters can be co-located in the color space without any interaction. The more edges interact visually in a given area, the less distance in the Lab color space can be assigned between them, as they need to be spaced more or less evenly throughout the color space. Figure 9 illustrates this aspect: while the algorithm accommodates many of the edge crossings in a complete eight degree graph, some crossing edges are inevitably assigned similar colors.

While Figure 10 shows a relatively good coloring for a more complex graph, Figure 11 reveals another shortcoming of this technique: crossing edges on the left are deemed far apart by the algorithm compared to the many tight-angled edges on the right side of the image. They will thus be assigned a lower embedding priority. This also motivates the choice of a simple linear distance metric; a polynomial would have amplified this effect even more.

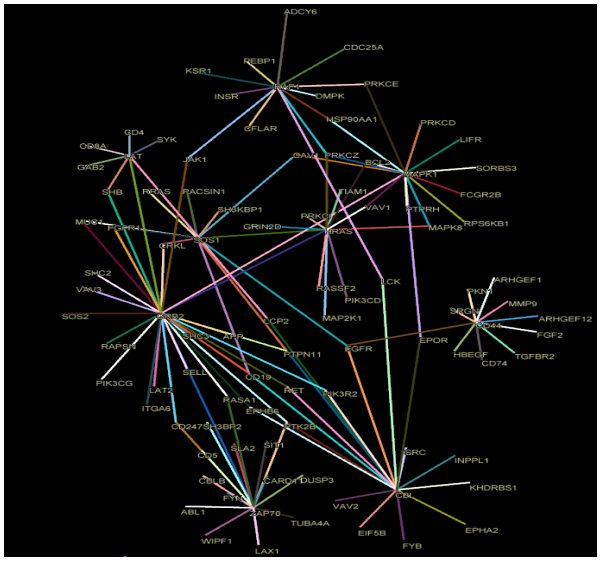


Figure 6. Protein interaction network derived from data available in the Human Protein Reference Database (HPRD) with described edge coloring solution.

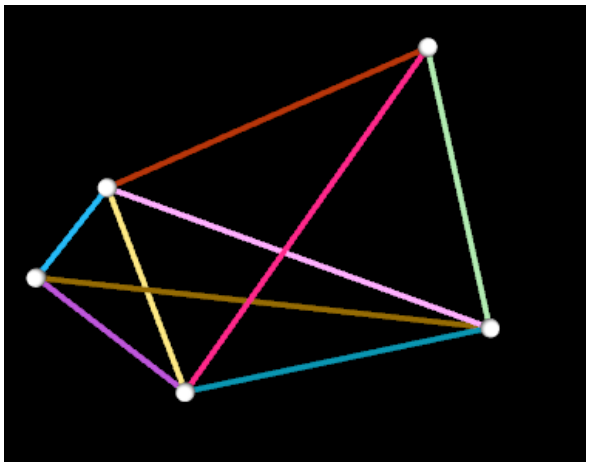


Figure 7. Simple example illustrating the coloring method. Edges crossing or stemming from the same node are assigned opposing colors.

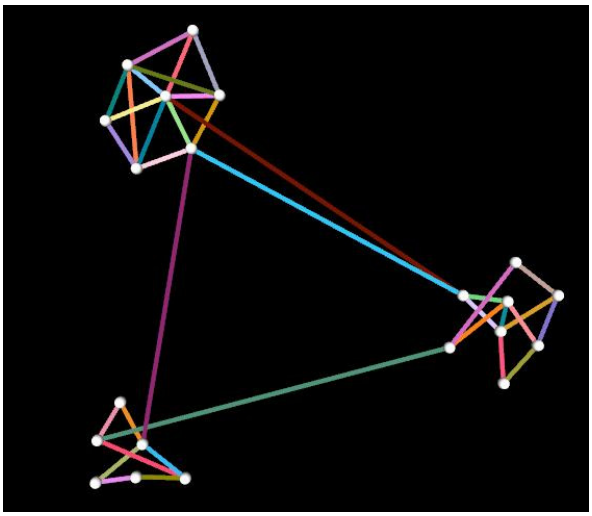


Figure 8. A graph of three sparsely connected clusters.

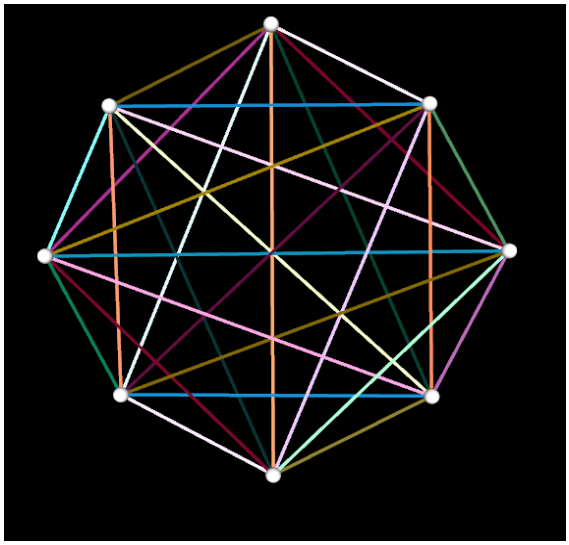


Figure 9. A complete 8-degree graph.

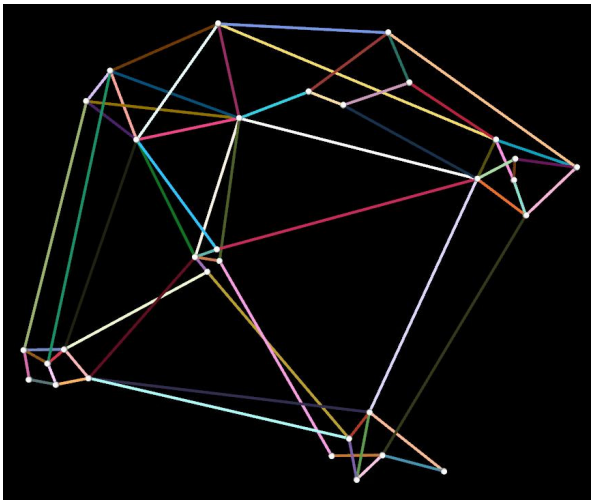


Figure 10. A more complex graph.

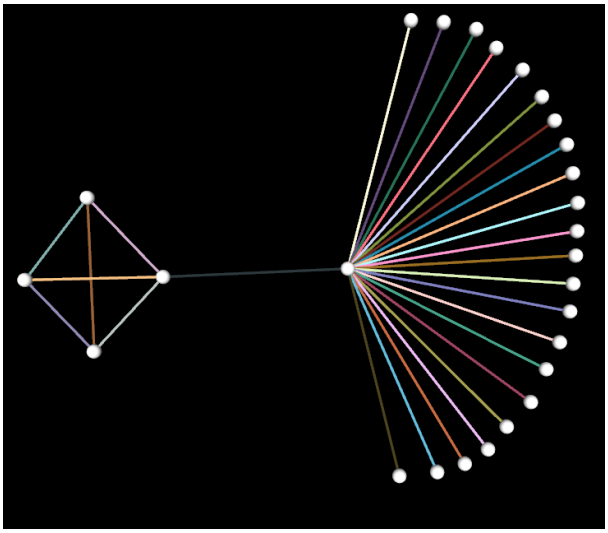


Figure 11. A problem with the coloring: the close edges in the bundle on the right cause the embedding algorithm to be insensitive to the crossing edges on the left.

2.1.6 Discussion

Some simple scenarios where edge coloring helps are presented here. However, a rigorous evaluation would be helpful to quantify the improvement and determine the range of benefits. It would be particularly interesting to perform some of the studies in [16] and quantify the effects of coloring on edge crossings.

While an intuitive approach to defining the "closeness" metric was given, it would be interesting to base this choice on a more rigorous study. Unfortunately, research so far can only give binary guidelines, in the sense that crossings do indeed affect graph-reading performance. A study on edge topologies that are hard to read in the absence of color would be interesting in its self, but would also help refine the definition of the closeness metric.

Direct embedding in the Lab space using the gamut as a containing space yields good results. By adjusting the embedding parameters the full visible range of Lab was useable and a low-error embedding was achievable. The alternative of using a generic

embedding to obtain 3D points and rescale them to fit inside the visible Lab gamut does not produce good results because of the irregular size of the visible gamut, and the fact that the high saturation colors are located far from the gamut center.

Finally, it should be noted that applying color to edges in the interest of improving graph drawing readability is not always an option. Oftentimes, in real-life applications, edge color is already used to encode a value corresponding to the relationship it stands for and cannot be used for additional purposes. However, there are plenty of opportunities to use this method when such constraints are not present.

2.2 Gestalt Principle of Closure Method

2.2.1 Introduction

Proposed here is a new graph drawing algorithm whose focus is on utilizing the Gestalt principle of closure to improve readability inhibited by edge crossings. At each edge crossing, assign one edge to be primary and the other to be secondary. The secondary edge will then have a gap in continuity on the portion immediately passing through the area of the edge crossing. The primary edge is trivially easy to trace through the area of the edge crossing, and the secondary edge is still easily perceived as a whole edge due to the Gestalt principle of closure. In the unusual case of multiple edges crossing at the same point, a number of them are assigned as primary, while the rest are assigned as secondary, then each set is treated as in the typical case.

2.2.2 The Algorithm

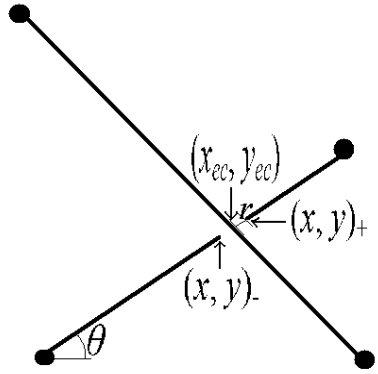
A sweep line algorithm can be used to efficiently find all edge crossings. For example, the Bentley-Ottman algorithm [30] performs with a time complexity of

$O((n+k)\log n)$, where n is the number of edges and k is the number of edge crossings. A simpler algorithm would be a pairwise comparison of edges where minimum bounding boxes are used to quickly eliminate spatially distant edges. Although it has a poorer time complexity, the latter option was chosen in this implementation since it offers a natural way to select the primary and secondary edges. In this implementation it was decided that the edge being compared to all others is the secondary edge. This choice naturally leads to the secondary edges indexing the edge crossing positions so that intermediate points can be easily determined using the same angle for each by using the following formulas:

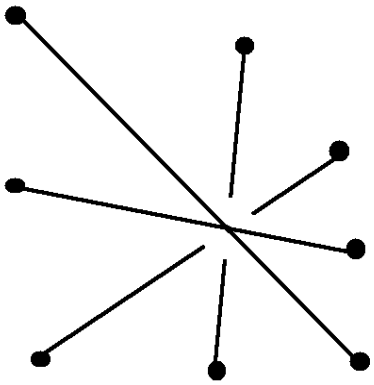
$$x = x_{ec} \pm r\cos(\theta)$$
$$y = y_{ec} \pm r\sin(\theta)$$

where (x_{ec}, y_{ec}) is an edge crossing, r is the break radius, and θ is the angle that the secondary edge makes with the x-axis. These intermediate points (x, y) lie on the intersections of the break area and the secondary edge, so that the entire edge can be represented as a series of line segments where the first line segment starts at the parent node's coordinates and ends at the first intermediate point, and the last line segment starts at the last intermediate point and ends at the child node's coordinates [Fig. 12]. When the intermediate points are being determined, some may need to be discarded if two break areas overlap so that a line segment is not drawn in this area. Drawing primary edges is trivial in this system since they are represented as secondary edges with no intersections, causing the lines to be drawn through the intersection points creating the desired Gestalt effect over the secondary edges with which they cross.

The graphs presented for this algorithm were all derived from the Human Protein Reference Database. The same graph generating program used by the first algorithm is used to create these ones as well.



a) The geometry of the secondary edge.



b) Multiple primary and secondary edges intersecting at the same edge crossing point.

Figure 12. Examples illustrating the algorithm for edge crossings that incorporates the Gestalt principle of closure.

Various graphs were drawn, and each was rendered with and without the gestalt effect [Fig. 13-16]. We were interested in the effect of the gestalt approach when applied to a graph with fewer edge crossings vs. a very large number of edge crossings.

2.2.3 User Study

A preliminary study was performed with fourteen participants having various educational backgrounds. Each participant was shown the four testing figures, in the order listed here, and asked to identify all nodes connected to a node chosen at random five times for each graph and rendering. The participants were asked to rate the ease of this task for each graph on a five-point Likert scale, where one denotes 'very difficult' and five denotes 'very easy.' The participants were also asked to give general feedback regarding the test at the end. Table 1 shows the results of this study. The column headings include a number corresponding to the testing figure and also identify whether the testing figure is 'traditional' (T) or has the Gestalt effect added (G) (i.e., 13T corresponds to Figure 13 left - traditional, whereas 16G corresponds to Figure 16 right - with breaks in edges at edge crossings). The cells with lighter shading highlight an improvement in readability over the traditional graphs whereas the cells with darker shading highlight difficulty when introducing Gestalt effect, as perceived by the participants in the study.

Table 1. Graph readability results (5 *is* '*very easy*' *and* 1 *is* '*very difficult*').

	13T	13G	14T	14G	15T	15G	16T	16G
1	5	5	**3**	**4**	**3**	**4**	5	5
2	4	4	5	5	**3**	**2**	1	1
3	5	5	**4**	**5**	**3**	**4**	3	3
4	5	5	5	5	**4**	**5**	**4**	**3**
5	5	5	**5**	**4**	**5**	**4**	**5**	**4**
6	4	4	5	5	**2**	**1**	**3**	**2**
7	5	5	5	5	**5**	**4**	**5**	**4**
8	5	5	5	5	4	4	4	4
9	5	5	5	5	5	5	5	5
10	4	4	4	4	4	4	4	4
11	**4**	**5**	**4**	**5**	**4**	**5**	**4**	**5**
12	**5**	**4**	**3**	**5**	**4**	**3**	5	5
13	5	5	5	5	4	4	**3**	**4**
14	5	5	5	5	4	4	4	4

More qualitative data has been recorded through the general feedback on the exit survey. Out of the 14 participants, 13 left feedback of which one did not relate to the study. Three of them felt that the gaps were distracting and aesthetically unpleasing making the tasks more difficult whereas two felt that the gaps made the graph more aesthetically pleasing though had no impact on the difficulty of the tasks. Three felt that the breaks were distracting in sparse drawings but were very effective in densely clustered drawings, while another four had no preference for either rendering after the study. These preliminary tests are encouraging especially for the graph drawings with clusters of edge crossings or larger graph drawings.

2.2.4 Conclusion and Future Work

In future work, considering some of the feedback received in the initial study, more tests could be performed in which edge crossings must meet certain requirements before a gap is introduced. Such requirements may be local node density, angle at which edges intersect, number of edge crossings on a single edge, and whether gaps have already been introduced on an edge.

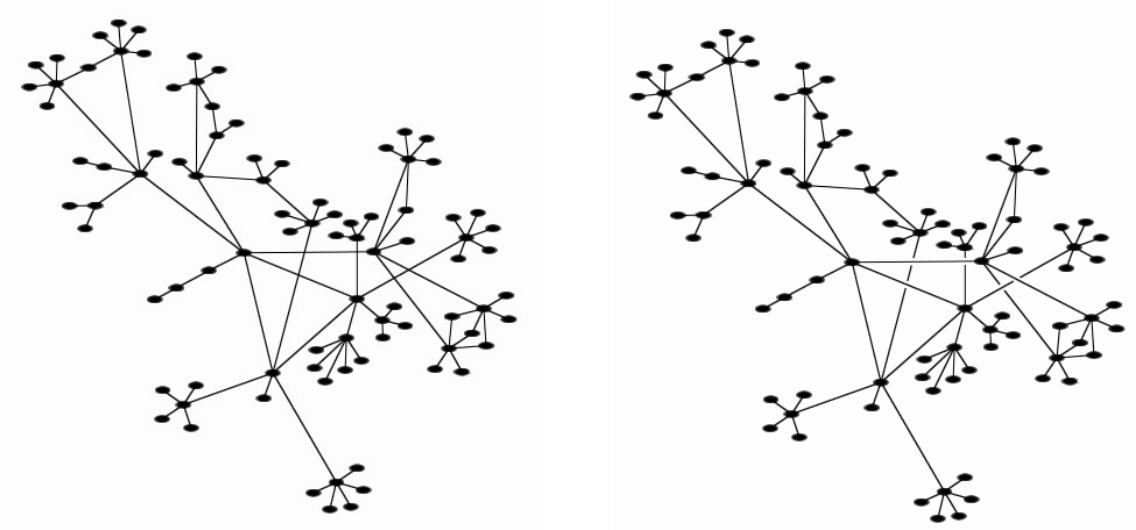


Figure 13. Graph drawing with few edge crossings (traditional on the left; Gestalt effect introduced on the right).

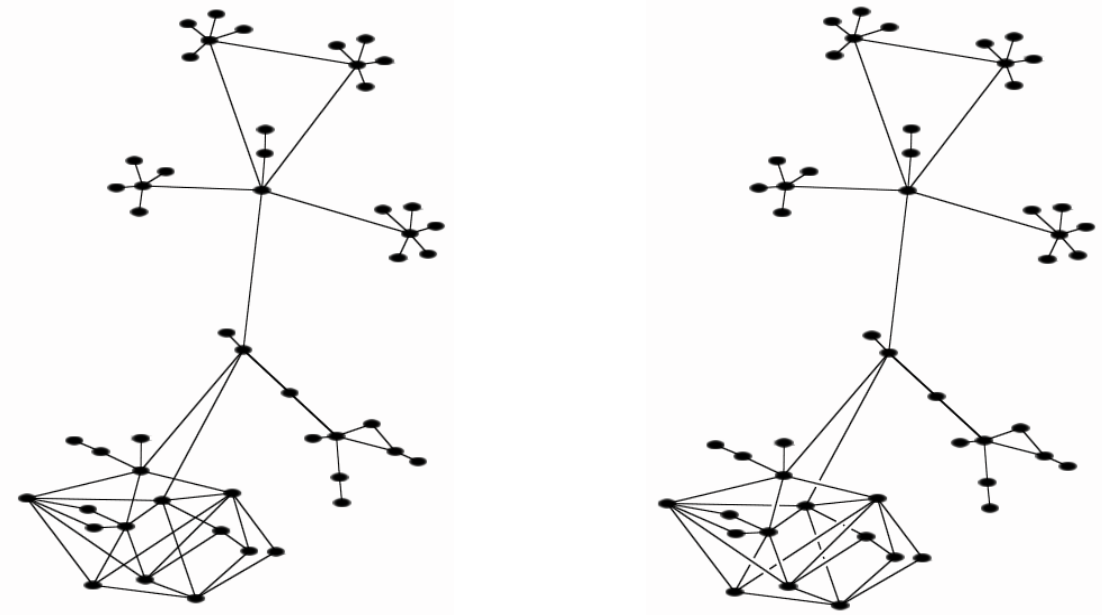


Figure 14. Graph drawing with a cluster of edge crossings (traditional on the left; Gestalt effect introduced on the right).

