Portable Graph Layout and Editing

Brendan Madden, Patrick Madden, Steve Powers, and Michael Himsolt

Tom Sawyer Software, 1828 Fourth Street, Berkeley, CA 94710
info@TomSawyer.COM

Abstract. The *Graph Layout Toolkit* and the *Graph Layout Toolkit* are portable, flexible toolkits for graph layout and graph editing systems. The Graph Layout Toolkit contains four highly customizable layout algorithms, and supports hierarchical graphs. The Graph Editor Toolkit is a tightly coupled interactive front end to the Graph Layout Toolkit.

1 Introduction

The visualization of graphs has become very important over the last few years. Some examples include project management, compiler and software development tools, work-flow, and reverse engineering applications. Other applications include those for network management, CAD and CASE, diagramming, and database design.

Many graph layout and editing systems have been developed during the last ten years, examples include *D-ABDUCTOR*, *dag*, *dot*, *daVinci*, *Diagram Server*, *EDGE*, *GEM*, *grab*, *GraphEd* and *vcg* (for an overview, see [DETT95]). Research projects generally have different design, documentation, packaging, and testing goals than those of commercial software.

In addition, research systems usually have the freedom to experiment with novel ideas with uncertain commercial value. However, our goals are often different as a commercial vendor. Our approach is generally to seek very high quality layout algorithms that have general applicability to groups of markets.

In a commercial environment, it is usually better to avoid techniques that have limited applicability. A novel approach to a graph drawing problem may confuse, and sell poorly, if the solution does not match the traditions or expectations of the particular customer. We have produced a family of layout styles to try to anticipate the needs of diverse markets with varying requirements.

In production applications, it is unlikely that any restrictions on input graphs such as planarity, maximum degree or biconnectivity will hold in general. In spite of that, we try to adopt many useful techniques from graph drawing literature into our framework when those techniques can be made generally applicable. However, it seems to be very difficult, and perhaps impossible, to satisfy all of the requirements of commercial applications.

Furthermore, the Graph Layout Toolkit and the Graph Editor Toolkit support a generalized framework to visualize information that spans across many linked graphs. A programmer or user may navigate easily from one graph to another, or many graphs may be interactively nested in the same plane if desired. Each graph also maintains its own layout tailoring properties.

Additionally, it is very important that layout and editing systems are portable. Ideally, they should work with a number of different compilers, operating systems, industry standard graphics class libraries, rapid application development tools, and be easily embedded into other applications.

Tom Sawyer Software's Graph Layout Toolkit and Graph Editor Toolkit are mature software packages that provide many of the above-mentioned features.

2 The Graph Layout Toolkit

The Graph Layout Toolkit [GLT95a, GLT95b] currently supports four different layout styles: circular, hierarchical, orthogonal, and symmetric. Each layout style derives from a virtual function driven layout class hierarchy that is loosely coupled with a graph management class hierarchy. This design allows the user to flexibly switch between the installed layout styles at any time.

The graph class hierarchy supports directed and undirected graphs with multiedges and reflexive edges. There are no imposed implementation limits on the size of a graph, its maximum degree, and no restrictions on the classes of graphs that can be specified. All layout algorithms also support multiple connected components, navigation, variable node dimensions, edges with bends, and reading and writing of graphs and their drawings from and to disk.

Edges may either be drawn as straight lines or may be represented as a sequence of lines. An edge with bends owns a graph, i.e. a path, that is a sorted chain of dummy nodes and edges. This technique ensures that paths can be managed through standard graph operations. Each new layout algorithm extends the generic graph and layout services that are provided by the framework. The virtual function interface and separate name spaces ensure that one layout algorithm does not adversely affect another.

The Graph Layout Toolkit provides operations that make it easy to integrate graph drawing techniques into various applications. It supports a number of features for portable interactive editing such as continuously up-to-date drawing, and cut, copy, paste, duplicate, and clear functionality. Users just plug in their domain-specific graphics calls. It further supports graph and layout replication features and subject/view relations where several objects may be attached to a node or an edge and get notified of changes. Graphical queries can be applied to graphs and displayed in different windows with minimal programming overhead.

