Chapter 13
The Grammar of Graphics

Leland Wilkinson

13.1 Introduction

The Grammar of Graphics, or GOG, denotes a system with seven orthogonal components (Wilkinson 1999). By *orthogonal*, we mean there are seven graphical component sets whose elements are aspects of the general system and that every combination of aspects in the product of all these sets is meaningful. This sense of the word orthogonality, a term used by computer designers to describe a combinatoric system of components or building blocks, is in some sense similar to the orthogonal factorial analysis of variance (ANOVA), where factors have levels and all possible combinations of levels exist in the ANOVA design. If we interpret each combination of features in a GOG system as a point in a network, then the world described by GOG is represented in a seven-dimensional rectangular lattice.

A consequence of the orthogonality of such a graphic system is a high degree of *expressiveness*. That is, it comprises a system that can produce a huge variety of graphical forms (chart types). In fact, it is claimed that virtually the entire corpus of known charts can be generated by this relatively parsimonious system, and perhaps a great number of meaningful but undiscovered chart types as well.

The second principal claim of GOG is that this system describes the meaning of what we do when we construct statistical graphs or charts. It is more than a taxonomy. It is a computational system based on the classical mathematics of representing functions and relations in Cartesian and other spaces. Because of this mathematical foundation, GOG specifications can serve as parsimonious and natural descriptions of famous statistical charts devised by Playfair, Minard, Jevons, Pearson, Bertin, Tukey, and other significant figures in the history of statistical graphics.

L. Wilkinson (✉)
SYSTAT Software Inc. Chicago, IL, USA
e-mail: Leland.Wilkinson@systat.com

J.E. Gentle et al. (eds.), *Handbook of Computational Statistics*, Springer Handbooks of Computational Statistics, DOI 10.1007/978-3-642-21551-3_13,
© Springer-Verlag Berlin Heidelberg 2012

13.1.1 Architecture

Figure 13.1 shows a *dataflow* diagram that contains the seven GOG components. This dataflow is a chain that describes the sequence of mappings needed to produce a statistical graphic from a set of data. The first component (Variables) maps data to an object called a *varset* (a set of variables). The next three components (Algebra, Scales, Statistics) are transformations on varsets. The next component (Geometry) maps a varset to a graph and the next (Coordinates) embeds a graph in a coordinate space. The last component (Aesthetics) maps a graph to a visible or perceivable display called a graphic.

The dataflow architecture implies that the subtasks needed to produce a graphic from data must be done in this specified order. Imposing an order would appear to be unnecessarily restrictive, but changes of this ordering can produce meaningless graphics. For example, if we compute certain statistics on variables (e.g., sums) before scaling them (e.g., log scales), we can produce statistically questionable results because the log of a sum is not the sum of the logs.

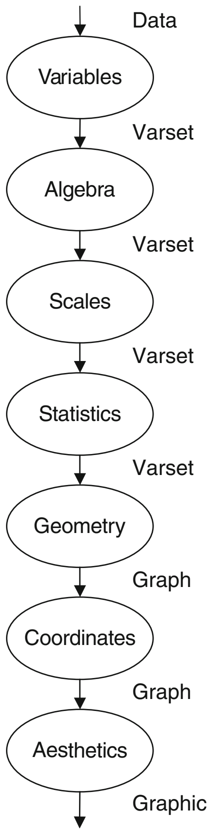

Fig. 13.1 Dataflow

The dataflow in Fig. 13.1 has many paths through it. We can choose different designs (factorial, nested, ...), scales (log, probability, ...), statistical methods (means, medians, modes, smoothers, ...), geometric objects (points, lines, bars, ...), coordinate systems (rectangular, polar, ...), and aesthetics (size, shape, color, ...). These paths reveal the richness of the system. The remainder of this article will summarize the seven GOG components, delineate these paths, and then briefly introduce sample applications.

13.2 Variables

We begin with data. We assume the data that we wish to graph are organized in one or more tables. The column(s) of each table represent a set of fields, each field containing a set of measurements or attributes. The row(s) of this table represent a set of logical records, each record containing the measurements of an object on each field. Usually, a relational database management system (RDBMS) produces such a table from organized queries specified in Structured Query Language (SQL) or another relational language. When we do not have data stored in a relational database (e.g., live data feeds), we need custom software to provide such a table using Extensible Markup Language (XML), Perl scripts, or other languages.

The first thing we have to do is convert such tables of data to something called a *varset*. A varset is a set of one or more variables. While a column of a table of data might superficially be considered to be a variable, there are differences. A variable is both more general (in regard to generalizability across samples) and more specific (in regard to data typing and other constraints) than a column of data. First, we define a variable, then a varset.

13.2.1 Variable

A statistical *variable* X is a mapping $f : O \to V$, which we consider as a triple:

$$X = [O, V, f]$$

The domain O is a set of objects.
The codomain V is a set of values.
The function f assigns to each element of O an element in V.

The image of O under f contains the *values* of X. We denote a possible value as x, where $x \in V$. We denote a value of an object as $X(o)$, where $o \in O$. A variable is *continuous* if V is an interval. A variable is *categorical* if V is a finite subset of the integers (or there exists an injective map from V to a finite subset of the integers).

Variables may be multidimensional. $\boldsymbol{X}$ is a p-dimensional variable made up of p one-dimensional variables:

$$
\begin{aligned}
\mathbf{X} &= (X_1, \ldots, X_p) \\
&= [O, V_i, f\,], \quad i = 1, \ldots, p \\
&= [O, \mathbf{V}, f\,]
\end{aligned}
$$

The element $\mathbf{x} = (x_1, \ldots, x_p)$, $\mathbf{x} \in \mathbf{V}$, is a p-dimensional value of $\mathbf{X}$. We use multidimensional variables in *multivariate analysis*.

