Constraint-Based Representations of Statistical Graphs

Allan R. Wilks

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

created: 18 August 1995

ABSTRACT

Pictor is an environment for statistical graphics that promotes simple commands for common uses, while also offering the ability to experiment with whole new paradigms. Its design comprises several levels, from its high-level graphics model to its low-level device model. Pictor describes graphs as graphical objects whose component pieces are related by several sorts of constraints. It is the purpose of this paper to describe in detail the constraint system that Pictor uses.

1. Introduction

Pictor is an environment for statistical graphics that promotes simple commands for common uses, while also offering the ability to experiment with whole new paradigms. Its design comprises several levels, from its high-level graphics model to its low-level device model. Pictor describes graphs as graphical objects whose component pieces are related by several sorts of constraints. It is the purpose of this paper to describe in detail the constraint system that Pictor uses.

In the next section we briefly review the ideas behind Pictor's graphics objects and then in the section following that, delve into the structure of them. Following this is a discussion of the various coordinate systems used by Pictor and how they are related. This prepares the groundwork for a detailed description of the various geometric constraints that may be specified when constructing a grob. There are a number of issues related to bounding boxes that are explained in section 6, and these lead to a further set of constraint types. With all of the constraint structure in place, section 7 shows how Pictor solves a linear programming problem to resolve all constraints in order to be able actually to plot a graphics object.

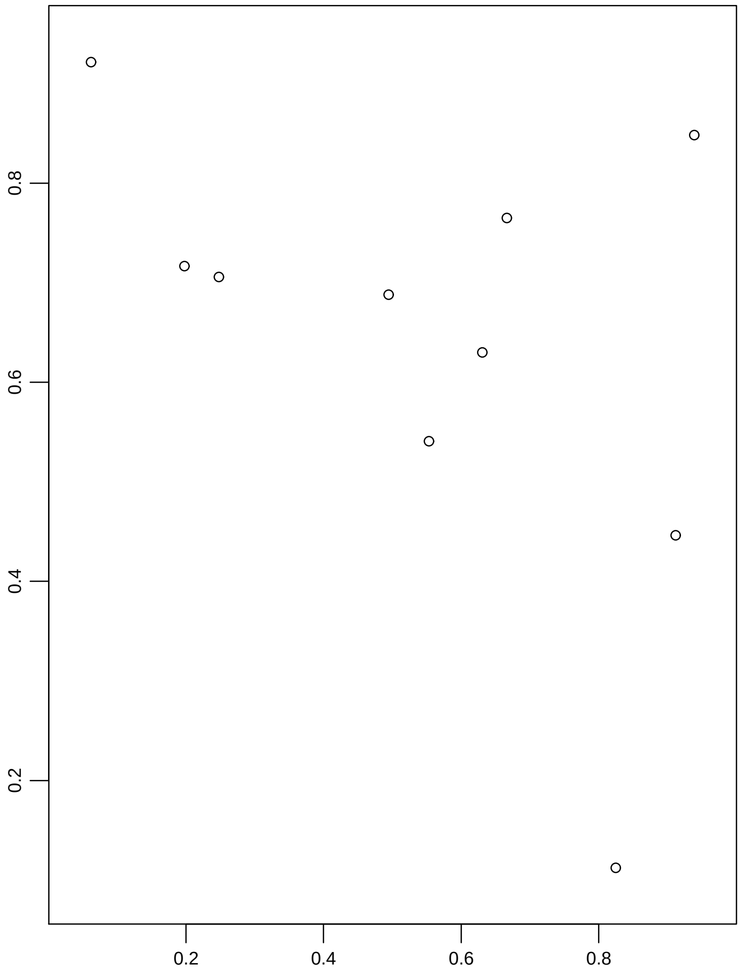
This paper assumes some familiarity with the basic low-level model that Pictor uses to describe pictures; for a detailed description of the model, see Wilks (1994a). Pictor has been implemented in Version 4 of S (Chambers, 1994 and Becker, et. al., 1988). A

*Lacaille, one of the first astronomers systematically to chart the southern skies, gave the name *Equuleus Pictoris*, meaning *Painter's Easel*, to the grouping of stars between declinations -43° and -64° and right ascensions $4^h 30^m$ and $6^h 50^m$. Today the name of this constellation has been shortened to simply *Pictor*, meaning *Painter* and can be found just north of the Large Magellanic Cloud.

programmer's guide for the implementation is available as Wilks (1994b).