The Graph Layout Toolkit also supports highly customizable Postscript^{TM} and Encapsulated Postscript^{TM} output. It generates gray scale or color output and can distribute a drawing over multiple pages of arbitrary size. It can also optionally scale and rotate, insert crop marks, and insert page labeling and numbering detail into the final drawing. It has an extensible design that enables application developers to load icons into nodes, and to write application-specific PostScript procedures.

The Graph Layout Toolkit is implemented in C++, and provides more than six hundred functions for each of the C++ and ANSI C application programmer interfaces. It has been ported to more than fifteen different compilers and runs on all standard operating systems.

2.1 Circular Layout

The circular layout algorithm is designed for the layout of ring and star network topologies. It is an advanced version of the one developed by Kar, Madden, and Gilbert [KMG89]. It functions by partitioning nodes into logical groups based on a number of flexible node grouping models. Each group of nodes is placed on radiating circles based on their logical interconnection. The partitioning is either performed with a pre-defined method, such as biconnectivity of the graph or the degree of nodes, IP addresses, IP subnet masks or another domain specific technique. It also supports manually configured clustering.

The algorithm minimizes cluster to cluster crossings as well as crossings within each cluster. In additions, it also employs tree balancing routines, and has ring and star detection and placement techniques within each cluster. Figure 1 shows two sample drawings.

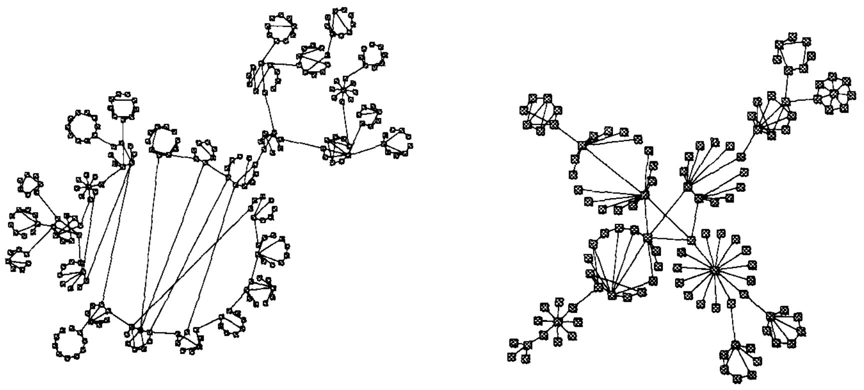


Fig. 1. Examples of circular layout.

2.2 Hierarchical Layout

The hierarchical layout algorithm is designed to lay out directed graphs. It has applications in project management software, compiler and software development tools, information management applications, work-flow, reverse engineering applications, database schemata and network management applications. Figure 2 shows two examples of this style.

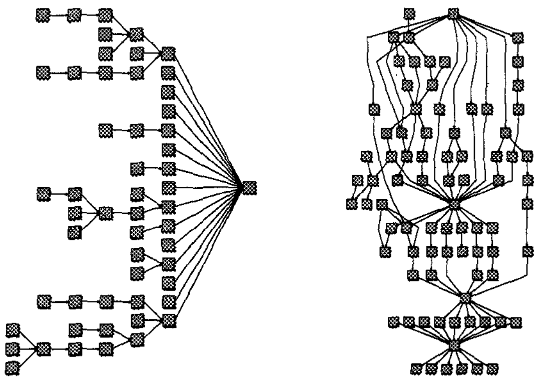


Fig. 2. Examples of hierarchical layout.

The hierarchical layout algorithm is a heavily optimized and extended version of the standard algorithm by Sugiyama, Tagawa and Toda [STT81] that organizes the nodes of a graph into levels, adds dummy nodes, as necessary, into the drawing to produce a proper hierarchy and reduces the number of crossings between levels. It supports both directed and undirected graphs and can optimally resolve cycles in directed graphs.

