Graphical Selection of Data Views

Catherine B. HURLEY

Graphical selection of data views is a fundamental task in interactive statistical graphics. Linked plots provide a form of indirect selection, where direct manipulation of objects displayed in one plot indirectly selects objects in other plots. A pointing device or brush is typically used for direct manipulation, and so this indirect selection is commonly known as linked brushing. Most commonly, linked brushing is applied to two or more scatterplots showing various pairs of variables from a multivariable dataset. This article describes a generalization of linked brushing for the setting where plots display different, though related datasets. With this form of linking, we can graphically explore relationships between datasets. Our linking system is extensible and handles any kind of display of any kind of dataset, as well as arbitrary relationships between those datasets.

Key Words: Brushing; Linking; Lisp; Software design.

1. INTRODUCTION

Graphical selection of objects is a key task in any system for interactive statistical graphics. Many systems offer a point-and-click mechanism for graphical selection, a rectangle or lasso for grouped selection, and a brush for a continuously changing selection. The selected objects are marked by highlighting, and are then available as operands to further operations. When plots are linked, selecting objects in one plot also highlights and indirectly selects corresponding objects (in the sense of representing the same dataset cases) in other plots. Brushing of linked plots is most commonly applied to two or more scatterplots showing various pairs of variables from a multivariable dataset. With it, we visually link corresponding points across multiple scatterplots, and thus explore multivariable associations. Such selection and brushing of linked plots is often referred to as linked brushing.

Brushing methods for exploratory data analysis were first introduced in the late 1970s and early 1980s (Newton 1978; McDonald 1982), and further developed into a brushing toolkit (Becker and Cleveland 1987). However, new varieties of selection and brushing methods are the subject of current research: Ward (1997) constructed multidimensional brushes, and Ward (1997) and Noirhomme-Fraiture and Rouard (1997) designed special glyphs for graphical selection. Some systems—for example, DataDesk (Velleman 1998)—allow the current selection to be combined with the previous selection using logical

Catherine B. Hurley is Lecturer, Department of Mathematics, National University of Ireland Maynooth, Maynooth, County Kildare, Ireland (E-mail: churley@maths.may.ie).

Journal of Computational and Graphical Statistics, Volume 9, Number 3, Pages 558–581

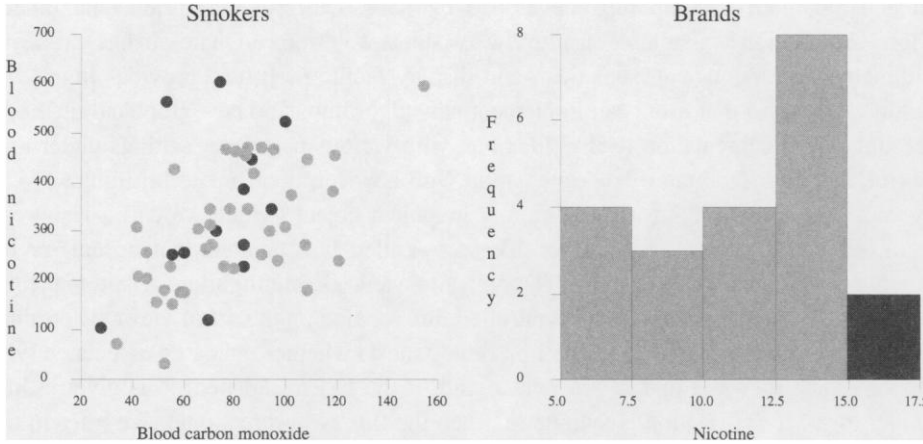

Figure 1. The left panel is a scatterplot of blood nicotine versus blood carbon monoxide for smokers; the right panel is a histogram of the nicotine levels found in the cigarette brands.

operations, and Wills (2000, p. 544 in this issue) devises a taxonomy of selection systems. In MANET, Theus, Hofmann, and Wilhelm (1997) took this idea even farther with selection sequences, where the system "remembers" the sequence of past selections.

Craig, Haslett, Unwin, and Wills (1989) generalized a different aspect of brushing: In their system, selecting cases in one plot of spatial locations causes a new point symbol to be added to a second plot, where the second plot traces an average or other summary measure of the selected locations over time. The plot of spatial locations is linked to the summary plot, even though a point symbol in the summary plot represents many spatial locations. Similarly, Haslett, Bradley, Craig, Unwin, and Wills (1991) used n-to-1 linking to link a plot of spatial locations to a variogram cloud.

This article discusses a general form of linked brushing which is appropriate when plots display different, though related cases. Our method handles arbitrary relationships between datasets, including relational database settings, and derived dataset settings such as summary plots and variogram clouds. A preliminary version of the algorithm was outlined by Hurley (1993), and later work gave some applications to spatial data (Hurley and Haslett 1995), and multilevel and multiway data (Hurley 1997).

As an example, consider a dataset on smoking styles, adapted from Hand and Taylor (1987). The dataset has information on 55 smokers. It also has measurements on the 21 cigarette brands smoked by the 55 smokers. Figure 1 shows two plots of the data. The first panel shows a scatterplot of the blood nicotine against blood carbon monoxide for the 55 smokers, the second panel shows a histogram of the nicotine values for the 21 brands. Here, brushing was used to color the subset of high nicotine brands in the second plot, causing the smokers of those brands in the first plot to become similarly colored. The plots show that the weak positive association between the blood variables also exists for smokers of high nicotine brands. In this example, the plots display smoker and cigarette brand cases, and the cases are related via the "smokes brand" relationship.

This article describes a general linking algorithm and its implementation in Quail. Quail (for QUantitative Analysis in Lisp) is a programming environment for statisti-

cal and quantitative computing, developed by R.W. Oldford, C. Hurley, and others. (More information is available at http://www.stats.uwaterloo.ca/Quail.) It has extensive arithmetical, mathematical, statistical, and display facilities. It also provides high-level building blocks so that users can implement new algorithms and new graphical methods. The linking implementation itself is flexible, with various parameter settings under user control, and so it facilitates experimentation with new forms of linked brushing.

With the Quail linking algorithm, any graphical object (which we call a *view*) can be linked via a pointer to any other. These so-called link pointers can be uni- or bi-directional. When view A has a link pointer to view B, highlighting A automatically highlights B. In our system, a user-controlled link relation on the set of views determines which views have link pointers. The link relation tests whether the cases associated with two candidate views A and B are related and, if the test is satisfied, A is given a link pointer to B. If the relation is symmetric, then the link is bi-directional. We refer to the setting where two or more displays showing the same dataset are linked on the basis of case identity as *standard linking*. For the Quail implementation of standard linking, we use a link relation that tests whether the cases associated with the candidate views are identical.

The outline of the article is as follows. In Section 2 we give some background on the design of the Quail system for statistical graphics. In Section 3 we describe the linking mechanism, and in Section 4 we discuss linked brushing and indirect selection of views via their links. In Section 5 we address the open-ended and extensible nature of the linking system, which works with new user-defined displays of arbitrary forms of statistical data. In Section 6 we give some examples, and in Section 7 we make some concluding remarks.

2. STATISTICAL GRAPHICS IN QUAIL

The statistical graphics system in Quail has an object-oriented design. We provide basic building blocks consisting of simple graphical objects such as point symbols and lines, and container objects within which the simple objects are positioned to display relationships, ultimately forming plots.

As an example, consider the scatterplot. There each glyph representing a case is its own data structure, or object, called a point symbol; similarly there is an axis object, and a label object. Even though a scatterplot is composed of point symbols, axes, and a few labels, it is convenient to introduce an intermediate object, called a 2-D point cloud object consisting of the point symbols. In general then, a statistical plot is a hierarchy of objects; the scatterplot object consists of axes, label, and a point cloud which itself consists of point symbols, while a scatterplot matrix consists of many point clouds and labels.

The plot and each of its components are termed a *view*. A *simple view* is a view such as a point symbol or label which contains no other views. All other views are referred to as *compound views*. In our software organization, information pertaining to a particular view is localized in slots of that view. A compound view has slots for its constituent views or *subviews* and the subview locations.