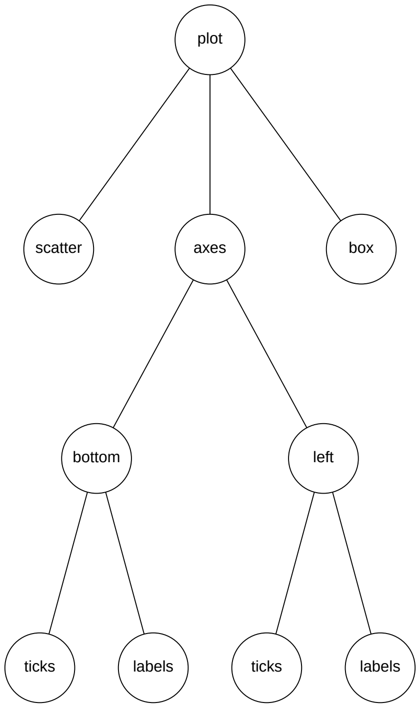
2. Graphics Objects

Pictor is a language for describing statistical graphs. For ease of exposition, we will refer to a statistical graph as a *picture*. A picture is represented in Pictor in a stylized form known as a *graphics object* or *grob* for short. To understand the structure of a grob, imaging trying to describe the following picture precisely to someone on the telephone:



You might begin by saying that there are four main things: a bunch of little circles; a box surrounding them; an *x*-axis; and a *y*-axis. Then you would be more specific about each of the four pieces. Concerning the little circles, you would tell where the center of each circle was, using the axes to get coordinates, and you would measure their size and report that in inches (say). The box is easy, as you could just give the coordinates of two corners in the same coordinate system as that of the circles.

To describe the *x*-axis, you might further break it down, saying that there are a set of ticks, and a set of labels for the ticks. You would say what the *x*-coordinates of the ticks are by reading off the tick labels and you would observe that they extend from the box downwards a certain distance, again in absolute units. The labels are centered at the ticks, a little below them. A similar description would apply to the *y*-axis, with the obvious modifications. This description may be represented schematically as follows:



Note that each part of the picture has been given a name, and that as we move from the top to the bottom of the tree, the picture is progressively broken into smaller and smaller pieces. The leaves of the tree correspond to Pictor's *drawing primitives*, which are fully described in Wilks (1994a). Broadly speaking, a drawing primitive is a collection of similar objects (line segments, arcs, rectangles, etc.) whose geometric information (endpoints, corners, etc.) is all assumed to be in the same, intrinsic coordinate system.

At several points in the imaginary telephone conversation, it was necessary to tie a piece of the picture to a previously described piece. For example, when describing the box, it was stated that the coordinates of the corners were in the same coordinate system used for the circles. Similarly, the horizontal locations of the ticks marks for the *x*-axis were stated to correspond to the horizontal locations of the circles, in the sense that a tick at a certain position on the *x*-axis and a circle at the same position in the scatter of circles will be on the same vertical line on the page.

The two main components of a Pictor grob, then, are the tree-like breakdown of the picture into successively more refined pieces, and the system of constraints that tie the pieces (more precisely, the coordinate systems of the pieces) together.

3. Grob Structure

The previous section gave a loose, qualitative overview of graphics objects. We now wish to be more precise and describe just how grobs are structured. First, some terminology. We will freely speak of grobs as trees, and this is to be understood in the sense of a hierarchical picture description as outlined above. Each leaf of the grob is a *drawing element* (describing a Pictor drawing primitive, as mentioned earlier), and the other tree nodes we will call *interior nodes.*

A drawing element contains geometric and possibly rendering information. Geometric information, like line segment endpoints or arc centers, consists of one or more coordinate pairs in a coordinate system that is intrinsic to the drawing element. Rendering information gives specific instructions for rendering the primitive, such as the width and color of lines and the font and pointsize to use for text. The structure of geometric and rendering information is described in complete detail in Wilks (1994a).

Interior nodes, by definition, have no drawing primitives associated with them. They can, however, contain rendering information. Any such information is meant to apply recursively to all children of that node, unless overridden by a specification lower in the tree. So, for example, a color specification at the root of the tree would apply to the entire grob, and this could be overridden for a particular subtree by giving a color specification at the root of that subtree.

Each node of a grob is given a name relative to its parent and its full name is just its relative name, together with the relative names of its ancestors in reverse order, separated by slashes. This is exactly analogous to the naming convention in a UNIX file system. The analogy extends to the naming of the root of the tree, which is always ‘‘/’’. (Notice that this does not correspond to the root name of ‘‘plot’’ used in the tree figure in section 2; the name used there was merely for illustrative purposes.)

The grob specifies a set of *geometric constraints*, each between a pair of its nodes; this is one of the places where the node names get used. A geometric constraint ties together the coordinate systems of two nodes in one of several ways; later we give a full description of how this is done. Note that a geometric constraint may involve an interior node, but that coordinate systems apparently only make sense for drawing elements. In a grob, the existence of a constraint involving an interior node carries with it the implication that all descendents of that node, and in particular the leaf descendents, share the same coordinate system. All other geometric constraints for the grob must be consistent with this assumption.