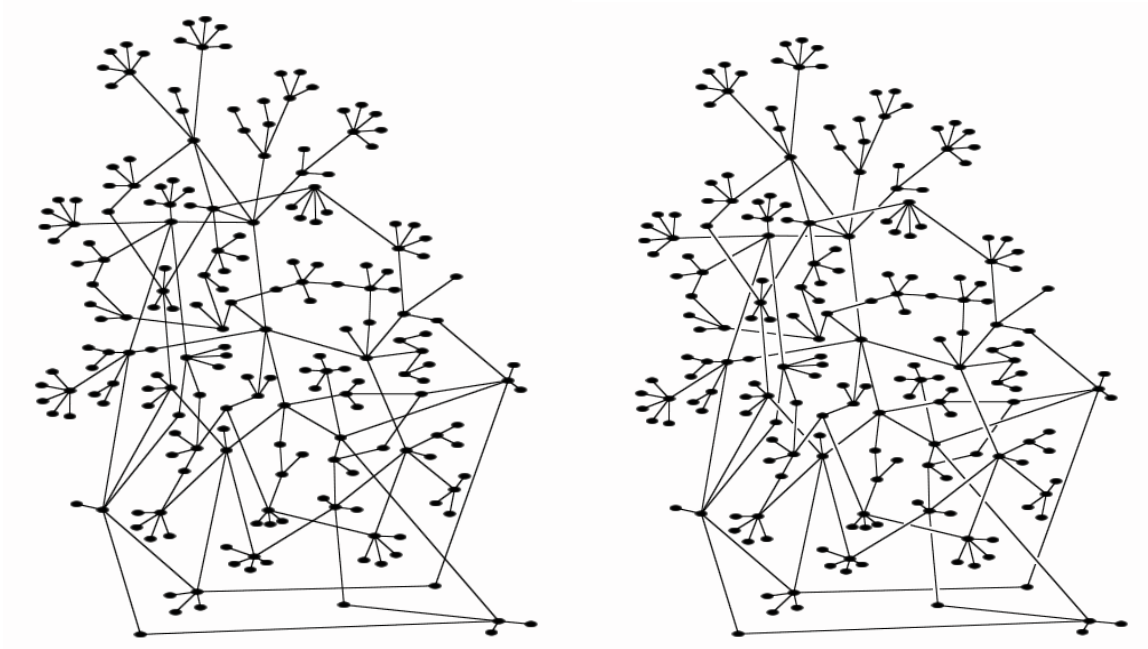


Figure 15. Larger graph drawing with edge crossings (traditional on the left; Gestalt effect introduced on the right).

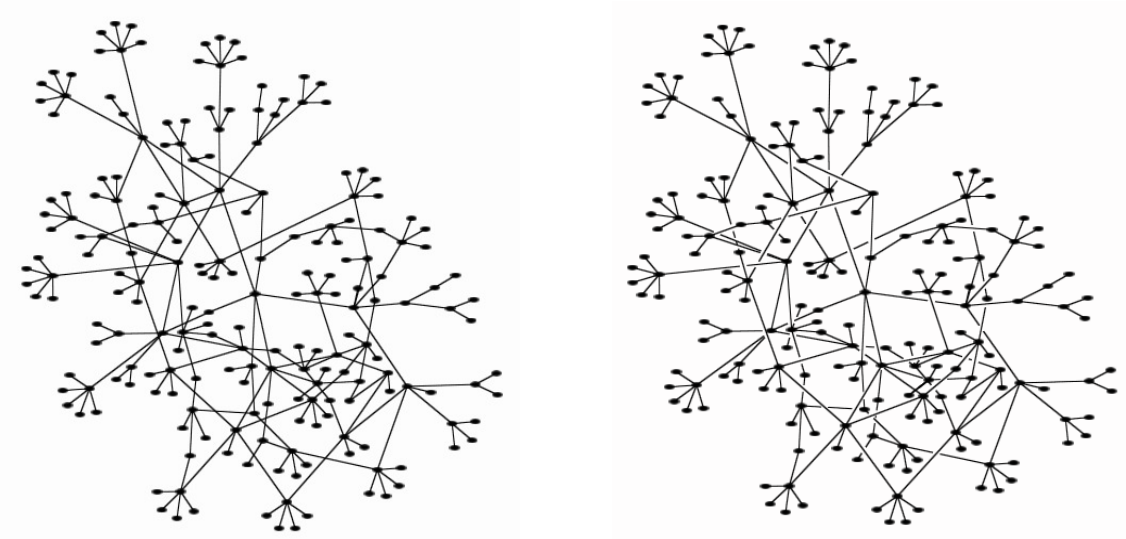


Figure 16. Graph drawing with a high density of edge crossings (traditional on the left; Gestalt effect introduced on the right).

Presented here was a method that uses the Gestalt principle of closure for increasing graph drawings' aesthetics and readability by introducing gaps into a graph at every edge crossing. Similar Gestalt principles applied as deformations to handwriting

samples or shapes have been proven to work well without altering human recognition abilities. The FDP algorithm was adapted to accommodate these gaps as part of the secondary edges when crossing primary edges.

A preliminary study was conducted to collect evidence that this method holds merit. Initial results and user feedback have been promising and provided important information for future work. Additional studies will be considered in the future on adding gaps at edge crossings after certain requirements are met. Additional experiments will be considered by varying the size of the edge breaks in order to determine any influence on graph readability.

Chapter 3

A Straight-Line Ordered Binary Tree Drawing Algorithm with Linear Area and Arbitrary Aspect Ratio

3.1 Introduction

Trees are data structures that are useful for encoding acyclic hierarchical relationships such as directory structures, business organizational structure, object-oriented software, genealogy, tournament brackets, and decision trees. These data structures contain nodes and edges that connect the nodes. In a directory structure, a directory would be a node and every subdirectory would be a node connected to it by an edge. The root is the directory at the highest level of the hierarchy from which all other nodes descend. A file in the directory structure is a node contained in a directory that cannot contain any further subdirectories. Such a node is called a leaf of the tree.

The trees dealt with in this chapter are binary trees, meaning each node may not have more than two children (nodes immediately descended from it). For example, a tournament bracket represents teams or players at each node where siblings, nodes which are children of the same node (their parent) are teams that are paired for a match. Their parent is the winner of that match, which continues to the root giving the winning team or player in the tournament. This structure is binary since no more than two teams play a match. Another example would be a yes/no decision tree where each node represents a decision and each child represents the next decision in the process depending on which decision was made at the parent node.

An ordered tree is one that requires an ordering to exist to interpret the information, for example the leftmost child may be the smallest of a series of numbers and the rightmost child may be the largest of a series of numbers. In a binary tree, the left

node may represent no and the right node may represent yes. More terminology and background information is provided in Preliminaries.

The approach to creating drawings in [32] and this chapter is known as Separation. Drawings are solved using a divide-and-conquer strategy where at each level

1. a separator edge, explained in Preliminaries, is found
2. the tree is split into subtrees by removing at least the separator edge
3. a desired aspect ratio is assigned to each subtree, explained in Preliminaries
4. the subtrees are recursively drawn
5. the removed edges are reintroduced and composed with the drawings of the subtrees

How to split the tree and compose the drawing are the trickiest parts of this process. As such they are the main focus of this chapter.

As electronic devices shrink in size, so do their monitors, so it is desirable that the most information as possible be contained in the least amount of space. For tree drawings on an integer grid, the optimal area is order n, where n is the total number of nodes in a tree. This would arise from placing one node at each grid location with no extra whitespace. The algorithm presented here will be shown to be of order n, i.e. optimal.

Aspect ratio is an important consideration as popular monitor sizes are either 4:3 or 16:9, while optimal aspect ratio is considered 1:1. The algorithm here tends toward a desired aspect ratio allowing for a better fit of the drawing based on monitor dimensions.

Table 2. Area and Aspect Ratio bounds for planar grid drawing algorithms of n-node binary trees for each combination of the four properties that may be guaranteed, upward, orthogonal, straight-line, and order-preserving drawings. Note that $ab \leq kn$ for some constant k and $0 < \varepsilon < 1$ arbitrarily.

Up-ward	Ortho-gonal	Straight-line	Order-preser-ving	Area	Aspect Ratio	Reference
yes	yes	yes	yes	$\Theta(n^2)$	$O(1)$	[35, 37]
no	yes	yes	yes	$O(n^{1.5})$	$O(n^{0.5} / n)$	[37]
yes	no	yes	yes	$\Theta(n \log n)$	$n / \log n$	[33]
no	no	yes	yes	$O(n \log n)$	$[1, n / \log n]$	[33]
				$O(n \log \log n)$	$n \log \log n \ / \log^2 n$	
				$O(n)$	$[n^{-\varepsilon}, n^{\varepsilon}]$	this chapter
yes	yes	no	yes	$O(n \log n)$	$\Theta(\log^2 n / (n \log \log n))$	[38, 39]
no	yes	no	yes	$O(n)$	$(9a+8) / (9b+8)$	[36]
yes	no	no	yes	$O(n \log n)$	$\log n / n$	[40]
					$\Theta(\log^2 n / (n \log \log n))$	[35, 38]
no	no	no	yes	$O(n \log \log n)$	$n \log \log n \ / \log^2 n$	[33]
yes	yes	yes	no	$O(n \log n)$	$[1, n / \log n]$	[34, 35]
no	yes	yes	no	$O(n \log \log n)$	$\Theta(\log^2 n / (n \log \log n))$	[34, 44]
yes	no	yes	no	$O(n \log \log n)$	$\Theta(\log^2 n / (n \log \log n))$	[44]
no	no	yes	no	$O(n)$	$[n^{-\varepsilon}, n^{\varepsilon}]$	[32]
yes	yes	no	no	$O(n \log \log n)$	$\Theta(\log^2 n / (n \log \log n))$	[38]
no	yes	no	no	$O(n)$	$\Theta(1)$	[41, 43]
yes	no	no	no	$O(n)$	$[n^{-\varepsilon}, n^{\varepsilon}]$	[38]

The previously best known bounds on non-upward planar straight-line order-preserving grid drawings of binary trees are $O(n \log \log n)$ for area and $O(n \log \log n / \log^2 n)$ for aspect ratio (Table 2). The algorithm here is a non-trivial improvement of the non-upward planar straight-line non-order-preserving grid drawing algorithm for binary trees (Table 2) of [32] with area $O(n)$ and aspect ratio between $n^{-\varepsilon}$ and n^{ε}, where $0 < \varepsilon < 1$. It will be shown that the algorithm presented here maintains these bounds on aspect ratio and area, not only improving on the previous bounds for non-upward planar straight-line order-preserving grid drawings of binary trees but finding the area bounds to be optimal.

3.2 Preliminaries

3.2.1 Terminology

In this chapter, a *tree* refers to a binary rooted ordered tree (one with a given root where each node has at most two children and *siblings*—nodes that share the same parent—are assigned a left-to-right ordering). This algorithm will embed a tree onto an orthonormal grid in the plane, where the $x(y)$-*axis* is horizontal(vertical). A *channel* is a line on the grid that is parallel to an axis. The *horizontal*(*vertical*) channel is parallel to the $x(y)$-axis.

Let T be a valid tree with n nodes. If n is zero, the tree is a *null tree* and has no drawing. If n is one, the tree is a *trivial tree* and has a trivial (single dot) drawing. For any tree with more nodes, let o be the root of the tree. Let the *leftmost*(*rightmost*) *node* of T be the node that is at the end of the maximal path from o consisting of all left(right) nodes. A *partial tree* of T is a connected subgraph of T and a *subtree* of T at some node w is a partial tree consisting of w and all of its descendants.

Let Γ be the drawing of T produced by the algorithm of this chapter. R is the smallest rectangle on the grid that completely encloses Γ. The *bottom*, *top*, *left*, and *right* boundaries of Γ are defined to be their respective sides of R. Similarly, the *bottom-left*, *bottom-right*, *top-left*, and *top-right* corners of Γ are defined to be their respective corners of R. A *good* aspect ratio of R is defined to be in the range $[n^{-\varepsilon}, n^{\varepsilon}]$ where $0 < \varepsilon < 1$ is a constant [32].

Given a node w in T, $p(w)$ is the parent of w, $s(w)$ is the sibling of w, $l(w)$ is the left child of w and $r(w)$ is the right child of w. u^* is a node of T with at most one child and is called the link node [32], which will be detailed later for how it can be used to combine two tree drawings into a larger drawing containing both trees. *lm* is the leftmost node of T and *rm* is the rightmost node of T. $a(w, x)$ is the common ancestor of nodes w and x such that no descendant of $a(w, x)$ is also a common ancestor of w and x; when w is x it is defined as $a = w = x$. $a(w, x)$ is simply referred to as a when the context is clear.

3.2.2 Definition of Feasibility

The three properties that constitute a feasible drawing in this algorithm are relaxed properties of a feasible drawing in [32]:

- **Property 1**: The root o is placed on the left boundary of Γ and the vertical channel above it is clear.
- **Property 2**: If u^* exists and is not o, then u^* is either placed on the bottom boundary of Γ and can be moved downward an arbitrary amount or is placed on the right boundary and can be moved rightward an arbitrary amount without causing any crossings.

- **Property 3**: If u^* exists and is o, then u^* is placed in the top-left corner of Γ and can be arbitrarily moved in either the horizontal or in the vertical channels without causing any crossings.

These represent the weakest feasibility properties imposed on drawings. It will be shown in the algorithm that the drawings are grouped into four categories, called *scopes*, with varying restrictions on these properties. In the feasibility lemmas, it will be shown how these variations lead to a planar drawing (a drawing with no edge crossings). Since it is known that every tree admits a planar drawing, which is more desirable than one with edge crossings, a feasible drawing of a tree is one that guarantees planarity. Further it is imposed that a feasible drawing has only straight-line edges. The other properties such as ordering, area and aspect ratio are not considered until feasibility is firstly established in Lemmas 1-5.

3.2.3 Separator Edge

Theorem 1 (Separator Edge): In [32], Theorem 1 states:

> *"Every binary tree T with n nodes, where* $n \geq 2$*, contains an edge e, called a* separator edge*, such that removing e from T splits it into two non-empty trees with* n_1 *and* n_2 *nodes, respectively, such that for some x, where* $1/3 \leq x \leq 2/3$*,* $n_1 = xn$*, and* $n_2 = (1 - x)n$*. Moreover, e can be found in* $O(n)$ *time."*

3.2.4 Definition of Ordering

Let the edge connecting a node x to its parent w be a reference line from which children of that node are placed in increasing counter-clockwise angular order. Let y and z be children of x. Angle wxy is the angle between edges wx and xy and angle wxz is the angle between edges wx and xz. Angle wxy does not equal angle wxz since that implies the siblings are on the same edge, which is invalid. If angle $wxy < wxz$, then y is said to be

the left child and z is said to be the right child, and vice versa. The root of the tree does not have a parent so assume wx is the vertical channel above x when x is the root. When x is the root wx will rotate with the image. For example the vertical channel above the root is wx, but upon a counter-clockwise rotation of 90 degrees about x (the root), it is rotated to the horizontal channel left of x. This guarantees that the angles do not change upon rotation so left remains left and right remains right. This behavior agrees with the usual understanding of a rotation preserving orientation (handedness).

3.2.5 Functions on Drawings

Let Rot90 be the rotation operation on Γ by 90 degrees counter-clockwise about o. As discussed in the definition of ordering, this preserves the left-to-right ordering of siblings. This is demonstrated for a 3-node binary tree in Figure 17.

Let V be a vertical flip of Γ about the horizontal channel through o. Let H be a horizontal flip of Γ about the vertical channel through o. It is well known that V and H (flips or reflections) create mirror images, which have opposite orientation from the original drawing. This causes any angle to negate. Since $wxy < wxz$ for two siblings y and z as described in the definition of ordering, $-wxy > -wxz$, so left children become right children and vice versa. This is demonstrated for a 3-node binary tree in Figure 18 (a-d).

To preserve order, let Γ_M be the drawing of M, the mirror of T. M is the tree with exactly the reverse ordering as T. Though in M we have $wxy < wxz$ for x and y giving x left of y, the data in x is that from what would be the right node in T and y contains the data that would be from the left node in T. For example in Figure 18 (a), swapping the

data in the left and right nodes puts the right node's information in the left node and vice versa as seen in Figure 18 (e).

Using Γ_M when flipping will give a drawing that is order-preserving with respect to Γ, since x left of y in Γ_M after a flip becomes x right of y, which is exactly the ordering we want. In Figure 18 (f) and (g) the ordering problem is solved when using the mirror because the out-of-order data in Figure 18 (e) is put in-order for these situations. Therefore $V(\Gamma)$ is defined to be $V(\Gamma_M)$ and $H(\Gamma)$ is defined to be $H(\Gamma_M)$ so that order is preserved when flipping. [33] further describes the utility of mirroring in solving the out-of-order problem caused by flipping.

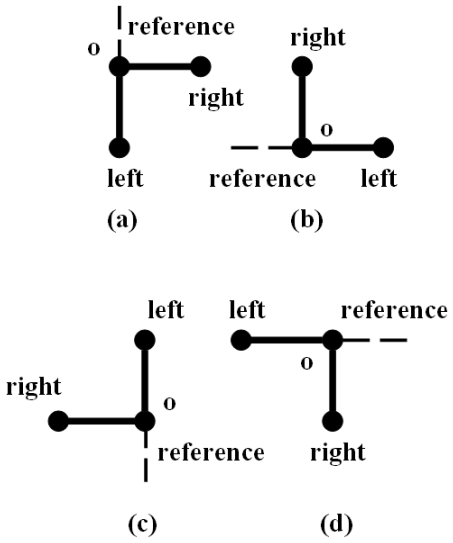


Figure 17. (a) the initial configuration of a three node binary tree. (b) one application of Rot90. (c) two applications of Rot90. (d) three applications of Rot90. Four applications of Rot90 do not affect (a).

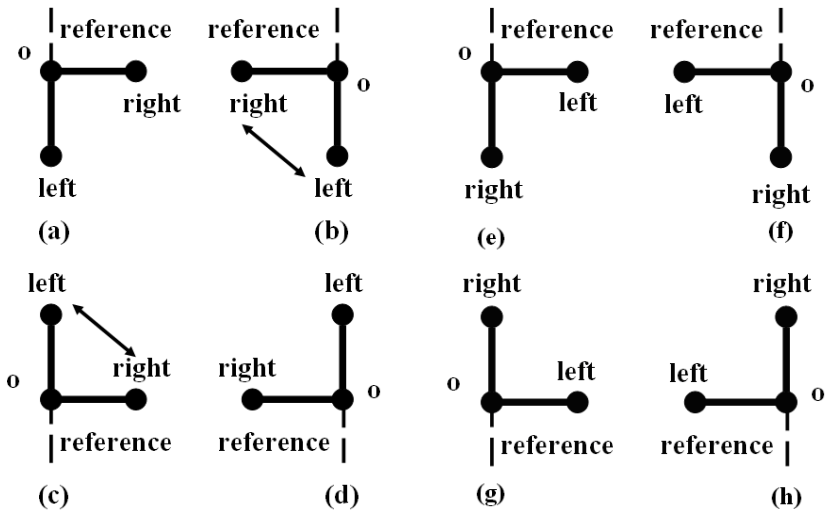


Figure 18. (a) the initial configuration of a three node binary tree. (b) one application of V. (c) one application of H. (d) both V and H applied, order does not matter and is equivalent to two applications of Rot90. The arrows in (b) and (c) demonstrate how one application of a flip creates a mirror drawing (left node is right and vice versa). (e) the mirror of (a). (f) one application of V on (e) and the in-order equivalent of (b). (g) one application of H on (e) and the in-order equivalent of (c). (h) both V and H applied on (e), where order does not matter.

3.2.6 Achieving Good Aspect Ratio

Let T_k be a partial tree with n_k nodes originating from T with n nodes. Its drawing has width w_k and height h_k, where these dimensions are defined as the maximum number of cells occupied on the grid horizontally and vertically. Therefore a null tree has a drawing with width and height of zero and a trivial tree has a width and height of one. If a non-null drawing has its top-left corner at the origin and has width w, it occupies the grid points along the x-axis from 0 to w-1, so a second drawing of width w' can be placed starting at grid point w and there will be a single unit space between the drawings though the combined width of the drawings will be $w + w'$. This allows adjoining drawings to have a total width or height that is the sum of their separate widths and heights though visually they still have a gap of one unit between them.

At a recursive level of the algorithm there are N partial trees that are either placed horizontally or vertically next to each other. If there were minimal spacing between these partial trees, then a horizontal placement would give for the recursive level an overall height of $\max(h_k)$ and an overall width of $\text{sum}(w_k)$ while a vertical placement would give an overall height of $\text{sum}(h_k)$ and an overall width of $\max(w_k)$ for $1 \leq k \leq N$. Given a desired aspect ratio A for the current recursive level, one has depending on the placement, $A = \text{width/height} = \text{sum}(w_k)/ \max(h_k)$ or $A = \text{width/height} = \max(w_k)/\text{sum}(h_k)$.