A *random variable* X is a real–valued function defined on a sample space Ω:

$$X = [\Omega, \mathbb{R}, f]$$

Random variables may be multidimensional. In the elementary probability model, each element $\omega \in \Omega$ is associated with a probability function P. The range of P is the interval [0, 1], and $P(\Omega) = 1$. Because of the associated probability model, we can make probability statements about outcomes in the range of the random variable, such as:

$$P(X = 2) = P(\omega : \omega \in \Omega, X(\omega) = 2)$$

13.2.2 Varset

We call the triple

$$\mathrm{X} = [V, \widetilde{O}, f]$$

a varset. The word *varset* stands for *variable set*. If X is multidimensional, we use boldface **X**. A varset inverts the mapping used for variables. That is,
The domain V is a set of values.
The codomain $\widetilde{O}$ is a set of all possible ordered lists of objects.
The function f assigns to each element of V an element in $\widetilde{O}$.

We invert the mapping customarily used for variables in order to simplify the definitions of graphics algebra operations on varsets. In doing so, we also replace the variable's set of objects with the varset's set of ordered lists. We use lists in the codomain because it is possible for a value to be mapped to an object more than once (as in repeated measurements).

13.2.3 Converting a Table of Data to a Varset

To convert a table to a varset, we must define the varset's domain of values and range of objects and specify a reference function that maps each row in the table to an element in the varset.

We define the domain of values in each varset by identifying the measurements its values represent. If our measurements include weight, for example, we need to record the measurement units and the interval covered by the range of weights. If the domain is categorical, we need to decide whether there are categories not found in the data that should be included in the domain (refused to reply, don't know, missing, . . .). And we may need to identify categories found in the data that are not defined in the domain (mistakes, intransitivities, . . .). The varset's domain is a key component in the GOG system. It is used for axes, legends, and other components of a graphic. Because the actual data for a chart are only an instance of what we expect to see in a varset's domain, we let the domain control the structure of the chart.

We define the range of potential objects in each varset by identifying the class they represent. If the rows of our table represent measurements of a group of school children, for example, we may define the range to be school children, children in general, people, or (at the most abstract level) objects. Our decision about the level of generality may affect how a graphic will be titled, how legends are designed, and so on.

Finally, we devise a reference system by indexing objects in the domain. This is usually as simple as deriving a *caseID* from a table row index. Or, as is frequently done, we may choose the value of a key variable (e.g., Social Security Number) to create a unique index.

13.3 Algebra

Given one or more varsets, we now need to operate on them to produce combinations of variables. A typical scatterplot of a variable X against a variable Y, for example, is built from tuples (x_i, y_i) that are elements in a set product. We use graphics algebra on values stored in varsets to make these tuples. There are three binary operators in this algebra: *cross*, *nest*, and *blend*.

13.3.1 Operators

We will define these operators in set notation and illustrate them by using a table of real data. Table 13.1 shows 1980 and 2000 populations for selected world cities. During various periods in US history, it was fashionable to name towns and cities after their European and Asian counterparts. Sometimes this naming was driven by immigration, particularly in the colonial era (New Amsterdam, New York, New London). At other times, exotic names reflected a fascination with foreign travel and culture, particularly in the Midwest (Paris, Madrid). Using a dataset containing namesakes will help reveal some of the subtleties of graphics algebra.

Table 13.1 Cities and their Populations

Country	City	1980 Population	2000 Population
Japan	Tokyo	21900000	26400000
India	Mumbai	8067000	18100000
USA	New York	15600000	16600000
Nigeria	Lagos	4385000	13400000
USA	Los Angeles	9523000	13100000
Japan	Osaka	9990000	11000000
Philippines	Manila	5955000	10900000
France	Paris	8938000	9624000
Russia	Moscow	8136000	9321000
UK	London	7741000	7640000
Peru	Lima	4401000	7443000
USA	Chicago	6780000	6951000
Iraq	Bagdad	3354000	4797000
Canada	Toronto	3008000	4651000
Spain	Madrid	4296000	4072000
Germany	Berlin	3247000	3324000
Australia	Melbourne	2765000	3187000
USA	Melbourne	46536	71382
USA	Moscow	16513	21291
USA	Berlin	13084	10331
USA	Paris	9885	9077
USA	London	4002	5692
USA	Toronto	6934	5676
USA	Manila	2553	3055
USA	Lima	2025	2459
USA	Madrid	2281	2264
USA	Bagdad	2331	1578

We begin by assuming there are four varsets derived from this table: `country`, `city`, `pop1980`, and `pop2000` (we use lower case for varsets when they are denoted by names instead of single letters). Each varset has one column. The varsets resulting from algebraic operations will have one or more columns.

There is one set of objects for all four varsets: 27 cities. This may or may not be a subset of the domain for the four associated variables. If we wish to generalize analyses of this varset to other cities, then the set of possible objects in these varsets might be a subdomain of the set of all cities existing in 1980 and 2000. We might even consider this set of objects to be a subset of all possible cities in all of recorded history. While these issues might seem more the province of sampling and generalizability theory, they affect the design of a graphics system. Databases, for example, include facilities for *semantic integrity constraints* that ensure domain integrity in data tables. Data-based graphics systems share similar requirements.

There are sets of values for these varsets. The `country` varset has country names in the set of values comprising its domain. The definition of the domain of the varset depends on how we wish to use it. For example, we might include spellings of city names in languages other than English. We might also include country names not contained in this particular varset. Such definitions would affect whether we could add new cities to a database containing these data. For `pop1980` and `pop2000`, we would probably make the domain be the set of positive integers.

Cross (∗)

Cross joins the left argument with the right to produce a set of tuples stored in the multiple columns of the new varset:

$$\begin{array}{|c|}\hline \mathrm{x} \\ \hline \mathrm{y} \\ \hline \mathrm{z} \\ \hline \end{array} \; * \; \begin{array}{|c|}\hline \mathrm{a} \\ \hline \mathrm{a} \\ \hline \mathrm{b} \\ \hline \end{array} \; = \; \begin{array}{|c|c|}\hline \mathrm{x} & \mathrm{a} \\ \hline \mathrm{y} & \mathrm{a} \\ \hline \mathrm{z} & \mathrm{b} \\ \hline \end{array}$$

The resulting set of tuples is a subset of the product of the domains of the two varsets. The domain of a varset produced by a cross is the product of the separate domains.