A graph can be laid out either horizontally or vertically. Properties of the drawing, such as edge to node attachment, node justification, the minimum slope of an edge or the spacing between nodes and levels can be controlled by the user, please refer to Fig. 10. Special tree balancing algorithms help to draw class hierarchies tidy. Recently, a multi-pass placement engine has been written to improve the placement of nodes that lack either parents or children. This eliminates some shortcomings of traditional barycentric techniques.

Programmers of network applications often create bipartite graphs when there is inherent semantic information in the graph, i.e. routers connecting to networks. If one uses the standard layout techniques from [STT81] for directed graphs, the layout that will result is usually an unreadable two-level hierarchy.

Therefore, we have extended the hierarchical layout style to unfold undirected network structures in a number of novel ways. One method employs a bipolar acyclic ordering for the biconnected components. Ethernet nodes typically have very high degree and variable width. Subtrees and end-nodes, i.e. terminals and hosts, are selectively inverted around these possibly variable-width Ethernet nodes, as shown in Figure 3.

These techniques have been applied to the Network Layout Assistant [NLA95] which is an add-on for a network management software package from Sun Microsystems named Solstice SunNet Manager^{TM}. In this system, Solstice SunNet Manager^{TM} discovers devices on a network from which the Network Layout Assistant generates an intelligent drawing.

2.3 Orthogonal Layout

The orthogonal layout algorithm is based on papers by Biedl and Kant [BK94] and Papakostas and Tollis [PT94]. Orthogonal layouts are specifically suited for database design, object oriented analysis and design, and CAD and CASE diagrams. Figures 4 and 5 show examples of drawings of this style.

The algorithm produces a constant number of bends per edge and has relatively high performance in practice. It has further been extended by Biedl and Papakostas at Tom Sawyer Software to produce more aesthetically pleasing drawings than those of the currently published B/K and P/T algorithms. With this algorithm, planar graphs can be drawn planar, and nonplanar graphs are also easily supported. Support has been recently added for nodes with degree greater than four and for row and column reuse. For high degree nodes, the user has the choice to keep node dimensions fixed to their specified input which implies decreasing the separation of grid lines. Alternatively, the user can allow the node area to increase to allow high degree while keeping the separation of grid lines fixed.

2.4 Symmetric Layout

The symmetric layout algorithm is generally designed to display networks. The drawings stress symmetric and isomorphic substructures, and its uniform distribution of nodes and edges, in general, yields aesthetically pleasing drawings. Both directed and undirected graphs are supported. Figure 6 shows two examples. The algorithm is originally based on work by Kamada and Kawai [KK89].

Several parameters of this algorithm are available in order to fine-tune the heuristics and the termination conditions. These include the strength of the spring constant and the numbers of iterations that are performed. Generally, it produces relatively few edge crossings. We have also been able to more than quadruple the performance over subsequent Graph Layout Toolkit releases. We will try to continue this trend over coming releases to quickly lay out very large graphs.

2.5 Hierarchical Graphs

Hierarchical graphs become important when information becomes too complex to be modeled with a single graph, or when information is more naturally modelled hierarchically. The *navigation manager* realizes hierarchical relations among graphs:

1. *Node to graph* navigation supports hierarchical relations from a node to a graph. Figure 7 shows several examples of node to graph relationships. These relations can be visualized by expanding the parent node so that it visually contains the child graph.

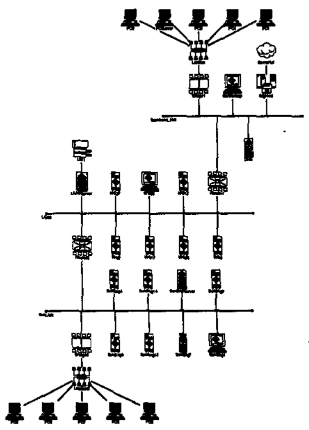


Fig. 3. A network drawing generated by the Network Layout Assistant. The Network Layout Assistant is a plug-in module for Solstice SunNet Manager^{TM} that draws networks with the Graph Layout Toolkit.