4. Coordinate Systems

Each leaf of a grob has geometric information (coordinate pairs) that exist in an coordinate system specific to that leaf. In addition, the display itself has a coordinate system, which we shall call the *display coordinate system* or the *absolute coordinate system*. The latter term comes from the fact that the display coordinate system is in absolute units. More precisely, its origin is in the lower left corner of the display (always assumed rectangular), x increases to the right, y increases up and one unit in each direction is one inch. (But for the substitution of ‘‘inch’’ for ‘‘point’’, this is precisely the default PostScript coordinate system; see Adobe, 1985).

In order to render a grob, it is necessary to relate each drawing element’s coordinate system to that of the display, that coordinate pairs in the leaf's intrinsic coordinate system may be given meaning as absolute coordinate pairs. This mapping from intrinsic coordinates to absolute coordinates is the *coordinate transformation* for the drawing element. Note that the coordinate transformation is unknown at the time the grob is created; the grob just contains geometric constraints, each of which gives information about one or more of the coordinate transformations. But the transformations themselves are not discovered until the grob is actually rendered.

Pictor allows only certain kinds of coordinate transformations. First, they must be affine. That is, if (x,y) is a coordinate pair in the intrinsic coordinate system of a leaf and (x',y') is the corresponding coordinate pair in absolute coordinates, then these pairs must be related as follows:

$$x' = ax + by + \tau_x$$

$$y' = cx + dy + \tau_y$$

for some fixed, but initially unknown numbers a,b,c,d,τ_x,τ_y. There are a set of these numbers for each drawing element and the coordinate system constraints that we shall soon describe each imply one or more relationships among the numbers. The process of finding all the unknown coordinate transformations will amount to solving an optimization problem.

This problem turns out to be quite difficult to solve in general and so we are led to make some simplifying assumptions, which will allow the problem to be formulated in a linear programming setting. The main reason that the problem is not linear is that a pair of arbitrary, affinely-related coordinate systems are generically rotated with respect to each other and this leads to non-linear constraints. So the first assumption we will make is that the amount by which the leaf's coordinate system is rotated with respect to the display coordinate system may be determined independently of the other unknown parameters in the problem. What this means will become clear shortly when we precisely describe the allowable constraints; it will be seen then that the constraints are designed in such a way as to allow all rotation constraints to be solved first.

Actually, this assumption is somewhat imprecise as the concept of the rotation difference in two coordinate systems is not even well-defined. In the equations above the basis vectors $(1,0)^t$ and $(0,1)^t$ are each rotated through different amounts. This leads us to insist that those two rotations be the same. Now, since $(1,0)^t$ maps to $(a,c)^t$ and $(0,1)^t$ maps to $(b,c)^t$, those two vectors must be orthogonal (since their antecedents were), which gives the *nonshear condition*: $ab + cd = 0$. Under this condition, it is then possible to determine a unique θ (the angle through which all vectors are rotated) such that the transformation equations may be rewritten as:

$$x' = \sigma_x \rho_x x \cos\theta - \sigma_y \rho_y y \sin\theta + \tau_x \qquad (*)$$

$$y' = \sigma_x \rho_x x \sin\theta + \sigma_y \rho_y y \cos\theta + \tau_y$$

Here, σ_x and σ_y are unknown, positive scale factors and ρ_x and ρ_y are either 1 or -1, to allow reflections. (Note that since these equations remain unchanged if (θ,ρ_x,ρ_y) is replaced by $(\theta+\pi,-\rho_x,-\rho_y)$, θ should be restricted to an interval of length π to be truly unique; in practice this ambiguity is irrelevant).

The final assumption we will make is that, like the coordinate system rotations, the reflections (encapsulated in ρ_x and ρ_y) are pre-determinate, independent of the scale (σ) and location (τ) parameters.

In summary, then, each drawing element's coordinate system is assumed to be related to the display coordinate system through a set of affine equations like (*) above and it is further assumed that the parameters θ, ρ_x and ρ_y may all be computed in advance and independently of the remaining parameters.

5. Coordinate system constraints

Pictor allows the coordinate systems of the drawing elements of a grob to be linked together through the set of geometric constraints given in Table 1. In the table, a constraint specification begins with the name of the constraint type (exact, scale, etc.). The constraint is between two nodes of the grob, or between a node and the display. The next part of the specification names the *from-node* and the *to-node* of the constraint; the display's name is just the null string. For ease of exposition, the nodes have been numbered from 1 to n in Table 1, and the display has been given number 0. This way, the from-node and to-node can be written as the numbers i and j. Finally, the specification may require additional numeric parameters; when necessary, these are given as α, β, and so on.