One may think of a cross as a horizontal concatenation of the table representation of two varsets, assuming the rows of each varset are equivalent and in the same order. The following example shows a crossing of two varsets using set notation with simple integer keys for the objects:

$$\begin{aligned} \mathrm{A} &= [\{red, blue\}, \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\}, \{red \rightarrow \langle 1,4\rangle, blue \rightarrow \langle 2,3\rangle\}] \\ \mathrm{B} &= [[-10,10], \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\}, \{-10 \rightarrow \langle 1\rangle, 5 \rightarrow \langle 2,3\rangle, 10 \rightarrow \langle 4\rangle\}] \\ \mathrm{A} * \mathrm{B} &= [\{red, blue\} \times [-10,10], \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\}, \\ &\qquad \{(red,-10) \rightarrow \langle 1\rangle, (blue,5) \rightarrow \langle 2,3\rangle, (red,10) \rightarrow \langle 4\rangle\}] \end{aligned}$$

If we plotted A ∗ B in two dimensions with a point graph, we would see n points between −10 and 10 stacked vertically above one or both of the two color names.

Figure 13.2 shows a graphic based on the algebraic expression `city*pop2000`. We choose the convention of representing the first variable in an expression on the horizontal axis and the second on the vertical. We also restrict the domain of `pop2000` to be [0, 32000000].

Although most of the US namesake cities have smaller populations, it is not easy to discern them in the graphic. We can separate the US from the other cities by using a variable called `group` that we derive from the country names. Such a new variable is created easily in a database or statistical transformation language with an expression like

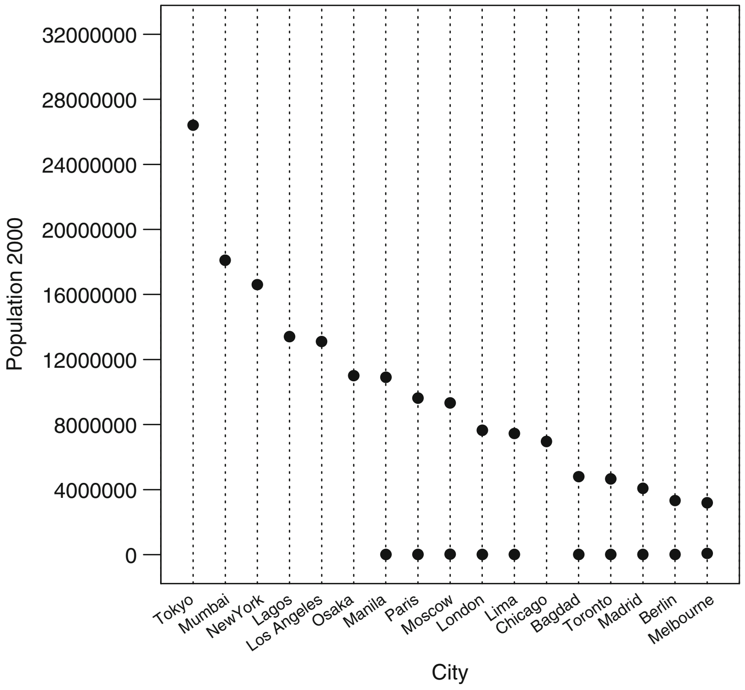

Fig. 13.2 `city * pop2000`

```
if (country == "USA") group = "USA";

else group = "World";
```

Figure 13.3 shows a graphic based on the three-dimensional algebraic expression `city * pop2000 * group`. This expression produces a varset with three columns. The first column is assigned to the horizontal axis, the second to the vertical, and the third to the horizontal axis again, which has the effect of splitting the frame into two frames. This general pattern of alternating horizontal and vertical roles for the columns of a varset provides a simple layout scheme for complex algebraic expressions. We may think of this as a generalization of the Trellis layout scheme (Becker et al. 1996). We could, of course, represent this same varset in a 3D plot projected into 2D, but the default system behavior is to prefer 2D with recursive partitioning. We will describe this in more detail in Sect. 13.9.

Chicago stands out as an anomaly in Fig. 13.3 because of its relatively large population. We might want to sort the cities in a different order for the left panel or eliminate cities not found in the US, but the algebraic expression won't let us do that. Because `group` is crossed with the other variables, there is only one domain of cities shared by both country groups. If we want to have different domains for the two panels, we need our next operator, *nest*.

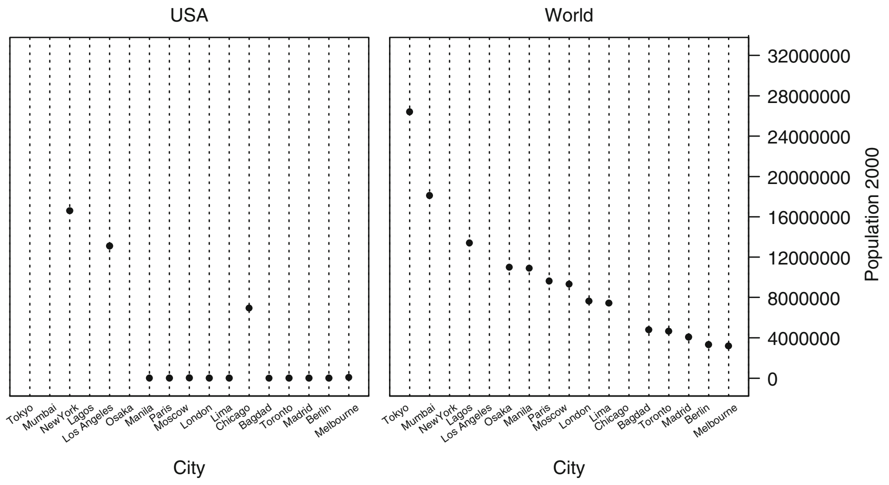

Fig. 13.3 `city * pop2000 * group`

Nest (/)

Nest partitions the left argument using the values in the right:

$$\begin{array}{|c|}\hline x \\ \hline y \\ \hline z \\ \hline \end{array} \; / \; \begin{array}{|c|}\hline a \\ \hline a \\ \hline b \\ \hline \end{array} \; = \; \begin{array}{|c|c|}\hline x & a \\ \hline y & a \\ \hline z & b \\ \hline \end{array}$$

Although it is not required in the definition, we assume the nesting varset on the right is categorical. If it were continuous (having interval domain) there would be an infinite number of partitions. We do require predefined nested domains. To construct a nested domain, three options are possible:

1. Data values – identify the minimal domain containing the data by enumerating unique data tuples.
2. Metadata – define the domain using external rules contained in a metadata resource or from known principles.
3. Data organization – identify nested domains using the predefined structure of a hierarchical database or OLAP cube.