Each view has a slot for a *viewed object*, which is the piece of (usually statistical)

data associated with the view. Indeed, a view is so-called because it provides a graphical representation of the viewed object, and in fact serves as a graphical user interface to that viewed object. Although we place no restriction on the type of the viewed object, viewed objects in statistical plots are portions of the dataset displayed. Typically, for a scatterplot, the viewed object is the dataset itself, for the point symbol it is a case, and for a 2-D point cloud it is a list of cases, one for each of its point symbols. When a scatterplot viewing a dataset is constructed, its subviews are automatically constructed with appropriate viewed objects.

Aggregate views representing collections of data occur frequently in statistical graphics; examples are a bar in a histogram or a label showing the text "Female" representing the subset of female cases. We refer to these collections of data as the *viewed elements* of a view. Since in principle any view may be an aggregate view, each view has an associated viewed element list. An aggregate view has multiple viewed elements which are usually cases extracted from the viewed object. A nonaggregate view has a single viewed element, which is usually the viewed object itself.

Each view also has a slot for *drawing style* attributes, that specify how the view is to be drawn. Drawing style attributes include color information, and whether the view should be highlighted or not, and visible or not. Some kinds of view have additional drawing styles; for instance, a point symbol has a shape that may be filled or drawn as an outline, while a label has a font. Most views are *single style* views with just one set of drawing style attributes. However, views such as a bar (in a histogram or a boxplot) are *multistyle* views with many sets of drawing style attributes, one for each element in the viewed element list.

There are various ways a multistyle view could make use of the multiple drawing style values. For example, a bar allocates a proportion of the bar to each viewed element and its corresponding drawing style. These bar segments could be drawn in order, but the visual effect may be unpleasantly stripey. Since the order of viewed elements is not usually of particular interest, the default bar drawing behavior groups together segments with the same drawing style values. Highlighted segments are drawn first, and this simplifies comparison of highlighted proportions across bars of a histogram.

3. LINKING VIEWS

We describe methods for building links between views which constrain those views to have common drawing styles. Other forms of links between views which contrain other viewing pipeline elements (such as choice of variables, cases and coordinate systems) exist in Quail (Hurley 1991). Wilhelm (in press) gives a formal description of various linking structures. This article addresses linking views via their drawing styles only.

In the Quail system, there are two different implementations of linking. The first (Hurley and Oldford 1991) provides an efficient way of linking two or more scatterplots of the same data. The second method is very general and is appropriate for linking arbitrary and different types of displays, possibly showing different datasets.

3.1 REUSING VIEWS

The first linking implementation is efficient but somewhat restrictive. It relies on the fact that a single view object may be a component of more than one compound view, and thus is displayed at multiple screen locations. The scatterplot matrix makes use of this feature: there is one point symbol for each case and this point symbol is a component of all the point clouds in the matrix. Since there is just one point symbol for the case, the point symbol has just a single drawing style and the change of highlight status via the brush, say, in one location is automatically reflected at the other point symbol locations. This linking implementation is appropriate for the standard linking setting when the displays to be linked both contain point clouds.

3.2 A GENERAL LINKING ALGORITHM

The second linking implementation is far more flexible and can be applied to any display, at any stage in an analysis. This method, first outlined in Hurley (1993), is appropriate in any setting where plots display related datasets. (Of course, it also handles standard linking.) Basically, any view A can be given a link pointer to any other view B, with the result that a change of drawing style in A is reflected in a change of drawing style in B. In the standard mode of operation, two views have link pointers to each other if they display the same data but, more generally, the user controls which views should have link pointers by specifying a link relation—that is, a function which, given a pair of views A and B, decides whether or not A should have a link pointer to B.

The algorithm uses a data structure called a *link table* to keep track of the link pointers between views. The link table is a graph where the views are vertices and the link pointers between views are edges. The graph is usually undirected, but may be directed. For a view A, let $L(A)$ denote the set of views which are connected by link pointers from A. (Note $A \notin L(A)$.) When the link table contains an edge connecting view A to view B, we say that A is *linked* to B. More generally, view A is said to be linked to view B when there is at least one pair of views, one each from the A and B hierarchies, which are connected by an edge. In the standard setting where two scatterplots of an n-case multivariable dataset are linked, there are $2n$ point symbols in the link table, and an undirected edge connects the ith point symbol from one plot with the ith point symbol from the other.

When view A is added to the link table, the link relation which is a property of the link table is used to determine its link pointers. For a view B in the link table, the link relation tests if the data associated with A and B in the form of their viewed elements are related. Specifically, the relation returns "true" if a user-specified function called the *link test* returns "true" for any pair of viewed elements a of A and b of B. The following pseudo-code gives more detail:

```
For B in link table
        If link-test (viewed element of A, viewed element of B) then
                A ⟶ B: add B to L(A)
        If link-test (viewed element of B, viewed element of A) then
                B ⟶ A: add A to L(B)
Add A to the link table
```

If the link test is known to be symmetric, then a slight simplification is possible, and the resulting graph will be undirected:

For B in link table
    If link-test (viewed element of A, viewed element of B) then
        $A \longleftrightarrow B$: add B to $L(A)$ and A to $L(B)$
Add A to the link table

The default link test is suitable for the standard linking setting. This link test, called `eq-dataset`, tests whether two viewed elements represent identical data. Figure 2 shows part of a link table using this link test. The function `eq-dataset` is symmetric in its arguments, so the graph is undirected. `Point-symbol-1` and `bar-1` both represent `case-a` (the viewed element), so they are linked. Similarly, `bar-1` and `bar-2` along with `point-symbol-2` represent `case-b`, so all pairs of these views are linked to each other. `Point-symbol-3` is still part of the link table, though it has no edges. Note that even though `eq-dataset` is transitive, the resulting graph structure is not, unless each view has only one viewed element.

3.3 SEMANTICS OF LINKED VIEWS

When the analyst changes the highlight status, color, or other public drawing style attribute of a view A, the new value is passed on to views which are links of A, namely $L(A)$. (We will refer to the view operated on by the analyst as the *active* view.) By default, all drawing style attributes of a view A are public, in the sense that they are passed to its links and may be changed from its links, with the exception of the font style of labels, which are designated as private. However, the analyst may change this designation for an individual view. Unless otherwise indicated, we will assume in this discussion that all drawing styles are public.

When the analyst changes the color of `bar-1` in the link table of Figure 2, `point-symbol-1`, `point-symbol-2`, and `bar-2` all receive the new color. Conversely, when she changes the color of `point-symbol-1`, the segment of `bar-1`

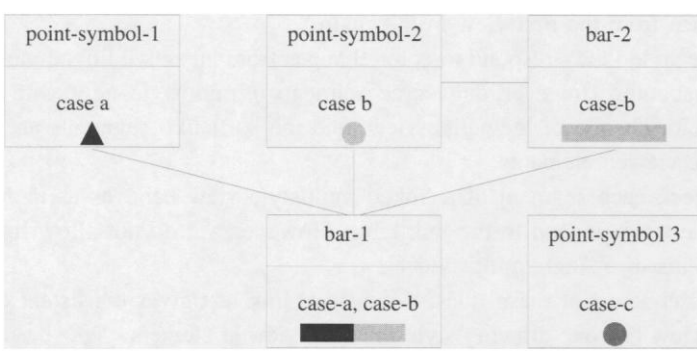

Figure 2. Part of a link table that uses link test `eq-dataset`.

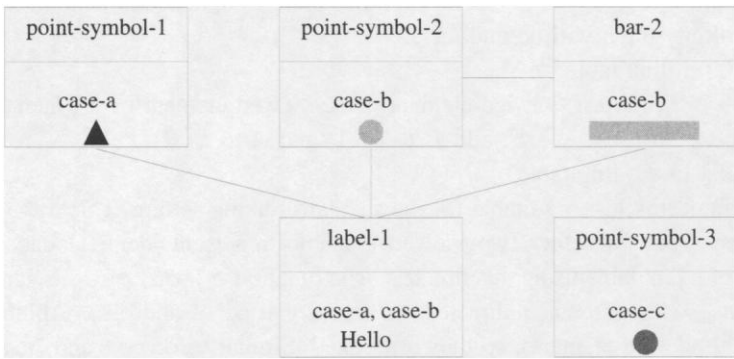

Figure 3. Part of a link table that uses link test `eq-dataset`.

representing `case-a` changes color, as shown in Figure 2. (If instead she changes the shape of `point-symbol-1`, `bar-1` is unaffected, since it does not have the shape drawing style attribute.) The views in this figure each have one color for every viewed element or case. When they are linked using the test `eq-dataset`, the colors are consistent in the sense that each case is represented by the same color in all views.

