Visualisation of Large and Complex Networks Using PolyPlane*

Seok-Hee Hong[1] and Tom Murtagh[2]

[1] National ICT Australia; School of Information Technologies,
University of Sydney, Australia
shhong@it.usyd.edu.au
[2] School of Information Technologies,
University of Sydney, Australia
tfm@it.usyd.edu.au

Abstract. This paper discusses a new method for visualisation of large and complex networks in three dimensions. In particular, we focus on visualising the *core tree structure* of the large and complex network. The algorithm uses the concept of *subplanes*, where a set of subtrees is laid out. The subplanes are defined using *regular polytopes* for easy navigation. The algorithm can be implemented to run in linear time. We implemented the algorithm and the experimental results show that it produces nice layouts of large trees with up to ten thousand nodes. We further discuss how to extend this method for more general case.

1 Introduction

Recent technological advances produce a lot of data, and have led to many large and complex network models in many domains; examples include social networks, biological networks and webgraphs.

Visualization can be an effective tool for analysis of such networks. Good visualisation reveals the hidden structure of the networks and amplifies human understanding, thus leading to new insights, new findings and possible prediction.

However, recent advances in technology have made available data on networks with millions of nodes; visualization of such large and complex networks is very challenging. Current methods exhibit at least one the following problems: poor scalability, lack of good navigation methods, poor integration with analysis methods, and lack of good 3D visualisation.

In this paper, we present a new method for visualisation of large and complex networks in three dimensions. In particular, we focus on visualising the *core tree*

* This research has been supported by a SESQUI grant from the University of Sydney, a research grant from the School of Information Technologies, Special Study Leave Program of the University of Sydney, and NICTA Summer Vacation Scholarship. Animated drawings are available from http://www.cs.usyd.edu.au/∼shhong/3dtreedraw.htm. National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

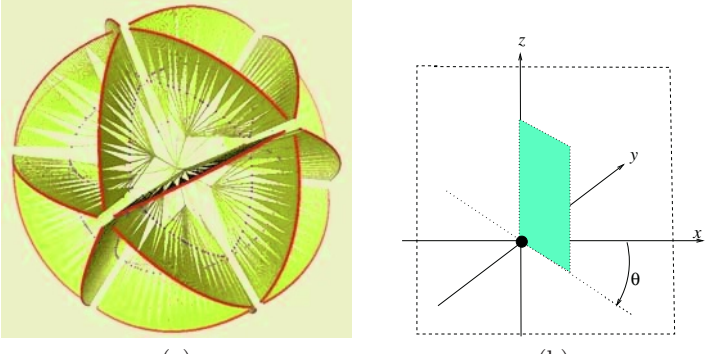
J. Pach (Ed.): GD 2004, LNCS 3383, pp. 471–481, 2004.

structure of the large and complex network to reduce both cognitive overload and visual complexity.

The tree is one of the most common relational structure. Many applications can be modeled as trees. Examples include family trees, hierarchical information, BFS tree of WWW graphs and phylogenetic trees. There are many layout methods for trees in two and three dimensions. However, most existing methods focus on two dimensions. Layout methods for trees in three dimensions are not well investigated.

We present a new drawing algorithm for trees in three dimensions. The algorithm uses the concept of the *subplanes*, where a set of subtrees are laid out. The main reason to use subplanes is to reduce visual complexity and for easy navigation. Note that 3D visualisation may suffer from occlusion and navigation problem. However, using subplanes defined by *regular polytopes*, the drawing is easy to navigate. Further, the algorithm can be implemented to run in linear time.

We implemented the algorithm and the experimental results show that it produces nice layouts of large trees with up to ten thousands nodes. Figure 1(a) shows a drawing of a tree with 6929 nodes using our algorithm. Here, we use the Icosahedron polytope to define 30 subplanes.



(a) (b)

Fig. 1. (a) Example output of the algorithm (b) example of a subplane.

