CG: A Graph Drawing System Using Graph-Grammar Parsing

Carolyn McCreary Fwu-Shan Shieh Helen Gill*

Dept. of Computer Science and Engineering
Auburn University, AL
{mccreary, fwushan}@eng.auburn.edu

The MITRE Corporation
McLean, Virginia
gill@MITRE.org

1 Introduction

CG (Clan-based Graph Drawing Tool) is an automated system capable of converting a textual description of a digraph into a well organized and readable drawing of the digraph. The objectives of our system are to provide an aesthetically pleasing visual layout for arbitrary directed graphs. CG incorporates techniques to place nodes in a balanced way, both vertically and horizontally, and to present edges with few unnecessary crossings, few long arcs, and hardly any bends in the existing long arcs.

We developed CG for use in our work with program dependency graphs. Like *dot* [6], CG takes a textual description of an arbitrary directed graph and produces a visual representation of it. The layout places nodes from top to bottom insuring that an ancestor of a node is always above it. CG uses a unique graph parsing method to determine intrinsic substructures in the graph and to produce a parse tree. The tree is given attributes that specify the node layout. CG is the first digraph drawing tool to use a graph grammar as the fundamental structure that describes the node layout. Brandenburg [1] defines a layout graph grammar as a graph grammar together with a layout specification. The layout specification associates a finite set of layout constraints with each production. Our approach is to classify the productions of the graph parse and associate layout attributes with each production type in the parse tree of the graph.

2 Node Layout

The node layout is determined by the combination of (1) parsing of the graph into logically cohesive subgraphs and (2) defining layout attributes to apply to the resulting parse tree. The parse is based on a simple graph grammar, and the attributes that are now programmed into CG produce a layout whose nodes are balanced both vertically and horizontally.

2.1 Graph Decomposition

Clan-based graph decomposition is a parse of a directed acyclic graph (DAG) into a hierarchy of subgraphs. The subgraphs defined by our graph grammar and parse are called clans[3,4]. 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) independent if every subgraph of C is a clan; or (iii) linear 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.

* currently at NSF

An alternate characterization of a clan is a subgraph where: (1) all sources have the same parents, (2) all sinks have the same children, (3) nodes in the clan that are not clan sources have no parents outside the clan and (4) nodes that are not clan sinks haveno children outside the clan. The algorithm used to find clans identifies groups of nodes for which the above characterization holds. It first finds sets of siblings (nodes that have the same parents) and set of mates (nodes that have the same children.) A potential clan is the intersection of the siblings and their descendants with the mates and their ancestors. If properties 3 and 4 hold for the potential clan, it is identified as a clan.

In CG's graph layout algorithm, the parse tree is used to specify both the graph node locations and the edge routing. A bounding box attribute for each tree vertex, x, defines the length and width of a rectangle containing the graph nodes in x's subtree. The length and width are computed for each linear and each independent clan. The children of a linear clan are placed in bounding boxes which are stacked vertically. The children of an independent clan are placed in bounding boxes which are placed side by side horizontally.

Decomposing Primitives.

Primitive clans represent a special challenge. They do not fall into the clear-cut categories of vertices that should be laid out horizontally or laid out vertically. In addition, primitive clans can be arbitrarily large, and there can be indefinitely many of them. While this poses no special challenge to parsing, and every directed graph can be parsed into clans from the three clan types, graph layout decisions must be made for these clans. Our approach is to eliminate primitive clans by decomposing them further by relaxing properties (3) and (4) above which conceptually adds edges from the source nodes of a primitive to the union some of the children of the sources. The sources of a potential clan are grouped into sets which have at least one child in common. Temporary edges are added from the sources in the set to the union of the children. By definition, the source sets are independent clans. The remainder of the clan is parsed again and if it is primitive, the primitive decomposition heuristic is recursively called. This method extracts series-parallel constructs from primitive clans and represents the serial and parallel structures in the parse tree. It also pushes source nodes down the graph so they will be placed in levels adjacent to their children whenever possible. The parse tree of any completely decomposed graph is compressed into a bipartite tree where the internal vertices alternate between layers containing linear nodes and independent nodes.

2.2 Spatial Analysis

The parse tree of the graph can be given a variety of geometric interpretations. For example, a rectangle or bounding box with known width and length can be associated with each clan. Synthetic attributes can be associated with the parse tree hierarchy to show the embedding of the bounding boxes. For illustrative purposes CG uses attributes we call natural that give a balanced layout, both vertically and horizontally. Linear clans require an area whose length is the sum of the lengths of the component clans and whose width is the maximum width of the component clans. Independent clans require 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.

Definition: Denote the bounding box attribute of node N in parse tree T by (L.N,W.N). We define the natural values of the attribute to be: (1) (L.N, W.N) = (1,1), if N has no children; (2) (L.N, W.N) <-- (L.C1+...+L.Ck, Max(W.C1,...,W.Ck)), if N is a linear node with children C1...Ck; (3) (L.N, W.N) <--- (Max(L.C1...L.Ck),W.C1+...+W.Ck), if N is an independent node with children C1 ...Ck.

To achieve an aesthetically pleasing layout, the nodes are centered within the bounding boxes. For child C of linear node N, the actual rectangle in which the node is to be centered has length L.C and width W.N. For child D of independent node I, the rectangle in which D is to be centered has length L.I and width W.D.