Suppose that the view marked `bar-1` was instead the single-style view `label-1`, displaying the text "Hello", as in Figure 3. Then the links between views are unchanged—notice the algorithms given in the previous subsection do not depend on the type of view. When the analyst changes the color of `label-1`, the views `point-symbol-1`, `point-symbol-2`, and `bar-2` all receive the new color, as before. However, when she changes the color of `point-symbol-1` to black, all of `label-1` is drawn in black, as shown. Note that the colors in Figure 3 are no longer consistent, because `case-b` appears in both black and gray views.

In general, drawing styles of views in a link table obey the following rules:

1. Drawing style changes to an active view are passed on to its linked views, but do not travel further to linked views of the linked views.

   Allowing changes to travel further could be chaotic; in the case of Figure 3, a change of `point-symbol-1` to red would propagate first to `label-1`, and then on to `point-symbol-2` and `bar-2`. (It might be interesting to devise a system where the color assigned to a related view is diluted according to its distance from the first view in the chain.)

2. Single style views respond to color changes from an active linked view by adopting that color. However, only some segments of multistyle, aggregate views adopt the color change of the active view, and the particular segments are determined by the viewed elements.

   In effect, each segment of a linked multistyle view behaves as if the segments themselves belonged to the link table. However, we do not allow links between segments of a single multistyle view.

As a consequence of these rules, a transitive link test gives consistent colors when each linked view has one drawing style for each viewed element.

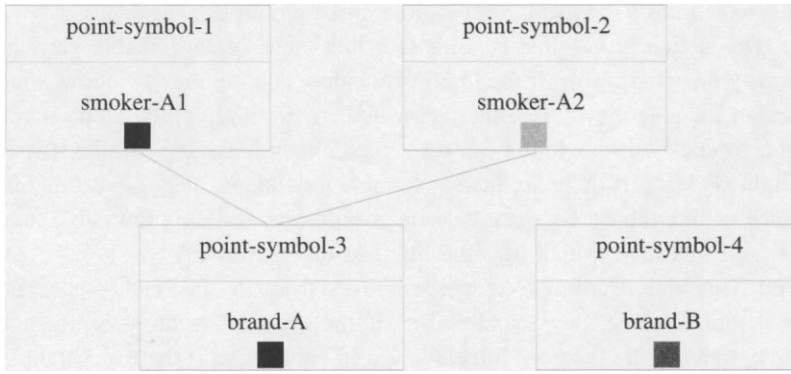

Figure 4. Part of a link table that uses link test `smokes-brand`.

3.4 ANOTHER LINK TEST

Consider the dataset on smoking styles described in the Introduction and shown in Figure 1. The left plot shows smoker cases and the right plot cigarette brand cases and these cases are related via the "smokes brand" relationship. Both plots are linked, into a link table using the `smokes-brand` link test. This test returns true, if given a smoker and a cigarette brand, the smoker smokes that brand.

Figure 4 shows part of a link table using link test `smokes-brand`. `Point-symbol-1` representing `smoker-A1` and `point-symbol-3` representing `brand-A` are linked to each other, because the `smoker-A1` smokes `brand-A`. Similarly for `point-symbol-2`.

When the analyst changes the color of `point-symbol-3`, `point-symbol-1` and `point-symbol-2` automatically receive the new color. Similarly, when she changes the color of `point-symbol-1`, `point-symbol-3` automatically receives the new color as shown, but the new color does not travel on to `point-symbol-2`.

An alternative link test to `smokes-brand` for the smoking styles dataset compares cases on the basis of their brand values. This test, call it `eq-brand` brand, returns true for two smokers using the same brand, or, when a smoker uses the given brand (as tested by `smokes-brand`). When this link test is used on the views of Figure 4, the only change is the addition of an extra edge between `point-symbol-1` and `point-symbol-2`.

3.5 USER INTERFACE

In the previous subsection, we described the various options available in our linking algorithm. These are summarized as follows:

Views which are designated *linkable* may be added to a *link table*. The links of a view in the link table are determined by its *link test*, which compares *viewed elements* of the views. Thereafter, when the user directly changes a *public style* of a linked view, the new style is passed to the linked views for which this is also a public style.

All of the items in italics may be controlled by the user, through a Lisp interface. At present, we provide a limited graphical user interface. The Quail menubar located on

the top of the screen has a submenu with various linking options. These are:

Link views: This action links—that is, adds to a link table—each linkable view in the topmost view window, or, if the topmost window is not a view window, links all on-screen linkable views. In either case, if there are highlighted views available (in the topmost view window or on screen), then linking is limited to those highlighted views. If there are many available link tables, then the user is offered a choice of link tables. By default, there is only one available link table, namely the default link table which uses the link test `eq-dataset`.

Unlink views: This action unlinks—that is, removes from its link table—each linked view in the topmost view window, or, if the topmost window is not a view window, unlinks all on-screen linked views. In either case, if there are highlighted views available (in the topmost view window or on screen) then unlinking is limited to those highlighted views.

New link table: This option creates a new link table. The user is prompted for the a link test and name for the new table.

Delete all link tables: This option unlinks all views and deletes all link tables, including the default link table. This action effectively undoes all "link" and "new link table" actions.

The "link" and "unlink" actions are also available on a menu associated with each view. In this case, the actions are limited to views in the hierarchy of the selected view.

4. SELECTION OF VIEWS

Graphical selection of objects in a statistical plot is available in many statistical software systems. Selection, or more precisely direct selection, of a view or views occurs when the analyst uses a pointing device (usually a mouse cursor or brush) to indicate views appearing on screen. In Quail, any view whether simple or compound may be selected, though brushes operate only in certain contexts such as point clouds. Indirect selection occurs as a consequence of linking: by directly selecting a view, the analyst indirectly selects those views which are links of the selected view.

4.1 DIRECT SELECTION OF VIEWS

A single view is (directly) selected by pointing at it with the mouse cursor and depressing a mouse button. In Quail, there are nine modes of selecting a single view, available by depressing one of three mouse buttons, with or without the shift and control modifier keys. (For a single-button mouse, modifier keys are used in conjunction with the mouse to simulate middle and right buttons.) Although some may find nine selection modes excessive, we believe that the organization of modes is logical and should not unduly burden the user. Table 1 summarizes the resulting nine modes of selection.

The left button (without any modifier keys) is reserved for highlighting the selected view: this is the action labeled (1) Replace-Select in Table 1. Middle and right buttons offer menus of operations which can be invoked on the selected view. The view modification menu includes options for changing various drawing style parameters, and

Table 1. Modes of Direct Selection of a Single View

	Mouse button		
Modifier	*Left*	*Middle*	*Right*
left	(1) Replace-Select	view modification menu	view management menu
shift	(2) Union-Select	(3) Intersection-Select	(4) Difference-Select
control	Viewed object menu	Inspect viewed object	Display viewed object

other options appropriate to the type of view. The view management menu is common to all types of view and includes options for moving, copying, linking, and unlinking the view. Pressing the shift key in conjunction with the mouse buttons permits logical operations that combine the selected view with previously highlighted views: these are the actions labeled (2) Union-Select, (3) Intersection-Select, and (4) Difference-Select in Table 1. Furthermore, pressing the control key in conjunction with the mouse buttons allows various forms of access to the viewed object of the selected view.

In addition to the point-and-click mechanism for selection of individual views, Quail offers a brushing rectangle for grouped selection. There are four modes of grouped selection: as with individual view selection, the left button is reserved for Replace-Select, and pressing the shift key in conjunction with the mouse buttons permits the operations Union-Select, Intersection-Select, and Difference-Select. As Wills (2000, p. 544 in this issue) points out, there are 16 possible logical operations, and 65,536 possible selection systems. (The system described here contains four of the modes of Wills' recommended five-mode system.)