Following each constraint prototype in Table 1 are one or more equations that translate the constraint into statements about the unknown parameters of the indicated coordinate transformations. The goal is to find a set of values for the unknown parameters such that all of these equations (and others to be described later) are simultaneously satisfied. The notation in the equations is the same as that in (*) from the last section, but superscripts have been added to denote which coordinate system the parameters belong to (so the superscripts are *not* powers). The values for the display coordinate system are fixed and known:

$$\theta^0 = 0; \quad \rho_x^0 = \rho_y^0 = 1; \quad \sigma_x^0 = \sigma_y^0 = 1; \quad \tau_x^0 = \tau_y^0 = 0.$$

5.1. exact, scale, match

When the coordinate systems of two nodes are identical, this is expressed through an exact constraint. The equations for the exact constraint in Table 1 just say that each parameter associated with coordinate system i is equal to the corresponding parameter for coordinate system j. The exact constraint plays a special role in constraint resolution, in that when i signifies an interior node of the grob for some constraint C, Pictor replaces i with the first leaf descendent i' of i and then automatically adds exact constraints between i' and each other leaf descendent of i. This is done for j as well, and finally, an exact constraint is added between i' and j'. The net result is that in the end, constraints are only between leaves or between a leaf and the display. This process takes place at the time the grob is rendered, not when it is created, so that it is the original C that is stored in the grob.

The scale constraint allows the specification of a general scaling between two coordinate systems. It means that a horizontal run of α and a vertical rise of β in coordinate system i corresponds to a run and rise of γ and δ in coordinate system j. Note that this only makes sense in the context of the nonshear condition.

Similarly, match establishes the coordinates of a single position in the two coordinate systems; the point (α,β) in coordinate system i is the same spot on the display as the point (γ,δ) in coordinate system j.

`exact` $i\ j$

$$\theta^i = \theta^j;\ \ \sigma^i_x = \sigma^j_x;\ \ \sigma^i_y = \sigma^j_y;\ \ \tau^i_x = \tau^j_x;\ \ \tau^i_y = \tau^j_y;\ \ \rho^i_x = \rho^j_x;\ \ \rho^i_y = \rho^j_y$$

`scale` $i\ j\ \alpha\ \beta\ \gamma\ \delta$

$$\sigma^i_x\rho^i_x\cos\theta^i\alpha - \sigma^i_y\rho^i_y\sin\theta^i\beta = \sigma^j_x\rho^j_x\cos\theta^j\gamma - \sigma^j_y\rho^j_y\sin\theta^j\delta$$
$$\sigma^i_x\rho^i_x\sin\theta^i\alpha + \sigma^i_y\rho^i_y\cos\theta^i\beta = \sigma^j_x\rho^j_x\sin\theta^j\gamma + \sigma^j_y\rho^j_y\cos\theta^j\delta$$

`match` $i\ j\ \alpha\ \beta\ \gamma\ \delta$

$$\sigma^i_x\rho^i_x\cos\theta^i\alpha - \sigma^i_y\rho^i_y\sin\theta^i\beta + \tau^i_x = \sigma^j_x\rho^j_x\cos\theta^j\gamma - \sigma^j_y\rho^j_y\sin\theta^j\delta + \tau^j_x$$
$$\sigma^i_x\rho^i_x\sin\theta^i\alpha + \sigma^i_y\rho^i_y\cos\theta^i\beta + \tau^i_y = \sigma^j_x\rho^j_x\sin\theta^j\gamma + \sigma^j_y\rho^j_y\cos\theta^j\delta + \tau^j_y$$

`xexact` $i\ j$

$$\theta^i = \theta^j;\ \ \sigma^i_x = \sigma^j_x;\ \ \tau^i_x = \tau^j_x;\ \ \rho^i_x = \rho^j_x$$

`xscale` $i\ j\ \alpha\ \beta$

$$\theta^i = \theta^j;\ \ \sigma^i_x|\alpha| = \sigma^j_x|\beta|;\ \ \rho^i_x\,\mathrm{sgn}\,\alpha = \rho^j_x\,\mathrm{sgn}\,\beta$$

`xmatch` $i\ j\ \alpha\ \beta$

$$\theta^i = \theta^j;\ \ \sigma^i_x\rho^i_x\alpha + \tau^i_x\cos\theta^i + \tau^i_y\sin\theta^i = \sigma^j_x\rho^j_x\beta + \tau^j_x\cos\theta^j + \tau^j_y\sin\theta^j$$