3 Drawing Arcs

There are two major problems to be solved in routing arcs: reducing edge crossings and "long arc problems". Producing optimal edge crossings is NP-hard, even for nodes in only two levels [2]. CG currently does not address this issue, but future versions will include modifications of the Barycentric ordering technique of Warfield [8] to be applied to clans.

The long arc problems include reducing arc lengths whenever possible and the proper routing of arcs that connect nodes not placed on adjacent levels. A drawing tool must insure that long arcs do not intersect nodes at intermediate levels. Bends in the arcs are often necessary for this routing, but may cause visual confusion. Bends should be minimized. CG places false nodes and artificial clans in the parse tree to address these problems.

The subclans of a linear clan can have arcs between them. If the connected subclans are not adjacent children in the clan, the arcs will span more than one level. However, within independent clans, no two vertices may be connected by an arc. For these reasons, the algorithm needs to consider only routing arcs between vertices within a common linear clan. Because independent clans are not connected, subclans of independent nodes that insure the edges do not intersect intermediate nodes, can be created for edge routing.

Two heuristics modify the parse tree by adding false vertices through which the long edges are routed. Heuristic A routes long edges within a linear clan and heuristic B routes long edges between different subclans.

Heuristic A is invoked when node or linear 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 1 shows a graph with and without the dummy nodes generated by this heuristic.

Heuristic B 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 linear node. Let ix and iy be the independent children of lca that are parents of x and y, respectively. Heuristic B adds dummy nodes to the parse tree in three ways. 1. For all linear 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 independent clans that are children of lca between ix and iy, a dummy node is added in the appropriate location. 3. For all linear 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. Figure 2 illustrates this situation and figure 1 shows a graph where the dummy nodes were generated by this heuristic.

The addition of the dummy nodes by Heuristic B can cause unnecessary curves in the long edges. After the dummy nodes are identified and all nodes are placed, CG checks the long edges to see if there are any true nodes in the rectangular region bounded by the x and

y-coordinates of the edge ends. If there are none, a straight edge connects the nodes.

The CG system eliminates sharp corners by using B-splines. When an arc connects an actual vertex with a dummy vertex, the spline technique described by Farin [5] is used.

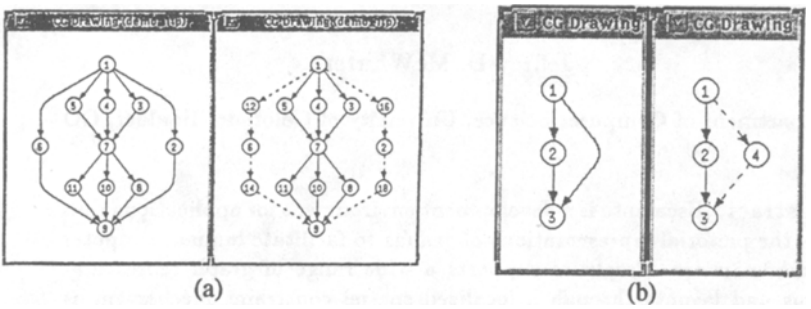
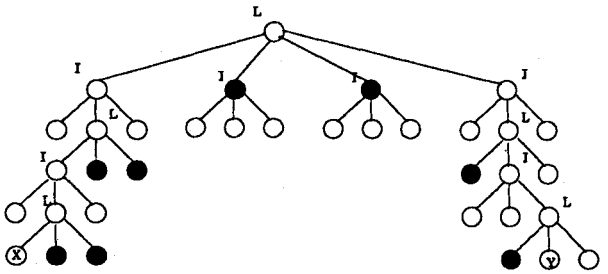


Fig. 1 Dummy Nodes Added by (a) Heuristic A, and (b) Heuristic B



● Dummy nodes will be added as children

Fig. 2. Parse Tree with Linear (L) and Independent (I) Nodes

4 References

1. F. J. Brandenburg, "Layout Graph Grammars: the Placement Approach", *Lecture Notes in Computer Science, 532, Graph Grammars and Their Application to Computer Science,* Springer-Verlag, Berlin, 1991..

2. P. Eades and N. Wormald, "Edge Crossings in Drawings of Bipartite Graphs", to appear in *Algorithmica.*

3. A. Ehrenfeucht and G. Rozenberg, "Theory of 2-Structures, Part I: Clans, Basic Subclasses, and Morphisms," *Theoretical Computer Science*, vol. 70, pp. 277-303, 1990.

4. A. Ehrenfeucht and G. Rozenberg, "Theory of 2-Structures, Part II: Representation Through Labeled Tree Families," *Theoretical Computer Science*, vol. 70, pp. 305-342, 1990.

5. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design.* Boston: Academic Press, 1988.

6. E. Koutsofios and S. North, "Drawing Graphs with *dot*", *Dot User's Manual,* AT&T Bell Labs, Murray Hill, N. J.

7. C. L. McCreary, C. L. Combs, and D. H. Gill, "Using Graph-Grammar Parsing for Automatic Graph Drawing", *Department of Computer Science and Engineering Tech Report, CSE 94-07, Auburn Univeristy* also submitted to *Journal of Visual Languages and Computing*..

8. J. N. Warfield, "Crossing Theory and Hierarchy Mapping," *IEEE Trans. on Syst., Man, and Cybernetics*, vol. 7, no. 7, pp. 505-523, Jul. 1977.