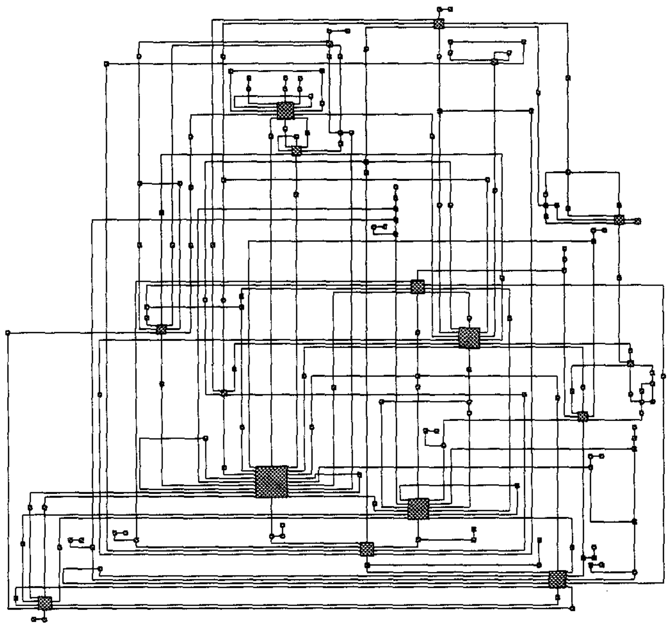


Fig. 4. Example of orthogonal layout.

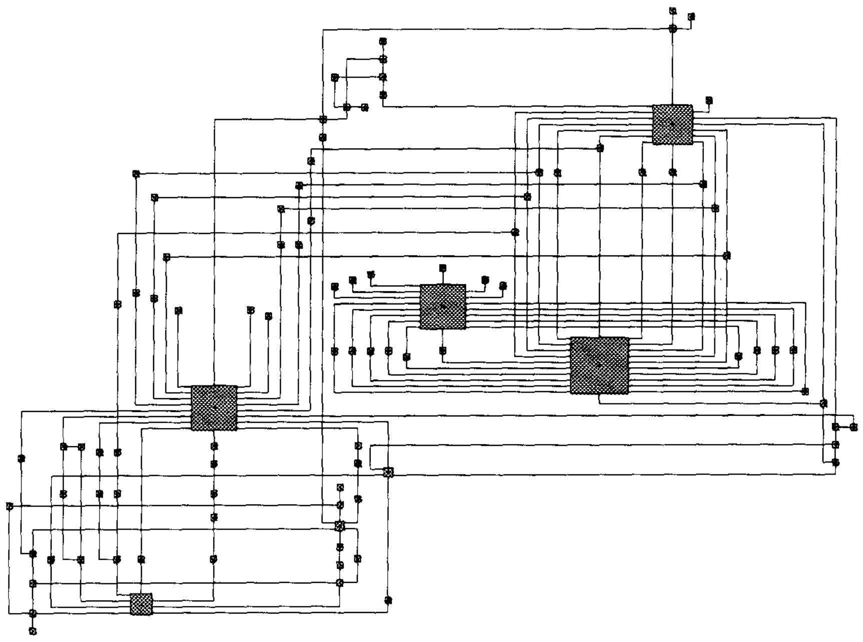


Fig. 5. Example of orthogonal layout.

2. *Edge to graph* navigation supports hierarchical relations from an edge to a graph. An edge (or node) to graph relation can be also be visualized by displaying the target graph in with another window or another application specific technique.

A navigation manager is organized as several graphs which model the hierarchical relation. One graph manages node to graph and edge to graph relations. Another recalculation graph manages which graphs are dependent on which in order to perform layouts in the correct order when graphs are nested. Arbitrarily many graphs can be recursively nested within a navigation manager to create large hypertext-like structures. Individual nodes and edges may also navigate to many graphs; further, many nodes and edges may navigate to the same graph.