`yexact` $i\ j$

$$\theta^i = \theta^j;\ \ \sigma^i_y = \sigma^j_y;\ \ \tau^i_y = \tau^j_y;\ \ \rho^i_y = \rho^j_y$$

`yscale` $i\ j\ \alpha\ \beta$

$$\theta^i = \theta^j;\ \ \sigma^i_y|\alpha| = \sigma^j_y|\beta|;\ \ \rho^i_y\,\mathrm{sgn}\,\alpha = \rho^j_y\,\mathrm{sgn}\,\beta$$

`ymatch` $i\ j\ \alpha\ \beta$

$$\theta^i = \theta^j;\ \ \sigma^i_y\rho^i_y\alpha - \tau^i_x\sin\theta^i + \tau^i_y\cos\theta^i = \sigma^j_y\rho^j_y\beta - \tau^j_x\sin\theta^j + \tau^j_y\cos\theta^j$$

`xyexact` $i\ j$

$$\theta^i - \pi/2 = \theta^j;\ \ \sigma^i_x = \sigma^j_y;\ \ \tau^i_x = \tau^j_y;\ \ \rho^i_x = \rho^j_y$$

`xyscale` $i\ j\ \alpha\ \beta$

$$\theta^i - \pi/2 = \theta^j;\ \ \sigma^i_x|\alpha| = \sigma^j_y|\beta|;\ \ \rho^i_x\,\mathrm{sgn}\,\alpha = \rho^j_y\,\mathrm{sgn}\,\beta$$

`xymatch` $i\ j\ \alpha\ \beta$

$$\theta^i - \pi/2 = \theta^j;\ \ \sigma^i_x\rho^i_x\alpha + \tau^i_x\cos\theta^i + \tau^i_y\sin\theta^i = \sigma^j_y\rho^j_y\beta - \tau^j_x\cos\theta^j + \tau^j_y\sin\theta^j$$

`yxexact` $i\ j$

$$\theta^i + \pi/2 = \theta^j;\ \ \sigma^i_y = \sigma^j_x;\ \ \tau^i_y = \tau^j_x;\ \ \rho^i_y = \rho^j_x$$

`yxscale` $i\ j\ \alpha\ \beta$

$$\theta^i + \pi/2 = \theta^j;\ \ \sigma^i_y|\alpha| = \sigma^j_x|\beta|;\ \ \rho^i_y\,\mathrm{sgn}\,\alpha = \rho^j_x\,\mathrm{sgn}\,\beta$$

`yxmatch` $i\ j\ \alpha\ \beta$

$$\theta^i + \pi/2 = \theta^j;\ \ \sigma^i_x\rho^i_x\alpha - \tau^i_x\sin\theta^i + \tau^i_y\cos\theta^i = \sigma^j_x\rho^j_x\beta + \tau^j_x\cos\theta^j + \tau^j_y\sin\theta^j$$

`aspect` $i\ j\ \alpha\ \beta$

$$\sigma^i_x\rho^i_x\cos\theta^i\beta - \sigma^i_y\rho^i_y\sin\theta^i\alpha\beta = \sigma^i_x\rho^i_x\sin\theta^i + \sigma^i_y\rho^i_y\cos\theta^i\alpha, \quad j = 0 \text{ only}$$

`rotate` $i\ j\ \alpha$

$$\theta^i + \alpha = \theta^j \quad (\mathrm{mod}\,\pi)$$

Table 1: Pictor's geometric constraints. Each constraint type is followed by the constraint equations.

5.2. xexact, xscale, xmatch

These three constraints are analogous to the first three, but they only apply to the horizontal direction in the two coordinate systems. All three imply that the x-axis in the two systems are parallel, and because of the non-shear condition, this can be expressed simply by saying that the systems are rotated by the same amount from the horizontal of the display ($\theta^i = \theta^j$).

The xexact constraint says that not only are the two x-axes parallel; in fact, they are exactly coincident—same origin, same unit size, same direction of positive x. A weaker condition is given by the xscale constraint, which only demands that α units in the first system is β units in the second; an interval of length α on the first x-axis will have the same physical size on the display as an interval of length β on the second. Notice that the equations are phrased in such a way that the positivity of the σs is preserved, while still honoring a possible difference in sign between α and β.