The following example shows a nesting of two categorical variables:

$$\mathrm{A} = [\{ant, fly, bee\}, \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\}, \{ant \to \langle 1\rangle, fly \to \langle 2,3\rangle, bee \to \langle 4\rangle\}]$$
$$\mathrm{B} = [\{noun, verb\}, \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\}, \{noun \to \langle 1,2,4\rangle, verb \to \langle 3\rangle\}]$$
$$\mathrm{A/B} = [\{(ant, noun), (fly, noun), (fly, verb), (bee, noun)\}, \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\},$$

$$\{(ant, noun) \rightarrow \langle 1 \rangle, (fly, noun) \rightarrow \langle 2 \rangle, (fly, verb) \rightarrow \langle 3 \rangle (bee, noun) \rightarrow \langle 4 \rangle\}]$$

Nesting defines meaning conditionally. In this example, the meaning of *fly* is ambiguous unless we know whether it is a noun or a verb. Furthermore, there is no verb for *ant* or *bee* in the English language, so the domain of A/B does not include this combination.

If A is a continuous variable, then we have something like the following:

$$\begin{aligned}
\mathrm{A} &= [[0,10], \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\}, \{0 \rightarrow \langle 1 \rangle, 8 \rightarrow \langle 2 \rangle, 1.4 \rightarrow \langle 3 \rangle, 3 \rightarrow \langle 4 \rangle, 10 \\
&\qquad \rightarrow \langle 5, 6 \rangle\}] \\
\mathrm{B} &= [\{1,2\}, \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\}, \{1 \rightarrow \langle 1,2,3 \rangle, 2 \rightarrow \langle 4,5,6 \rangle\}] \\
\mathrm{A/B} &= [\{[0,8] \times \{1\}, [3,10] \times \{2\}\}, \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\}, \\
&\qquad \{(0,1) \rightarrow \langle 1 \rangle, (8,1) \rightarrow \langle 2 \rangle, (1.4,1) \rightarrow \langle 3 \rangle, (3,2) \rightarrow \langle 4 \rangle, (10,2) \rightarrow \langle 5,6 \rangle\}]
\end{aligned}$$

In this example, the elements of the nesting A/B result in intervals conditioned on the values of B. A represents 6 ratings (ranging from 0 to 10) of the behavior of patients by two psychiatrists. B represents the identity of the psychiatrist making each rating. The intervals $[0, 8]$ and $[3, 10]$ imply that psychiatrist 1 will not use a rating greater than 8 and psychiatrist 2 will not use a rating less than 3. Nesting in this case is based on the (realistic) assumption that the two psychiatrists assign numbers to their perceptions in a different manner. A rating of 2 by one psychiatrist cannot be compared to the same rating by the other, because of possible differences in location, scale, and even local nonlinearities. Much of psychometrics is concerned with the problem of equating ratings in this type of example so that nesting would not be needed, although it is not always possible to do so plausibly.

The name *nest* comes from design-of-experiments terminology. We often use the word *within* to describe its effect. For example, if we assess schools and teachers in a district, then *teachers within schools* specifies that teachers are nested within schools. Assuming each teacher in the district teaches at only one school, we would conclude that if our data contain two teachers with the same name at different schools, they are different people. Those familiar with experimental design may recognize that the expression A/B is equivalent to the notation A(B) in a design specification. Both expressions mean *A is nested within B*. Statisticians' customary use of parentheses to denote nesting conceals the fact that nesting involves an operator, however. Because nesting is distributive over blending, we have made this operator explicit and retained the conventional mathematical use of parentheses in an algebra.

Figure 13.4 shows a graphic based on the algebraic expression `city/group` $*$ `pop2000`. The horizontal axis in each panel now shows a different set of cities: one for the USA and one for the rest of the world. This graphic differs from the one in Fig. 13.3 not only because the axes *look* different, but also because the meanings

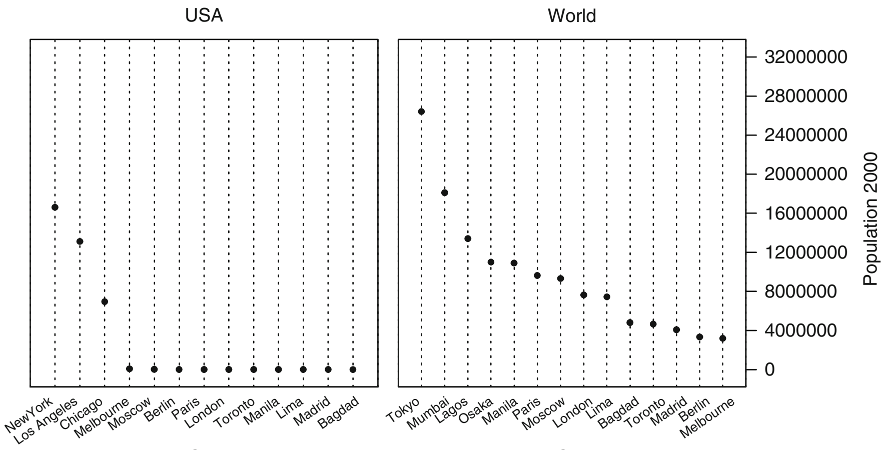

Fig. 13.4 `city/group * pop2000`

of the cities in each panels *are* different. For example, the city named Paris appears twice in both figures. In Fig. 13.3, on the one hand, we assume the name Paris in the left panel is comparable to the name Paris in the right. That is, it refers to a common name (Paris) occurring in two different contexts. In Fig. 13.4, on the other hand, we assume the name Paris references two different cities. They happen to have the same name, but are not equivalent. Such distinctions are critical, but often subtle.