This paper is organized as follows. In Section 2, we review previous drawing algorithms for trees. The main results of the paper is in Section 3: here we present a new drawing algorithm for drawing trees in three dimensions. An implementation and experimental results are described in Section 4. In Section 5, we discuss how to extend this method for more general case.

2 Related Work

There are many tree drawing algorithms and systems are available [2, 3, 7–9, 11–13]. For a survey, see [5]. These can be classified as 2D visualisation [2, 3,

7, 8, 11, 12] or 3D visualisation [9, 13], rooted tree [7, 12, 13] or free (unrooted) tree [3, 8, 11], binary tree [12] or general tree [3, 7, 8, 11, 13], and their aesthetics or optimization goal, such as efficient use of spaces [7, 11].

First we consider 2D tree drawing algorithms. The *radial drawing* algorithm is suitable for drawing free trees in two dimensions [3]. It uses concentric circles and then recursively draw each subtree in a wedge of the circle. However, there is some unused space in order to guarantee no edge crossings. For rooted binary trees, one can use *Tidier Drawing* algorithm [12].

Treemaps use a space filling technique for the visualisation of the tree in two dimensions [7]. It uses all available space, but it may be difficult to understand the relationship between the nodes[11]. The *hyperbolic tree browser* uses hyperbolic geometry for layout in two dimensions and classical *focus+context* techniques [8]. It produces nice layouts and can be used to visualise large hierarchies, up to a thousand or so nodes.

Recently, the *space optimization tree* was presented for viewing very large hierarchies in two dimensions [11]. The method uses the space in an optimized way and can display trees with up to 55000 nodes.

In three dimensions, *cone trees* are the best known [13]. They allow *focus+context* view and provide rotation operations. However, it uses only the surface of the cone and there is some unused space in 3D. They are able to display trees with thousands of nodes. The *H3* method uses hyperbolic geometry in three dimensions [9]. It produces a three dimensional layout for the spanning trees of large directed graphs, and also provides *focus+context* view.

However, in general, 3D drawing algorithms for trees are not well investigated compared to the methods in two dimensions. In this paper, we present a new drawing algorithm for trees in three dimensions.

3 The PolyPlane Algorithm

The algorithm uses the concept of *subplanes*, which are defined by regular polytopes. Roughly speaking, we choose the root of the tree and then partition the subtrees of the root. Then we assign each set of subtrees to a subplane and we draw each set on the subplane. An example of a subplane is shown in Figure 1(b).

The algorithm described in this paper treats the input tree as a rooted tree. The deletion of the root results in *subtrees* $T_1, T_2, \ldots, T_m$. The drawing algorithm draws the root of T at the origin o, and distributes the subtrees $T_1, T_2, \ldots, T_m$ onto disjoint subplanes P_j, which are equally spaced around the z axis.

The algorithm uses a two dimensional drawing algorithm, `Draw2D`, as a subroutine. This algorithm draws a rooted tree in a subplane. For the purposes of this paper there are no specific requirements for `Draw2D`; there are many linear time algorithms available (see [2]). Thus, the main algorithm can be described as follows.

`Algorithm PolyPlane`

1. Compute the core tree structure T of a graph G.
2. Choose the root r of the tree T.

3. Choose the regular polytope which defines the subplanes. Let j be the number of subplanes defined from the regular polytope.
4. Choose a partitioning $\mathcal{S} = S_1 \cup S_2 \cup \ldots \cup S_j$ of the set $\{T_i : 1 \leq i \leq m\}$ of subtrees rooted at r.
5. For all i, $1 \leq i \leq j$, consider the subtrees in S_i to be a single tree T_i' with a common root r. Use `Draw2D` to draw T_i' in the subplane P_i.

It is clear that the drawings in the subplanes have no edge crossings. Further, as long as we use a linear time algorithm `Draw2D`, and compute the partitioning at step 3 in linear time, the whole algorithm takes linear time.

Note that the algorithm is very *flexible*, as there are many steps at which an arbitrary choice can be made at each step. We now explain each step in details.

