Directed Graphs Drawing by Clan-Based Decomposition

Fwu-Shan Shieh and Carolyn L. McCreary
Department of Computer Science and Engineering
Auburn University, Alabama, USA
{fwushan, mccreary}@eng.auburn.edu

Abstract. This paper presents a system for automatically drawing directed graphs by using a graphanalysis that decomposes a graph into modules we call clans. Our system, CG (Clan-based Graph Drawing Tool), uses a unique clan-based graph decomposition to determine intrinsic subgraphs (clans) in the original graph and to produce a parse tree. The tree is given attributes that specify the node layout. CG then uses tree properties with the addition of "routing nodes" to route the edges. The objective of the system is to provide, automatically, an aesthetically pleasing visual layout for arbitrary directed graphs. Using the clan-based decomposition, CG's drawings are unique in several ways: (1)The node layout can be balanced both vertically and horizontally; (2) Nodes within a clan, a subgraph of nodes that have a common relationship with the rest of the nodes in the graph, are placed close to each other in the drawing; (3) Nodes are grouped according to a two-dimensional affinity rather than a single dimension such as level or rank [13]; (4) The users can contract a clan into a single node and later expand the node to show the subgraph in its original clan; and (5) Crossings reduction processing by clan-based graph decomposition is faster than Sugiyama, Tagawa, and Toda [20, 21] barycentric ordering algorithm.

In addition to the capabilities of the old drawing system [16], several features have been added: (1) The modified barycentric technique is used to reduce crossings. In the modified technique, the components of matrix representation are clans instead of nodes. The users can (2) specify a node's size, shape, and label, and a edge's label in the textual input file; (3) contract a clan (subgraph) into a single node; (4) extract a clan (subgraph) and hide the rest of the graph; and (5) save the drawing into a file in Postscript format.

1. Introduction

Directed graphs, or digraphs, are an excellent means of conveying the structure and operation of many types of systems. They are capable of representing not only the overall structure of such a system, but also the smallest details in a simple and effective way. However, drawing digraphs by hand can be tedious and time consuming, because much time can be spent just trying to plan how the graph should be organized on the page, especially if the number of nodes and edges is large. In addition, it is difficult for a user to draw a graph when the data is generated by applications (e.g., compiler-generated parse trees[1] and dialogue state diagrams generated by reverse engineering [3]). We have developed an automated system capable of converting a textual description of a digraph into a well organized and readable drawing of the digraph.

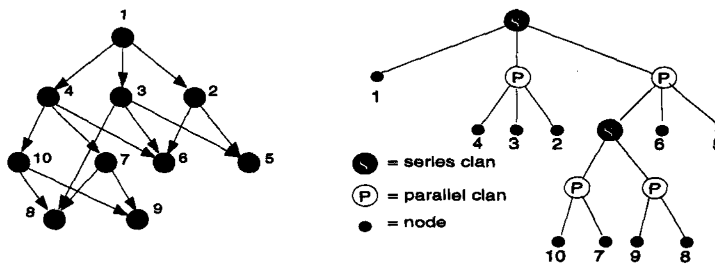
Many researchers have studied this problem and many graph drawing systems have been developed [see 6 for complete list]. The aesthetic criteria of the systems vary. The objectives may include requirements of uniform edge length, minimum number of edge crossings, straight edges, grid drawings (edges are either horizontal or vertical), minimal bends in the edges, etc. Cruz and Tamassia [4], Tamassia, Batini, and Battista [22], and Messinger [17] have lists of aesthetic criteria of drawings. Some criteria limit the input graphs to a particular class such as planar graphs, trees, graphs with maximum degree of four, or some application-specific graphs such as Petri nets, data-flow diagrams, DBMS models diagrams, digital system schematic diagrams, PERT diagrams, flowcharts, etc. Originally, CG is designed for program dependency graphs of parallel computation, and has been adopted by social networks, automatic graphical user interface design, reverse engineering graphical representations [3]. Like dot [13, 14] and its predecessor DAG [12], CG takes a textual description of an arbitrary directed graph (digraph) and produces a visual representation of it.