Therefore for $x_k = n_k / n$,

$$A_k = x_k A \text{ for horizontal placement}$$

and

$$A_k = (1 / x_k) A \text{ for vertical placement}$$

since one can arbitrarily split these into $w_k = n_k$ and $h_k = n / A$ or $w_k = n A$ and $h_k = n_k$ giving:

$$\text{sum}(w_k)/ \max(h_k) = \text{sum}(n_k) / (n / A) = (n / n) A = A$$

or

$$\max(w_k)/\text{sum}(h_k) = n A / \text{sum}(n_k) = (n / n) A = A$$

If n_k is small enough, then A_k can be approximated by the edges of the good aspect ratio range to give $A_k = n_k^{-\varepsilon}$ for horizontal placement and $A_k = n_k^{\varepsilon}$ for vertical placement. This ensures that A_k keeps a good aspect ratio by avoiding smaller values from entering into the division. The threshold for determining if a partial tree is large is:

$$A \geq 1, n_k \geq (n / A)^{1 / (1 + \varepsilon)}$$

$$A < 1, n_k \geq (n A)^{1 / (1 + \varepsilon)}$$

In [32], the above scheme is used where horizontal placement is used when $A \geq 1$ and vertical placement for the other case. In the algorithm of this chapter, there are times when either horizontal or vertical placement is used for any A. This would only cause a problem if there were no variety of placements as the aspect ratio would become unachievable through constant stretching of one of the directions. The situation that causes this problem only occurs in special cases denoted in the algorithm under the β-scope section. Using only these special cases would cause the algorithm to fail, but since they only occur in one scope, the problem of starting in one of these special cases is eliminated since the algorithm does not start from within this scope.

3.3 The Algorithm

Let T be the tree to draw with n nodes, root o, and link node u^*. Let $0 < \varepsilon < 1$ be a constant and let $n^{-\varepsilon} \leq A \leq n^{\varepsilon}$ be the *desirable aspect ratio* of T. The algorithm uses a simple divide-and-conquer strategy to construct Γ, a feasible drawing of T. The algorithm consists of a modification of the original steps in [32]:

- Split Tree: Split T into several partial trees by removing select nodes and their incident edges. No partial tree has more than (2/3) n nodes due to an application of Theorem 1 described earlier. There are four scopes in the algorithm described later, which specify four subgraphs of T that are constructed using different assumptions then brought together in an overarching decomposition.
- Assign Aspect Ratios: Using the aspect ratios for partial trees described earlier, assign each partial tree a desirable aspect ratio A_k.

- Draw Partial Trees: Recursively create feasible drawings for each of the partial trees T_k using their desirable aspect ratios A_k.
- Compose Drawings: Arrange the feasible drawings and replace the nodes and edges removed in the split in a way described later that creates Γ, a feasible drawing of T. In most arrangements, when $A < 1$ the partial tree drawings are stacked and are placed side-by-side in the other case. There are some arrangements that always require the partial trees to be placed either horizontally or vertically.

Γ will not necessarily have aspect ratio A, but will fit within rectangle R with the desired aspect ratio as will be demonstrated in the proof of correctness later. Each of the steps mentioned will now be detailed.

3.3.1 Split Tree

Let T be the tree rooted at o with leftmost node lm. A and ε have been chosen by the user. Using Theorem 1, find the separator edge between nodes u and v, where $u = p(v)$. The algorithm begins in the *general scope* where the three feasibility properties are restricted to those found in [32]:

- **Property 1**: The root o is placed on the top-left corner of Γ.
- **Property 2**: If u^* is not o, then u^* is placed on the bottom boundary of Γ and can be moved downward an arbitrary amount.
- **Property 3**: If u^* is o, then u^* is placed in the top-left corner of Γ and can be arbitrarily moved in either the horizontal or in the vertical directions.

Also, $lm = u^*$ and $a = a(u^*, u)$. There are two cases for a, either $a \neq u$ implies u is not an ancestor of u^* or $a = u$ implies u is an ancestor of u^*. If $a = u$ and $l(u) = v$, then the separator edge occurs in the leftmost path of T. [32] denotes this case as Case 2. Here this situation is presented first since it requires no additional scopes, which are different algorithms that are applied to specific subtrees depending on the characteristics they need to achieve as will be shown at the beginning of the section of each scope. Since Γ_M is needed to flip Γ so as to preserve ordering of siblings, flips cannot be used on partial trees A and C in the following, since Γ_M does not have the property $lm = u^*$, though B never connects to other partial trees through a link node so it can be flipped. This leads to a slightly different description of the algorithm than what is provided in [32] where the changes are incorporated to avoid flips in A and C.

3.3.1.1 General Scope

Case 1 (*separator edge in leftmost path*)

A-B

In subcase A, $a = u$ and v is the root of C. The link node of partial tree A must be the leftmost node of A, as desired, since a is on the leftmost path of T. $u^* = lm$ so it is obviously the valid link node for C. $r(u)$ exists as well in subcases A and B. In subcase B, v is the root of C and also $v = u^*$; since it is drawn in the top-left corner of C, it must be moved down to the bottom boundary. See Figure 19 for when $A < 1$ and see Figure 20 for when $A \geq 1$. The star in these figures and all following figures represents u^*. Figure 19 is straight-forward, but Figure 20 requires some more explanation. There are four options in this case based on whether $o = p(a)$ and $v = u^*$ or not. For Figure 20(a), A' and o are together A, but have been split and considered separately to fit the situation. For Figure

20(b), o is exactly A (A is trivial). For Figure 20(a-b), if $v = u^*$, then it must be moved to the bottom boundary and is the bottom-left corner of C as was mentioned previously in this section. For Figure 20(c), the link node of A must be extended downward so as to be level with a (represented in the picture with a stretched box) and a vertical flip must be applied to B so that its root is on the bottom boundary. Figure 20(d) is when $v = u^*$, so v is the bottom-left corner.

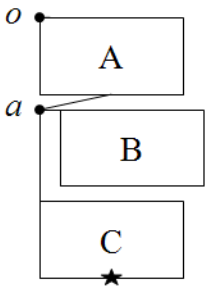


Figure 19. $A < 1$. $a = u$. $l(u) = v$. v is root of C. v may be u^*.

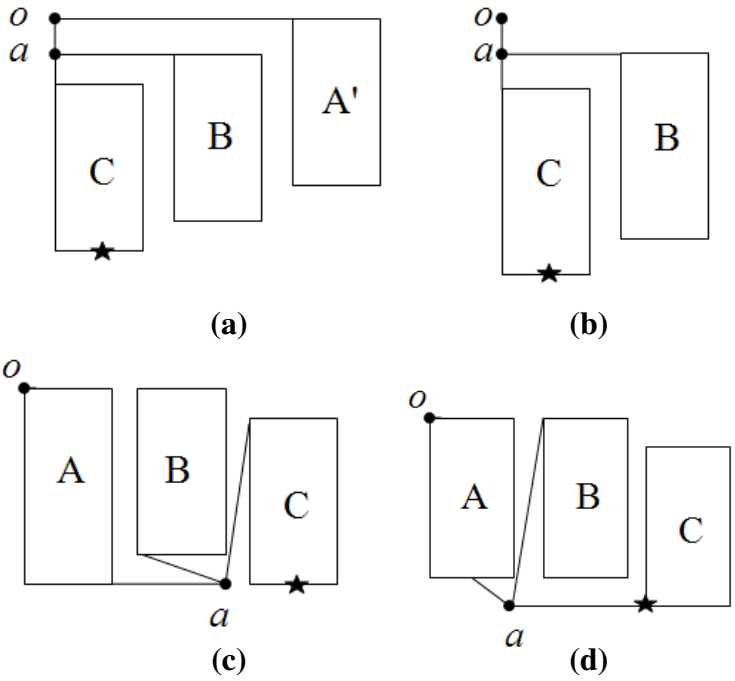


Figure 20. $A \geq 1$. $a = u$. $l(u) = v$. v is root of C. (a): $o = p(a)$ and $r(o)$ exists. (b): $o = p(a)$ and $r(o)$ does not exist. (c): $o \neq p(a)$ and $v \neq u^*$. (d): $o \neq p(a)$ and $v = u^*$.

C-D

In subcases C and D the partial tree B is null. Subcase C has $v \neq u^*$ and subcase D has $v = u^*$. See Figure 21 for when $A < 1$ and see Figure 22 for when $A \geq 1$. The rationale for these cases is apparent from the discussion in subcases A and B.

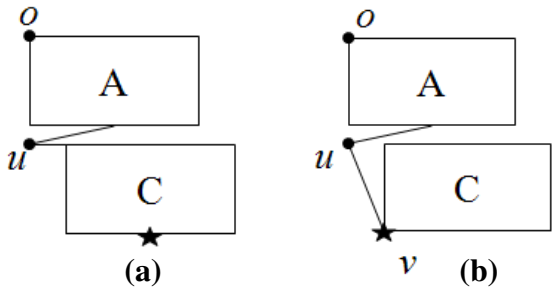


Figure 21. $A < 1$. (a): $v \neq u^*$ (subcase C). (b): $v = u^*$ (subcase D).

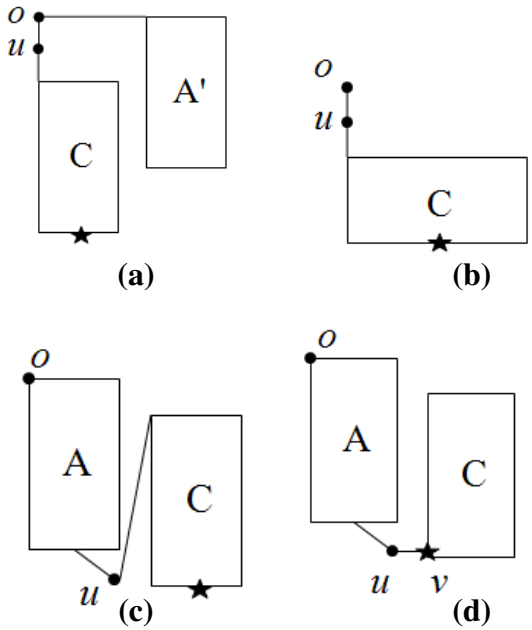
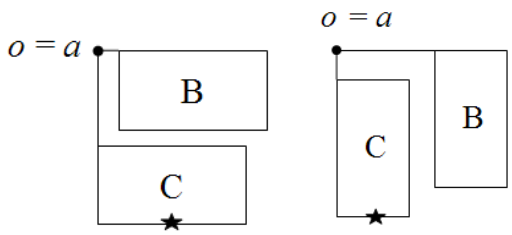


Figure 22. $A \geq 1$. (a): $o = p(u)$ and $r(o)$ exists. (b): $o = p(u)$ and $r(o)$ does not exist. (c): $o \neq p(u)$ and $v \neq u^*$. (d): $o \neq p(u)$ and $v = u^*$.

E-F

In subcases E and F the partial tree A is null. Subcase E has $v \neq u^*$ and subcase F has $v = u^*$. See Figure 23. The rationale for these subcases is straightforward.

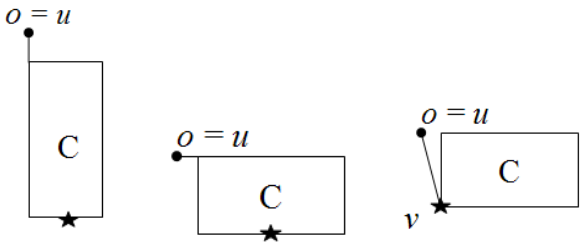


(a) **(b)**

Figure 23. (a): $A < 1$. (b): $A \geq 1$. In both options, when $v = u^*$ the root of C moves to the bottom-left corner.

G-H

In subcases G and H the partial trees A and B are null. Subcase G has $v \neq u^*$ and subcase H has $v = u^*$. See Figure 24. The rationale for these cases is straightforward.



(a) **(b)** **(c)**

Figure 24. (a): $A < 1$. (b-c): $A \geq 1$. When $v = u^*$ in (a), the root v of C becomes the bottom-left corner of C.

Case 2 (*separator edge not in leftmost path*)

Note that in Case 2 the B-scope (detailed later) is needed since the separator edge is within B (unless v is the root of B, though the B-scope handles that case by returning a general scope drawing). It is important that the separator edge is split at each recursive level, so even though the B-scope may look like a separate recursive level, its first level is processed at the same recursive level as the current level. This type of scope will be

referred to as *immediate*. Scopes that do not start until the next recursive level, like the general scope will be referred to as *delayed.* The general scope and β-scope are delayed. The B-scope and α-scope are immediate.

A-B

This is exactly the same set-up as in Case 1 subcases A and B. See Figures 19 and 20. The only difference here is that a may not be u and v is contained within B instead of C. This requires that the partial tree B is sent to the B-scope (detailed later) so that it is split immediately.

C-D

In subcases C and D, partial tree C is null, thus $a = u^*$. In subcase C, $o \neq p(a)$. In subcase D, $o = p(a)$. See Figure 25 for when $A < 1$ and see Figure 26 for when $A \geq 1$. In Figure 25, subcase D works because feasibility property 3 ensures that the vertical channel directly below o is empty.

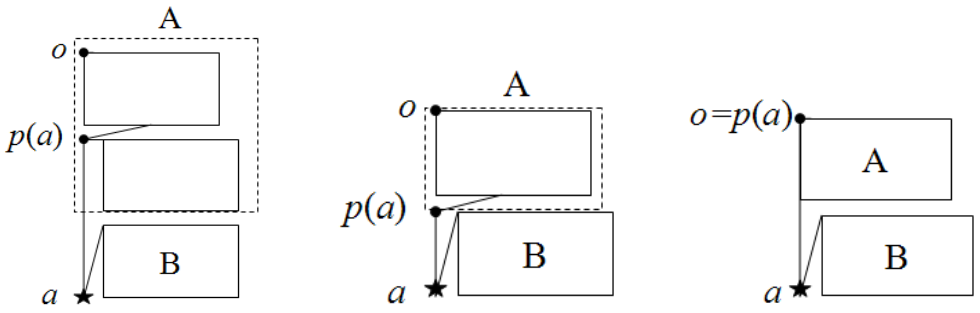


Figure 25. $A < 1$. (a-b): subcase C. (c): subcase D. In subcase C, $p(a)$ is not guaranteed to be on the left boundary, so A must be split and its pieces considered separately.

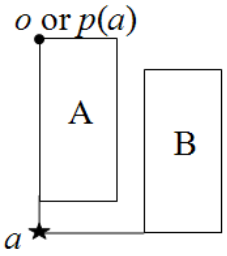


Figure 26. $A \geq 1$. In subcase C, $o \neq p(a)$ so $p(a)$ is actually on the bottom boundary (not necessarily the bottom-left corner as shown). In subcase D, $o = p(a)$ so by feasibility property 3, the vertical channel below o is empty allowing a to connect through the left boundary. A vertical flip is applied to B so that it connects on the bottom-left corner.

E

In subcase E, partial trees A and C are null, therefore $a = u^* = o$. See Figure 27.

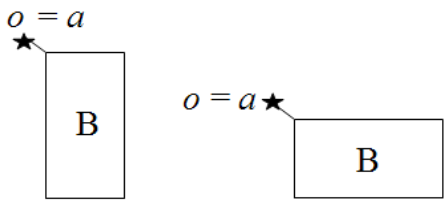


Figure 27. (a): $A < 1$. (b): $A \geq 1$.

F-G

These have the same set-up as subcases E and F in Case 1. See Figure 23.

3.3.1.2 B-Scope

- **Property 1**: The root o is placed on the top-left corner of Γ.
- **Property 2 / Property 3**: u^* is a do not care condition so these are ignored.

As will be shown later, the β-scope only guarantees that o is placed on the left boundary and that the vertical channel above it is empty, so β cannot be the partial tree from o of B to $p(u)$ as in [32]. Instead it is the partial tree from a child of o of B to $p(u)$. Also $p(u)$ cannot be the root of β, since the link node cannot equal the root in that scope. β is a

delayed scope and α is an immediate scope. Any unlabeled partial trees are general scope and are sibling subtrees to the subtree mentioned in the figure notes (e.g. Figure 29).

Case 1 ($u = o$ or $v = o$)

See Figure 28. Remember that general scope is delayed and α-scope is immediate.

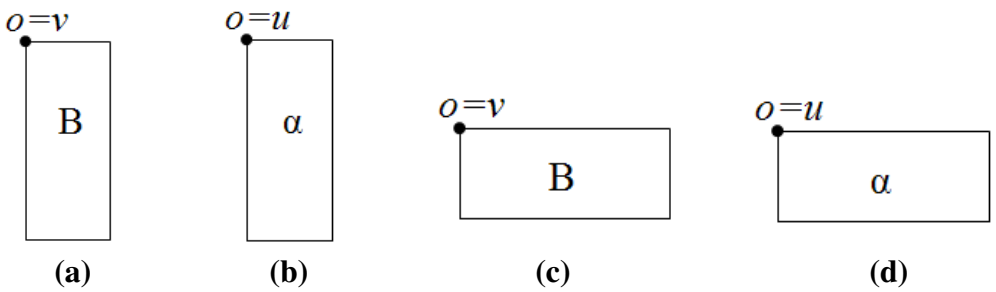


Figure 28. (a): $A < 1$. $o = v$. General scope drawing. (b): $A < 1$. $o = u$. α-scope drawing. (c): $A \geq 1$. $o = v$. General scope drawing. (c): $A \geq 1$. $o = u$. α-scope drawing.

Case 2 ($o = p(u)$ or $o = p(p(u))$)

If $o = p(u)$ then β does not exist. If $o = p(p(u))$ (shorthanded as $o = p^2(u)$) then β must be specially handled since the link node cannot equal the root in β-scope. The figure called αβ is a logical convention that does not exist in practice and only exists in the figure for simplification. Figure 29 shows the construction of αβ when $o = p(u)$ and Figure 30 shows simplified diagrams for when $o = p^2(u)$ where the partial trees marked αβ are actually one of the diagrams in Figure 29.

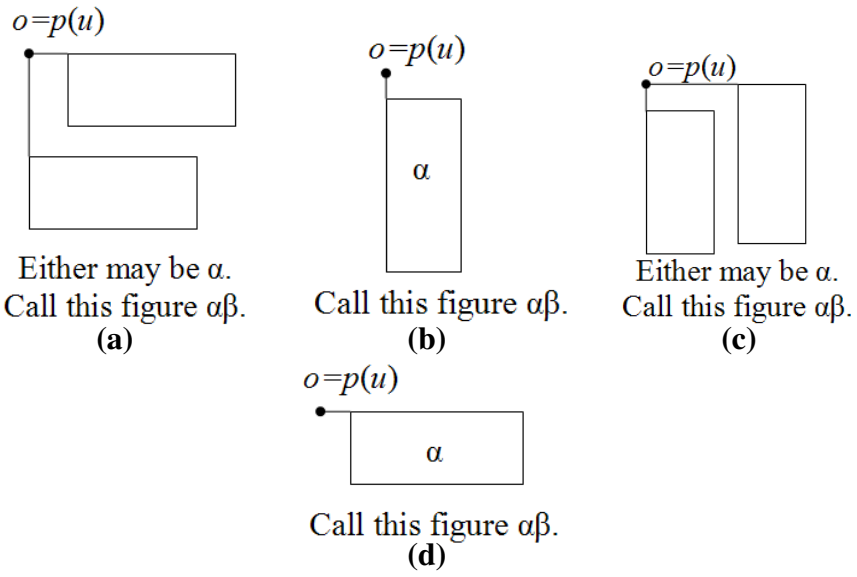


Figure 29. (a-b): $A < 1$. (c-d): $A \geq 1$. Where either subtree may be α-scope, the other is its sibling and is general scope.