3.1 Computation of the Core Tree Structure

We first discuss how to compute the core tree structure of the large and complex networks. One can use a spanning tree, Steiner tree, BFS (Breath-First-Search) tree, or DFS (Depth-First-Search) tree based on the application domain.

For example, for weighted graphs, one may use the maximum weight spanning tree or minimum weight spanning trees. These can be computed in polynomial time. For the simplest, one can use a BFS or DFS Tree which can be computed in linear time.

3.2 Choice of the Root Node

The choice of the root may depend on the application domain. For example, the input tree may already have a designated root from hierarchy.

For free (unrooted) trees, one may choose the *center* of the tree as root. Every tree T has a center c, that is, a vertex such that the maximum distance from c to the leaves of the tree is minimized. Further it can be found in linear time.

In practice, one may choose *the (domain dependent) most important vertex* as the root. For example, highest degree vertex based on degree centrality, or betweenness centrality in social network analysis.

3.3 Choice of the Regular Polytopes

We use regular polytopes to define subplanes. The main reason to use regular polytopes is to provide an easy navigation method. There are many regular polytopes. However, all are variations on just three types: pyramids, prisms, and the Platonic solids. We now explain each polytope in detail.

Regular Pyramid A *regular pyramid* is a pyramid with a regular g-gon as its base. There is only one *g-fold rotation axis*, called the *vertical rotation axis*, passing through the apex and the center of its base. The axis defines the g-fold rotational symmetries of the regular pyramid polytope. An example is shown in Figure 2(a). Also, there are g reflection planes, called *vertical reflection* planes, each containing the principal axis.

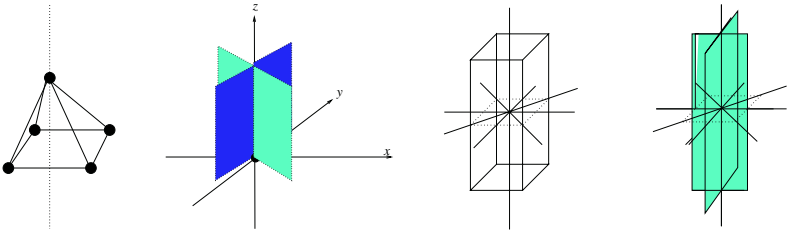


Fig. 2. (a) pyramid polytope (b) subplanes of pyramid polytope (c) prism polytope and subplanes.

We can define the g reflection planes, each of which is a rotation of $2\pi i/g$, $i = 0, 1, \ldots, g-1$. The basic idea is to construct a regular pyramid drawing of a tree by placing the center of the tree at the apex of the pyramid, and the g partitioned subtrees on the subplanes that contain the side edges of the pyramid. An example is illustrated in Figure 2(b).

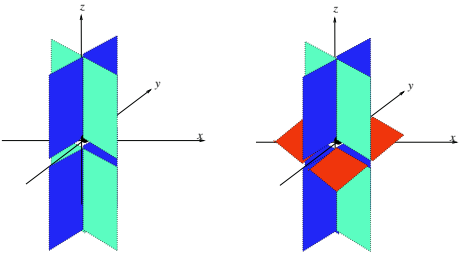
Regular Prism A *regular prism* has a regular g-gon as its top and bottom face. There are $g+1$ rotation axes and they can be divided into two classes. The first one, called the *principal* axis or *vertical rotation* axis, is a g-fold rotation axis which passes through the centers of the two g-gon faces. The second class, called *secondary* axes or *horizontal rotation* axes, consists of g 2-fold rotation axes which lie in a plane perpendicular to the principal axis. Also, there are g reflection planes, called *vertical reflection* planes, each containing the principal axis, and another reflection plane perpendicular to the principal axis, called *horizontal reflection*. Figure 2(c) shows the rotation axes in the prism polytope.

We can define the $2g$ reflection planes, each of which is a rotation of $2\pi i/g$, $i = 0, 1, \ldots, g-1$ in both directions of z coordinates. The basic idea is to construct a regular prism drawing of a tree by placing the center of the tree at the centroid of a prism, and $2g$ partitioned subtrees on the subplanes that contain the side edges of the prism.