The remainder of the paper is divided into several sections which are related to CG: (section 2) clan-based graph decomposition; (section 3) node layout; (section 4) edge routing; (section 5) crossings reduction; (sections 6) subgraphs contract / extract; (section 7) cyclic directed graphs; and (section 8) example of applications.

2. Clan-Based Graph Decomposition

In general, a graph can be decomposed in two ways: (1) application-specific decompositions suggested by the semantics of the input graph; and (2) graph-theoretic decompositions based on syntactic decomposing algorithms [18]. CG uses a new method called clan-based parsing [15, 16]. Clan-based graph decomposition is a parse of a directed acyclic graph (DAG) into a hierarchy of subgraphs. These new subgraphs generated by the decomposition are called clans [9, 10].

Let G be a DAG. A subset $X \subseteq G$ is a clan iff for all $x, y \in X$ and all $z \in G - X$, (a) z is an ancestor of x iff z is an ancestor of y, and (b) z is a descendant of x iff z is a descendant of y. A simple clan C, with more than three vertices, is classified as one of three types. It is (i) primitive if the only clans in C are the trivial clans; (ii) parallel if every subgraph of C is a clan; or (iii) series if for every pair of vertices x and y in C, x is an ancestor or descendant of y. Any graph can be constructed from these simple clans. Applying clan-based graph decomposition algorithms, any DAG can be decomposed into a tree of subgraphs (clans) whose leaves are trivial clans (graph nodes) and whose internal nodes are complex clans (series or parallel) built from their descendants [15, 16]. The primitive clans are decomposed into series and parallel clans by augmenting edges from all the source nodes of the primitive to the union of the children of the sources [15, 16]. After adding edges (2, 7), (2, 10), (3, 7), (3, 9), (3, 10), and (4, 5) into figure 1(a), sets {2, 3, 4}, {8, 9}, {7, 10}, {7, 8, 9, 10}, and {5, 6, 7, 8, 9, 10} are some of the nontrivial clans. Figure 1(b) is the parse tree of the figure 1(a) graph. After graph decomposition, the series clan are displayed vertically and connected by inter-clan edges, and the parallel clans are displayed horizontally and there are no edges connected between them.

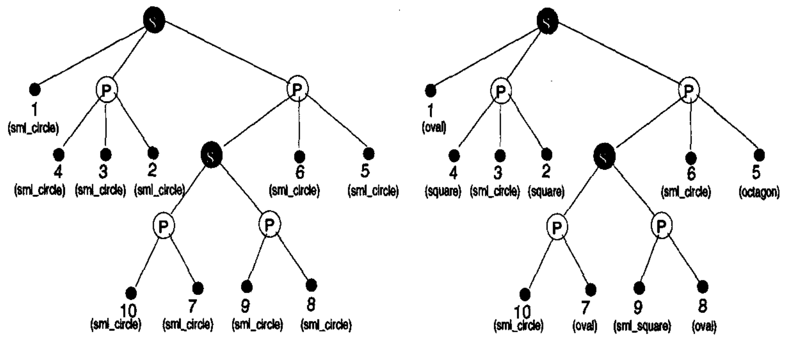


(a) graph (b) parse tree

Figure 1. Graph and Its Parse Tree

3. Node Layout

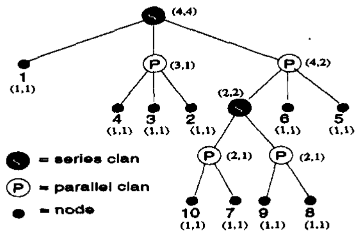
The parse tree of the graph is used to preserve node attributes (such as shape, and label) and to provide geometric interpretations to the graph. A node's shape and label are used to determine the size of its bounding box. The default shape for a node is a circle and default label is a number. The user can specify the shape and label for each node in the textual description input file. Figure 2 shows two parse trees with the node shape attribute specified for figure 1. In figure 2(a), the nodes shapes are from system default values. In figure 2(b), the shapes are user specified.