Blend (+)

Blend produces a union of varsets:

$$\begin{array}{|c|}\hline \mathrm{x}\\ \hline \mathrm{y}\\ \hline \mathrm{z}\\ \hline\end{array} + \begin{array}{|c|}\hline \mathrm{a}\\ \hline \mathrm{a}\\ \hline \mathrm{b}\\ \hline\end{array} = \begin{array}{|c|}\hline \mathrm{x}\\ \hline \mathrm{y}\\ \hline \mathrm{z}\\ \hline \mathrm{a}\\ \hline \mathrm{a}\\ \hline \mathrm{b}\\ \hline\end{array}$$

Blend is defined only if the order of the tuples (number of columns) in the left and right varsets is the same. Furthermore, we should restrict blend to varsets with composable domains, even though we do not need this restriction for the operation to be defined. It would make little sense to blend Age and Weight, much less Name and Height.

In vernacular, we often use the conjunction *and* to signify that two sets are blended into one (although the word *or* would be more appropriate technically). For example, if we measure diastolic and systolic blood pressure among patients in various treatment conditions and we want to see blood pressure plotted on a common axis, we can plot diastolic *and* systolic against treatment. The following example shows a blending of two varsets, using integers for keys:

$$\begin{aligned}
\mathrm{A} &= [[0,120],\{\langle\cdot\rangle,\langle\cdot,\cdot\rangle,\ldots\},\{0\to\langle 1\rangle,120\to\langle 2\rangle,90\to\langle 3,4\rangle\}]\\
\mathrm{B} &= [[10,200],\{\langle\cdot\rangle,\langle\cdot,\cdot\rangle,\ldots\},\{10\to\langle 1\rangle,200\to\langle 2,3\rangle,90\to\langle 4\rangle\}]\\
\mathrm{A}+\mathrm{B} &= [[0,200],\{\langle\cdot\rangle,\langle\cdot,\cdot\rangle,\ldots\},\\
&\quad \{0\to\langle 1\rangle,10\to\langle 1\rangle,120\to\langle 2\rangle,90\to\langle 3,4,4\rangle,200\to\langle 2,3\rangle\}]
\end{aligned}$$

Figure 13.5 shows an example of a blend using our cities data. The graphic is based on the algebraic expression `city`$*$(`pop1980`$+$`pop2000`). The horizontal axis represents the cities and the vertical axis represents the two repeated population measures. We have included different symbol types and a legend to distinguish the measures. We will see later how shape aesthetics are used to create this distinction.

As with the earlier graphics, we see that it is difficult to distinguish US and world cities. Figure 13.6 makes the distinction clear by splitting the horizontal axis into two nested subgroups. The graphic is based on the algebraic expression (`city/group`)$*$(`pop1980`$+$`pop2000`). Once again, the vertical axis represents

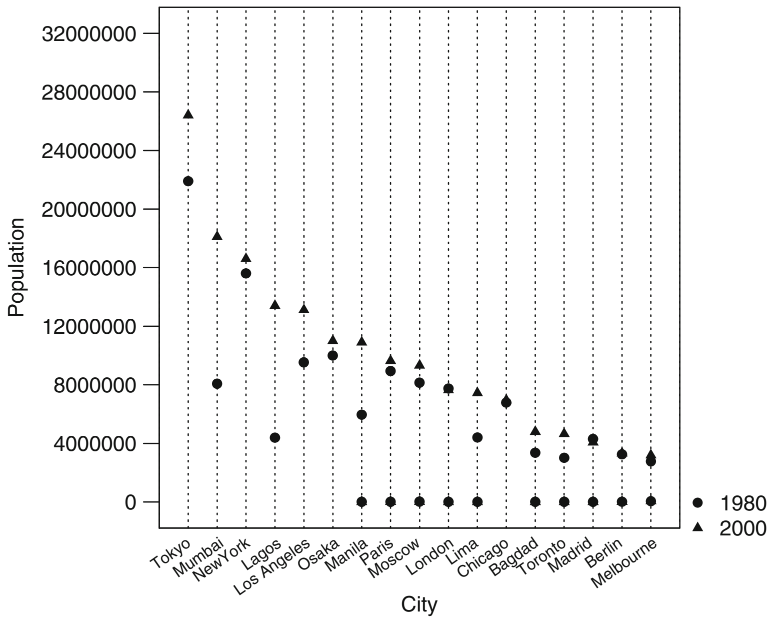

Fig. 13.5 `city` $*$ (`pop1980` $+$ `pop2000`)

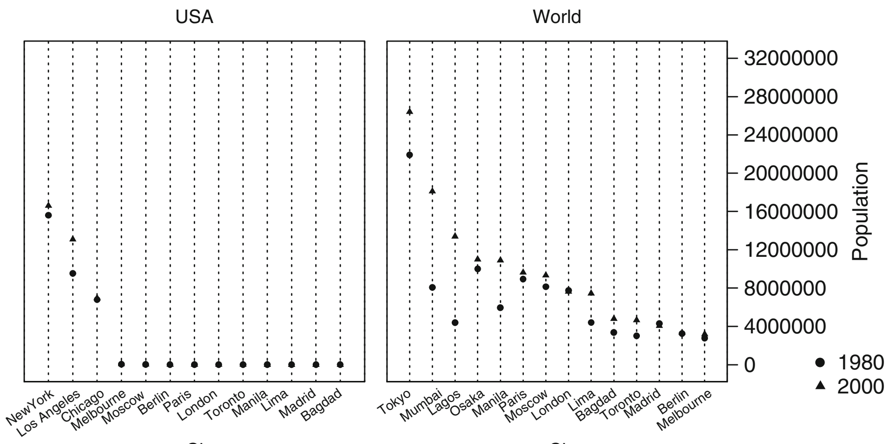

Fig. 13.6 (`city/group`) $*$ (`pop1980` $+$ `pop2000`)

the two repeated population measures blended on a single dimension. We see most of the cities gaining population between 1980 and 2000.