An example is illustrated in Figure 3. In fact, there are two variations. One can define $2g$ subplanes as illustrated in Figure 3(a), or $3g$ subplanes as in Figure 3(b), which include g subplanes on the xy plane.

The Platonic Solids Basically, we use the rotation axes of the regular polytopes to define the subplanes. Using the regular g-gon pyramid polytope, we can define g subplanes, and using the regular g-gon prism polytope, we can define either $2g$ or $3g$ subplanes. For the Platonic solids, we can define more subplanes, as there are more rotation axes.

The tetrahedron has four 3-fold rotation axes and three 2-fold rotation axes. It has 12 rotational symmetries and 24 symmetries in total. The cube has three 4-fold rotation axes, four 3-fold axes, and six 2-fold rotation axes. It has 24 rotational symmetries and a full symmetry group of size 48. The icosahedron has six 5-fold rotation axes, ten 3-fold rotation axes, and fifteen 2-fold rotation



(a) (b)

Fig. 3. Example of the subplanes of the regular prism polytope.

axes. It has 60 rotational symmetries and a full symmetry group of size 120. Note that the cube and the octahedron are dual solids, and the dodecahedron and the icosahedron are dual.

As an example, we consider the case of the cube. We can define 24 subplanes using the cube. The cube has three 4-fold axes, and these can define six half-axes. We can use each of the half-axis, to define 4-subplanes. This means that we can define 24 subplanes in total.

Similarly, we can define 12 subplanes using the tetrahedron, 3 subplanes each around four 3-fold axes. We can also define 60 subplanes using the icosahedron, as we can define 5 subplanes each around six 5-fold axes and these six 5-fold axes define 12 half-axes.

3.4 The Partitioning Algorithm

Once we have chosen the regular polytope and fixed the number g of the subplanes, we then need to divide the subtrees into g subsets.

For this step, we need to find a balanced partitioning of the subtrees. This problem can be formulated as a traditional bin-packing problem. Note that the bin-packing problem is NP-hard [4]. However, many heuristics and approximation algorithms are available [1, 4]. For our implementation, we use first-fit and best-fit [4]. One main advantage for using these heuristics is that they run in linear time. One may use other well-known approximation algorithms for more sophisticated balancing. For details, see [1].

Note that in many applications, the partitioning may be given based on clustering or analysis of the large and complex networks. For example, in social network analysis, clustering can be defined using centrality or status measure. In biological networks, clustering can be defined by users or functionality.

3.5 The 2D Drawing Algorithm

Once we have chosen the regular polytope and computed a partitioning of the subtrees into g subsets, we then choose a 2D drawing algorithm for trees to draw each subset in a *subplane*. Formally, a subplane can be defined as a maximal simply connected open subset of a reflection plane that does not intersect any other reflection plane.

Many linear time algorithms are available to implement Draw2D(see [2]). For our implementation, we choose the radial drawing algorithm [3] to create the drawings in the subplanes as wedges. To guarantee no edge crossings, we allow a small space between each pair of subplanes.

4 Implementation and Experimental Results

We implemented the new layout algorithm as a part of the system *3DTree-Draw* [10]. In fact, the system 3DTreeDraw implements two 3D tree drawing algorithms.

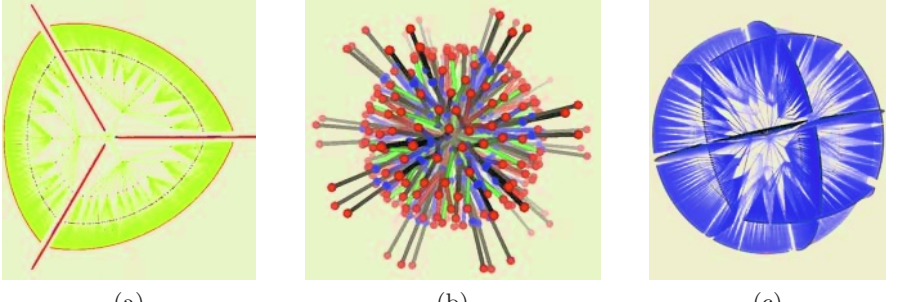
The first one is a symmetric drawing algorithm [6], which finds the maximum number of symmetries in a tree and then constructs a maximally symmetric drawing of trees in three dimensions. The second algorithm is the algorithm PolyPlane, which is presented in this paper.