The selection modes numbered (1) through (4) change the highlight status of subsets of views. Let S denote the current view selection. This is either a single view selected by pointing at it with the mouse cursor and depressing a mouse button, or a group of views selected with a brushing rectangle, in both cases with or without a modifier key. Let H denote the result of the previous selection. This contains the views which are completely highlighted and the highlighted segments of multistyle views before the new selection is made. Let H_{new} denote the result of the selection operation. This consists of the views which are completely highlighted and the highlighted segments of multistyle views, when the selection action is completed.

Table 2 describes the selection operations Replace-Select, Union-Select, Intersection-Select, and Difference-Select in the absence of linking. Each operation combines the views in the sets S and H resulting in the new selection, H_{new}. Note that, in the absence of linking, the operations affect the highlight status of views in S or H only.

As defined above, Replace-Select conforms to standard user practice: clicking on an object highlights that object and downlights all other objects, and clicking on "empty space" where there is no object simply downlights all objects. In Quail, however, mouse clicks on view images rarely result in $S = \emptyset$. Since we need a fast way to downlight all views, we make an exception to the Replace-Select rule given above: if S consists of a single view which is already completely highlighted, all views including S are downlighted.

Table 2. Selection Modes That Combine Current and Previous Selections

Action	*Result* H_{new}	*Algorithm*
(1) Replace-Select	S	`Downlight` H `Highlight` S
(2) Union-Select	$S \cup H$	`Highlight` S
(3) Intersection-Select	$S \cap H$	`Downlight` $H \cap \overline{S}$
4) Difference-Select	$H \cap \overline{S}$	`Downlight` S

In the description so far, S consists of a single view, or a number of views at the same depth in a view hierarchy that may be selected via a brushing rectangle. There is no such restriction on the views in H, since collections of highlighted views may be formed by repeating the actions Replace-Select, Union-Select, Intersection-Select, and Difference-Select. Once a collection of highlighted views H is formed, it may be saved and then retrieved at a later stage. The retrieved set of views may be (replace) selected as in $H_{\text{new}} = S$, where now S denotes the saved set, or using any of Union-Select, Intersection-Select, or Difference-Select, as described above. Therefore, the four selection operations may be applied to arbitrary sets of views.

4.2 Indirect Selection of Views

Direct selection of a view A indirectly selects its linked views $L(A)$. As explained in Section 3.3, when the analyst selects a view and changes its drawing style, the drawing styles of linked views are changed as a consequence. (Other changes to the selected view do not affect its linked views.) However, we need to clarify the effect of Replace-Select, Union-Select, Intersection-Select, and Difference-Select in the presence of linking.

Recall that $L(A)$ denotes the set of views that are connected by link pointers from the view A. However, if B in $L(A)$ is a multistyle view, changing a drawing style attribute for A may not affect all segments of B, as demonstrated by Figure 2 (p. 563). Therefore, it is convenient to redefine $L(A)$ to include only those segments of multistyle views which are responsible for the link pointers from A. Furthermore, when S is a set of views, let $L(S)$ denote $\cup_{A \in S} L(A)$.

In the case of Replace-Select, highlighting a set of views S automatically highlights views (or segments) in $L(S)$. So, this suggests that the result H_{new} should be $S \cup L(S)$. More generally, there are two obvious ways of generalizing our selection operations in the presence of linking: Either (1) replace S by $S \cup L(S)$ or (2) redefine the result as $H_{\text{new}} \cup L(H_{\text{new}})$, where H_{new} is as defined previously. Since the user chooses the set S, it seems reasonable to assume that the user is focused on S and as a consequence $L(S)$, suggesting that option (1) is more appropriate. In the presence of linking then, we simply replace each S in Table 2 by $G(S) \equiv S \cup L(S)$.

As before, the four selection operations change the highlight status of subsets of views. In the presence of linking, only the highlight status of views or segments in $G(S)$ or H is altered, other views are unchanged. Note that the result of the four operations does not depend on the type of view S.

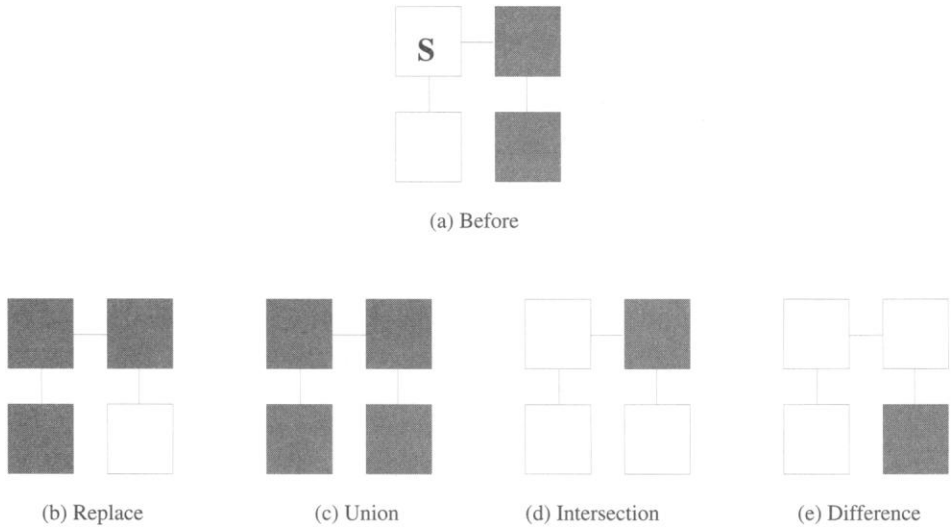

Figure 5. Part of a link table before and after the four Selection actions.

The four selections operations are easily explained by examining Figure 5. This shows part of a link table with four views and three (undirected) links. Initially, two of the four views are highlighted. These views are marked with a gray fill in panel (a) and form the set H. The user then selects the view marked **S**, using one of the four selection modes. Each of the four lower panels shows the result H_{new} after a selection action; again these views are marked with a gray fill.

As we see from Figure 5(a), after Replace-Select, S and its links are the only highlighted views. Note that options (1) and (2) are equivalent here. As explained in Section 4.1, we make an exception to the Replace-Select rule in the case where S consists of single view which is already completely highlighted. Then all views including S are downlighted.

Union-Select highlights S and $L(S)$, other views are unchanged. With Intersection-Select, only the subset of highlighted views or segments which are in S or $L(S)$ remain highlighted. Difference-Select removes S and $L(S)$ from the set of views or segments H which are highlighted.

The result of the selection operation H_{new} consists of all views or view segments which are highlighted, either directly, via operating on that view or indirectly because the view or view segment was linked to a selected view. This result may be further combined with a new selection, using one of the four selection operations. At this stage, it is not relevant how views got into H_{new}, via direct or indirect selection. This seems reasonable, because the user sees highlighted views on the screen but may not remember the sequence of actions used to arrive there.

5. EXTENSIBLE LINKING

The Quail system is intended to be open-ended, so users can use it to construct new kinds of views displaying new kinds of statistical data, as appropriate to the application

at hand. Similarly, the linking system itself is open-ended, and works on new views, new data, and even new link tests.

As described in Section 3, the linking mechanism works for any view in any plot, and because of the object-oriented nature of the system, this extends to new views.

The linking system works with new kinds of data and new link tests, because the plot system deals with an abstraction or model of the dataset, rather than the actual data structure itself. The abstract version of the data is implemented by the plot-data interface (Hurley 1993), which contains the functions the plot system uses to extract information from the data. By adding appropriate functions to the plot-data interface, plot construction and behavior become effectively independent of the physical dataset implementation.

The plot-data interface makes it easier to implement new data display methods and to construct plots of new forms of statistical data. Correspondingly, the interface makes it easier to implement new link tests and to apply existing built-in link tests to new forms of statistical data.

5.1 THE PLOT-DATA INTERFACE

The elements of our data model are cases, variables and the dataset itself. A case contains observations on an individual, a variable is an index into the case used to extract an observation, and a dataset is composed of cases. We also regard an individual case as a dataset. Additionally, a dataset (and hence a case) may have an identifier, which describes or names the dataset.

The plot-data interface is easily implemented using the generic functions of Common Lisp (Steele 1990). Some key generic functions in the interface are:

`dataset-p`: Each dataset object for which the plot-data interface is implemented should satisfy the `dataset-p` predicate.

`list-cases`: Given a dataset as argument, this function returns a list of its cases. It is used in scatterplot construction for instance, where cases are extracted from the dataset to become the viewed objects of the point symbols.