13.3.2 Rules

The following rules are derivable from the definitions of the graphics operators:

Associativity

$$(X * Y) * Z = X * (Y * Z)$$
$$(X/Y)/Z = X/(Y/Z)$$
$$(X + Y) + Z = X + (Y + Z)$$

Distributivity

$$X * (Y + Z) = X * Y + X * Z$$
$$X/(Y + Z) = X/Y + X/Z$$
$$(X + Y) * Z = X * Z + Y * Z$$
$$(X + Y)/Z = X/Z + Y/Z$$

Commutativity

$$X + Y = Y + X$$

Identity

The identity element for blend is an empty list. Cross and nest have no identity.

Precedence

Nest takes precedence over cross and blend. Cross takes precedence over blend. This hierarchical order may be altered through the use of parentheses.

13.3.3 SQL Equivalences

Given a table X and a table Y in a database, we can use SQL to perform the operations in chart algebra. This section outlines how to do this.

Cross

Cross can be accomplished by a *cross join*:

```
SELECT a.*, b.*
FROM X a, Y b;
```

Of course, this operation is inefficient and requires optimization. Alternatively, one can do a simple *join* and generate the missing tuples with an iterator when needed.

Nest

Nest can be accomplished through a *nest* operation. The nest operator requires that the database allow tables as primitives, either as relation-valued attributes (Date and Darwen 1992) or as nested tables (Makinouchi 1977), (Abiteboul et al. 1989).

Alternatively, we can accumulate the subset of tuples in a nest operation with a simple *join*:

```
SELECT a.*, b.*
FROM X a, Y b
WHERE a.rowid = b.rowid;
```

If we use this latter method, we must distinguish the entries used for tags and those used for values.

Blend

Blend is performed through `UNION`. If `UNION all` is not available, we can concatenate key columns to be sure that all rows appear in the result set.

```
SELECT * from X
UNION all
SELECT * from Y;
```

Composition and Optimization

SQL statements can be composed by using the grammar for chart algebra. Compound statements can then be submitted for optimization and execution by a database compiler. Alternatively, pre-optimization can be performed on the chart algebra parse tree object and the optimized parse tree used to generate SQL. Secondary optimization can then be performed by the database compiler.

13.3.4 Related Algebras

Research on algebras that could be used for displaying data has occurred in many fields. We will summarize these approaches in separate sections.

Table Algebras

The US Bureau of Labor Statistics pioneered a language for laying out tables (Mendelssohn 1974). While not a formal algebra, this Table Production Language (TPL) contained many of the elements needed to assemble complex tables. Gyssens et al. (1996) outlined an algebra for displaying relational data; this algebra closely followed TPL, although the latter is not referenced. Wilkinson (1996) presented an algebra for structuring tables and graphics.

Design Algebras

Nelder (1965) and Wilkinson and Rogers (1973) developed a language for implementing factorial and nested experimental designs, following Fisher (1935). The operators in this language are similar to the cross and nest operators in the present paper. The algebraic design language was implemented in the GENSTAT statistical computer program for generating and analyzing general linear statistical models.

Query Algebras

Pedersen et al. (2002) described an algebra for querying OLAP cubes. The result sets from their algebraic expressions could be used for graphic displays. Agrawal et al. (1997) used a similar algebra for statistical modeling of data contained in a cube.

Display Algebras

Mackinlay (1986) developed an algebra for querying relational databases and generating charts. His general goal was to develop an intelligent system that could offer graphical responses to verbal or structural queries. Roth et al. (1994) followed a similar strategy in developing graphical representations of relational data. They extended Mackinlay's and others' ideas by using concepts from computational geometry.

13.3.5 Algebra XML

A parse tree for a given algebraic expression maps nicely to XML in a manner similar to the way MathML (http://www.w3.org/TR/MathML2/) is defined. We have developed an implementation, called VizML (http://xml.spss.com/ visualization), that includes not only the algebraic components of the specification, but also the aesthetic and geometric aspects. Ultimately, VizML makes it possible to embed chart algebraic operations in a database.

13.4 Scales

Before we compute summaries (totals, means, smoothers, ...) and represent these summaries using geometric objects (points, lines, ...), we must scale our varsets. In producing most common charts, we do not notice this step. When we implement log scales, however, we notice it immediately. We must log our data before averaging logs. Even if we do not compute nonlinear transformations, however, we need to specify a measurement model.

The measurement model determines how distance in a frame region relates to the ranges of the variables defining that region. Measurement models are reflected in the axes, scales, legends, and other annotations that demarcate a chart's frame. Measurement models determine how values are represented (e.g., as categories or magnitudes) and what the units of measurement are.

13.4.1 Axiomatic Measurement

In constructing scales for statistical charts, it helps to know something about the function used to assign values to objects. Stevens (1946) developed a taxonomy of such functions based on axioms of measurement. Stevens identified four basic scale types: *nominal*, *ordinal*, *interval*, and *ratio*.

To define a nominal scale, we assume there exists at least one equivalence class together with a binary equivalence relation ($\sim$) that can be applied to objects in the domain (e.g., the class of this object is the *same* as the class of that object). For a domain of objects D and a set of values $X(d), d \in D$, we say that a scale is nominal if

$$d_i \sim d_j \iff X(d_i) = X(d_j), \ \forall\, d_i, d_j \in D\,.$$

To define an ordinal scale, we assume there exists a binary total order relation ($\succ$) that can be applied to objects in the domain (e.g., this stone is *heavier than* that stone). We then say that a scale is ordinal if

$$d_i \succ d_j \iff X(d_i) > X(d_j), \ \forall\, d_i, d_j \in D\,.$$

To define an interval scale, we assume there exists a symmetric concatenation operation ($\oplus$) that can be applied to objects in the domain (e.g., the length of this stick *appended to* the length of that stick). We then say that a scale is interval if

$$d_i \oplus d_j \sim d_k \iff X(d_i) + X(d_j) = X(d_k), \ \forall\, d_i, d_j, d_k \in D\,.$$

To define a ratio scale, we assume there exists a magnitude comparison operation ($\oslash$) that can be applied to objects in the domain (e.g., the *ratio* of the brightness of this patch to the the brightness of that patch). We then say that a scale is ratio if

$$d_i \oslash d_j \sim d_k \iff X(d_i)/X(d_j) = X(d_k), \ \forall\, d_i, d_j, d_k \in D\,.$$

Axiomatic scale theory is often invoked by practitioners of data mining and graphics, but it is not sufficient for determining scales on statistical graphics produced by chart algebra. The blend operation, for example, allows us to union values on different variables. We can require that blended variables share the same measurement level (e.g., diastolic and systolic blood pressure), but this will not always produce a meaningful scale. For example, we will have a meaningless composite scale if we attempt to blend height and weight, both presumably ratio variables. We need a different level of detail so that we can restrict the blend operation more appropriately.