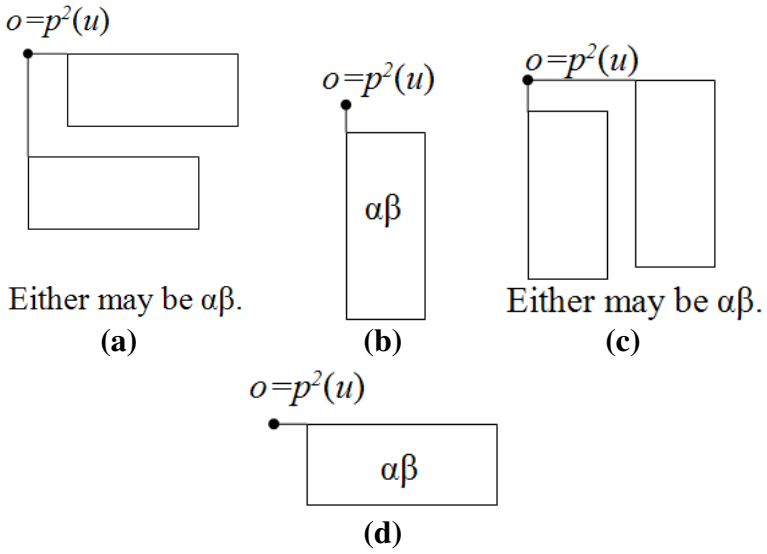


Figure 30. (a-b): $A < 1$. (c-d): $A \geq 1$. Where either subtree may be αβ, the other is its sibling and is general scope.

Case 3 (β-*scope is used*)

Since the root of β exists on the left boundary, it is necessary to know the northern or southern extent of β from its root in addition to its height so it can be appropriately

spaced. See Figures 31 and 32 for the different aspect ratio scenarios. In Figure 31, if $p(u)$ is on the right boundary of β, then use a horizontal flip on α so that the root is in the top-right corner. In Figure 32, use a 90 degree rotation and a vertical flip on β to move its root to the upper boundary. Since the vertical channel is clear above the root in the initial image, the horizontal channel is clear to the left of the root in the rotated and flipped image. The connections to the root of β that appear to attach to the top-left corner actually are connecting to the top boundary along the empty horizontal channel. If the link node of β is on the bottom boundary, it should be shifted so that if α had a larger height than β, β now has the same height as α. α only needs to be vertically flipped when the link node is on the bottom boundary, though for convenience it is always flipped vertically in this case.

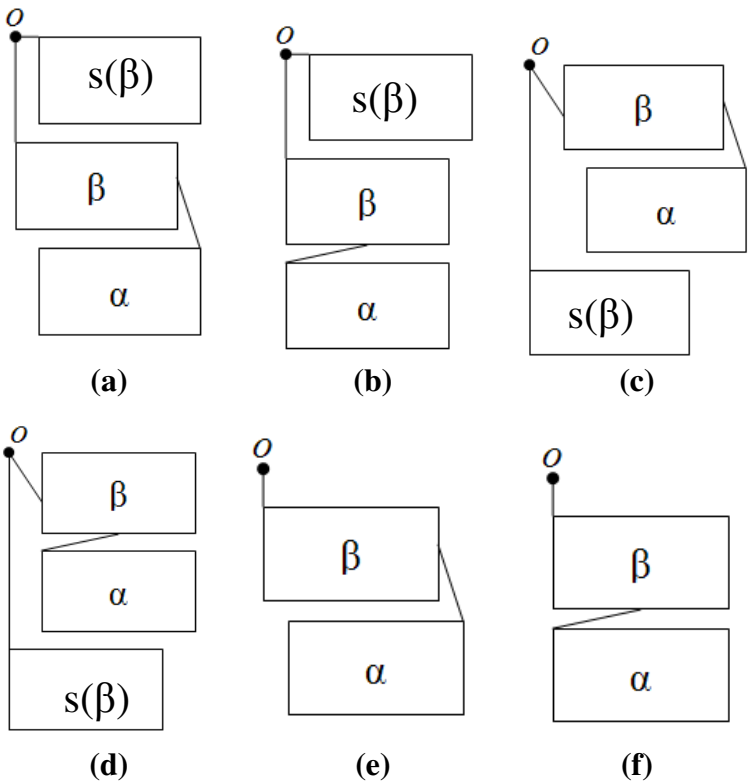


Figure 31. $A < 1$. (a-b): β connects to the root on the left and $p(u)$ may exist on either of two boundaries. (c-d): β connects to the root on the right and $p(u)$ may exist on either of two boundaries. (e-f): the root of β has no sibling.

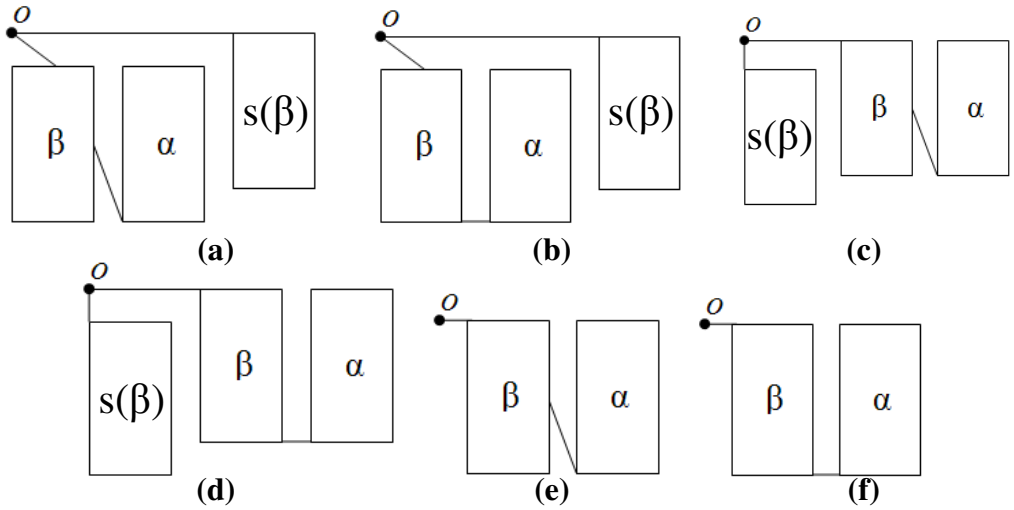


Figure 32. $A \geq 1$. (a-b): β connects to the root on the left and $p(u)$ may exist on either of two boundaries. (c-d): β connects to the root on the right and $p(u)$ may exist on either of two boundaries. (e-f): the root of β has no sibling.

3.3.1.3 α-Scope

- **Property 1**: The root *o* is placed on the top-left corner of Γ.
- **Property 2 / Property 3**: u^* is a do not care condition so these are ignored.

This scope is an immediate scope used to draw the subtrees below *u* in the B-scope, so it is the simplest scope to describe. See Figure 33.

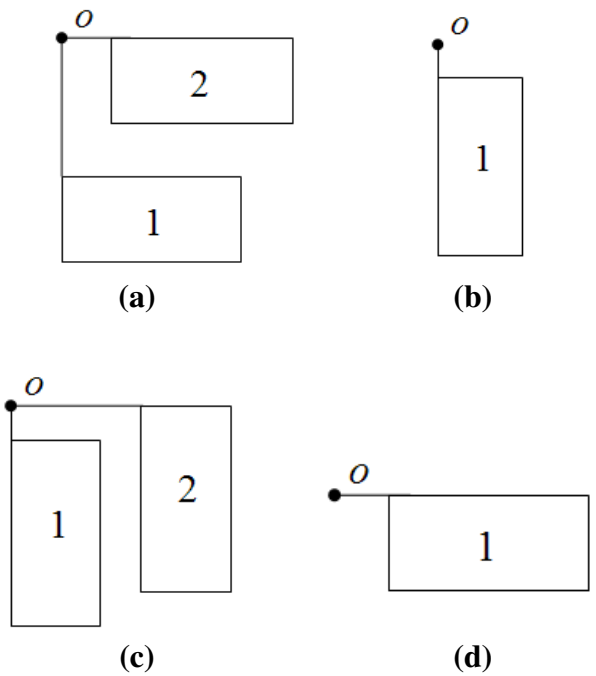


Figure 33. (a-b): $A < 1$. (c-d): $A \geq 1$. When both subtrees exist, 1 is the subtree on the left and 2 is the subtree on the right. When only one exists, 1 is the subtree that exists (and has root *v*).

3.3.1.4 β-Scope

- **Property 1**: The root *o* is placed on the left boundary of Γ and the vertical channel above it is clear.
- **Property 2**: If $u^* \neq o$, then u^* is either placed on the bottom boundary of Γ and can be moved downward an arbitrary amount or is placed on the right boundary and can be moved rightward an arbitrary amount.

- **Property 3**: $u^* = o$ is explicitly handled before entering this scope and within it, therefore at the beginning of every recursive level, this condition is impossible.

In this scope $a = a(u^*, v)$. If $a = u^* = v$ then $u^* = v$. If $a = u^*$ then u^* is an ancestor of v. If $a = v$ then u^* is a descendent of v. If a is neither, then one subtree of a is β-scope (or explicitly handled if the subtree is rooted at u^* since this condition is not allowed in β-scope) and the other subtree is B-scope. Also, the cases are checked in order, for example if $u^* = a = lm$, then Case 1 is used instead of Case 3.

Case 1 ($u^* = lm$)

The drawing belongs to the general scope. Refer to that section.

Case 2 ($u^* = rm$)

Use a 90 degree rotation and vertical flip to produce an in-order general scope drawing with the root at the top-left corner and the link node on the right boundary.

Case 3 ($u^* = a$)

See Figure 34. Figure 34(a,c) have u^* with no children (thus making $a = u^* = v$). The dotted lines represent the fact that the edge will either come from the bottom boundary or the right boundary. Figure 34(b,d) have $r(u^*)$ exists. Figure 34(e) has any aspect ratio in which $l(u^*)$ exists. Figure 34(f) has any aspect ratio in which $r(o) = a$. In this case, $r(a)$ must exist or else $a = u^* = rm$, which is already handled by Case 2. If $o = p(a)$, then one of Figure 34(e-f) must be used, since all other options lead to either $u^* = rm$ or $u^* = lm$, which would have already been handled in Case 1 or Case 2. It is allowable to use β-scope for the bottom-left option when $o = p(a)$, since it is guaranteed that $l(o) = a$ thus making it a Case 1 drawing (in the general scope).

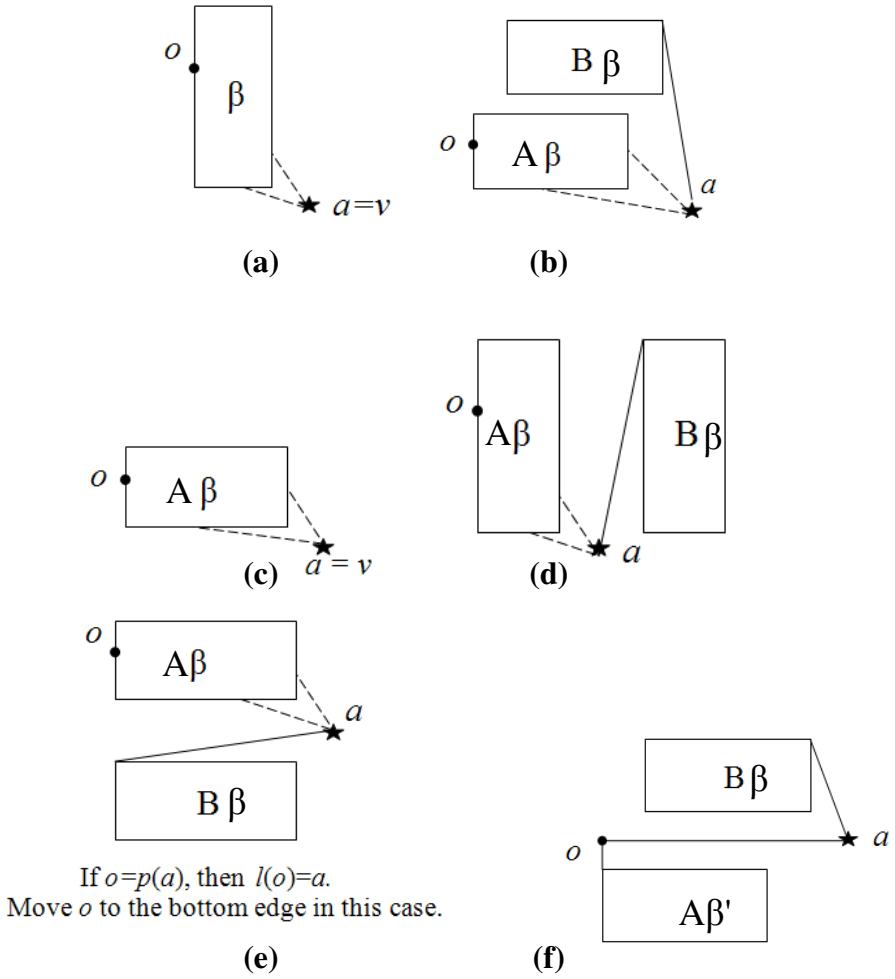


Figure 34. (a-b): $A < 1$. (c-d): $A \geq 1$. (e-f): any aspect ratio. Aβ is drawn with β-scope and Bβ is drawn with B-scope. These are labeled as such to relate to similar roles in the general scope diagrams. (b-f): Bβ is horizontally flipped. (f): Aβ' and *o* make Aβ.

Case 4 ((1) *o* = *a*. (2) *r*(*o*) = *a and l*(*o*) *exists*. (3) *l*(*o*) = *v* = *a*. (4) *l*(*o*) = *a and r*(*o*) *exists and the right subtree of a contains the separator edge and A* ≥ **1)

See B-scope Case 2 for a discussion of the αβ diagram simplification. This case uses a similar simplification called the Root diagram. See Figure 35 for the construction of the Root diagram ($o = a$) and Figure 36 for when $o = p(a)$. The two stars on the β subtree represent the fact that the link node may appear on the bottom or right boundary.

Also the β subtree must be stretched so that the link node is on the bottom or the right boundary of the total drawing.

After this case, it is guaranteed that a is not the root. Also a is only $r(o)$ if it is a single child. Figure 36(c) allows for a horizontal placement for a specific situation that arises when $A \geq 1$. Figure 36(b) exists to allow a simple solution to $l(o) = v = a$.

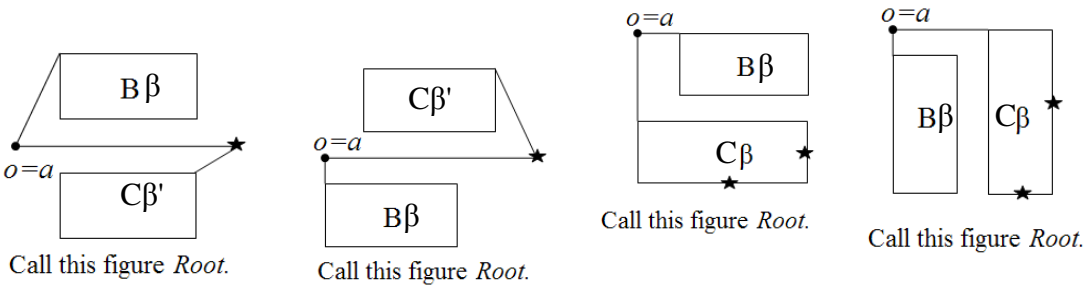


Figure 35. All options are for any aspect ratio. (a-b): Cβ' and the link node together make Cβ; also use a horizontal flip on Cβ'. (d): use a 90 degree rotation and vertical flip on Cβ so its root is on the top boundary.

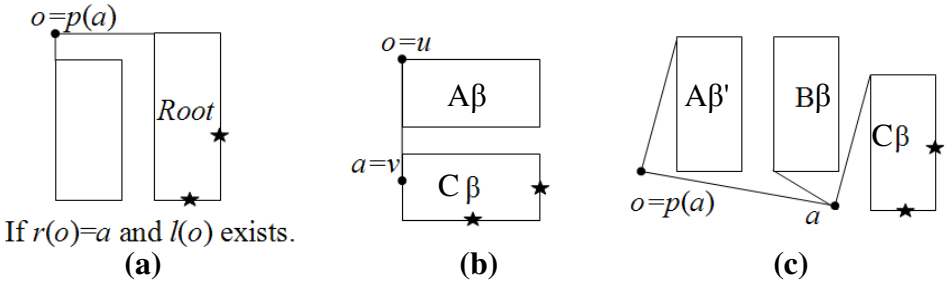


Figure 36. (a-b): any aspect ratio. (c): $A \geq 1$. (a): use a 90 degree rotation and vertical flip on Root when it is created from Figure 35(a-b). (b): works due to feasibility property 3 in the general scope guaranteeing the vertical channel below the root is empty when it is the link node. (c): use a vertical flip on Bβ so its root is on the bottom-left corner.

Case 5 ($A < 1$)

See Figure 37. Since $o = p(a)$ only when $l(o) = a$ or a is an only child, the partial tree containing the root will either have a right subtree in which the root has to be moved to the bottom boundary since it will also be the link node, or this partial tree is trivial. $a = v$ is specially handled if a has an only child or is bottom-connected to the partial tree rooted at o. This represents Figure 37(k-o). Figure 37(a-f) detail the different configurations when the right subtree of a contains the link node. Figure 37(a-c) detail when the right subtree of a has a root that is not the link node (left subtree of this root contains link node, only one subtree of the root, right subtree of this root contains link node, in that order). Figure 37(d-f) detail when the right subtree of a has a root that is the link node (right subtree exists, left subtree exists, the drawing is trivial, in that order). Figure 37(g-j) detail the different configurations when the left subtree of a contains the link node. The first of these details when the root is not the link node and the rest of these detail when the root is the link node (left subtree exists, right subtree exists, the drawing is trivial, in that order). The caption of Figure 37 explains how to handle Cβ(') when the root is also the link node.