`value-of`: This function takes a case and index (the variable) as arguments and returns a value. It is used in a scatterplot to obtain the x and y point symbol coordinates.

`identifier-of`: Given a dataset as argument, this function returns an identifier, which are used by labels for instance. Any type of object could be used as an identifier, but, most commonly, the identifier is a string naming the dataset.

We have implemented the plot-data interface for arrays. (The arrays of Quail are a richer version of Common Lisp arrays.) For a 2-D array, the interface assumes that the cases are rows and the variables columns. We also provide a function called `dataset` that associates case and variable identifiers with the rows and columns of a 2-D array and an identifier with the dataset itself, without changing the physical structure of that array. Alternatively, a k-dimensional array could contain the responses for $m \leq k$-way data. The function `mway-dataset` associates the factors with sucessive array dimensions, and assumes that remaining array dimensions contain variable values. Examples of the plot-data interface are given in the Appendix.

5.2 DATA COMPARISONS

The link test `eq-dataset` is the default data comparison function. This test is a generic function, and its method when its arguments are Lisp objects of arbitrary type is shown in the following. The method returns `t` (for true) when both objects satisfy `dataset-p` and are the same object, as tested by the (built-in) Lisp function `eql`.

```
(defmethod eq-dataset((a t) (b t))
  (and (dataset-p a) (dataset-p b)
       (eql a b)))
```

We can also use built-in link tests with new forms of statistical data. Suppose we wish to use a different kind of data structure to contain a dataset. To construct a plot of this dataset, we must first implement the necessary functions in the plot-data interface. Any built-in link test will then work for this new form of statistical data. Alternatively, the built-in link tests are generic functions and we may choose to provide new methods appropriate for the new data structures.

Next, we explain how to use the interface to implement a new link test. Suppose, for instance, we wish to define a link test `eqd-identifier` which returns true when two datasets have the same identifier. For example, consider two related datasets, containing patient information for two different years. A patient in both datasets has one case object for each year, and these case objects are not `eq-dataset`. If both cases use the same identifier, however, then they will satisfy the `eqd-identifier` test. As a second example, consider a dataset and a new derived dataset obtained perhaps from a principal component analysis. The derived dataset cases contain the principal component scores for the original dataset and carry over the same case identifiers, thus the ith original case and the ith derived case satisfy `eqd-identifier`.

The `eqd-identifier` function is easily implemented using the functions in the plot-data interface as follows:

```
(defun eqd-identifier(a b)
  (and (dataset-p a) (dataset-p b)
       (eq-identifier (identifier-of a) (identifier-of b))))
```

The function `eqd-identifier` first checks that both of its arguments satisfy `dataset-p`. Then it checks to see if the arguments have the same identifier, using the function `eq-identifier`. This latter function (also belonging to the plot-data interface) returns true if the identifiers are the same string for instance, but it could also be used to compare more general identifier objects.

6. EXAMPLES

6.1 SMOKING STYLES DATA

Recall the smoking styles data of the Introduction. The dataset has information on 55 smokers. It also has measurements on the 21 cigarette brands smoked by the 55 smokers.

A smoker's cigarette brand name, age, sex, and three blood measurements taken after

smoking were recorded. This information is contained in a 55 × 6 array. For each cigarette brand, information on brand name, and three other brand characteristics was recorded, and this is contained in a 21 × 4 array. The `dataset` function of Section 5.1 is used to associate case and variable identifiers with the rows and columns of these arrays. The smoker's names were not recorded and so we give smoker cases identifiers such as `smoker-1`, `smoker-2`, and so on. The brand names are used as case identifiers for the brand array.

Next, we define the link test `smokes-brand` which returns true, if given a smoker and a cigarette brand, the smoker smokes that brand.

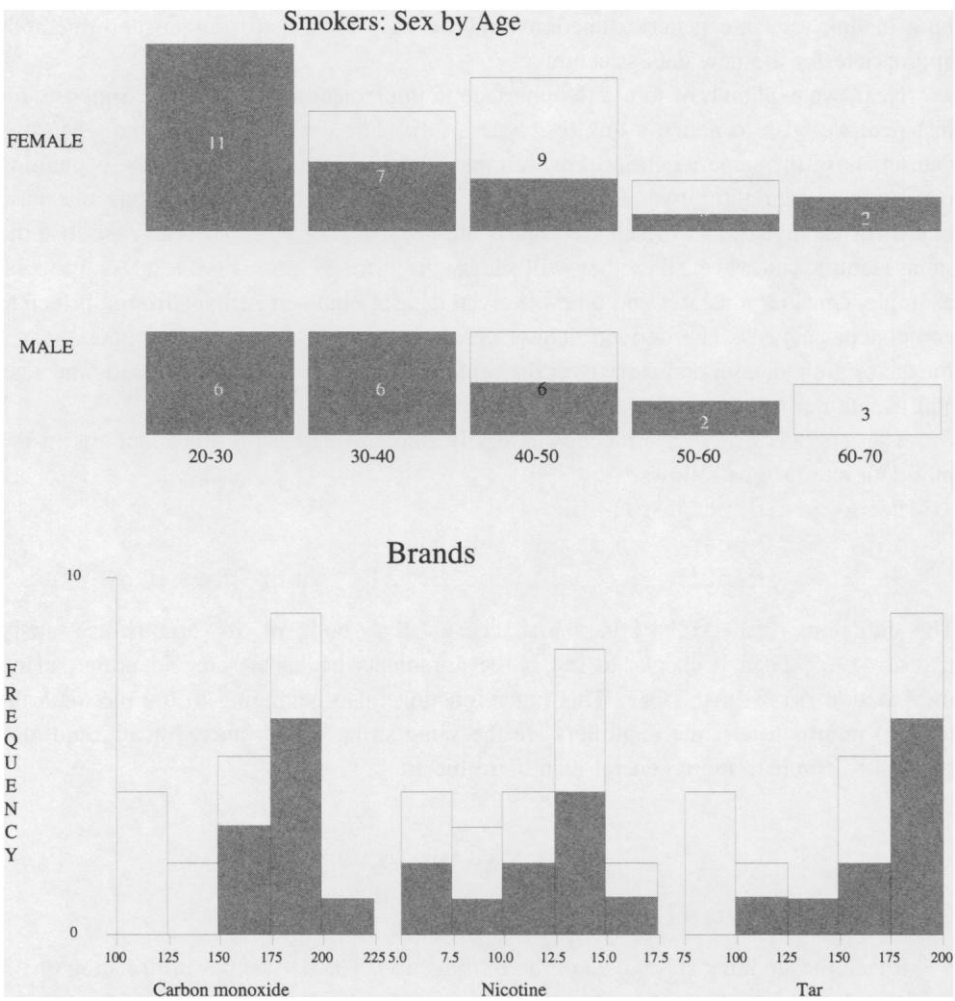

Figure 6. The top panel shows a barchart of smokers categorized by age group and sex. The lower panel shows histograms of three brand variables. The gray fill shows characteristics of brands smoked by females in the 20–30 age group.

```
(defun smokes-brand(a b)
  (and (dataset-p a) (dataset-p b)
     (or (equal (identifier-of a)
          (value-of b "brand" :default "na"))
         (equal (identifier-of b)
          (value-of a "brand" :default "na")))))
```

Brand information is a variable of smoker cases and an identifier of brand cases. The above function first checks that both arguments are datasets, so that the `value-of` and `identifier-of` functions may be invoked. The value of the default keyword to the `value-of` function (here `"na"`) is returned when the case does not have a variable brand.