The Graph Layout Toolkit supports interactive collapse and expand operations with continuously up-to-date drawings to show and hide detail in large structures. Nodes are scaled to fit around their contained graphs based on a post-order traversal of the nodes in the graph that maintains the navigation recalculation information. Recalculation operations are extremely fast under small changes to drawings as each graph maintains information that indicates whether it needs to be laid out again.

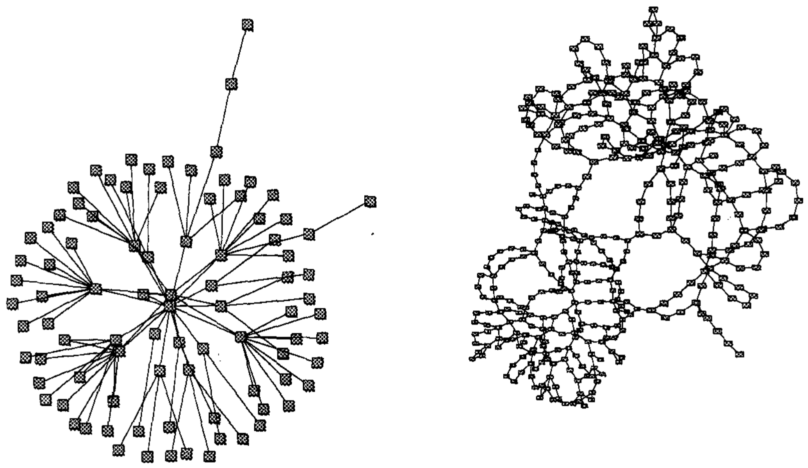


Fig. 6. Examples of symmetric layout.

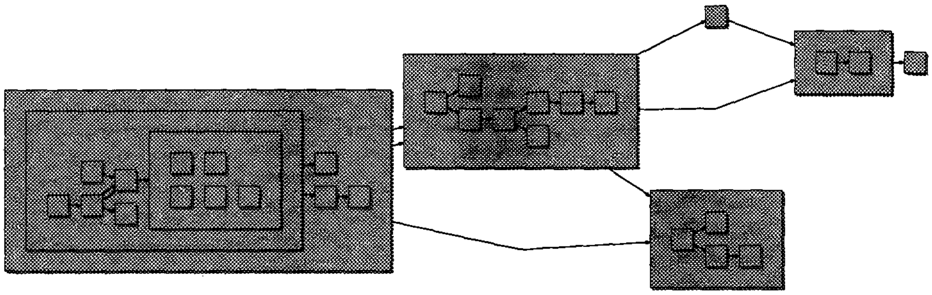


Fig. 7. An example of a hierarchical graph with several *node to graph* relations. Note the graphical visualization where the parent nodes contain the children graphs.

Each graph is laid out in an arbitrary world coordinate system and transformed dynamically into the coordinate system of the bottom-most graph in the drawing. This design helps to keep children graphs and their layouts invariant under collapse and expand operations.

Since the layout algorithms process each graph separately, layout works faster on hierarchically organized graphs than on flat graphs, and the structure is preserved. Navigation operations are efficient enough to manage hundreds of small graphs at the same time with an interactive graph editor.

3 The Graph Editor Toolkit

The Graph Editor Toolkit [GET95] is a portable front end for the Graph Layout Toolkit. It provides a user interface to create, display and edit graphs. Figure 8 shows a screen capture of the editor. Figures 9 and 10 show two dialogs of the editor. It is designed to be extended at the back end to tie it to various data sources and to be customized at the front end to add application semantics.