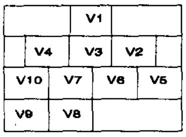
(a) node shape defined by system (b) node shape specified by the user

Figure 2. Parse Trees with Node Shape Attribute

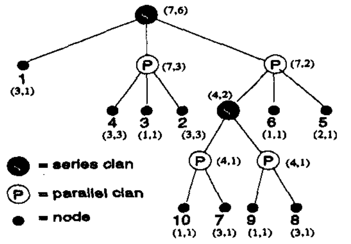
A "bounding box" with computed dimension is associated with each clan and the nodes in the clan are assigned locations within the bounding box. A bounding box is used to specify the allotted area for a subgraph. To calculate the bounding box for the parse tree, the bounding boxes of leaves are computed first. For a leaf node N, the bounding box length (N.l) and width (N.w) can be determined by N's shape, label size, application-specific settings, or user-specific settings. A series clan is bounded by a rectangle whose length is the sum of the lengths of the component clans and whose width is the maximum width of the component clans. An parallel clan is placed in an area whose width is the sum of the widths of the component clans and whose length is the maximum of the lengths of the component clans. After all the bounding boxes have been computed, CG uses the parse tree with the computed bounding boxes to map the graph onto coordinates in the planar window [15, 16]. Figure 3(a) and 3(c) show the parse trees with bounding boxes computed from figure 2's defined shapes. The bounding boxes of figure 3(a) are computed from figure 2(a)'s shape settings and figure 3(c) are from figure 2(b)'s. Figure 3(b) and 3(d) are node layouts for 3(a) and 3(c) respectively.



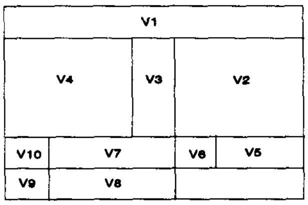
(a) parse tree associated with bounding box computed from figure 2(a)'s node shapes



(b) node layout of (a)



(c) parse tree associated with bounding box computed from figure 2(b)'s node shapes



(d) node layout of (c)

Figure 3. Parse Tree with Bounding Box and Node Layout

4. Edge Routing

Using the parse tree to place nodes is simple and elegant and provides for an aesthetically pleasing balanced placement. If adjacent nodes are connected by straight edges, several unacceptable visualizations may occur when the nodes are placed according to the location attributes.

- Edges could pass through nodes on their path.
- Edges might be superimposed upon other edges
- Unnecessarily long edges may be drawn.
- There may be an unnecessary number of edge crossings.

The first 3 problems listed above are caused by "long" edges, i.e. edges connecting nodes whose levels (y-values) differ by more than one. The traditional solution is to place dummy nodes at each intermediate level and route the long edge through the intermediate nodes [21]. One of the problems with this approach is that the long edges, by passing through nodes placed at arbitrary horizontal displacements, may contain unnecessary bends and may cross other edges unnecessarily. CG provides inter-clan and short-clan heuristics to solve the long edge problems [16].

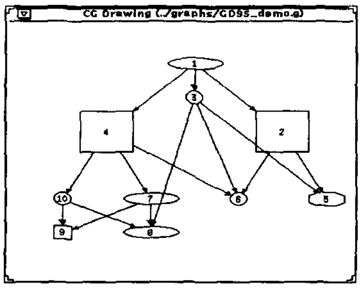
Inter-clan routes edges between nodes in different linear clans. For edge (x,y), let *lca* be the nearest common ancestor. By definition, *lca* must be a series node. Let P_x and P_y be the parallel children of *lca* that are parents of x and y, respectively. Inter-clan adds dummy nodes to the parse tree in three ways.

1. For all series clans in the traversal from x to *lca*, dummy nodes are added as children in each clan to the right of the ancestor of x.
2. For all parallel clans that are children of *lca* between P_x and P_y, a dummy node is added in the appropriate location.
3. For all series nodes in the traversal down the tree from *lca* to y, dummy children are added for each node to the left of y's ancestor.