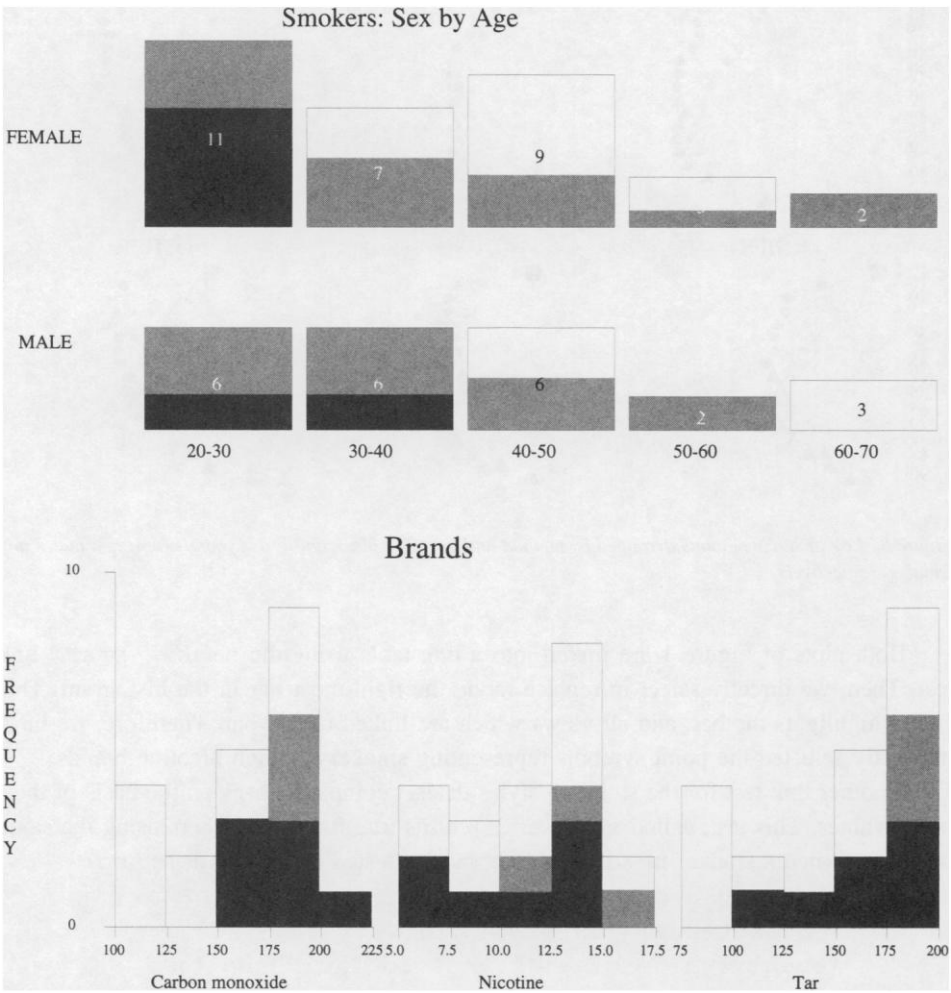

Figure 7. The top panel shows a barchart of smokers categorized by age group and sex. The lower panel shows histograms of three brand variables. The dark gray fill shows characteristics of brands smoked by females aged 20–30 and no other females, while the lighter gray shows characteristics of the remaining brands smoked by females aged 20–30.

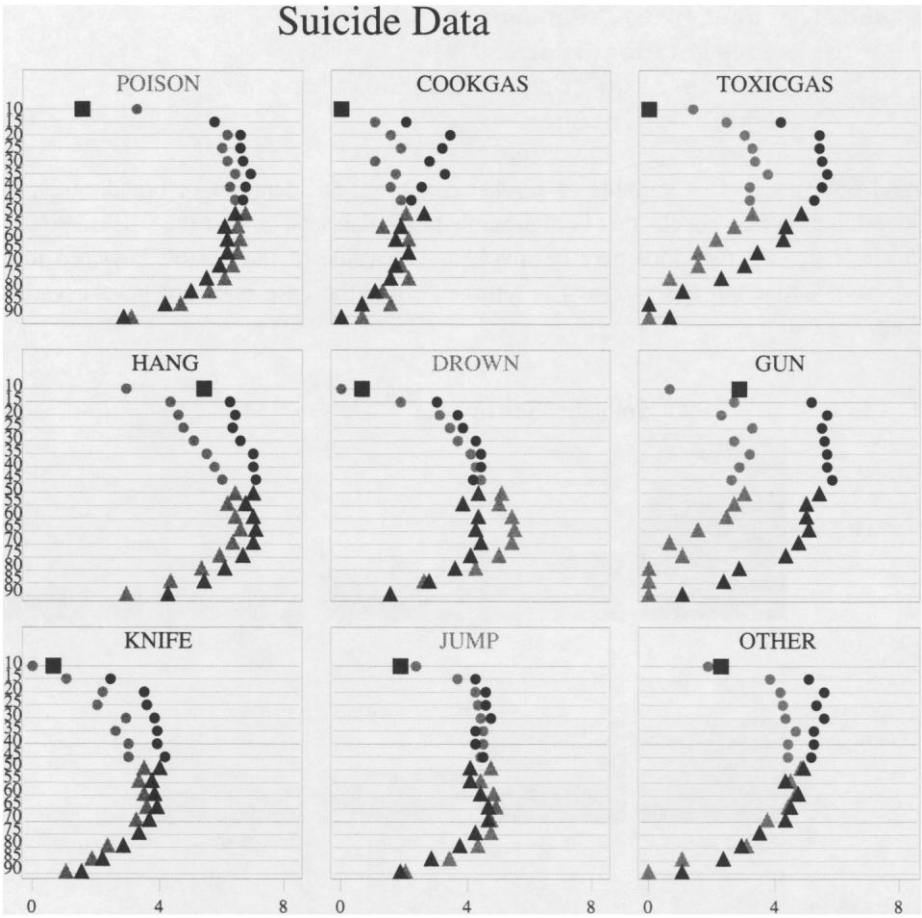

Figure 8. Log of suicide counts arranged by method and age. The black and gray symbols represent males and females, respectively.

Both plots of Figure 1 are linked into a link table using the `smokes-brand` link test. Then, we directly select in replace mode the right-most bar in the histogram. This action highlights the bar, and all views which are linked to this bar. Therefore, we have indirectly selected the point symbols representing smokers of high nicotine brands.

Another link test for the smoking styles dataset compares cases on the basis of their brand values. This test, called `eq-brand`, returns true for two smokers using the same brand, or, when a smoker uses the given brand (as tested by `smokes-brand`).

```
(defun eq-brand(a b)
  (and (dataset-p a) (dataset-p b)
       (equal (value-of a "brand" :default "na1")
              (value-of b "brand" :default "na2"))))
```

The link test `eq-brand` was used to link the plots of Figure 6. This shows barcharts of the age groups for the male and female smokers, and histograms of three brand variables. If we Replace-Select the bar for females in the 20–30 age category, this action

Age	Method	Sex
10	POISON	MALE
15	COOKGAS	FEMALE
20	TOXICGAS	
25	HANG	
30	DROWN	
35	GUN	
40	KNIFE	
45	JUMP	
50	OTHER	
55		
60		
65		
70		
75		
80		
85		
90		

Figure 9. Levels of the factors age, method, and sex in the suicide data.

shows that the brands smoked by young females are commonly smoked by males and females of most age groups, and that these brands have relatively high tar and carbon monoxide values. We then used a Difference-Select action on the bars for females in the older categories, and colored the result a dark gray to distinguish it from the result of Replace-Select, as shown in Figure 7. We can then compare the characteristics of brands smoked by young women and no other women (dark gray) with the characteristics of brands smoked by young women and other women (light gray).

6.2 SUICIDE DATA

We will examine a multiway dataset consisting of the numbers of suicides in West Germany in 1974–1977 (Van der Heijden and de Leeuw 1985). The suicides are classified by age (17 groups), method (9 categories), and sex. Therefore, this multiway dataset has one response variable (`count`) and $17 \times 9 \times 2 = 306$ cases, one for each combination of the three factor variables.

Figure 8 shows a multiway dot plot of the log counts (with one added, because of the presence of zero values). Each panel shows a different method of suicide, where the male and female values for each age group are shown in black and gray, respectively. The display of Figure 9 listing factor levels provides a useful method of selecting point symbols for required combinations of values. Highlighting a label for a factor level in Figure 9 automatically highlights all point symbols for that factor level. Furthermore, when the labels for `drown` and `age 30` are highlighted in intersection mode, this action automatically highlights the point symbols for `drown` at `age 30`. When in addition the `female` label is highlighted in intersection mode, we have located the point symbol for females who drowned in the 30 age group.

For the suicide data, it appears as if the method of suicide varies with sex and age.