The third constraint matches the points α and β on the two x-axes. Since they are parallel, a line perpendicular to one is perpendicular to both; if such a line passes through the point $(\alpha, 0)$ on the first axis, it is constrained to pass through $(\beta, 0)$ on the second. If (α, y^i) is an arbitrary point on the line in system-i coordinates then the constraint says that we may find a y^j such that (β, y^j) are the point's coordinates in system j. Hence, letting θ be the common rotation parameter,

$$\sigma_x^i \rho_x^i \alpha \cos\theta - \sigma_y^i \rho_y^i y^i \sin\theta + \tau_x^i = \sigma_x^j \rho_x^j \beta \cos\theta - \sigma_y^j \rho_y^j y^j \sin\theta + \tau_x^j,$$
$$\sigma_x^i \rho_x^i \alpha \sin\theta + \sigma_y^i \rho_y^i y^i \cos\theta + \tau_y^i = \sigma_x^j \rho_x^j \beta \sin\theta + \sigma_y^j \rho_y^j y^j \cos\theta + \tau_y^j.$$

Adding $\cos\theta$ times the first equation and $\sin\theta$ times the second yields the relation given in the table. Additionally, the first equation, together with the Table condition, implies the existence of a y^j making the second equation true. This justifies the Table condition.

5.3. yexact, yscale, ymatch

These three conditions are exactly analogous to those just described, except that they operate on the y-axis of the two coordinate systems, instead of the x-axis.

5.4. xyexact, xyscale, xymatch

If the x-axis of coordinate system i is parallel to the y-axis of coordinate system j then the coordinate systems are rotated $\pi/2$ with respect to each other. These three constraints are designed to tie the two axes together. We make the convention that the i-system is rotated $\pi/2$ *more* than the j-system; hence the condition in the table that $\theta^i - \pi/2 = \theta^j$. The equation for the xymatch constraint is justified in much the same way as that for the match and ymatch equations.

5.5. yxexact, yxscale, yxmatch

These three constraint types are analogous to the previous three, but now the convention is that the i-system is rotated $\pi/2$ *less* than the j-system.

5.6. aspect

One of the most important and often most neglected controls in statistical graphics is the aspect ratio. It is surprising how much information can be revealed (or suppressed!) by modifying it. See Cleveland (1994) for some stunning examples of this fact.

The aspect ratio is the ratio of the height to the width of a certain box or, equivalently, the slope of a certain line segment (the diagonal of the box beginning at the lower left corner) in a picture. There are two ways to express it, depending on whether the endpoints of the line segment are given in intrinsic or absolute coordinates. In the first case, we would want the line segment to have a certain absolute slope and in the second case we would want the (absolute) line segment to have a certain slope in intrinsic coordinates. The line segment might be the diagonal of a bounding box for the data or for a square box, or something more complicated like an average of several line segments that will appear in the graph.

The aspect constraint says that a line segment of slope α in coordinate system i and a line segment of slope β in coordinate system j must have the same actual slope (in absolute coordinates) when drawn. Unfortunately, this condition is too general, in the sense that it leads to equations that are nonlinear in the unknown parameters. Hence we restrict j to be 0; i.e., β must be a slope in absolute coordinates. This is by far the most useful case anyway; little is lost by making the restriction. The equation given in the table follows easily after a little algebra.

5.7. rotate

The final constraint type is very simple. It states that coordinate system i must be rotated by an angle α in order to have axes parallel to those of coordinate system j. Again note that the nonshear condition is needed in order for this to make sense.

6. Bounding Boxes

We have seen that the coordinate systems of the leaves of a grob are tied together via a set of coordinate system constraints. There are two reasons that more is needed. First, the constraints are usually not enough to completely specify where every piece is go on the page; at the very least, an overall scaling is usually missing. Second, these geometric constraints are sometimes not what is really wanted; often one wants merely to specify proximity in terms of bounding boxes. Both of these topics require a discussion of bounding boxes, to which we now turn.

Each leaf has two bounding boxes associated with it: its *geometric bounding box* and its *rendering bounding box*. Recall that a drawing primitive describes a geometric object, and contains, in addition, a set of rendering information used to draw the object. The geometric bounding box is the smallest rectangle enclosing the geometric object, while the rendering bounding box is the smallest rectangle enclosing the rendered object; it includes all the *ink* around the geometric points.

These are invariably different, though often not much so. Consider, for example, a drawing primitive consisting of a single line segment. Then the two endpoints of the segment are opposite corners of the geometric bounding box, but the rendering bounding box is a little bigger, to account for the thickness of the line with which the segment is rendered.

The geometric bounding box is expressed in intrinsic coordinates and can be computed simply and directly from the geometric data of the drawing primitive. The rendering bounding box is expressed in absolute coordinates, on the other hand, and therefore cannot be known in advance since the coordinate transformation for the primitive is initially unknown. As we shall see, however, the rendering bounding box is required for computing the coordinate transformation. This seeming circularity may be resolved as follows.