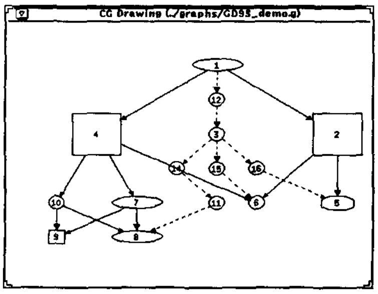
Short-clan is invoked when node or series clan C has bounding box height less than the bounding box height of its parent. Dummy nodes are added both at the top and bottom of the clan. For each clan source, dummy nodes are added for each in-coming edge, and for each clan sink, dummy nodes are added for each out-going edge. Figure 4 shows drawings before and after applying CG routing heuristics.

5. Reducing Edge Crossings

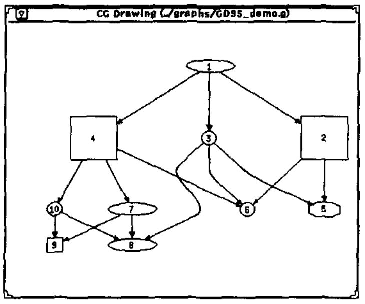
Minimizing the number of edge crossing is a NP-hard problem [7, 8]. Warfield [23] developed a heuristic method, barycentric ordering, for two-level graphs. A value called barycenter is computed for each of the vertices in the two levels. For each vertex, this value is a weighted average of the horizontal positions of the vertices in the adjacent level to which the vertex is connected. The barycenters of each of the vertices in a level are computed and the vertices are sorted according to their barycenters. Carpano [2], and Sugiyama, Tagawa, and Toda [20, 21] generalized the two-level



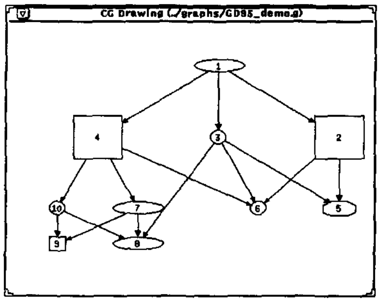
(a) original drawing (edge (3,5) & (3,8) pass node 2 & 7 respectively)



(b) dummy nodes 11 added by inter-clan heuristic, and 12, 15, & 16 added by short-clan heuristic



(c) new drawing with dummy nodes added

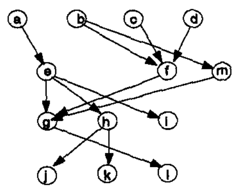


(d) final drawing with unnecessary bends removed

Figure 4. Drawings Before and After Applying Inter-clan and Short-clan Heuristics

barycentric method to reduce edge crossings for k-level hierarchical graphs. CG adopts the concept of barycentering to be used in conjunction with clans. A matrix is used to describe the connections of subgraphs (clans) instead of nodes. The value [i, j] of a matrix is defined as the number of the connecting edges between $clan_i$ and $clan_j$.

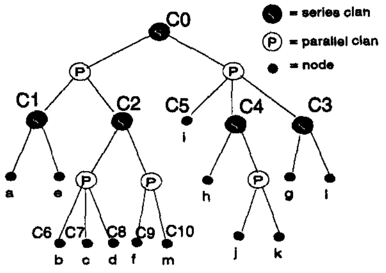
In CG, matrices are formed for the component clans of series clans not of parallel clans, because, by definition, there are no connections between parallel clans. There are fewer matrices used in CG and each is no larger than a matrix of individual nodes, because they are formed by subgraphs. As a result, CG's crossings reduction processing is faster than the original barycentric method. Figure 5 shows the matrix representation of the original barycentric technique and CG. After adding edges (c, m) and (d, m) into figure 5(a), the parse is produced as figure 5(c). According to the parse tree, only two matrices are required, M_1 for series clan C_0 and M_2 for C_2. There is no matrix needed for series clans C_1, C_5, C_4, and C_3, because each of them contains two components and at least one of the two components is a single node. In the matrix M_1, the value of $[C_2, C_3]$ is 2, because there are two edges between clan C_2 and C_3.