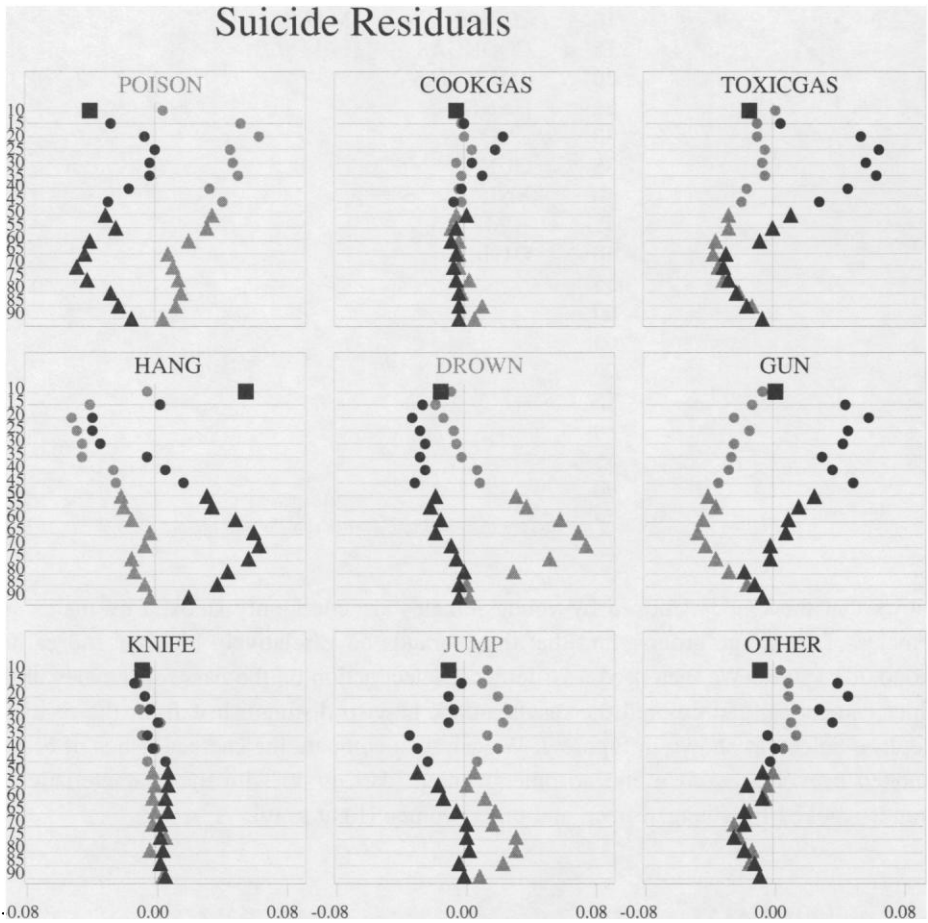

Figure 10. The plot shows the residuals assuming age and sex are independent of method, arranged by method and age. The black and gray symbols represent males and females, respectively.

To examine the nature of the dependence, we compute the (Pearson) residuals from the fit obtained by assuming method is independent of age × sex. The plot of residuals is shown in Figure 10. We then perform a correspondence analysis to summarize the variation in these residuals, and plotted the first two dimensions as in Figure 11.

When the plots in Figures 10 and 11 are linked to those in Figures 8 and 9, the point symbols on the right side in the CA plot inherit a gray color, indicating that they represent female age groups, while those on the left side inherit a black color, indicating that they represent male age groups. When we change the shape of the point symbols with positive scores for CA2, we discover that these correspond to the older age groups, and that the CA2 scores tend to increase with age, both for males and females. The point in the upper left side is clearly an outlier—it is marked with a black solid box, and corresponds to males in the youngest age category. From an examination of the residual plot, this category of males is over-represented for `hang` and under-represented in categories `toxicgas`, `gun`, and `other` relative to other young males.

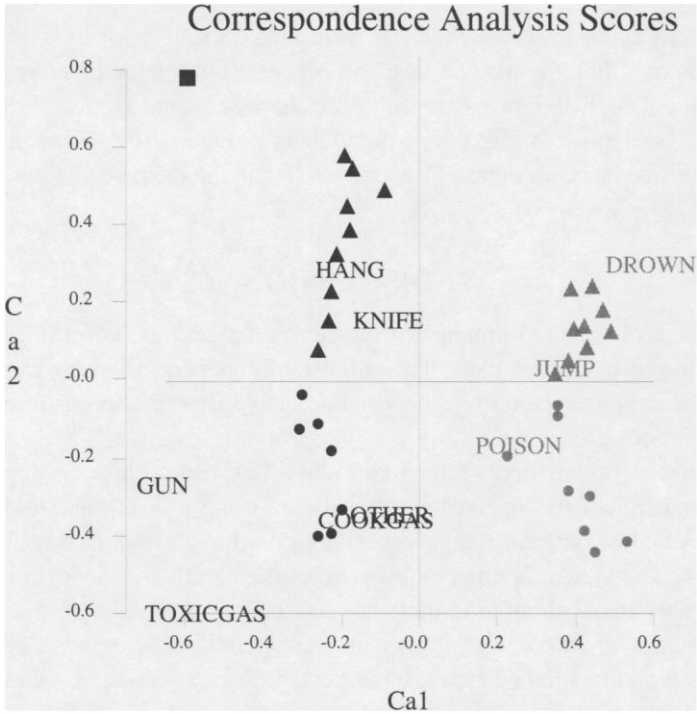

Figure 11. The plot shows the first two dimensions of correspondence analysis scores, computed from the residuals assuming age and sex are independent of method.

The correspondence analysis plot also groups suicide methods with similar age × sex profiles. The success of this grouping can be checked by examining the multiway plot of residuals. Here the patterns of residuals for `gun` and `toxicgas` look similar, except for males in the 60 to 75 age groups. At first glance, the profiles for `hang` and `drown` look very similar, but on closer inspection we see the patterns for males and females are reversed.

The plots above display a number of different data arrays. First, there is the raw suicide data which is stored in a 17 × 9 × 2 array. We then use the function `mway-dataset` (see Section 5.1) which builds a plot-data interface and associates a case with each element of the array. Second, the computation of residuals produces a new 17 × 9 × 2 array of residuals and, as before, we use the function `mway-dataset` to build a plot-data interface. Third, the correspondence analysis produces two arrays, the first of dimension 17 × 2 × 9 containing the nine CA-scores for age × sex, and the second of dimension 9 × 9 containing the nine CA-scores for method. Once again, we use the function `mway-dataset` to build a plot-data interface for these arrays with the appropriate cases and identifiers.

Each of these data arrays contains different cases and so their plots cannot be linked using `eq-dataset`. However, identifiers have been assigned to the cases via the `mway-dataset` function, and the identifiers are related. Therefore, we use a link test called `subset-identifiers` that, given two cases, returns true if the identifiers of one case are a subset of the identifiers of the other. In the above displays, all point

symbols and the factor levels of Figure 9 were linked using `subset-identifiers`. Similarly, we could link the method labels in Figure 10 to related point symbols in the multiway plots, though this is less useful since the association is clear from the panel labels. Instead, we chose to link the method labels in Figure 10 to those in Figure 11. Then, the right side labels in Figure 11 corresponding to the methods favored by females, were colored gray.

7. DISCUSSION

The purpose of standard linking is to explore relationships between the aspects of a dataset portrayed in various plots. The purpose of generalized linking as described in this article is to explore relationships between possibly different datasets through aspects portrayed in various plots.

The linking algorithm stores linked views in a link table. Links between views are determined by comparisons of their associated data using a link test. The link table is then an undirected or directed graph, depending on whether the link test is symmetric or asymmetric. The system permits multiple link tables, each with its own collection of views, but a view may belong to at most one link table at a time.

The semantics of linked views are that user-invoked drawing style changes on a view are passed on to its linked views. In particular, direct selection of a view highlights that view and its linked views, thus indirectly selecting the linked views. Note that it is the views themselves that are selected, not the associated data. This distinction is important when it comes to understanding the semantics of the Intersection-Select and Difference-Select of Section 4.2.

The linking algorithm handles aggregate views in a systematic way, whether they have multiple drawing styles or not. The methods will work with new user-defined types of view and types of data, as long as these new types use the protocols specified by the plot-data interface of Section 5.1. A user may also supply a new link test, appropriate to the application at hand. Note that though our linking methods are described in the context of statistical displays of statistical data, there is nothing statistical about the algorithms and any view (in a plot or not) representing any data object can be added to a link table and assigned links according to its data object.