13.4.2 Unit Measurement

An alternative scale classification is based on units of measurement. Unit scales permit standardization and conversion of metrics. In particular, the International System of Units (SI) (Taylor 1997) unifies measurement under transformation rules encapsulated in a set of base classes. These classes are *length*, *mass*, *time*, *electric current*, *temperature*, *amount of substance*, and *luminous intensity*. Within the base classes, there are default metrics (meter, kilogram, second, etc.) and methods for

Table 13.2 Typical unit measurements

Length	Mass	Temperature	Time	Volume	Currency
meter	kilogram	kelvin	second	liter	dollar
point	gram	rankine	minute	teaspoon	euro
pica	grain	celsius	hour	tablespoon	pound
inch	slug	fahrenheit	day	cup	yen
foot	carat		week	pint	rupee
yard			month	quart	dinar
mile			quarter	gallon	
furlong			year	bushel	
fathom			century	barrel	

converting from one metric to another. From these base classes, a set of derived classes yields measurements such as *area*, *volume*, *pressure*, *energy*, *capacitance*, *density*, *power*, and *force*. Table 13.2 shows some examples of several SI base classes, derived classes, and an example of an economic base class that is not in SI. The *currency* class is time dependent, since daily exchange rates determine conversion rules and an inflation adjustment method varies with time.

Most of the measurements in the SI system fit within the interval and ratio levels of Stevens' system. There are other scales fitting Stevens' system that are not classified within the SI system. These involve units such as *category* (state, province, country, color, species), *order* (rank, index), and *measure* (probability, proportion, percent). And there are additional scales that are in neither the Stevens nor the SI system, such as *partial order*.

For our purposes, unit measurement gives us the level of detail needed to construct a numerical or categorical scale. We consider unit measurement a form of strong typing that enables reasonable default behavior. Because of the class structure and conversion methods, we can handle labels and relations for derived quantities such as miles-per-gallon, gallons-per-mile, and liters-per-kilometer. Furthermore, automatic unit conversion within base and derived classes allows meaningful blends. As with domain check overrides in a database (Date 1990), we allow explicit type overrides for the blend operation.

13.4.3 Transformations

We frequently compute transformations of variables in constructing graphics. Sometimes, we employ statistical transformations to achieve normality so that we can apply classical statistical methods such as linear regression. Other times, we transform to reveal local detail in a graphic. It helps to apply a log transform, for example, to stretch out small data values in a display. We might do this even when not applying statistical models.

These types of transformations fall within the scale stage of the grammar of graphics system. Because GOG encapsulates variable transformations within this

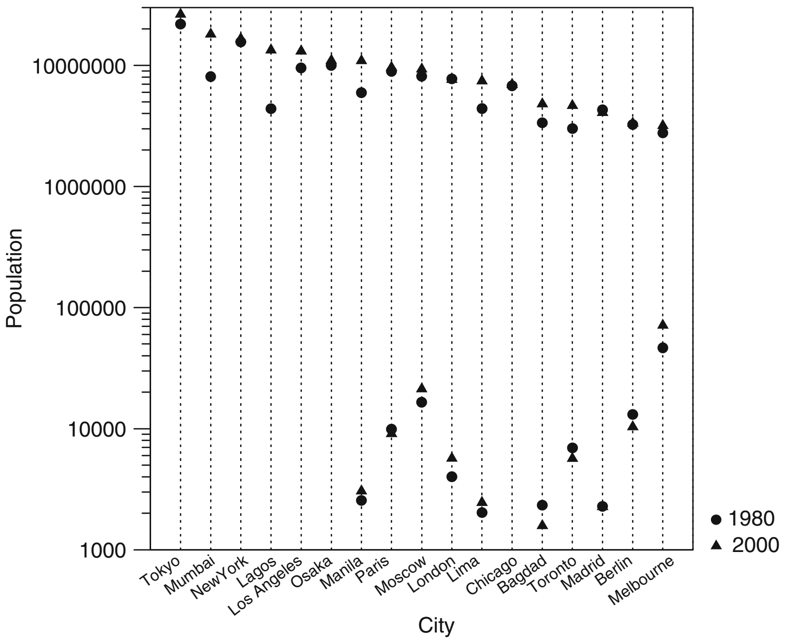

Fig. 13.7 `city` $*$ (`pop1980` $+$ `pop2000`), *ylog*

stage, it accomplishes two tasks at the same time: 1) the values of the variables are transformed prior to analysis and display, and 2) nice scale values for axes and legends are computed based on the transformation. Figure 13.7 shows an example of this process for the city data. In order to highlight population changes in small cities, we represent the populations on a log scale. The algebraic expression is the same as in Fig. 13.5: `city` $*$ (`pop1980` $+$ `pop2000`). Now we see that most of the cities gained population between 1980 and 2000 but half the US namesakes lost population.

13.5 Statistics

Visualization and statistics are inseparable. Statisticians have known this for a long time, but non-statisticians in the visualization field have largely ignored the role of statistics in charts, maps, and graphics. Non-statisticians often believe that visualization follows data analysis. We aggregate, summarize, model, and *then* display the results. In this view, visualization is the last step in the chain and statistics is the first.

In GOG, statistics falls in the middle of the chain. The consequence of this architecture is that statistical methods are an integral part of the system. We can construct dynamic graphics, in which statistical methods can be changed (for exploratory purposes) without altering any other part of the specification and without restructuring the data. By including statistical methods in its architecture, GOG also makes plain the independence of statistical methods and geometric displays. There is no necessary connection between regression methods and curves or between confidence intervals and error bars or between histogram binning and histograms.