The Graph Editor Toolkit derives its classes from Graph Layout Toolkit classes and extends them with methods to handle user interface operations such as dispatching user input and drawing nodes and edges. It consists of four independent subsystems:

- The *Engine* is a portable module that determines how the editor reacts to input. The engine extends the Graph Layout Toolkit's classes into user interface aware graph, node, and edge classes. Potential users of the editor may extend these classes to adapt them for their own need.
- The *DocView manager* implements a document/view model that allows one graph to be displayed in more than one window, but with different visual representations.
- The *Graphics manager* can display nodes and edges as arbitrary graphic widgets. Each node is represented by a matrix of graphic primitives. In the standard cases, this is a 1×2 matrix that consists of a bitmap and a text label. Databases may use $2 \times n$ tables to display data entries.
- The *Dialog manager* interacts with the windowing system to generate windows, menus, and dialogs.

Naturally, the Graphics and the Dialog manager depend on the underlying toolkit and window system, and are less portable than the Engine and DocView managers.

The Graph Editor Toolkit is currently available as a stand-alone application or as a library for Borland ObjectWindows^{TM} based programs for Microsoft Windows 3.1, Windows '95, and Windows NT.

4 Outlook

There are many improvements and extensions planned for the Graph Layout Toolkit and the Graph Editor Toolkit. Extensions will include incremental and interactive algorithms, constraint systems, enhanced navigation systems, and labeling systems. A Graph Editor Toolkit portable editor framework is being developed and the graphics portions are being ported to the Microsoft Foundation Classes for Windows, Microsoft OLE, and OSF/Motif^{TM} for UNIX platforms.

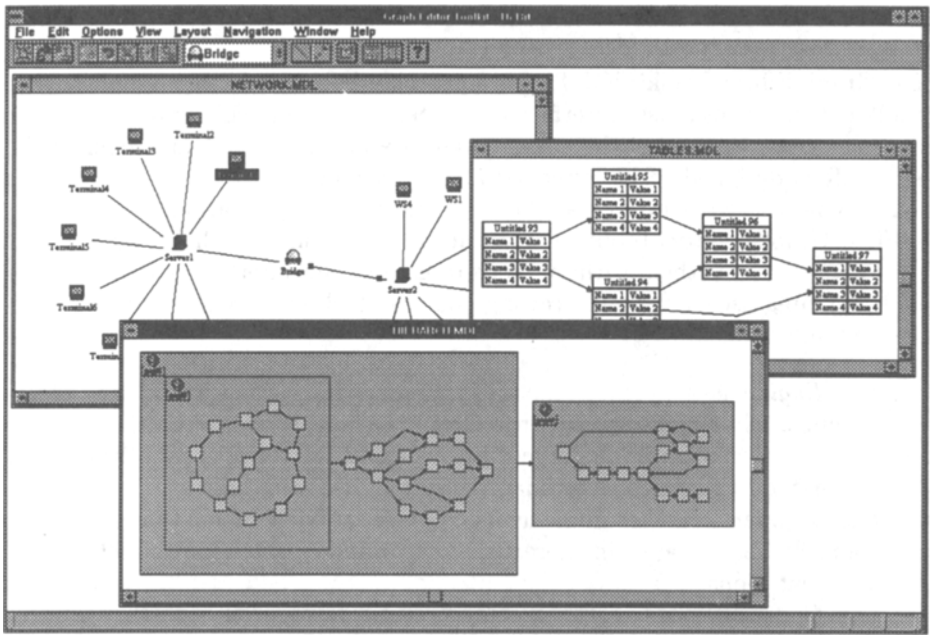
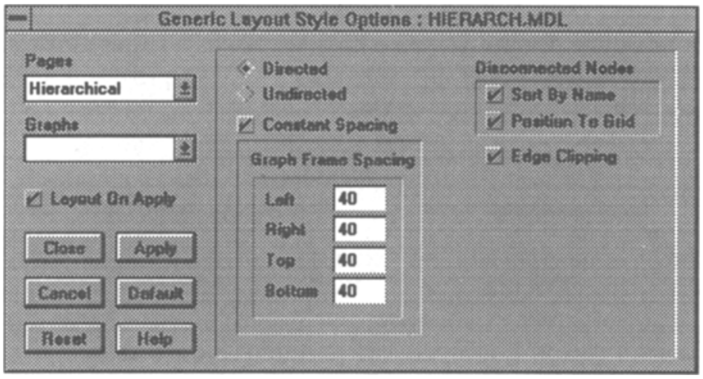


Fig. 8. The Graph Editor Toolkit for Microsoft Windows^{TM} 3.1.