The system also provides simple zoom in and zoom out functions, as well as rotation of the 3D drawing. This rotation function is sufficient for navigation, as the subplanes were defined using regular polytopes which make the drawing easy to navigate. It also provides a function that you can save the result as a bmp file.

We use two different data sets, regular data sets and real world data sets from software engineering, webgraph and social network domains. We now present experimental results in details.

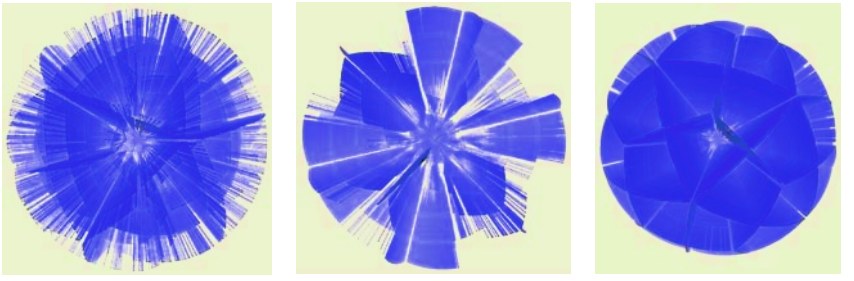
Firstly, we use randomly generated regular data sets, from a few hundred up to a hundred thousands nodes. The experimental results show that it produces aesthetically pleasing drawings of trees with up to ten thousands nodes. For the regular data sets, the drawings produce balanced appearance. Figure 4(a) shows a tree with 8613 nodes, using a regular 3-gon prism polytope with 6 subplanes.

One can define more subplanes based on variations of the Platonic solids. For example, in Figure 1, we can define three planes in each of the triangular shape space recursively. This improves resolution. However, we observed that too many planes make navigation a bit difficult. See Figure 4 (b) for an example. The tree has 483 nodes and it was drawn using the icosahedron polytopes.



(a) (b) (c)

Fig. 4. (a) tree with 8613 nodes drawn with prism polytope (6 subplanes) (b) tree with 483 nodes drawn with the icosahedron polytope (c) tree with 6929 nodes drawn with the cube and the octahedron polytopes (36 subplanes).



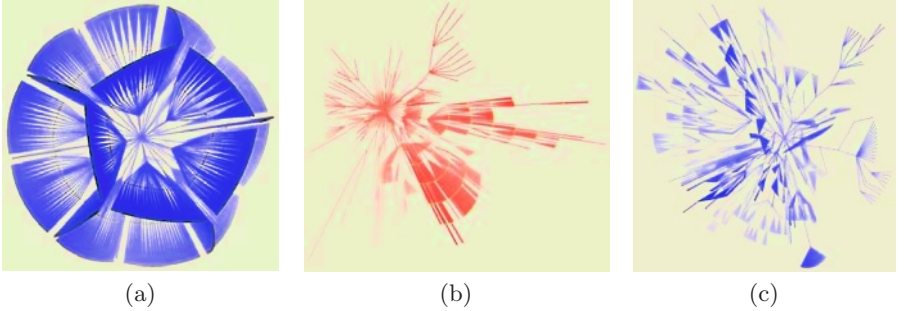
(a) (b) (c)

Fig. 5. Trees drawn the dodecahedron and the icosahedron polytopes (90 subplanes) (a) tree with 22001 nodes (b) tree with 59732 nodes (c) tree with 139681 nodes.