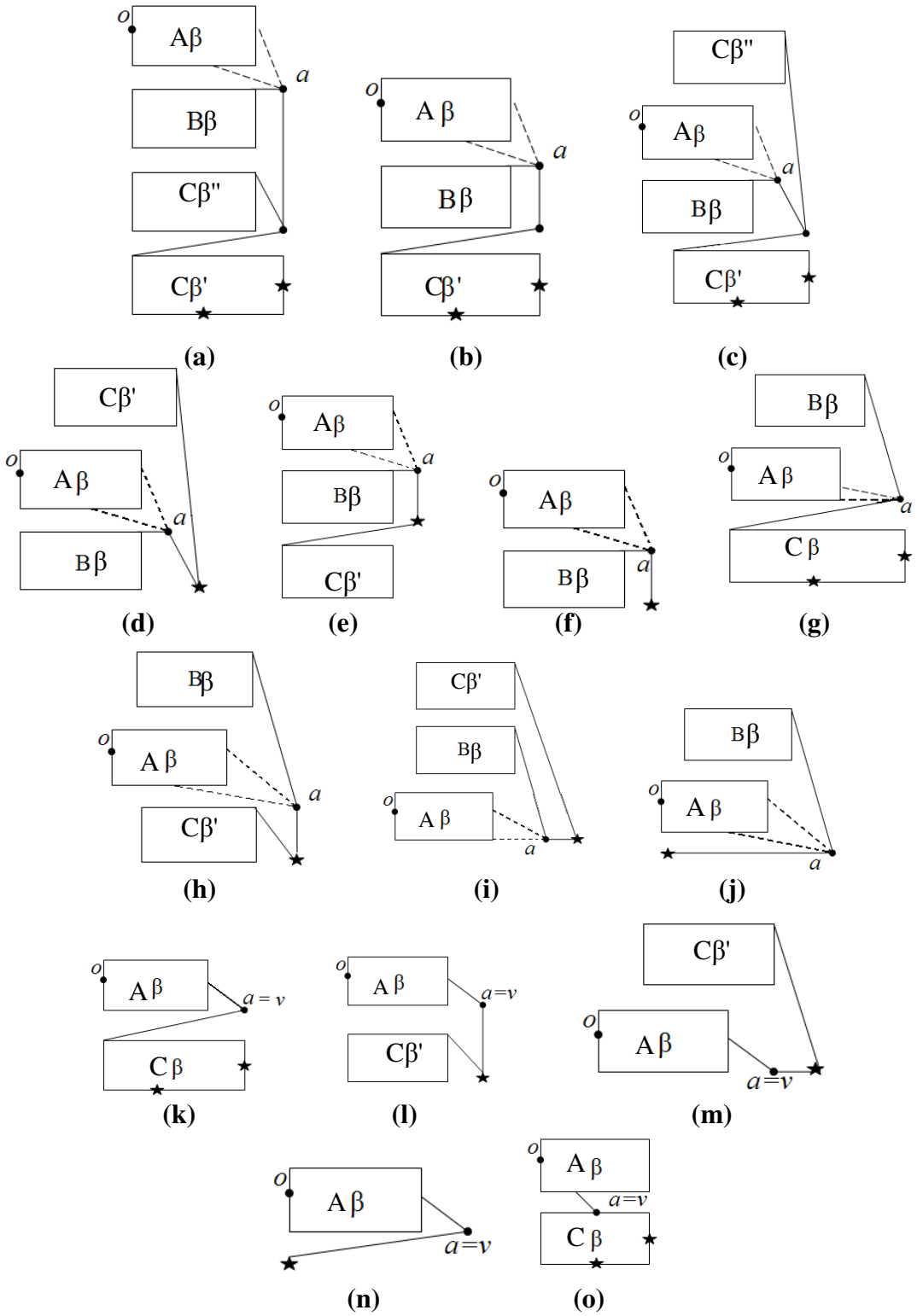


Figure 37. A < 1. (a-f): have the left subtree of *a* containing the separator edge. (g-j): have the right subtree of *a* containing the separator edge. (k-n): have $a = v$ and *a* has no

siblings. (o): has $a = v$ and has a sibling and is bottom connected to the partial tree with root o. If $a = v$ and has a sibling and is right connected to the partial tree with root o, then one of the first 10 options is used. A horizontal flip is used for any subtree that is connected at the top-right corner. A 90 degree rotation and vertical flip are used on Cβ(') when it needs to be connected at its top boundary. If Cβ has the link node at its root it can be safely drawn if its root has no children or a right child. If it has a left child it can be safely drawn (Case 2) as long as the root is moved to the right boundary (by feasibility property 3). Bβ and Cβ('/'') may need to be horizontally flipped so they can be connected from the right.

Case 6 ($A \geq 1$)

See Figure 38. Since $o = p(a)$ exists only when a is a single child of o the drawing in this instance is trivial, allowing the correct flipping of β, which would otherwise be unachievable.

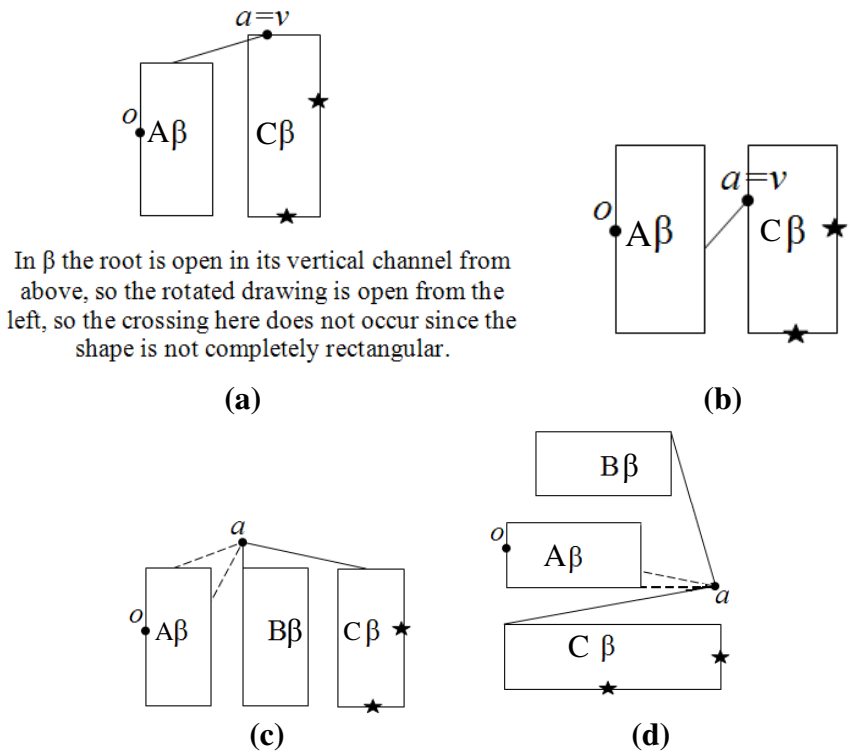


Figure 38. $A \geq 1$. (a-c): Aβ is vertically flipped and o is moved left one so the vertical channel above it after the flip is guaranteed to be empty. Cβ has a 90 degree rotation and a vertical flip performed on it so that it is top-connected in (a, c, d). Bβ is horizontally flipped in (d).

3.3.2 Assign Aspect Ratios

The aspect ratios assigned to each partial subtree were described earlier in Preliminaries. If the split requires a vertical placement, then the vertical placement aspect ratios are assigned. If the split requires a horizontal placement, then the horizontal placement aspect ratios are assigned. It should also be noted that using the vertical and horizontal flips do not affect aspect ratio, though an odd number of 90 degree rotations switches the placement of width and height. Therefore, whenever rotating, the assigned aspect ratio should be $1/A_k$ to assure the drawing after the rotation has aspect ratio A_k.

3.3.3 Draw Partial Trees

The partial trees T_k as described by the *Split Tree* process are recursively drawn using desirable aspect ratios A_k, or the multiplicative inverses, as discussed in *Assign Aspect Ratios*. The base case is the drawing of the trivial tree, a single node represented by a single dot.

3.3.4 Compose Drawings

Referring back to the *Split Tree* process, the drawings are ordered in relation to each other and the removed nodes and their incident edges are placed back into the drawing. The spacing between drawings of partial trees and nodes is minimal unless more space is needed as obviated by the figures in *Split Tree* so that the link node and root are both guaranteed to be on a boundary of the overall drawing at each recursive level. All edges in the figures that appear to be drawn in the horizontal or vertical channels are actually drawn in those channels. Any edge that is not guaranteed to be in one of these channels is drawn on a slant in the figures. The partial trees are labeled with what scope

they were drawn with except for partial trees 1, 2, A, C which are general scope partial trees with naming convention matching [32].

3.4 Proof of Correctness

Lemma 1: *The general scope satisfies its feasibility properties.*

Proof:

The trivial tree is trivially feasible. For any larger tree, by inspection of the general scope, the root is always placed in the top-left corner when explicitly present (Figure 20 options 1 and 2, Figure 22 options 1 and 2, Figure 23, Figure 24, Figure 27). When it is implicit (partial tree A exists), A's root is the root of the overall drawing. A is drawn with general scope, therefore it will eventually resolve into one of the drawings in the general scope with the root explicitly present. Therefore the root is always in the top-left corner when using general scope.

Figure 27 is the only time when the root explicitly equals the link node. Property 3 of the general scope is satisfied here. This also shows that in this situation the link node can be made to be in the bottom boundary. Figures 25 and 26 are the only times the link node is explicitly present. Both figures draw the link node on the bottom boundary where it can obviously be moved downward freely. All other figures in the general scope show that the bottom boundary of partial tree C is placed on the bottom boundary of the drawing at that level, moving the root of C to the bottom boundary when it is also the root of C. Therefore the implicit drawings always have C's bottom boundary on the bottom boundary and when they resolve to the explicit cases they are drawn on the bottom boundary or can be moved there by feasibility property 3 of the general scope. Therefore Property 2 is satisfied since the implicit cases always resolve to the explicit

cases and the link node is on the bottom boundary every time or can be moved there when it is also the root of C. All three feasibility properties are therefore satisfied.

Lemma 2: *The* α*-scope satisfies its feasibility property.*

Proof:

The root is explicitly placed in the top-left corner in the α-scope [Fig. 33] so every drawing in this scope satisfies the feasibility property of α-scope.

Lemma 3: *The* B*-scope satisfies its feasibility property.*

Proof:

Figure 28 shows that Case 1 resolves to either the general scope or the α-scope, so by Lemma 1 and Lemma 2 the root is in the top-left corner for this case. Figures 29 to 32 explicitly place the root in the top-left corner so every drawing from these figures places the root in the top-left corner. Every drawing of the B-scope thus satisfies the feasibility property of B-scope.

Lemma 4: *The* β*-scope satisfies its feasibility properties.*

Proof:

Case 1 and Case 2 are resolved by Lemma 1. The top-left corner belongs to the left boundary and the vertical channel above this is obviously empty. Case 1 places the link node in the bottom boundary and Case 2 places the link node in the right boundary due to rotation. Since the link node can move downward before rotation, it can move rightward after rotation. In Case 3 the link node is explicitly drawn in either the bottom or the right boundary (the bottom-right corner counts as either) and by examination can be shown to be movable in either the downward direction when on the bottom boundary or the rightward direction when on the right boundary. By the same reasoning it can be

shown that the explicit link node placements in Cases 4 and 5 also share this property. Every implicit placement of the link node can be seen to touch the bottom and right boundary simultaneously by stretching the link node rightward when on the right boundary and downward when on the bottom boundary. Therefore, the link node in this scope is guaranteed to obey β-scope feasibility property 2. For every case except Case 6, it is easy to see that the vertical channel above the root is clear and that it is placed on the left boundary in every instance therefore satisfying property 1. Since the root is on the left boundary it is safe to move it left one space to create an empty vertical channel beneath it so that when it is flipped vertically in Case 6 it is still true that the root is on the left boundary and has an empty vertical channel above it. Therefore every drawing satisfies β-scope feasibility property 1. It is explicitly stated with explanation when it was safe to enter the β-scope with $u^* = o$, which is only from within the β-scope itself in certain situations. It is also mentioned how u^* is restricted from being o before entering the β-scope from a different scope. Therefore feasibility property 3 indeed can be ignored.

Lemma 5 (Feasibility): *The drawing of a tree using this algorithm is feasible.*

Proof:

The trivial tree is trivially planar. For any nontrivial tree, by Lemmas 1-4 every scope maintains its feasibility properties. In the general scope, the only other scope visible is the B scope, which acts like a general scope drawing since its root is also at the top-left corner and it never links to anything by its link node. It is easy to see that no edge crossings can happen while connecting to the root of a partial tree in the general scope or the B-scope when it connects from above or from the left due to feasibility property 1 in

these two scopes. From feasibility properties 2 and 3 of the general scope, the link node can always be moved to the bottom edge. By these properties it is obvious that no edge crossings occur when they connect from below. It may appear that the left option in Figure 20 does not connect below A, but A is drawn stretched which means that the link node was moved below the rest of the image so it can connect horizontally in this case. Knowing now the exception in Figure 20 as well as how A must connect below and B and C must connect at either the top-left corner (or bottom-left corner after a vertical flip or moving the root to the bottom edge) it can be seen from the figures for the general scope that this indeed happens. By inspecting the edges around *a*, it can be seen that they cannot conflict with each other since they all stem from *a* and go in different directions. They also do not interfere with the partial trees since they are shown to only touch the drawings of the partial trees at their connection points, which were just demonstrated as being conflict-free points. The general scope must produce only planar drawings, and since every edge is drawn straight as well the general scope must produce feasible drawings.

Convincing oneself that the general scope produces feasible drawings is the most challenging part because there is no basis to build from, though once convinced that the general scope indeed works the other scopes are much simpler to prove as feasible. The α-scope is comprised of at most two general-scope subtrees and at least one general-scope subtree. Both cases are analogous to Figure 23 and the first two options of Figure 24, which were shown to be feasible drawings in the general scope, so immediately the α-scope must be feasible.

The first two cases in the B-scope are readily demonstrated as feasible. The six options in Case 3 for $A < 1$ enumerate the configurations of the β subtree either connecting to the root from right or left and connecting to α from bottom or right and whether the auxiliary tree is present or not. The other six options for $A \geq 1$ are analogous to the first six options. From the description of this case, it is seen that the rotations and reflections justify where the partial trees are being connected, remembering that β-scope guarantees root placement on the left border with the vertical channel open from above. For $A \geq 1$, a 90 degree rotation and vertical flip are used to move the root to the top boundary with the left horizontal channel open so that the root can be connected horizontally from the left. These operations will move the link node from either right boundary to bottom boundary or vice versa. The α subtree is flipped either vertically or horizontally so that it can connect from the bottom or right edge of β simultaneously. After understanding how β and α fit together to produce no crossings, the same techniques from the general scope can be applied to show that no violations occur between the root, auxiliary subtree and β. It is then shown that B-scope is feasible.

The β-scope can also be shown to be feasible using the same reasoning that was used to demonstrate feasibility for the other scopes. Cases 1 and 2 are immediately feasible. The other cases can be demonstrated exactly as before. The drawings of the partial trees do not touch. The connection points occur on edges so they do not cause interference with the internal parts of the partial trees, the edges cannot cross each other because they all join at a common area and are pointed in differing directions, and the edges only touch the partial trees at their connection points unless explicitly stated in the

algorithm that a channel is guaranteed open along an edge as in the B-scope when the root of β is on the top edge and the horizontal channel is guaranteed open to the left.

The proof started with the feasibility properties of each scope and showed how these were sufficient to guarantee edges at a recursion level will not cross within drawings of partial trees. Then the edges were shown to not cross each other relying on the diagrams of each case of the algorithm. These together offer a proof that any drawing produced by this algorithm is planar. No edge is ever bent in any diagram and since the diagrams are planar, they never need to be bent to guarantee planarity so the algorithm also has straight-line edges. These were the two conditions necessary for the algorithm to be deemed feasible.

Lemma 6 (Time): From [32] Lemma 2:

> "*Given an n-node binary tree T with a link node u**, [*the a*]*lgorithm* […] *will construct a drawing* Γ *of T in O*(*n* log *n*) *time.*"

This statement is applicable to the algorithm of this paper as well since it uses the same recursion depth and each recursive step is linear in time as is also the case in [32].

Lemma 7 (Ordering): *The drawing of a tree using this algorithm preserves left-to-right ordering of siblings.*

Proof:

A set of siblings contain between 0 and 2 nodes inclusive in a binary tree. If there are no siblings in a set, then the parent node is a leaf and the siblings are trivially ordered. If there is one sibling in a set, then it is trivially ordered if its parent is not a link node. If the parent is a link node, then after composition the set may be increased to two nodes if the link node was used to link to the root of another subtree. It is therefore only the case of two siblings in a set that needs to be addressed.

If there are two siblings in a set, it is easy to determine that they are properly ordered when their parent is not the link node by inspecting how the tree is composed.

The explicit cases fall into two categories. Remembering the definition of ordering, *wx* is the reference line for siblings *y* and *z* with parent *x*. The first category has *w* above *x* and one node is in the vertical channel below *x* and the other is in the horizontal channel to the right of *x*. Most situations are covered by this case. It is obvious that starting above *x* and moving counter-clockwise, the vertical channel below *x* is encountered before the horizontal channel right of *x*, so the node below *x* is the left node and the other one is the right node. Most options in cases, like the six options in Figure 31, exist so that a partial tree, like β, can occur either as the right or left node (on the right horizontal channel or the vertical channel below the parent). In the general case C is defined to be left of *a* and B is defined to be right of *a*, so C is always drawn in the vertical channel below *a* when in this category.

The second category is *w* is on the horizontal channel left of *x*, the left node *y* is to the right and above *x*, and the right node has a larger angle than *wxy*. It is obvious that this setup guarantees over 90 degrees for the right node's subtree to be drawn, which is all that is needed for a rectangle. See *a* in the bottom-left option of Figure 20 for an example.

The third category is *w* is left and above *x*, the left node *y* is on the horizontal channel right of *x*, and the right node has a larger angle than *wxy*. It is obvious that this also leaves over 90 degrees for the right node's subtree to be drawn. See *a* in the bottom-right option of Figure 20 for an example.

The fourth category is *w* is left and at or above the position of *x* and the left node is left and below or at the position of *x*. The right node has angle larger than *wxy*. See *a* in most options of Figure 37 for multiple examples of this.

The final category is *w* is left and at or below the position of *x* and the left node is in the vertical channel below *x*. The right node has angle larger than *wxy*. See *a* in the top-left option of Fig. 38 for an example of this.

These five categories cover all of the configurations for explicit sibling pairs. The algorithm details all possible layouts of nodes where ordering is managed by one of the five categories above. When a layout can occur to the left or right, an option exists to show how to proceed for either situation, which explains the many options present in Figure 37. In the general scope the partial tree C is to the left of the partial tree B in every case. In the B-scope an option exists for whether the β-subtree is left or right of the root. In the β-scope an option exists from each case that allows the β-subtree below *a* to appear on either the left or right. In the α-scope when subtrees 1 and 2 both exist, subtree 1 is the left subtree and subtree 2 is the right subtree. Since it is known ahead of time how the tree will be ordered, the algorithm will know which subtree is left and which subtree is right based on their roots, which are siblings. For every option of every case, it is known which partial tree in the schema is left and right so it is only a matter of assigning the left schema to the left subtree, etc.

The more interesting case is the one where there is a hidden sibling. This is the case when the link node has a child then connects to another drawing essentially adding on a second child. One child must be placed without knowing the exact position of its sibling since that sibling was removed during decomposition, to appear later in

composition. It is known which child was removed and which child remains since the ordering of the tree structure is known and it is only desired to be replicated in the drawing of the tree. If the left child is removed, then the right child must be placed in such a way that it is guaranteed to still be the right child after composition and vice versa. See Figure 39 for the geometry of the link node when it is on the bottom boundary. The horizontal line is the bottom boundary of the drawing containing the link node extended infinitely. Let *wx* be the ray originating at *x* and going through *w* or the vertical channel above *x* if *w* does not exist. The child removed during decomposition will appear somewhere in region III by the design of the algorithm. If this is the left node then the right child is in the drawing including *x* and must appear in region I so that during composition region III is guaranteed to be completely left of the right child. If the child removed during decomposition is the right child, then the left child is in the drawing including *x* and must appear in region II so that region III is guaranteed to be completely right of the left child. This same logic applies for when the link node is on the right boundary by rotating the geometry by 90 degrees though still drawing *wx* vertical in the case that *w* does not exist. In this case, region I does not exist, but this situation only arises when *x* is the root and the child removed was the right child, in which case the remaining child is drawn in region II below the position of *x*.

When in α-scope or B-scope, there is no link node so this problem is not present. In the general scope the link node is always the leftmost node in the tree so the right node is the one that is present. It is clear that the right node is never drawn in region III since that is outside of the drawing. Case 2 (C), (D), and (E) are the only cases that explicitly draw the link node. In (C) and (D) the right subtree spawning from the link node is in

region I as it should be. In (E) the link node will be moved to the bottom boundary when being utilized which will also put the subtree in region I as desired. In the β-scope the link node can have either a left or right child in its drawing. The options that have the child in region I assume this is the right child and the options that have the child in region II assume that this is the left child of the link node.

Since each set of siblings can be correctly ordered within each drawing and between drawings, then it is demonstrated that this algorithm preserves left-to-right ordering of siblings.

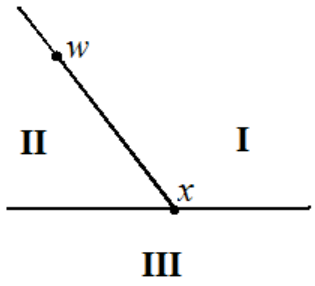


Figure 39. Link node geometry where *x* is the link node and *w* is its parent.