(a) graph

$$
\begin{array}{c|ccc}
 & e & f & m \\
\hline
a & 1 & 0 & 0 \\
b & 0 & 1 & 1 \\
c & 0 & 1 & 0 \\
d & 0 & 1 & 0
\end{array}
\qquad
\begin{array}{c|ccc}
 & g & h & i \\
\hline
e & 1 & 1 & 1 \\
f & 1 & 0 & 0 \\
m & 1 & 0 & 0
\end{array}
\qquad
\begin{array}{c|ccc}
 & j & k & l \\
\hline
g & 0 & 0 & 1 \\
h & 1 & 1 & 0 \\
i & 0 & 0 & 0
\end{array}
$$

(b) matrix representation of (a) by using barycentric ordering algorithm



(c) parse tree

$$
\begin{array}{c|ccc}
 & C3 & C4 & C5 \\
\hline
C1 & 1 & 1 & 1 \\
C2 & 2 & 0 & 0
\end{array}
\qquad
\begin{array}{c|cc}
 & C9 & C10 \\
\hline
C6 & 1 & 1 \\
C7 & 1 & 0 \\
C8 & 1 & 0
\end{array}
$$

(M1) (M2)

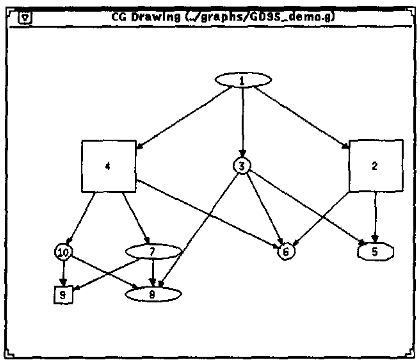
(d) matrix representation of (a) in CG

Figure 5. Graph and Matrix Representation

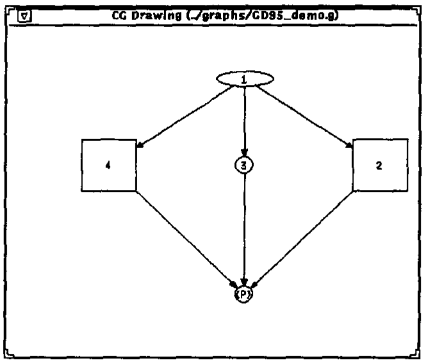
6. Subgraphs (Clans) Contract/Extract

Because the graph layout is based on a parse tree created by extracting subgraphs, it is possible to abstract those subgraphs and represent them by a single node, or just to display the selected subgraph. CG supports subgraph contraction, expansion, and extraction.

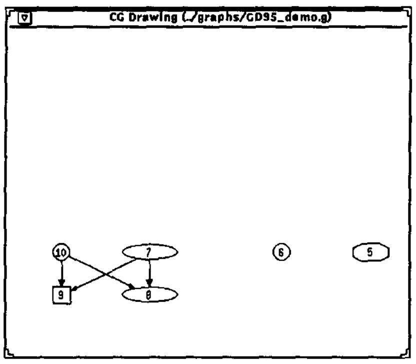
Since clans are defined as sets of nodes with identical ancestors and descendants within the rest of the graph, clans can easily be contracted to a single node. By selecting a single node, the user can contract the smallest non-trivial clan containing that node into a single node. Any node not in the clan that was connected to a clan source or sink will be connected to the contracted node. By allowing segments of the graph to be contacted, the user can simplify dense graphs for viewing by contracting those parts which are not relevant to the investigation. Contracted nodes can be expanded to show the original clan configuration. Similarly, CG can extract and display only the clan, ignoring the rest of the graph. In figure 6(b), the user contracts the clan containing node 6. In figure 6(c), the user displays the clan containing node 6. The parse tree of figure 6(a) is in figure 1(b).



(a) original drawing



(b) contract a subgraph



(c) display subgraph

Figure 6. Subgraph Contract/Extract

7. Cyclic Directed Graphs

The clan-based graph decomposition can be used to draw cyclic directed graphs by reversing certain edges [19]. A simple transformation is required to apply the graph decomposition method to cyclic graphs. Cycles can be found in a depth-first graph traversal. To break a cycle, the edge that identifies the cycle is given the reverse orientation. When the layout is ready, its orientation will be corrected. This method of breaking cycles will show a cycle not as a circular arrangement of nodes, but as a vertical line of nodes with an edge connecting the bottom to the top. This view is consistent with some applications such as the visualization of program control flow graphs. The general philosophy of a top to bottom flow for directed graphs is supported by this layout, with only few edges reversing that direction.