6.1. Bounding box inequalities

Consider as an example, the problem of determining the rendering bounding box of a text primitive consisting of n text strings at locations (x_i, y_i), $i = 1, \ldots, n$. We ask which of the text strings will determine the left edge of the rendering bounding box, that is, which of the text strings, when rendered, will have ink closest to the left edge of the display? This depends both on the size of the rendered text (which in turn depends on the pointsize and the font) and on the coordinate transformation; if the rendered text strings are of different lengths then different coordinate transformations can bring different strings to the left edge.

Clearly we cannot say *a priori* which string will determine the left edge. What we can say, once we know the θ and the ρ_x and ρ_y of the coordinate transformation, is that for string i, the ink extends from (x_i, y_i) (intrinsic coordinates) w_i (absolute) units to the left, where w_i depends on the pointsize, font, textrotation and textadjust rendering parameters. Suppose that all ink must be to the right of $X_{\min}$ (absolute). $X_{\min}$ could be 0, corresponding to the left edge of the display, but we might also like to allow for a left margin. We then have, for string i, the *inequality* constraint (superscripts have been dropped for clarity):

$$\sigma_x x_i \cos\theta - \sigma_y y_i \sin\theta + \tau_x \geq X_{\min} + w_i$$

This is an inequality in the unknowns σ_x, σ_y and τ_x and there is one of them for each of the n strings. There is an additional set of inequalities relative to the right edge of the page, using $X_{\max}$ and a different set of w_i. Similarly, there are two more sets of equations for the top and bottom of the page. In all, this gives $4n$ inequalities that must all be simultaneously satisfied.

Each kind of drawing element gives rise to a set of inequalities like those for text strings. The important thing to note is that the data for all these inequalities—the (x_i, y_i, w_i) triples—implicitly define the rendering bounding box, but that without actually computing the coordinate transformations, the box cannot be known. Hence, we store the rendering bounding box as four sets of triples, one for each side of the display. The way in which the triples themselves are computed varies between kinds of drawing elements, but the general idea is the same in each case.

6.2. Simplifying the computation

The number of inequalities that arise from the above considerations is potentially quite large. Consider, for example, a leaf that consists of the marks on a scatterplot, and suppose that there are 1000 points being plotted. The rendering bounding box for this leaf consists of 4000 triples (12000 numbers) and gives rise to 4000 linear inequalities. This will clearly strain even the best of linear programming solvers. Fortunately, there is a way out of this problem.

Consider again the triples (x_i, y_i, w_i), $i=1,\ldots,n$ in the earlier example of determining the leftmost edge of the ink in n text strings. Let $f_i = x_i \rho_x \cos\theta$, $g_i = -y_i \rho_y \sin\theta$ and $h_i = -w_i$. Observe that f_i, g_i and h_i are all computable before the coordinate transformations have been computed. Then for given values of σ_x, σ_y and τ_x, the left edge of the rendering bounding box will be determined by the i that minimizes $\sigma_x f_i + \sigma_y g_i + h_i + \tau_x$. This means that the plane with normal $(\sigma_x, \sigma_y, 1)$ passing through (f_i, g_i, h_i) has all the (f, g, h) triples on or to one side of it, which implies that (f_i, g_i, h_i) is an extreme point (on the boundary of the convex hull) of the (f, g, h) triples. Since σ_x and σ_y are constrained to be positive, it is a special kind of extreme point, what we will call a *least extreme point*: a vertex of a *least face* of the boundary of the convex hull of the (f, g, h) triples, i.e., a face whose outward pointing normal has all nonpositive components.

The upshot of this is that any set of (x_i, y_i, w_i) triples may be reduced to a smaller, equivalent set by selecting only those points for which the corresponding (f_i, g_i, h_i) triple is a least extreme point of all the (f, g, h) triples. It will usually be the case that the reduction is quite dramatic. A common example is when $\theta = 0$ and all the w_i are equal; then the original set of triples will reduce to just one triple.

6.3. Rendering constraints

We have seen how to generate a set of linear equalities and inequalities starting from the geometric constraints of Table 1. As mentioned at the beginning of this section, the Table 1 constraints are often not sufficient (or convenient) to say how pieces of the grob should be tied together. Hence we introduce the notion of *rendering constraints*. Whereas a geometric constraint ties together the geometry of two pieces of a grob, a rendering constraint ties together their rendering bounding boxes.

Rendering constraints are expressed in much the same way as geometric constraints; the complete list of types is:

bexact	bmatch	bscale
bxexact	bxmatch	bxscale
byexact	bymatch	byscale

As can be seen, these are just a subset of the geometric constraint types with a b prepended. The syntax is the same as for geometric constraints, with the crucial difference that where coordinates need to be specified, they are given relative to the bounding box, with $(0,0)$ at the lower left corner of the box and $(1,1)$ at the upper right.