Figure 4(c) shows a tree with 6929 nodes, using a variation of the cube and the octahedron polytopes with 36 subplanes. Figure 5(a) shows a tree with 22001 nodes, using a variation of the dodecahedron and the icosahedron polytopes with 90 subplanes. Figure 5(b) shows a tree with 59732 nodes, using a variation of the dodecahedron and the icosahedron polytopes with 90 subplanes. Figure 5(c) shows a tree with 139681 nodes, using a variation of the dodecahedron and the icosahedron polytopes with 90 subplanes. Figure 6(a) shows a tree with 6929 nodes, using the dodecahedron polytope with 30 subplanes.

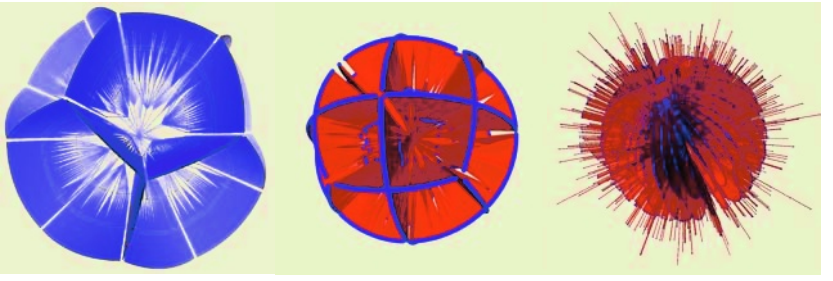
Finally, we apply our algorithm to visualise large and complex network from real world data. Figure 6(b) shows a home directory with 1385 nodes, using the icosahedron polytope with 30 subplanes.

Figure 6(c) shows a BFS tree of the School of IT, University of Sydney webgraph, with 4485 nodes, using the cube polytope with 12 subplanes. This is rooted at the main home page, and the core tree structure is a BFS tree. Figure 7(a) shows a tree of depth-limited search of the it.usyd.edu.au website with 146716 nodes, using a variation of the dodecahedron polytope with 30 subplanes.



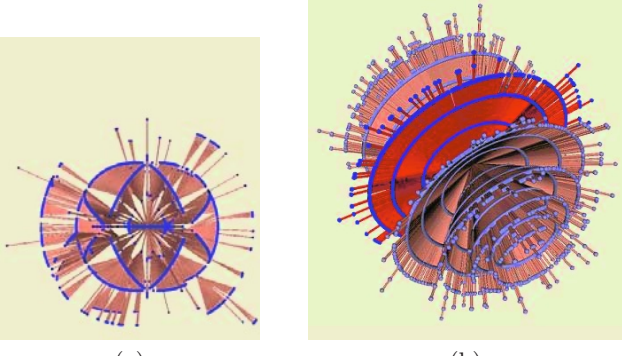
(a) (b) (c)

Fig. 6. (a) tree with 6929 nodes drawn with the dodecahedron polytope (30 subplanes) (b) a home directory with 1385 nodes drawn with the icosahedron polytope (30 subplanes) (c) BFS tree of School of IT website with 4485 nodes drawn with the cube polytope (12 subplanes).



(a) (b) (c)

Fig. 7. (a) tree of depth-limited search of website with 146716 nodes drawn with the dodecahedron polytope (30 subplanes) (b) Erdos Number visualisation using the icosahedron polytope (30 subplanes) (c) Kevin Bacon number visualisation using the prism.



(a) (b)

Fig. 8. Kevin Bacon number visualisations using (a) the icosahedron polytope with 30 subplanes (b) multiple planes defined by concentric cones.

Figure 7(b) shows Erdos number visualisation of mathematician collaboration network. We root at Erdos and compute a BFS tree. We then use the icosahedron polytope with 30 subplanes. Note that the current database keeps a record up to Erdos number 2, hence it displays a balanced appearance.