8. Example of Applications

One of application areas that are currently interested in CG is that of social networks. Figure 7. is an example of data reported in the 1940's by anthropologists [5]. They tallied the co-attendence of 18 women over a series of 14 small informal social events. Freeman and White began with their person by event matrix and constructed a Galois lattice that represents the person-person, the event-event and the person-event dependencies [11]. It shows that there are two pretty clear-cut sub-groups of women, and three kinds of events: those involving one group of women, those involving the other group, and those events that bridge the two. The 65 points in the graph picture the overall dependency structure. Events are labeled as A to N and women labeled as 1 to 18. Each point represents some collection of women and some collection of events. The uppermost point is the collection of all women and the null set of events. The lowermost point is the universal set of events and the null set of women. Each woman (or set of women) participated in those events labeled at or above her labeled point in the line diagram and each event (or set of events) included all the women labeled at or below its point.

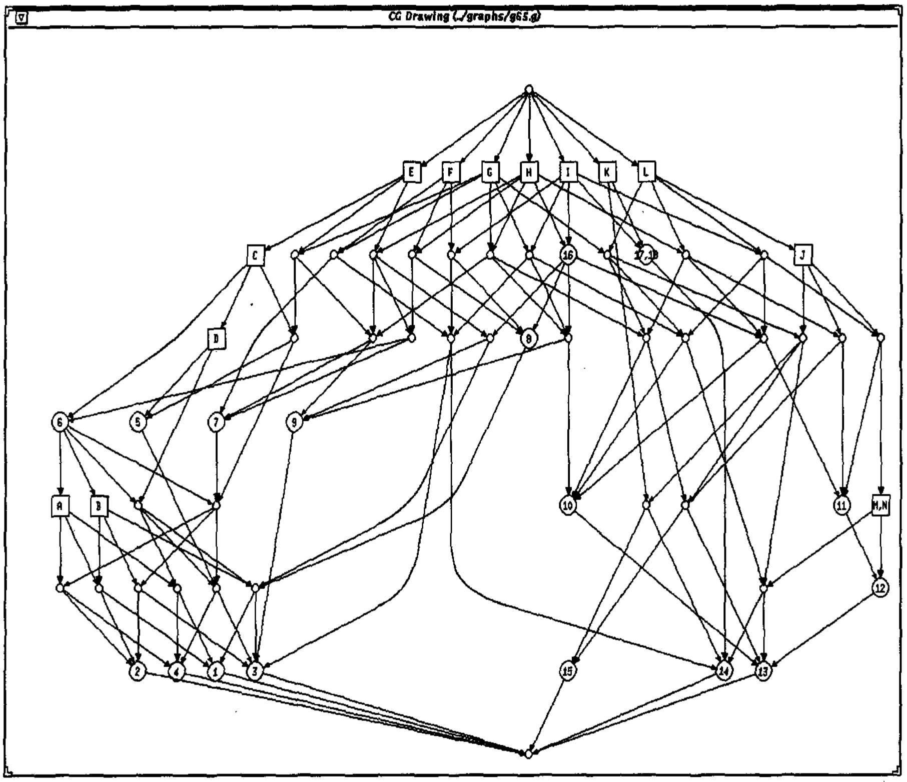


Figure 7. Social Networks Lattice

Acknowledgements

Thanks to Jeff Okerson for helping us to add the feature of saving the drawing to file in PostScript format.