Here are some examples. To assure that the right edge of the rendering bounding box in coordinate system *i* and the left edge in coordinate system *j* coincide, use:

bxmatch *i j* 1 0

To do the same thing, but with a small, scalable gap between the boxes:

bxmatch *i j* 1.05 0

This says that the left edge of *j* is a distance to the right of the right edge of *i* equal to 5% of the width of rendering bounding box *i*. If, in addition to lining up the right and left edges of *i* and *j*, we want their tops and bottoms to match:

bxmatch *i j* 1 0
byexact *i j*

Perhaps the material in coordinate system *j* is a legend for the material in *i* and we wish to tie the upper left corners of the two coordinate systems together:

bxmatch *i j* 1 1 1 1

Or, to move the legend a little inside the bounding box of *j*:

bxmatch *i j* 1 1 1.05 1.05

7. Finding coordinate transformations

Geometric and rendering constraints provide a natural way to tie together the various pieces of a grob. When it comes to the point of rendering the grob, however, it is necessary to compute for each leaf of the grob the coordinate transformation that takes coordinates in the leaf to display coordinates. This is done in several steps.

First, a constraint involving an interior node of a grob is propagated to its leaf descendents. This implies that all leaf descendents of the node will have the same coordinate system. In order to reduce the ultimate number of constraints generated in this way, the constraint is rewritten to only involve the first leaf descendent, and then the rest of the leaf descendents are noted as having the same coordinate transformation as the first, whatever that turns out to be; as such, the remaining leaves will not enter into the computation. (In order to make this work, it is also necessary to transfer any other constraints involving the temporarily-ignored leaves to the first leaf, as well). The end product of this step of the computation is that all constraints now only go between two leaves or between a leaf and the display.

The second step is to compute θ, ρ_x and ρ_y for each of the leaves remaining in the problem. By default these are 0, 1 and 1, but constraints may change them, as per the Table. Sometimes a constraint supplies a fixed value for θ (or ρ_x or ρ_y) and other times it is a relative value between two coordinate systems. The computation proceeds by passing through all the constraints, setting absolute values when possible, and setting relative values when one of the two values is known. This process is repeated until all values of θ are known. It is easy to determine during this step if the constraints are overdetermined or underdetermined with respect to θ, ρ_x or ρ_y.

Third, an LP problem is solved. The unknown variables are the σ_x, σ_y, τ_x and τ_y. Linear equalities arise from the geometric bounding box constraints, exactly as given in the Table. Rendering bounding box constraints also give rise to linear equalities, but the situation is slightly more complex; for a particular such constraint, an equality must be generated for each pair of triples, one from one of the two leaves and one from the other. Finally, a set of linear inequalities is generated from the restriction that each rendering bounding box must appear on the display: its left edge must be to the right of the left edge of the display, and so on. The cost function for the LP problem is simply the sum of the (absolute) widths and heights of the rendering bounding boxes; the idea is to make the pieces of the picture as large as possible on the display while still satisfying all the constraints.

The output of the LP problem is a set of coordinate transformations. These are added to the appropriate leaves of the grob and the remaining leaves get their imputed values according to the reduction in the first step. The grob may now be rendered by simply applying the computed coordinate transformation for a leaf to each coordinate pair appearing in the leaf.

8. Conclusion

Pictor's constraint system arises from a single design decision—the desire not to have to pre-specify regions of the display where various pieces of the graph will go. Conventional systems for statistical graphics will, for example, set aside one area of the display for the points and another for axes, labels and titles. But if no axes, labels and titles are requested, this leaves large amounts of unused space around the edge of the plot. By having plotting functions generate picture descriptions in terms of high-level constraints, region reservation is effectively deferred to the time of plotting, resulting in better use of limited space, and consequently, better data analysis.

A second, important consequence of constraint-based representation of statistical graphs is that since it more closely matches the way we *think* about a graph, it is therefore easier to construct new kinds of graphs, leading to a richer environment for graphical experimentation.

References

Adobe Systems Incorporated (1985), *PostScript Language Reference Manual*, Addison-Wesley, Reading, Massachusetts.

Becker, Richard A., John M. Chambers, Allan R. Wilks (1988), *The New S Language*, Wadsworth, Pacific Grove, California.

Chambers, John M. (1994), *Overview of Version 4 of S*, AT&T Bell Labs Research Report, Murray Hill, New Jersey.

Cleveland, William S. (1994), *The Elements of Graphing Data*, Second edition, Hobart Press, Summit, New Jersey.

Wilks, Allan R. (1994a), *Controlling Graphics Devices for Statistical Graphics*, AT&T Bell Labs Research Report, Murray Hill, New Jersey.

Wilks, Allan R. (1994b), Pictor *Programmer's Guide*, AT&T Bell Labs Research Report, Murray Hill, New Jersey.