Graph Drawing

Lecture 15
Date: April 5, 1993
Scribe: George Wynne

1 Introduction

Before this lecture we have studied problems concerned with getting information out of a drawn graph. Now we want look at the opposite problem: given information about a graph, draw it. This type of problem has the following applications:

- software engineering (FIELD, e.g.)
- subroutine-call graphs
- data-flow diagrams
- entity-relationship diagrams
- project management: PERT diagrams
- knowledge representation
- circuit schematics

2 Types of Graphs

Polyline Drawing - edges can have a chain of line segments.

Figure 1: Polyline drawing
**Straight-line Drawing** - all edges are represented by a single straight line. This is the easiest type of graph to implement.

![Figure 2: Straight-line drawing](image)

**Orthogonal Drawing** - edges are represented only by horizontal and vertical lines. Therefore, the only angles are right angles, and the resolution is better than the resolution of straight-line drawings. However, when a vertex has more than four edges, some of the edges will overlap. One way to avoid this problem is to represent each vertex with a rectangle, so that distinct edges can leave the vertex in the same direction.

![Figure 3: Orthogonal drawing](image)

![Figure 4: Orthogonal drawing with rectangular vertices](image)
3 Resolution

We need several minimal standards for resolution, so that the graph can be drawn and seen. Such standards are:

- integer coordinates for vertices and bends
- given minimum distance between vertices
- given minimum distance between vertices and nonincident edges
- given minimum angle formed by consecutive incident edges

Theorem 1 The minimal angle that can be formed in an $n$ by $n$ grid

$$\alpha = \frac{1}{n(n-1)}$$

Proof:

$\alpha$ is the minimal angle.

$$\alpha = \beta - \gamma$$

$$\sin \beta = \frac{1}{n-1} \implies \beta \approx \frac{1}{n-1}$$

$$\sin \gamma = \frac{1}{n} \implies \gamma \approx \frac{1}{n}$$

$$\alpha \approx \frac{1}{n-1} - \frac{1}{n} = \frac{1}{n(n-1)}$$
**Theorem 2** If a regular graph has degree $d$ (all vertices have $d$ edges), then there exists an angle $\leq \frac{\pi}{d - 1}$

**Proof:**

Pick a point on the convex hull. It has one angle $\geq \pi$

Therefore, the other $d - 1$ angles $\leq \pi$

so the smallest angle $\leq \frac{\pi}{d - 1}$ \hspace{1cm} \Box

4  **Aesthetic Criteria**

We have many aesthetic criteria for making a given graph as clear as possible. These criteria include:

- minimizing the number of crossings
- minimizing the area
- minimizing the number of bends (in orthogonal drawings)
- maximizing the smallest angle
- maximizing the display of symmetries

We often cannot satisfy all of these criteria; we must make tradeoffs. Some of these criteria alone, such as minimizing crossings and minimizing area, are NP-hard problems. However, testing the planarity of a graph can be done in linear time, and minimizing the number of bends in an orthogonal drawing can be done in polynomial time.

We can use constraints - additional input to the drawing algorithm - when we have some knowledge about the semantics of the graph. For instance, these constraints can guide the algorithm to draw the vertices in a prescribed shape, or to keep some vertices close together.
5 Rooted Trees

Rooted trees are planar, upward (meaning that the parent is above the children), straight-line trees. Nodes of the same level are horizontally aligned. Applications of rooted trees include:
- display of data structures
- breakdown of hardware components
- organization charts
- table of contents

5.1 Rooted Trees: a Simple Drawing Algorithm

In this simple algorithm, for each node $v$, we just let $y(v) =$ the level (distance from the root) and $x(v) =$ the inorder rank. This algorithm gives a graph that is planar, straight-line, and upward, and this graph displays symmetries and isomorphic subtrees. However, the width of this tree is $n$. Also, the area of this tree is $O(n \log n)$ if the tree is balanced, and $O(n^2)$ if the tree is not balanced. Other drawing algorithms use space more efficiently.
5.2 Rooted Trees: a Recursive Drawing Algorithm

Reingold and Tilford designed a better, recursive algorithm for drawing rooted trees (1983). After drawing the left and right subtrees, place the drawings of the subtrees at horizontal distance 2. Place the root one level above and half-way between the children. However, if there is only one child, place the root at horizontal distance 1 from the child.

Like the simple drawing algorithm, this algorithm gives a planar, straight-line, upward graph, and this graph displays symmetries and isomorphic trees. However, this graph has the added advantage that it centers the parents over the children. While this algorithm can be implemented to run in $O(n)$ time, the efficient parallelization of this algorithm is still an open problem.
Furthermore, this graph has a “small” width - at least as small as the graph constructed by the simple drawing algorithm. But this width is not minimal. In the following example, the small, unshaded three-node subtree needs to moved to the left for the overall width to be minimized. In fact, it has been shown that minimizing the width is NP-hard if integer coordinates are required.

Figure 6: Drawing by the recursive algorithm

Figure 7: Minimum width drawing
5.3 Other Drawing Algorithms

Other drawing algorithms have been designed over the last several years. For example, we have:

- $O(n)$-area planar orthogonal grid drawings (non-upward) [Valiant 1979]
- $O(n\log n)$-area planar straight-line grid upward drawings of binary trees
  [Crescenzi Di Battista Piperno 1991]
- $O(n)$-area planar straight-line grid drawings for complete binary trees
  and Fibonacci trees [Crescenzi Di Battista Piperno 1991]
- $O(n\log\log n)$-area planar orthogonal grid upward drawings [Garg Tamassia 1991]

Figure 8: The tree drawn by Crescenzi, et al.