Lemma 8 (Area and Aspect Ratio): From [32] Lemma 4:

> "*Let T be a binary tree with a link node* u^*. *Let n be the number of nodes in T. Let* ε *and A be two numbers such that* $0 < \varepsilon < 1$, *and A is in the range* $[n^{-\varepsilon}, n^{\varepsilon}]$. *Given T*, ε, *and A as input*, [*the a*]*lgorithm* […] *will construct a drawing* Γ *of T that can fit inside a rectangle R with O*(*n*) *area and aspect ratio A*."

Proof:

The proof in [32] is modifiable to this algorithm, as will be shown. Using the same idea, we start with $D(n)$ as the area of R. It will be demonstrated that

$D(n) \le c_1 n - c_2 n^{\beta} \ \forall n \ge n_0$ given c_1, c_2, n_0, β are positive constants and $\beta < 1$,

demonstrating $D(n) = O(n)$.

For the cases where $A \geq 1$ and placement is horizontal and $A < 1$ and placement is vertical, the proof follows exactly from [32] with a slight change to constants. Let T be the tree being drawn by this algorithm. T is either general scope or β-scope, since the other two scopes are immediate (e.g. the partial tree B in Case 2 of the general scope when drawn using Case 3 of the B-scope is actually partial trees 1 and 2 from the α-scope, the β-subtree and the $s(\beta)$ subtree because B is immediately recursed one level). If T is general scope it is always true that the conditions to use the proof from [32] are met. If T is β-scope, it may not be true that these conditions are met, e.g. Figure 34 (e-f), although the proof is similarly structured.

In either case, let T_k be a partial tree of T as specified by the tree splitting process of the algorithm (recursing one level for partial trees that are drawn in an immediate scope). For example, in general scope Case 2 using B-scope Case 3 T_k may be one of T_A, T_C, T_β, $T_{s(\beta)}$, T_1, and T_2. Let T_k have n_k nodes and $x_k = n_k / n$. Let ς be the maximum number of T_k needed to split T. Figure 37 (a) potentially has 7 partial trees (the two pieces of $T_{C\beta}$, $T_{A\beta}$, and up to 4 pieces of $T_{B\beta}$, e.g. T_β, $T_{s(\beta)}$, T_1, and T_2 in Case 3 of the B-scope); this is the maximum number of partial trees generated by any figure so $\varsigma = 7$. Let ζ and ξ respectively be the maximum number of extra horizontal and vertical channels needed to compose T. It is tedious to go through every case to compute these but exact numbers are not necessary for the proof. It is only necessary to know these values are small (around 5). The last definition needed before beginning is letting $P_k = c_1 n - c_2 n^\beta / x_k^{1-\beta}$.

From Theorem 1 it must be true that $n_k \leq (2/3)n$ which implies $x_k \leq 2/3$. Hence, $P_k \leq c_1 n - c_2 n^\beta / (2/3)^{1-\beta} = c_1 n - c_2 n^\beta (3/2)^{1-\beta} \equiv P'$. From the inductive hypothesis, each

drawing $\Gamma(T_k)$ will fit inside a rectangle R_k with optimal area $D(n_k) \le c_1 n_k - c_2 n_k^{\beta} = c_1 x_k n - c_2 (x_k n)^{\beta} = x_k \left(c_1 n - c_2 n^{\beta} / x_k^{1-\beta} \right) = x_k P_k \le x_k P'$ and aspect ratio A_k as defined by the aspect ratio assignments given in the algorithm.

Let W_k and H_k be the width and height respectively of R_k. There are four cases to consider from the combination of choice of whether placement will be horizontal or vertical and whether T_k is large or small as defined in the aspect ratio assignments given in the algorithm. For horizontal placement and T_k is small, $n_k < \begin{cases} (n/A)^{1/(1+\varepsilon)} & A \ge 1 \\ (nA)^{1/(1+\varepsilon)} & A < 1 \end{cases}$ and $A_k = n_k^{-\varepsilon}$. Defining W_k and H_k by the relation of area and aspect ratio, we get:

$$W_k = \sqrt{A_k D(n_k)} \le \sqrt{n_k^{-\varepsilon} x_k P'} = \sqrt{n_k^{1-\varepsilon} P' / n} <$$
$$\left\{ \begin{array}{ll} \sqrt{(n/A)^{(1-\varepsilon)/(1+\varepsilon)} P'/n} = \sqrt{\left(1/A^{(1-\varepsilon)/(1+\varepsilon)}\right) P'/n^{2\varepsilon/(1+\varepsilon)}} & A \ge 1 \\ \sqrt{(nA)^{(1-\varepsilon)/(1+\varepsilon)} P'/n} = \sqrt{A^{(1-\varepsilon)/(1+\varepsilon)} P'/n^{2\varepsilon/(1+\varepsilon)}} & A < 1 \end{array} \right\} \le \sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}}$$

$$H_k = \sqrt{D(n_k)/A_k} \le \sqrt{n_k^{\varepsilon} x_k P'} = \sqrt{n_k^{1+\varepsilon} P'/n} <$$
$$\begin{cases} \sqrt{(n/A)^{(1+\varepsilon)/(1+\varepsilon)} P'/n} = \sqrt{P'/A} & A \ge 1 \\ \sqrt{(nA)^{(1+\varepsilon)/(1+\varepsilon)} P'/n} = \sqrt{AP'} & A < 1 \end{cases}$$

For horizontal placement and T_k is large, $A_k = x_k A$, this time giving:

$$W_k = \sqrt{A_k D(n_k)} \le \sqrt{x_k A x_k P'} = x_k \sqrt{AP'}$$
$$H_k = \sqrt{D(n_k)/A_k} \le \sqrt{x_k P'/(x_k A)} = \sqrt{P'/A}$$

For the case of vertical placement, when T_k is small, A_k is the multiplicative inverse of what it was for horizontal placement, so W_k and H_k swap results. For large T_k, x_k is switched giving:

$$W_k \le \sqrt{AP'}$$
$$H_k \le x_k \sqrt{P'/A}$$

Let W' and H' be the width and height respectively of the rectangle R' that encloses $\Gamma(T)$, the drawing of T. For horizontal placement:

$$H' \le \max(H_k) + \xi \le \sqrt{P'/A} + \xi \quad \forall A$$
$$W' \le \sum_{T_k \text{ large}} W_k + \sum_{T_k \text{ small}} W_k + \zeta \le \sqrt{AP'} + \varsigma\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + \zeta \quad \forall A$$

since $\sum_{T_k \text{ large}} x_k \le 1$ and $\sum_{T_k \text{ small}} 1 \le \varsigma$.

For vertical placement:

$$W' \le \max(W_k) + \zeta \le \sqrt{AP'} + \zeta \quad \forall A$$
$$H' \le \sum_{T_k \text{ large}} H_k + \sum_{T_k \text{ small}} H_k + \xi \le \sqrt{P'/A} + \varsigma\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + \xi \quad \forall A$$

R' does not necessarily have an aspect ratio of A, but it is enclosed by a rectangle R with width W and height H that does have aspect ratio A. It must be assured that W and H contain their primed counterparts entirely. For horizontal placement:

$$W = \sqrt{AP'} + \varsigma\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + \zeta + \xi A \quad \forall A$$
$$H = W/A = \sqrt{P'/A} + \left(\varsigma\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + \zeta\right)/A + \xi \quad \forall A$$

which both certainly contain their primed counterparts entirely. For vertical placement:

$$H = \sqrt{P'/A} + \varsigma\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + \xi + \zeta/A \quad \forall A$$
$$W = HA = \sqrt{AP'} + A\left(\varsigma\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + \xi\right) + \zeta \quad \forall A$$

Notice that if the horizontal definition swaps the definition of H and W and ξ and ζ then inserts the multiplicative inverse of A, the vertical placement formulas are acquired. This means that horizontal placement with $A \ge 1$ is equivalent to vertical placement and $A < 1$. This case was proved in [32] (where with no loss of generality ζ=2, ξ=1, and ς=5). It must

be demonstrated that horizontal placement with A < 1 (and therefore vertical placement with $A \geq 1$) is also feasible. This reduces to demonstrating only the horizontal case of:

$$
\begin{aligned}
&D(n) = WH = \\
&\left(\sqrt{AP'} + \varsigma\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + \zeta + \xi A\right)\left(\sqrt{P'/A} + \left(\varsigma\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} + \zeta\right)/A + \xi\right) \quad A<1 \\
&= P' + 2\varsigma P'/\sqrt{An^{2\varepsilon/(1+\varepsilon)}} + 2\xi\sqrt{AP'} + \varsigma^2 P'/\left(An^{2\varepsilon/(1+\varepsilon)}\right) + 2\varsigma\xi\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} \\
&\quad + \xi^2 A + 2\zeta\xi + \zeta^2/A + 2\zeta\sqrt{P'/A} + 2\varsigma\zeta\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}}/A \\
&\leq P' + c_3 P'/\sqrt{An^{2\varepsilon/(1+\varepsilon)}} + c_4\sqrt{AP'} + c_5 P'/\left(An^{2\varepsilon/(1+\varepsilon)}\right) + c_6\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} \\
&\quad + c_7 A + c_8 + c_9/A + c_{10}\sqrt{P'/A} + c_{11}\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}}/A
\end{aligned}
$$

and since $n^{-\varepsilon} \leq A < 1$ and $P' < c_1 n$:

$$
\begin{aligned}
&D(n) \leq P' + c_3 P'\sqrt{n^{\varepsilon} n^{2\varepsilon/(1+\varepsilon)}} + c_4\sqrt{P'} + c_5 n^{\varepsilon} n^{2\varepsilon/(1+\varepsilon)} P' + c_6\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} \\
&\quad + c_7 + c_8 + n^{\varepsilon} c_9 + c_{10}\sqrt{n^{\varepsilon} P'} + c_{11} n^{\varepsilon}\sqrt{P'/n^{2\varepsilon/(1+\varepsilon)}} \\
&\leq P' + c_3 c_1 n\sqrt{n^{\varepsilon - 2\varepsilon/(1+\varepsilon)}} + c_4\sqrt{c_1 n} + c_5 n^{\varepsilon - 2\varepsilon/(1+\varepsilon)} c_1 n + c_6\sqrt{c_1 n^{1-2\varepsilon/(1+\varepsilon)}} \\
&\quad + c_7 + c_8 + n^{\varepsilon} c_9 + c_{10}\sqrt{n^{1+\varepsilon} c_1} + c_{11} n^{\varepsilon}\sqrt{c_1 n^{1-2\varepsilon/(1+\varepsilon)}}
\end{aligned}
$$

which simplifies to:

$$
\begin{aligned}
&D(n) \leq P' + c_1 c_3 n^{\varepsilon/2+1/(1+\varepsilon)} + \sqrt{c_1} c_4 n^{1/2} + c_1 c_5 n^{(1+\varepsilon^2)/(1+\varepsilon)} + \sqrt{c_1} c_6 n^{(1/2)(1-\varepsilon)/(1+\varepsilon)} \\
&\quad + c_7 + c_8 + n^{\varepsilon} c_9 + \sqrt{c_1} c_{10} n^{\varepsilon/2+1/2} + \sqrt{c_1} c_{11} n^{\varepsilon+1/(1+\varepsilon)-1/2}
\end{aligned}
$$

and only keeping highest order terms one can find c_{12} to c_{15} such that:

$$
D(n) \leq P' + c_{12} n^{\varepsilon/2+1/(1+\varepsilon)} + c_{13} n^{(1+\varepsilon^2)/(1+\varepsilon)} + c_{14} n^{\varepsilon/2+1/2} + c_{15} n^{\varepsilon+1/(1+\varepsilon)-1/2}
$$

$P' = c_1 n - c_2 n^{\beta}(3/2)^{1-\beta} = c_1 n - c_2 n^{\beta}(1 + c_{16})$ for some c_{16} such that $1 + c_{16} = (3/2)^{1-\beta}$.

Therefore:

$$
D(n) \leq c_1 n - c_2 n^{\beta}(1 + c_{16}) + c_{12} n^{\varepsilon/2+1/(1+\varepsilon)} + c_{13} n^{(1+\varepsilon^2)/(1+\varepsilon)} + c_{14} n^{\varepsilon/2+1/2} + c_{15} n^{\varepsilon+1/(1+\varepsilon)-1/2}
$$

simplifies to:

$$
D(n) \leq c_1 n - c_2 n^{\beta} - \left(c_2 c_{16} n^{\beta} - c_{12} n^{\varepsilon/2+1/(1+\varepsilon)} - c_{13} n^{(1+\varepsilon^2)/(1+\varepsilon)} - c_{14} n^{\varepsilon/2+1/2} - c_{15} n^{\varepsilon+1/(1+\varepsilon)-1/2}\right)
$$

and given $n \geq n_0$ for some constant n_0 chosen large enough and $1 > \beta \geq \varepsilon/2 + 1/(1+\varepsilon)$, since β needs to give an order at least as large as that of the highest power in the parenthesized part, a solution can be found to:

$c_2 c_{16} n^{\beta} - c_{12} n^{\varepsilon/2+1/(1+\varepsilon)} - c_{13} n^{(1+\varepsilon^2)/(1+\varepsilon)} - c_{14} n^{\varepsilon/2+1/2} - c_{15} n^{\varepsilon+1/(1+\varepsilon)-1/2} \geq 0$, which then gives:

$D(n) \leq c_1 n - c_2 n^{\beta}$, the desired result. Since $\beta < 1$, the desired result is also linear, therefore the area of the enclosing rectangle is linear (and therefore optimal).

From here, it is easy to see that the desired aspect ratio is achieved since one of the following was chosen by definition:

$$H \equiv W / A \quad W \equiv HA$$

By definition of *A*, this gives:

$A \equiv W / H = \begin{cases} W/(W/A) = A \\ HA/H = A \end{cases}$, which is indeed true.

Theorem 2 (Main Theorem): From [32] now that the same results have also been developed in these lemmas:

> "*Let T be a binary tree with n nodes. Given two numbers A and* ε*, where* ε *is a constant, such that* $0 < \varepsilon < 1$, *and* $n^{-\varepsilon} \leq A \leq n^{\varepsilon}$, *we can construct in O*(*n* log *n*) *time, a planar straight-line grid drawing of T with O*(*n*) *area and aspect ratio A*."

This result is further strengthened by including Lemma 7 so that the planar straight-line grid drawings preserve ordering of binary rooted and ordered trees.

Corollary to Theorem 2: Setting $A = 1$ and choosing an arbitrary ε within the specified bounds, then by Theorem 2 an order-preserving planar straight-line grid drawing of a binary rooted and ordered tree can be constructed in $O(n \log n)$ time with area $O(n)$ and optimal aspect ratio $A = 1$.

3.5 Experimental Results

An implementation of [32] was run side-by-side with an implementation of the algorithm in this chapter. Maximum width and height were recorded for different types of binary trees ranging from less than 100 nodes to tens of thousands of nodes. The different types of trees used were AVL, Fibonacci, Left-Heavy, Right-Heavy, Complete, and Random. Each tree generated three drawings with 0.5, 1, and 2 for their respective desired aspect ratios. It is expected that the number of nodes for each type plotted against aspect ratio will converge to the desired aspect ratio. Since area is expected to be linear to the number of nodes, the area divided by the number of nodes, called the K-value, is expected to converge to a constant as the number of nodes is increased. The following are the plots obtained.

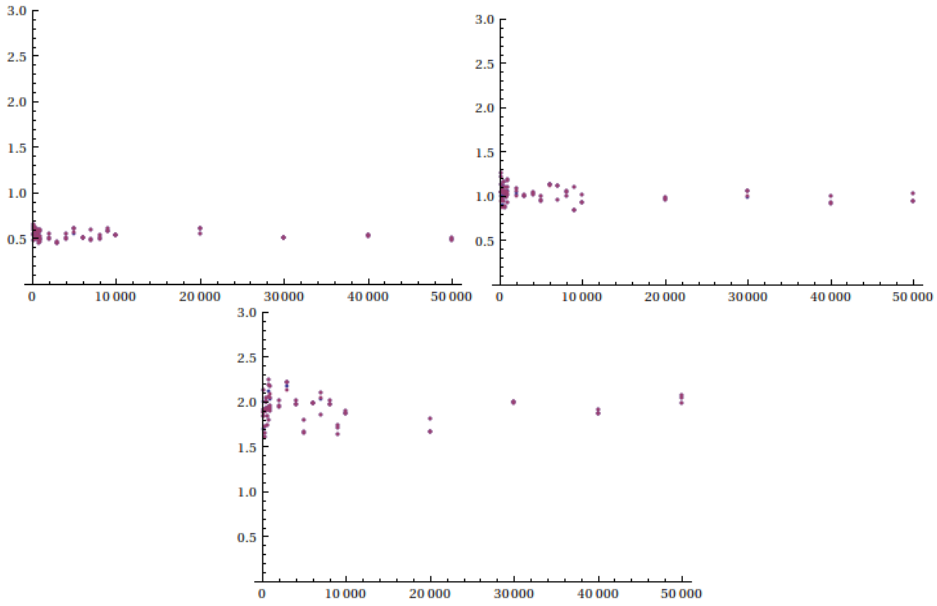


Figure 40. AVL tree aspect ratios for desired aspect ratios of 0.5, 1.0 and 2.0. The aspect ratios are perfectly overlaid.

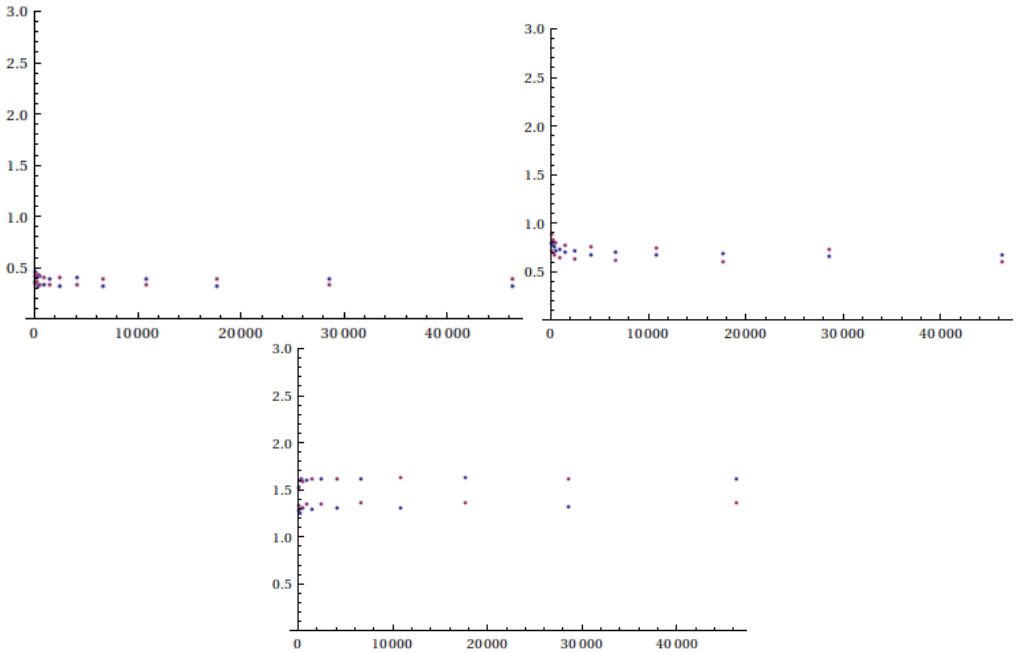


Figure 41. Fibonacci tree aspect ratios for desired aspect ratios of 0.5, 1.0 and 2.0. Blue represents the original algorithm and purple represents the new algorithm.