Generic Layout Style Options : HIERARCH.MDL

Pages: Hierarchical

Graphs:

☑ Layout On Apply

◆ Directed

◇ Undirected

☑ Constant Spacing

Graph Frame Spacing

Left	40
Right	40
Top	40
Bottom	40

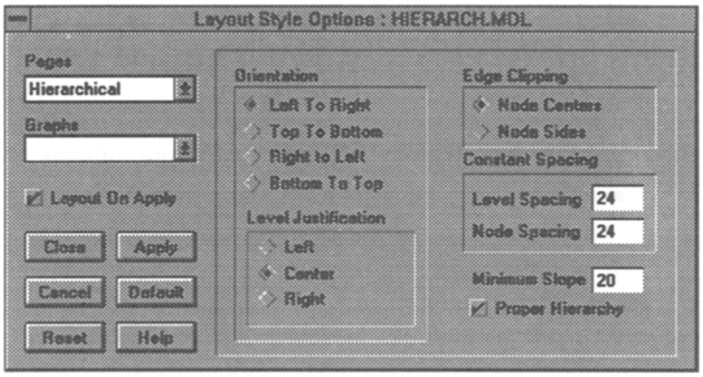
Disconnected Nodes

☑ Sort By Name

☑ Position To Grid

☑ Edge Clipping

Fig. 9. This Graph Editor Toolkit dialog is used to set the basic layout options.


Layout Style Options : HIERARCH.MDL

Pages: Hierarchical

Graphs:

[✓ Layout On Apply]

Orientation: [✓ Left To Right] Top To Bottom Right to Left Bottom To Top

Level Justification: Left [✓ Center] Right

Edge Clipping: [✓ Node Centers] Node Sides

Constant Spacing: Level Spacing 24, Node Spacing 24

Minimum Slope 20

[✓ Proper Hierarchy]

Fig. 10. This Graph Editor Toolkit dialog is used to set the options for the hierarchical layout style.

References

[BK94] Biedl, T., Kant, G.: *A Better Heuristic for Orthogonal Graph Drawings*, In: Proc. of the 2nd European Symp. on Algorithms (ESA 94), Lecture Notes in Computer Science **855** Springer-Verlag (1994) 124–135.

[DETT95] Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: *Algorithms for drawing graphs: An annotated bibliography.* In Computational Geometry: Theory and Applications **4**, (1994), 235–282.

[GET95] Tom Sawyer Software: *Graph Editor Toolkit Manuals*, Berkeley, CA (1995) (to appear).

[GLT95a] Tom Sawyer Software: *Graph Layout Toolkit User's Guide*, Berkeley, CA (1992 - 1995).

[GLT95b] Tom Sawyer Software: *Graph Layout Toolkit Reference Manual*, Berkeley, CA (1992 - 1995).

[KK89] Kamada, T., Kawai, S.: *An Algorithm for Drawing General Undirected Graphs*, Information Processing Letters **31** (1989) 7–15.

[KMG89] Kar, G., Madden, B.P., Gilbert, R.S.: *Heuristic Layout Algorithms for Network Management Presentation Services*, IEEE Network November (1988) 29–36.

[NLA95] Tom Sawyer Software: *Network Layout Assistant User's Guide* (1993 - 1995).

[PT94] Papakostas, A., Tollis, I.G.: *Improved algorithms and bounds for orthogonal drawings*, In: Proc. Graph Drawing '94, Lecture Notes in Computer Science **894**, Springer-Verlag (1994) 40–51. (a revised version is in progress).

[STT81] Sugiyama, K., Tagawa, S., Toda, M.: *Methods for visual understanding of hierarchical systems.* IEEE Transactions on Systems, Man and Cybernetics **11** (1981) 109–125.