References

1. K. Andrews, R. R. Henry, and W. K. Yamamoto, "Design and implementation of the UW illustrated compiler, " Univ. Washington, Dep. of Computer Science, Tech. Rep. 88-03-07, Mar. 1988.
2. M. J. Carpano, "Automatic Display of Hierarchized Graphs for Computer Aided Decision Analysis", IEEE Trans. on Systems Nab and Cybernetics, Vol. SMC-10, No. 11, 705-715, 1980.
3. J. H. Cross II and R. S. Dannelly, "Reverse Engineering Graphical Representations of X Source Code," will be appeared in International Journal of Software Engineering and Knowledge Engineering, Spring, 1996.
4. I. F. Cruz, and R. Tamassia, "How to Visualize a Graph: Specification and Algorithms, " Tutorial on Graph Drawing, available at http://www.cs.brown.edu/calendar/gd94, 1994.
5. A. Davis, B. Gardiner and M. Gardiner, 1941, Deep South, Chicago: U. of Chgo. Press
6. G. Di Battista, P. Eades, R. Tamassia, I. Tollis, "Algorithms for Drawing Graphs: an Annotated Bibliography", June, 1994, available via ftp from wilma.cs.brown (128.148.33.66) in file pub/papers/compgeo/gdbiblio.tex.Z.
7. P. Eades and D. Kelley, "Heuristics for drawing 2-layered graphs," Ars Combinatoria , Vol. 21-A, 89-98, 1986.
8. P. Eades and N. Wormald, "Edge Crossings in Drawings of Bipartite Graphs", Algorithmica, Vol. 11, No. 4, 379-403, April 1994.
9. A. Ehrenfeucht and G. Rozenberg, "Theory of 2-Structures, Part I: Clans, Basic Subclasses, and Morphisms," Theoretical Computer Science, Vol. 70, 277-303, 1990.
10. A. Ehrenfeucht and G. Rozenberg, "Theory of 2-Structures, Part II: Representation Through Labeled Tree Families," Theoretical Computer Science, Vol. 70, 305-342, 1990.
11. L. C. Freeman and D.R.White, 1993, "Using Gaiois Lattices to Represent Network Data," in P.V. Marsden, ed., Sociological Methodology 1993, Oxford: Blackwell, 127-146.
12. E. R. Gansner, S. C. North, and K. P. Vo, "DAG - A Program that Draws Directed Graphs," Software - Practice and Experience, Vol 18, No. 11, 1047-1062, Nov. 1988.
13. E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo, "A Technique for Drawing directed Graphs," IEEE Transactions on Software Engineering, Vol. 19, No. 3, 214-230, 1993.
14. Koutsofios and S. North, "Drawing Graphs with dot", Dot User's Manual, AT&T Bell Labs, Murray Hill, N. J., 1994

15. C. L. McCreary and A. Reed"A Graph Parsing Algorithm and Implementation," Tech. Rpt. TR-93-04, Dept. of Comp. Sci and Eng., Auburn U. 1993.
16. C. McCreary, F. S. Shieh, and H. Gill, "CG: a Graph Drawing System Using Graph-Grammar Parsing," Lecture Notes in Computer Science, Vol. 894, 270-273, Springer-Verlag, 1995.
17. E. B. Messinger, "Automatic Layout of Large Directed Graphs," Univ. of Washington, Ph.D. dissertation, 1988.
18. E. B. Messinger, L. A. Rowe, and R. R. Henry, "A Divide-and-Conquer Algorithm for the Automatic Layout of Large Directed Graphs," IEEE Trans. on Sys. Man, and Cyb., Vol. 21 No. 1,, 1-12, 1991.
19. L. A. Rowe, M. Davis, E. Messinger, and C. Meyer, "A Browse for Directed Graphs," Software-Practice and Experience, Vol. 17(1), 61-76, January 1987.
20. K. Sugiyama, S. Tagawa, and M. Toda, "Effective Representation of Hierarchies," in Proc. IEEE Int. Conf. Cybernetics and Society, New York, Oct. 1979.
21. K. Sugiyama, S. Tagawa and M. Toda, "Methods for Understanding of Hierarchical system structures," IEEE Trans. on Sys. Man, and Cyb., SMC-11, 109-125, 1981.
22. R. Tamassia, G. D. Battista, and C. Batini, "Automatic Graph Drawing and Readability of Diagrams," IEEE Trans. on Sys. Man, and Cyb., Vol. 18, No. 1, 61-79, 1988.
23. J. N. Warfield, "Crossing Theory and Hierarchy Mapping," IEEE Trans. on Syst., Man, and Cybernetics, Vol. 7, No. 7, 505-523, Jul. 1977.