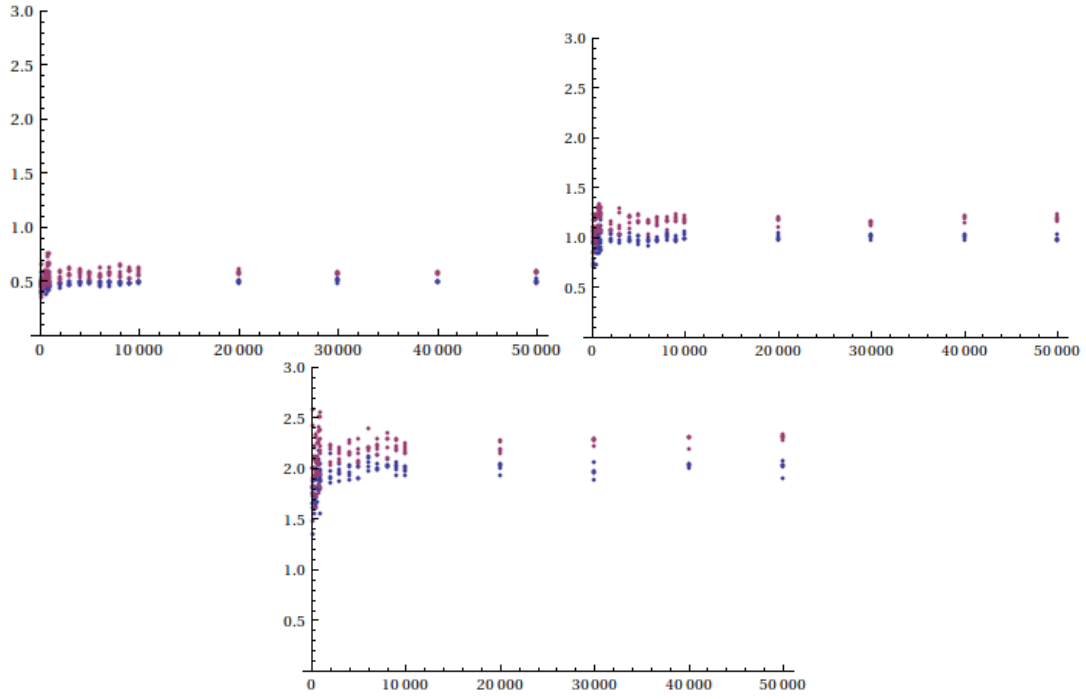


Figure 42. Left-Heavy tree aspect ratios for desired aspect ratios of 0.5, 1.0 and 2.0. Blue represents the original algorithm and purple represents the new algorithm.

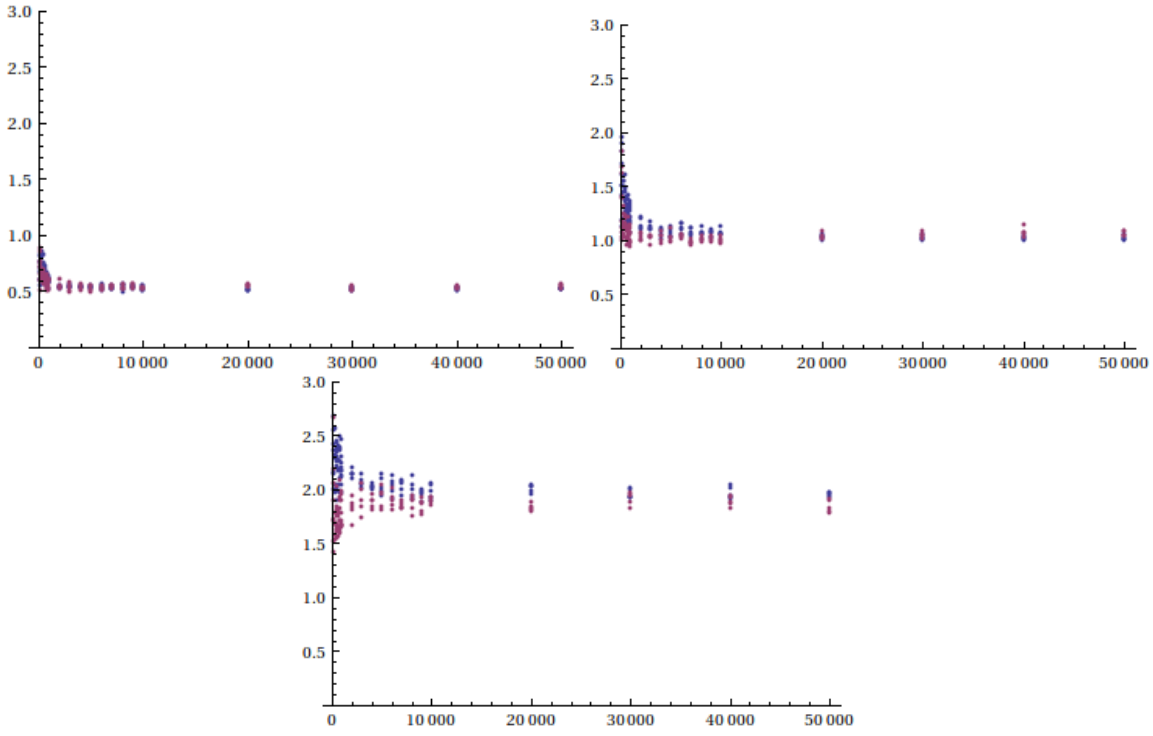


Figure 43. Right-Heavy tree aspect ratios for desired aspect ratios of 0.5, 1.0 and 2.0. Blue represents the original algorithm and purple represents the new algorithm.

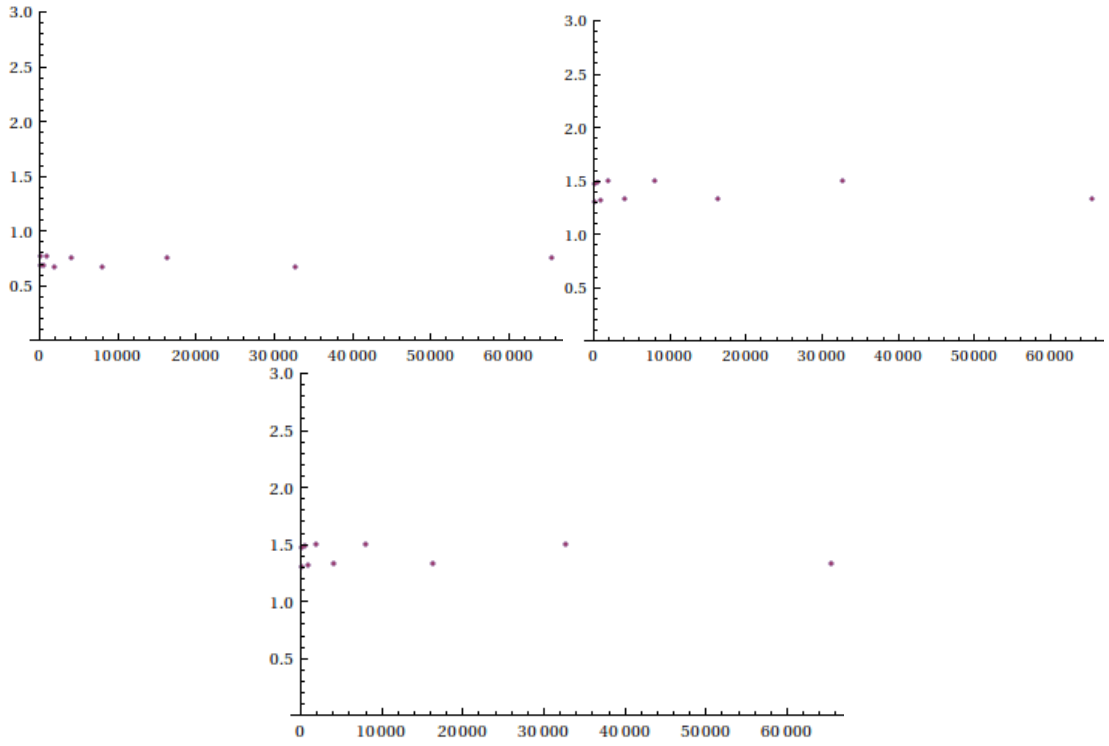


Figure 44. Complete tree aspect ratios for desired aspect ratios of 0.5, 1.0 and 2.0. The aspect ratios are perfectly overlaid.

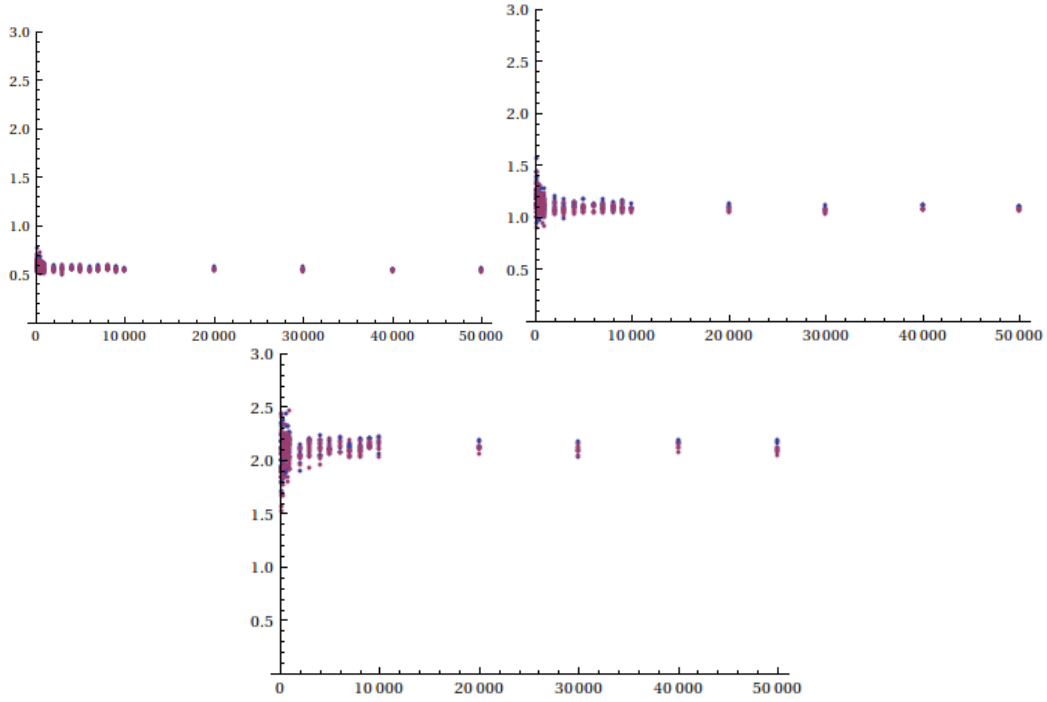


Figure 45. Random tree aspect ratios for desired aspect ratios of 0.5, 1.0 and 2.0. Blue represents the original algorithm and purple represents the new algorithm.

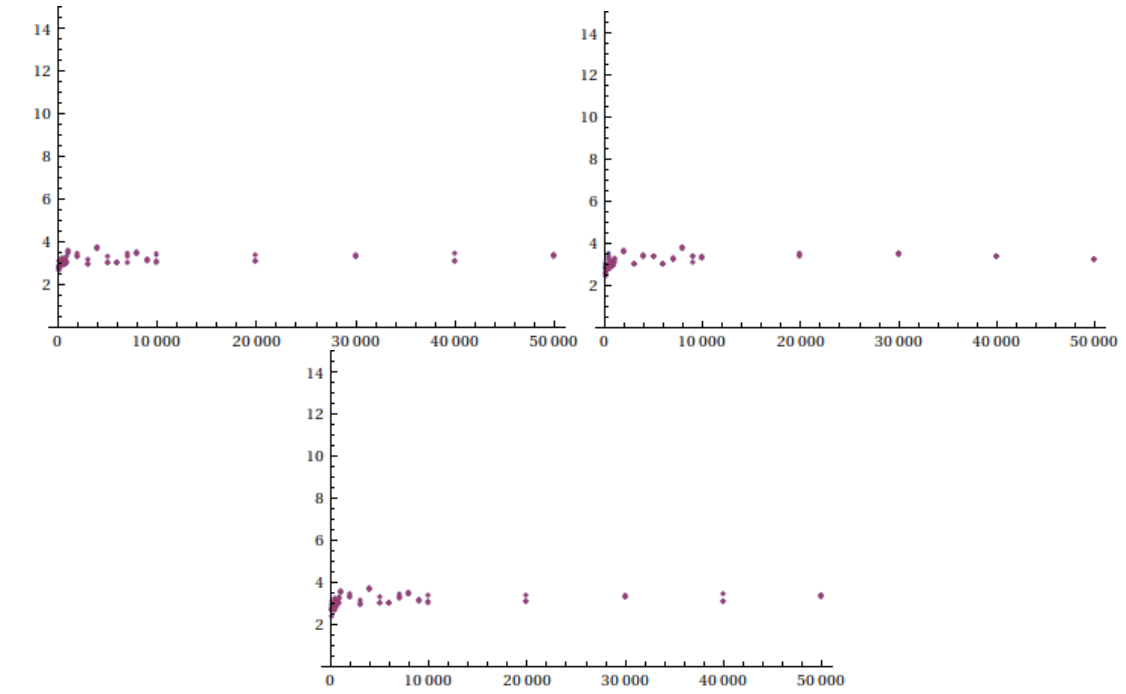


Figure 46. AVL tree K-Values for desired aspect ratios of 0.5, 1.0 and 2.0. The K-Values are perfectly overlaid.

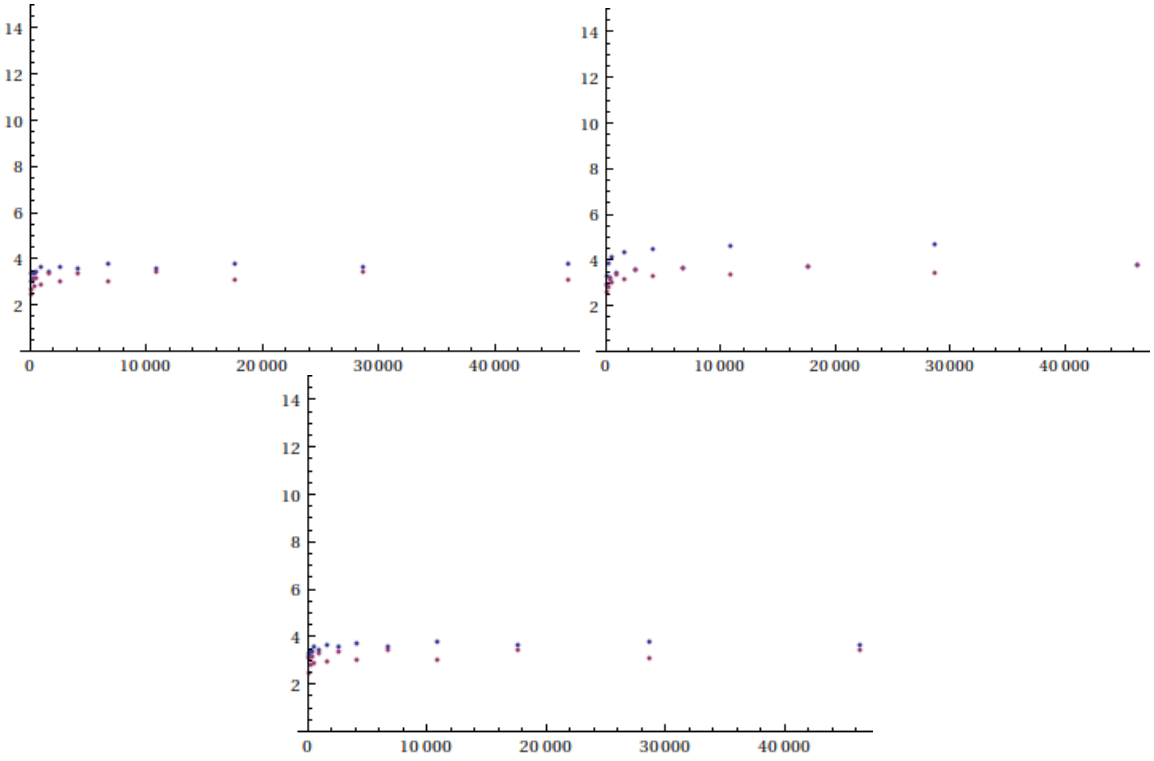


Figure 47. Fibonacci tree K-Values for desired aspect ratios of 0.5, 1.0 and 2.0. Blue represents the original algorithm and purple represents the new algorithm.

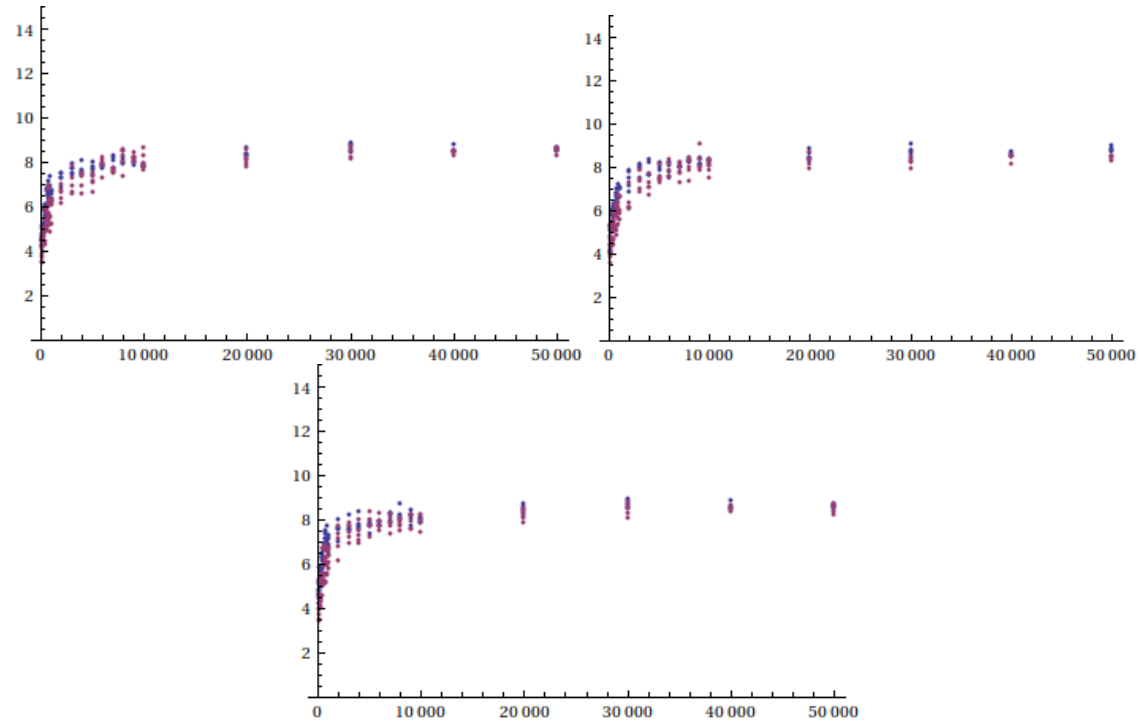


Figure 48. Left-Heavy tree K-Values for desired aspect ratios of 0.5, 1.0 and 2.0. Blue represents the original algorithm and purple represents the new algorithm.