Figure 7(c), 8(a), and 8(b) show "Kevin Bacon number visualisations" of Hollywood movie actor collaboration network. We root at Kevin Bacon and then compute a BFS tree to visualise Kevin Bacon Number. We use three different polytopes: Figure 7(c) uses the prism polytope, Figure 8(a) uses the icosahedron polytope with 30 subplanes, and Figure 8(b) uses a variation of the `PolyPlane` algorithm with multiple planes defined by concentric cones.

In summary `PolyPlane` produces nice layouts. In particular, the effective use of subplanes reduces both visual and cognitive complexity and provides easy navigation.

5 Conclusion and Current Work

In this paper, we present a simple method to visualise the core tree structure of a large and complex network in three dimensions. The algorithm uses the concept

of subplanes which are defined using regular polytopes. The algorithm is easy to implement and runs in linear time.

The algorithm is flexible, as one can choose the regular polytope and the 2D drawing algorithm for their own purpose. For example, for rooted trees, the pyramid polytope is more suitable. For dense trees with small diameter, the prism polytope or one of the Platonic solids is preferred. To improve resolution, one can define more subplanes using the method described in Section 4. However, there is a trade off between the number of planes and the navigation problem.

In summary, `PolyPlane` has the following advantages. It is flexible, easy to implement and can run in linear time. It can scale very well and is suitable for visualising a tree with high degree nodes, short diameter, or short average path length.

Our current work is to implement good navigation methods for `PolyPlane` and extend the method to cover more general cases. That is, to draw the whole network using multiple planes.

We consider many different variations as extensions. One is to use two parallel planes, to draw an *important subgraph* of the network on the top plane and draw the remaining subgraph of the network in the bottom plane. The important subgraph can be a set of vertices, a set of edges, or a small subgraph of the network. Note that this method can be generalised using up to k planes and a method to draw hierarchical graphs in three dimensions.

Another extension is to extend this multi plane idea to draw clustered graphs in three dimensions. Both involve some fundamental problems that need to be solved.

References

1. E. G. Coffman, M. R. Garey and D. S. Johnson, Approximation Algorithms for Bin Packing: A Survey, Approximation Algorithms for NP-Hard Problems, D. Hochbaum (editor), PWS Publishing, Boston, pp. 46-93. 1997.
2. G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, 1998.
3. P. Eades, Drawing Free Trees, Bulleting of the Institute of Combinatorics and its Applications, pp. 10-36, 1992.
4. M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP Completeness, *Freeman*, 1979.
5. I. Herman, G. Melançon G, M. Marshall, Graph Visualization in Information Visualization: a Survey, IEEE Transactions on Visualization and Computer Graphics, 6, pp. 24-44, 2000.
6. S. Hong and P. Eades, Drawing Trees Symmetrically in Three Dimensions, *Algorithmica*, vol. 36, no. 2, 2003.
7. B. Johnson and B. Shneiderman, Tree-maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures, Proc. of IEEE Visualization'91, IEEE, Piscataway, NJ, pp. 284-291, 1991.
8. J. Lamping, R. Rao and P. Piroli, A Focus + Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies, Proc. of ACM CHI'95 Conference: Human Factors in Computing Systems, ACM, New York, NY, pp. 401-408, 1995.

9. T. Munzner, H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space, Proc. of the 1997 IEEE Symposium on Information Visualization, pp. 2-10, 1997.
10. T. Murtagh and S. Hong, 3DTreeDraw: A Three Dimensional Tree Drawing System, Proc. of SoCG, pp. 380-382, 2003.
11. Q. V. Nguyen and M. Huang, A Space-Optimized Tree Visualization, Proc. of IEEE Symposium on Information Visualization (InfoVis2002), pp. 85-92, 2002.
12. E. Reingold and J. Tilford, Tidier Drawing of Trees, IEEE Transactions on Software Engineering, 7, pp 223-228, 1981.
13. G. Roberston, J. Mackinlay and S. Card, Cone Trees: Animated 3D Visualizations of Hierarchical Information, Proc. of SIGCHI'91, pp. 189-194, 1991.