A number of extensions to our linking methods are possible. We could investigate allowing drawing style changes from an active view to travel on from its linked views onto their linked views and so on, where the style values are somehow diluted as the distance increases. A more promising approach might be to allow weighted links between views so the link table is a weighted graph. Then the drawing styles should propagate from the active view in a fashion that reflects the weights of the links. These possibilities, to be useful, would clearly require a graphical user interface which is far more extensive than the version described in Section 3.5.

Our linking algorithm does not rule out different behaviors for the linked views. As described in Section 3.3, an active view simply passes on drawing styles to its linked views. Each view type makes use of its drawing styles in an appropriate fashion and it is irrelevant whether the drawing style values came from a linked view or directly from the analyst. Some researchers—for example, Eick and Wills (1995) and Unwin,

Hawkins, Hofmann, and Siegl (1996)—advocate that the highlighted data (specifically, the subset of a dataset appearing in highlighted views or view segments) be displayed in a way that facilitates comparison with the remaining subset, or the entire dataset. For example, the MANET system draws a second boxplot of the highlighted data over the boxplot of all the data. This is not the default behavior of Quail boxplots, which simply highlights a proportion in the center of each box (which is a multistyle bar object). We could implement the MANET behavior by linking a (multistyle) boxplot itself rather than its constituent boxes, and then requiring the boxplot to draw the highlighted data in a second boxplot.

APPENDIX: PLOT-DATA INTERFACE EXAMPLES

We have implemented the plot-data interface for arrays. (The arrays of Quail are a richer version of Common Lisp arrays.) Suppose there is a 4 × 2 array called `d`. The interface assumes that the cases are rows and the variables columns. Below we show some Lisp expressions and their results, which demonstrate the interface. (Note that `<-` is the Quail assignment operator. Unlike the usual Lisp `setq`, it does not print out the result.)

```
? (<- d (array '((1 2 ) (3 4) (5 6) (7 8))))  ; a 4x2 array
? (<- c0 (first (list-cases d)))  ;the first case
? (value-of c0 1)
     2      ;the value of the first case for the second variable
? (identifier-of c0)
    NIL   ; c0 has no identifier
? (identifier-of d)
     NIL   ; d has no identifier
? (scatterplot :data d :x 0 :y 1)   ; displays a scatterplot of d.
```

We also provide a function called `dataset` that associates case and variable identifiers with the rows and columns of a 2-D array and an identifier with the dataset itself, without changing the physical structure of that array. Below we show some Lisp expressions and their results, which demonstrate the effect of this function for the array `d` above:

```
? (dataset d :name "Test data"
          :identifiers '("Bob" "Tom" "Kate" "Rob")
          :variables '("height" "weight"))
      #2m((1 2) (3 4) (5 6) (7 8))      ; d
? (<- c0 (first (list-cases d)))      ;the first case
? (value-of c0 "weight")
     2        ;the value of the first case for "weight"
? (identifier-of c0)
          "Bob"   ; c0 has an identifier
? (identifier-of d)
     "Test data"   ; d has an identifier
? (scatterplot :data d :x "height" :y "weight")
  ; displays a scatterplot of d.
```

Alternatively, the array d could contain the responses for two-way data where the rows are levels of the first factor and the columns levels of the second. The function mway-dataset associates the factors with sucessive array dimensions, and assumes any remaining array dimensions contains variable values. The following code associates a case with three observations (one for the response, and one for each factor) with each element of the 2-D array d given above.

```
? (mway-dataset d :factors '("Temp" "Pressure")
                  :factor-levels '((20 30 40 50) ("L" "H"))
                  :variable "Response"
                  :name "Two-way Data")
      #2m((1 2) (3 4) (5 6) (7 8))      ; d
? (<- c0 (first (list-cases d)))      ;the first case
? (value-of c0 "Response")
     1          ;the response of the first case
? (identifier-of c0)
    ("20" "L")    ; c0 has Temp=20 and Pressure = L
? (identifier-of d)
     "Two-way Data"    ; d has an identifier
? (scatterplot :data d :x "Temp" :y "Response")
  ; displays a scatterplot of d.
```

ACKNOWLEDGMENTS

This work was partially supported by Enterprise Ireland Grant SC/97/605. The author thanks R.W. Oldford for helpful discussions.

[Received February 2000. Revised June 2000.]

REFERENCES

Becker, R. A., and Cleveland, W. S. (1987), "Brushing Scatterplots," *Technometrics*, 29, 127–142.

Craig, P., Haslett, J., Unwin, A., and Wills, G. (1989), "Moving Statistics: An Extension of Brushing for Spatial Data," in *Computer Science and Statistics: Proceedings of the 21st Symposium on the Interface*, pp. 170–174.

Eick, S. G., and Wills, G. J. (1995), "High-Interaction Graphics," *European Journal of Operational Research*, 445–459.

Hand, D. J., and Taylor, C. C. (1987), *Multivariable Analysis of Variance and Repeated Measures*, New York: Chapman and Hall.

Haslett, J., Bradley, R., Craig, P., Unwin, A., and Wills, G. (1991), "Dynamic Graphics for Exploring Spatial Data with Application to Locating Global and Local Anomalies," *The American Statistician*, 45, 234–242.

Hurley, C. (1991), "Applications of Constraints in Statistical Graphics," in *Proceedings of the Section on Statistical Graphics*, Alexandria, VA: American Statistical Association, pp. 877–881.

Hurley, C. B. (1993), "The Plot-Data Interface in Statistical Graphics," *Journal of Computational and Graphical Statistics*, 2, 365–379.

——— (1997), "Brushing Tools for Multilevel and Multiway Data," in *Proceedings of the Section on Statistical Graphics*, Alexandria, VA: American Statistical Association, pp. 15–24.

Hurley, C. B., and Haslett, J. (1995), "Graphical Exploration of Drillhole Data," in *Computing Science and Statistics: Proceedings on the Interface*, pp. 289–292.

Hurley, C. B., and Oldford, R. W. (1991), "A Software Model for Statistical Graphics," in *Computing and Graphics in Statistics, IMA Volumes in Mathematics and its Applications*, vol. 36, eds. A. Buja and P. Tukey, New York: Springer-Verlag.

McDonald, J. A. (1982), "Interactive Graphics for Data Analysis," unpublished PhD thesis, Stanford University.

Newton, C. M. (1978), "Graphics: From Alpha to Omega in Data Analysis," in *Graphical Representation of Multivariable Data*, ed. P. C. C. Wang, New York: Academic Press.

Noirhomme-Fraiture, M., and Rouard, M. (1997), "Zoom-Star: A Solution to Complex Statistical Object Representation," in *INTERACT 97 Proceedings*, pp. 100–101.

Steele, G. L. (1990), *Common Lisp, The Language* (2nd ed.), Bedford MA: Digital Press.

Theus, M., Hofmann, H., and Wilhelm, A. F. (1997), "Selection Sequences—Interactive Analysis of Massive Datasets," in *Computer Science and Statistics: Proceedings of the 29th Symposium on the Interface*, pp. 439–444.

Unwin, A. R., Hawkins, G., Hofmann, H., and Siegl, B. (1996), "Interactive Graphics for Datasets With Missing Values—MANET," *Journal of Computational and Graphical Statistics*, 5, 113–122.

Van der Heijden, P. G. M., and de Leeuw, J. (1985), "Correspondence Analysis Used Complementary to Log-Linear Analysis," *Psychometrika*, 50, 429–447.

Velleman, P. F. (1998), *Learning Data Analysis with DataDesk, Student Version 6.0*, Reading, MA: Addison-Wesley.

Ward, M. O. (1997), "Creating and Manipulating N-Dimensional Brushes," in *Proceedings of the Section on Statistical Graphics*, Alexandria, VA: American Statistical Association, pp. 6–14.

Wilhelm, A. F. X. (in press), " A Data Model for Interactive Satistical Graphics," in *Proceedings of the Section on Statistical Graphics*, Alexandria, VA.

Wills, G. (2000), "Natural Selection: Interactive Subset Creation," *Journal of Computational and Graphical Statistics*, 9, 544–557.