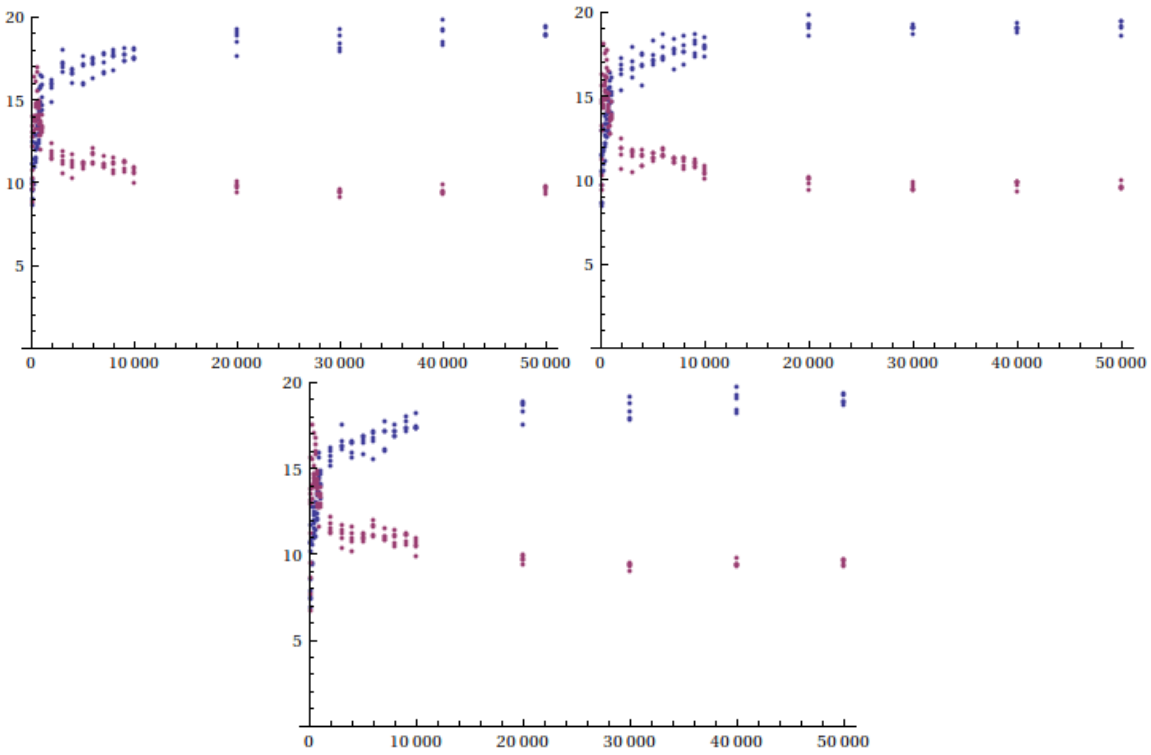


Figure 49. Right-Heavy tree K-Values for desired aspect ratios of 0.5, 1.0 and 2.0. Blue represents the original algorithm and purple represents the new algorithm.

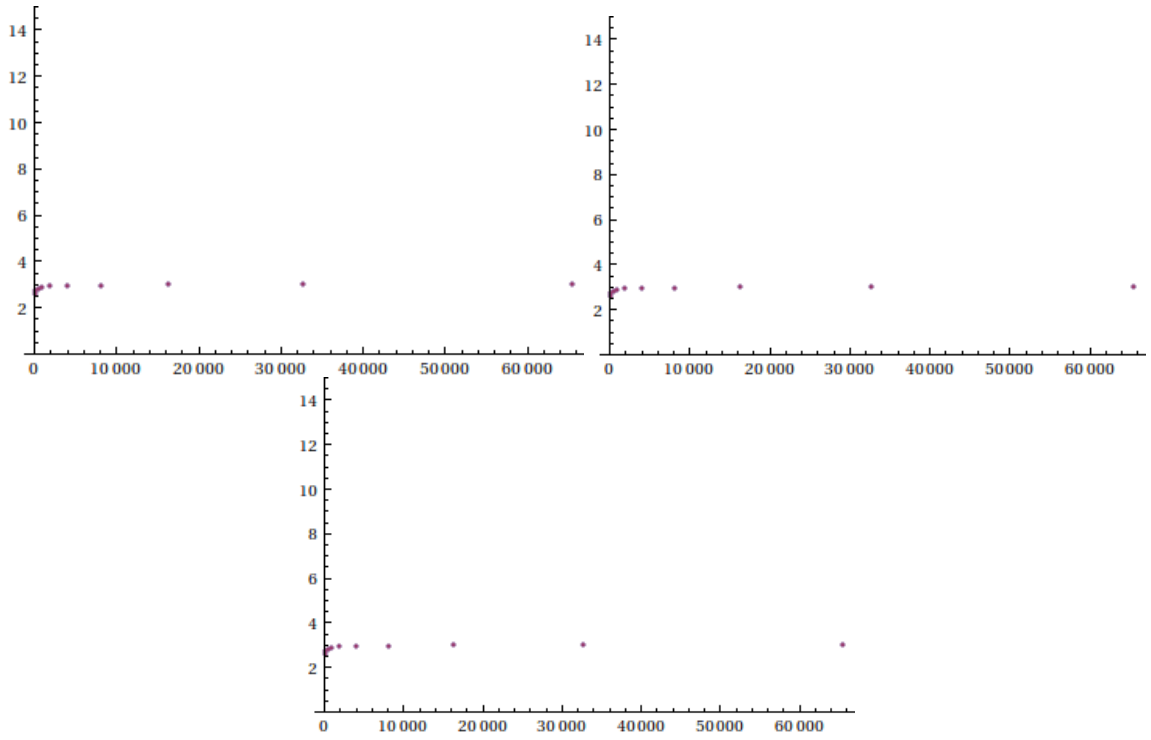


Figure 50. Complete tree K-Values for desired aspect ratios of 0.5, 1.0 and 2.0. The K-Values are perfectly overlaid.

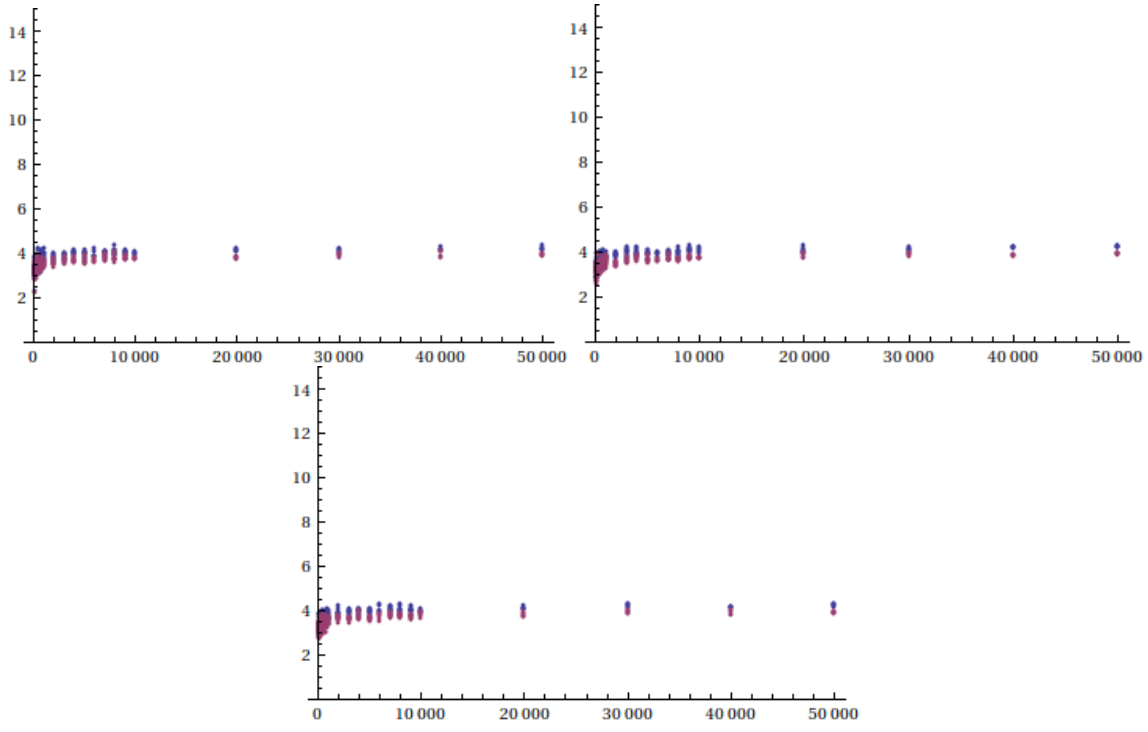


Figure 51. Random tree K-Values for desired aspect ratios 0.5, 1.0 and 2.0. Blue represents the original algorithm and purple represents the new algorithm.

In the implementation used, an exact 1/3-2/3 cut was not required as long as the cut was within those bounds, so the AVL and Complete trees always had a link node at the root for both algorithms, since they are both balanced trees. This results in the general scope, Case I, subcases E-H to be the only situations that can arise. These cases are exactly equivalent to Case II, subcases E-H from [32], so the drawings are identical. This is why the plots for these are perfectly overlaid for aspect ratio and K-Value.

Both algorithms do not guarantee that the aspect ratio will be exactly the desired aspect ratio, but should approach it so that it can fit in a box of the desired aspect ratio. The Fibonacci and Complete trees were consistently furthest from the desired aspect ratio in both algorithms.

Both algorithms suggest a constant K-Value for increasing number of nodes. This value tends to be between 3 and 4 for both algorithms when the tree style is not left or right heavy. In these cases, the original algorithm had K-Values of about 9 and 19 respectively, while the newer algorithm seems to improve this to about 9 and 10 respectively.

Figure 52 shows the difference between the two algorithms for a random binary tree of 100 nodes. The left drawing has a node circled in red in which the left and right children were drawn out of order, whereas the same node is circled on the right and has been correctly drawn.

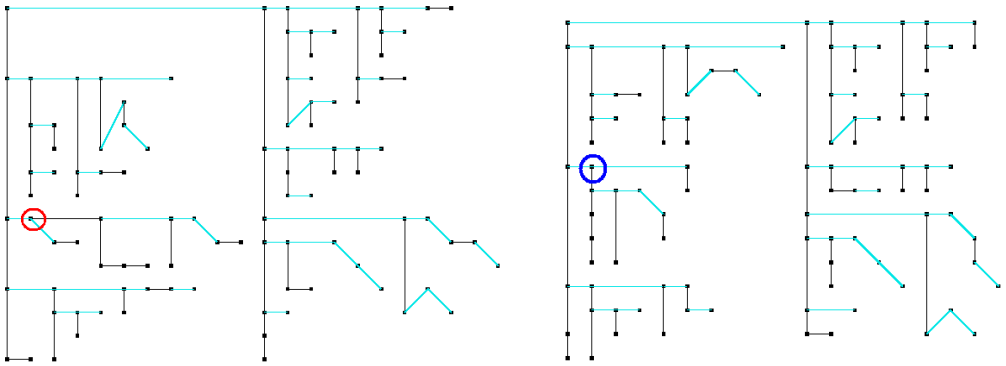


Figure 52. Left the original out-of-order algorithm. Right the ordered algorithm. Light blue is a right child and black is a left child. The red circle draws attention to an out-of-order pair, which is circled in blue on the right to demonstrate being ordered there.

3.6 Conclusion

The work in [32] has been extended to allow for order preserving planar straight-line grid drawings of binary rooted and ordered trees. Given that a tree has n nodes this solution is found in $O(n \log n)$ time with $O(n)$ area and user specified aspect ratio $n^{-\varepsilon} \leq A \leq n^{\varepsilon}$ where $0 < \varepsilon < 1$, which are the same bounds placed on the out-of-order solution.

Chapter 4

Summary

Graphs (and trees) have been demonstrated to be widely important in many different applications from many disciplines. It was also discussed that Information Visualization is an important area of research in finding better ways of effectively communicating the data contained within graphs. Gestalt psychology was also introduced as a complementing field to study techniques that may benefit visualizations. From here, aesthetic requirements were presented as a way of measuring the effectiveness and aesthetic value of graph and tree drawings.

Edge crossings were noted as being especially detrimental to non-planar graph drawings. Alleviating this problem by diminishing unintentional gestalt effects is the consideration of Chapter 2. Here a method of embedding graph edges into a color space was presented. The motivation being coloring edges that may hinder effectiveness by having low angular resolution or long lengths, for example, have perceptually different colors to allow for easier differentiation. The second method presented used breaks in edges as an alternative that does not rely on colors, being better for black-on-white printings and for color-blind users.

Chapter 3 focuses on a novel algorithm for improving the known bounds on planar straight-line order-preserving grid drawings of binary trees to optimal for both area (linear) and aspect ratio (1:1).

Future work should include an intensive user study of the effectiveness of the methods in Chapter 2 as well as researching other Gestalt applications to the edge crossing problem, and exploring different separator values for the tree drawing algorithm.

List of References

[1] M.W. Bern, D. Eppstein, and B. Hutchings. Algorithms for coloring quadtrees. *Algorithmica*, Volume 32, Issue 1, pages 87-94, 2002.

[2] M. Chalmers. A linear iteration time layout algorithm for visualizing high-dimensional data. In *Proc. of VIS'96*, pages 127-ff. IEEE CS Press, 1996.

[3] C. Demiralp, S. Zang, D. Tate, S. Correia and D. H. Laidlaw. Connectivity-aware sectional visualization of 3D DTI volumes using perceptual flat-torus coloring and edge rendering. In *Proceedings of Eurographics*, 2006.

[4] M.B. Dillencourt, D. Epstein and M. T. Goodrich. Choosing colors for geometric graphs via color space embeddings. In *Proceedings of the 14th International Symposium on Graph Drawing*, pages 294–305, 2006.

[5] P. Eades. A Heuristic for Graph Drawing. In *Congressus Numerantium,* Volume 42, pages 149-160, 1984.

[6] D. Eppstein. Testing bipartiteness of geometric intersection graphs. In *Proc. 15th Symp. Discrete Algorithms*, pages 853-861. ACM and SIAM, 2004.

[7] S. Felsner, F. Hurtado, M. Noy, and I. Streinu. Hamiltonicity and colorings of arrangement graphs. In *Proc. 11th Symp. Discrete Algorithms*, pages 155-164. ACM and SIAM, 2000.

[8] H. de Fraysseix, J. Pach, and R. Pollark. How to draw a planar graph on a grid. *Combinatorica*, Volume 10, pages 41-51, 1990.

[9] T. Fruchterman and E. Reingold. Graph Drawing by Force-Directed Placement. In *Software—Practice and Experience*, Volume 21, Issue 11, pages 1129-1164, 1991.

[10] C. Healey. Choosing effective colors for data visualization. In *Proceedings of the 7th conference on Visualization*, pages 263-ff, 1996.

[11] I. Jolliffe. Principal Component Analysis. Springer-Verlag, 1986.

[12] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, Volume 31, Issue 1, pages 7-15, 1989.

[13] E. Kandogan. Star Coordinates: A multidimensional visualization technique with uniform treatment of dimensions. In *Proceedings of the IEEE Information Visualization Symposium*, pages 9-12, 2000.

[14] H. Levkovitz and G. Herman. Color scales for image data. In *IEEE Computer Graphics and Applications* Volume 12, pages 72-80, 1992.

[15] A. Morrison and M. Chalmers. A pivot-based routine for improved parent-finding in hybrid MDS. *Information Visualization*, Volume 3, Issue 2, pages 109-122, 2004.

[16] H. Purchase. Which aesthetic has the greatest effect on human understanding? In *Proceedings of the 5th International Symposium on Graph Drawing*, pages 248–261, 1997.

[17] P. Rheingans and B. Tebbs. A tool for dynamic explorations of color mappings. *Computer Graphics*, Volume 24, Issue 2, pages 145-146, 1990.

[18] P. Robertson. Visualizing color gamuts: A user interface for the effective use of perceptual color spaces in data displays. *IEEE Computer Graphics and Applications*, Volume 8, Issue 5, pages 50-64, 1988.

[19] J. Sammon. A Nonlinear Mapping for Data Structure Analysis. In *IEEE Trans. Computers*, Volume 13, pages 401-409, 1964.

[20] K. Seisenberger. Termgraph: Ein System zur Zeichnerischen Darstellung von Strukturierten Agenten und Petrinetzen. Diplomarbeit, Universität Passau, 1991.

[21] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, Volume 16, pages 421-444, 1987.

[22] E. Tejada, R. Minghim, and L.G. Nonato. On improved projection techniques to support visual exploration of multidimensional data sets. *Information Visualization*, Volume 2, Issue 4, pages 218-231, 2003.

[23] D. Tunkelang. A practical approach to drawing undirected graphs. Carnegie Mellon University, 1994.

[24] C. Ware. Color sequences for univariate maps: Theory, experiments and principles. In *IEEE Computer Graphics and Applications*, Volume 8, pages 41–49, 1988.

[25] D. Woods. *Drawing Planar Graphs*. PhD thesis, Stanford University, 1982.

[26] C.G. Boeree. *Gestalt Psychology,* <http://webspace. ship.edu/cgboer/gestalt.html>, 2000.

[27] Am. Rusu and V. Govindaraju. Visual CAPTCHA with handwritten image analysis, *Proceedings of the 2nd International Workshop on Human Interactive Proofs,* Lecture Notes in Computer Science, Vol. 3517, pp. 42-52, 2005.

[28] Am. Rusu and V. Govindaraju. A human interactive proof algorithm using handwriting recognition, *Proceedings of the 8th International Conference on Document Analysis and Recognition*, IEEE Computer Society, Vol. 2, pp. 967-971, 2005.

[29] Am. Rusu, A.O. Thomas and V. Govindaraju. Generation and use of handwritten CAPTCHAs, *International Journal on Document Analysis and Recognition*, Springer-Verlag, Vol. 13, Issue 1, pp. 49-64, 2010.

[30] J. Bentley and T. Ottman. Algorithms for counting and reporting geometric intersections, *IEEE Transactions on Computers*, Vol. C-28, Issue 9, pp. 643-647, 1979.

[31] Am. Rusu and R. Docimo. Securing the Web using human perception and visual object interpretation, *Proceedings of the 2009 13th International Conference on Information Visualization*.

[32] A. Garg and A. Rusu. Straight-line Drawings of Binary Trees with Linear Area and Arbitrary Aspect Ratio. *Journal of Graph Algorithms and Applications*, 8(2):135-160, 2004.

[33] A. Garg and A. Rusu. Area-efficient order-preserving planar straight-line drawings of ordered trees. *International Journal of Computational Geometry and Applications*, 13(6):487-505, 2003.

[34] T. Chan, M. Goodrich, S. Rao Kosaraju, and R. Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings. S. North, editor, *Graph Drawing* (*Proceedings of Graph Drawing '96*), volume 1190 of *Lecture Notes in Computer Science*, 63-75, 1997.

[35] P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Computational Geometry Theory and Application*, 2:187-200, 1992.

[36] D. Dolev and H. W. Trickey. On linear area embedding of planar graphs. Technical report. Stanford University, Stanford, USA. 1981.

[37] F. Frati. Straight-line orthogonal drawings of binary and ternary trees. Seok-Hee Hong and Takao Nishizeki, editors, *15th International Symposium on Graph Drawing*, volume 4875 of *Lecture Notes in Computer Science*, 76-87, 2007.

[38] A. Garg, M. T. Goodrich, and R. Tamassia. Planar upward tree drawings with optimal area. *International Journal of Computational Geometry and Applications*, 6:333-356, 1996.

[39] S. K. Kim. Simple algorithms for orthogonal upward drawings of binary and ternary trees. *Proceedings of the 7th Canadian Conference on Computational Geometry*, 115-120, 1995.

[40] S. K. Kim. Order-preserving, upward drawing of binary trees using fewer bends. *Discrete Applied Mathematics Journal*, 143(1-3):318-323, 2004.

[41] C. E. Leiserson. Area-efficient graph layouts (for VLSI). *Proceedings of the 21st Annual IEEE Symposium on the Foundations of Computer Science*, 270-281, 1980.

[42] C.-S. Shin, S. K. Kim, and K.-Y. Chwa. Area-efficient algorithms for straight-line tree drawings. *Computational Geometry Theory and Application*, 15:175-202, 2000.

[43] L. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, C-30(2):135-140, 1981.

[44] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry Theory and Application*, 4:235-282, 1994.

[45] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.

[46] H. C. Purchase, R. F. Cohen, and M. I. James. An experimental study of the basis for graph drawing algorithms. *ACM Journal of Experimental Algorithmics*, 2(4), 1997.

[47] A. Rusu, R. Jianu, A. J. Fabian, and D. Laidlaw. *Proceedings of the 13th International Conference on Information Visualization*, 2009.

[48] Am. Rusu, A. J. Fabian, R. Jianu, and Ad. Rusu. Using the Gestalt Principle of Closure to Alleviate the Edge Crossing Problem in Graph Drawings. *Proceedings of the 15th International Conference on Information Visualization*, 2011.