In GOG, the statistics component receives a varset, computes various statistics, and outputs another varset. In the simplest case, the statistical method is an identity. We do this for scatterplots. Data points are input and the same data points are output. In other cases, such as histogram binning, a varset with n rows is input and and a varset with k rows is output, where k is the number of bins ($k < n$). With smoothers (regression or interpolation), a varset with n rows is input and and a varset with k rows is output, where k is the number of knots in a mesh over which smoothed values are computed. With point summaries (means, medians, ...), a varset with n rows is input and a varset with one row is output. With regions (confidence intervals, ranges, ...), a varset with n rows is input and and a varset with two rows is output.

Understanding how the statistics component works reveals an important reason for mapping values to cases in a varset rather than the other way around. If

$$\mathrm{A} = [\mathbb{R}, \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\}, \{1.5 \rightarrow \langle 1\rangle, 2.7 \rightarrow \langle 2\rangle, 1.8 \rightarrow \langle 3\rangle\}]\ ,$$

then

$$\mathrm{mean(A)} = [\mathbb{R}, \{\langle\cdot\rangle, \langle\cdot,\cdot\rangle, \ldots\}, \{2.0 \rightarrow \langle 1, 2, 3\rangle\}]\,.$$

Notice that the list of caseIDs that is produced by *mean*() is contained in the one row of the output varset. We do not lose case information in this mapping, the way we do when we compute results from an ordinary SQL query on a database or when we compute a data cube for an OLAP or when we pre-summarize data to produce a simple graphic. This aspect of GOG is important for dynamic graphics systems that allow *drill-down* or queries regarding metadata when the user hovers over a particular graphic element.

Figure 13.8 shows an application of a statistical method to the city data. We linearly regress 2000 population on 1980 population to see if population growth is proportional to city size. On log-log scales, the estimated values fall on a line whose slope is greater than 1, suggesting that larger cities grow faster than smaller. Ordinarily, we would draw a line to represent the regression and we would include the data points as well. We would also note that Lagos grew at an unusual rate (with a Studentized residual of 3.4). Nevertheless, our main point is to show that the statistical regression produces data points that are exchangeable with the raw data insofar as the entire GOG system is concerned. How we choose to represent the regressed values graphically is the subject of the next section.

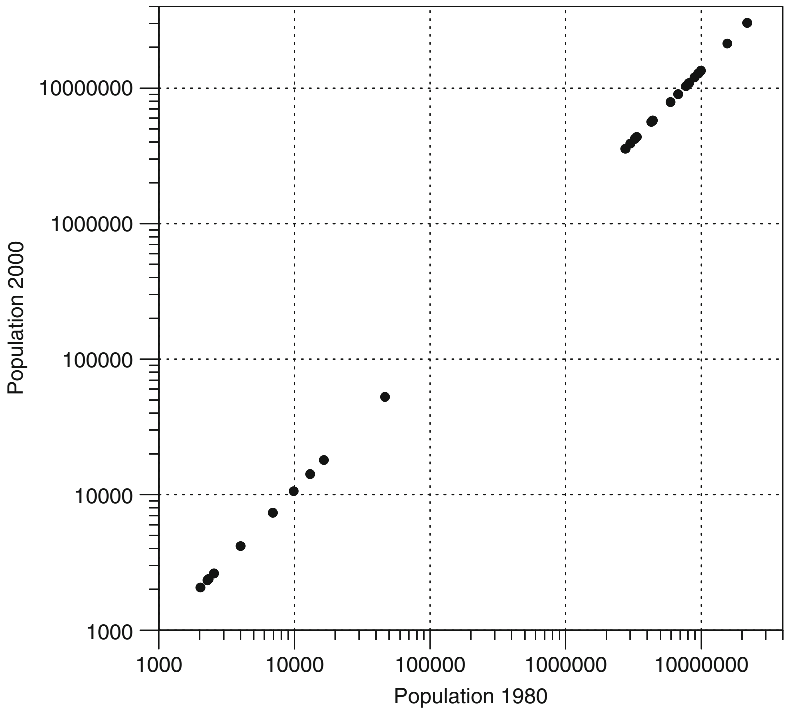

Fig. 13.8 `pop1980` $*$ *estimate*(`pop2000`), *xlog*, *ylog*

13.6 Geometry

GOG presumes no connection between a statistical method and a geometric representation. Histogram bins need not be represented by histograms. Tukey schematic plots (his original word for box plots) need not be represented by boxes and whiskers. Regressions need not be represented by lines or curves. Separating geometry from data (and from other graphical aspects such as coordinate systems) is what gives GOG its expressive power. We choose geometric representation objects independently of statistical methods, coordinate systems, or aesthetic attributes.

As Fig. 13.1 indicates, the geometry component of GOG receives a varset and outputs a geometric graph. A geometric graph is a subset of $\mathbb{R}^n$. For our purposes, we will be concerned with geometric graphs for which $1 \leq n \leq 3$. Geometric graphs are enclosed in bounded regions:

$$B^n \subset [a_1, b_1] \times \ldots \times [a_n, b_n]$$

These intervals define the edges of a bounding box or region in n-dimensional space. There are two reasons we need bounded regions. First, in order to define certain useful geometric graphs, we need concepts like the *end* of a line or the *edge* of a rectangle. Second, we want to save ink and electricity. We don't want to take forever to compute and draw a line.

Geometric graphs are produced by graphing functions $F : B^n \rightarrow \mathbb{R}^n$ that have geometric names like *line*() or *tile*(). A geometric graph is the image of F. And a graphic, as used in the title of this chapter, is the image of a graph under one or more aesthetic functions. Geometric graphs are not visible. As Bertin (1967) points out, visible elements have features not present in their geometric counterparts.

Figures 13.9 and 13.10 illustrate the exchangeability of geometry and statistical methods. The graphics are based on UN data involving 1990 estimates of female life expectancy and birth rates for selected world countries. Figure 13.9 shows four different geometric graphs – *point*, *line*, *area*, and *bar* – used to represent a confidence interval on a linear regression. Figure 13.10 shows one geometric

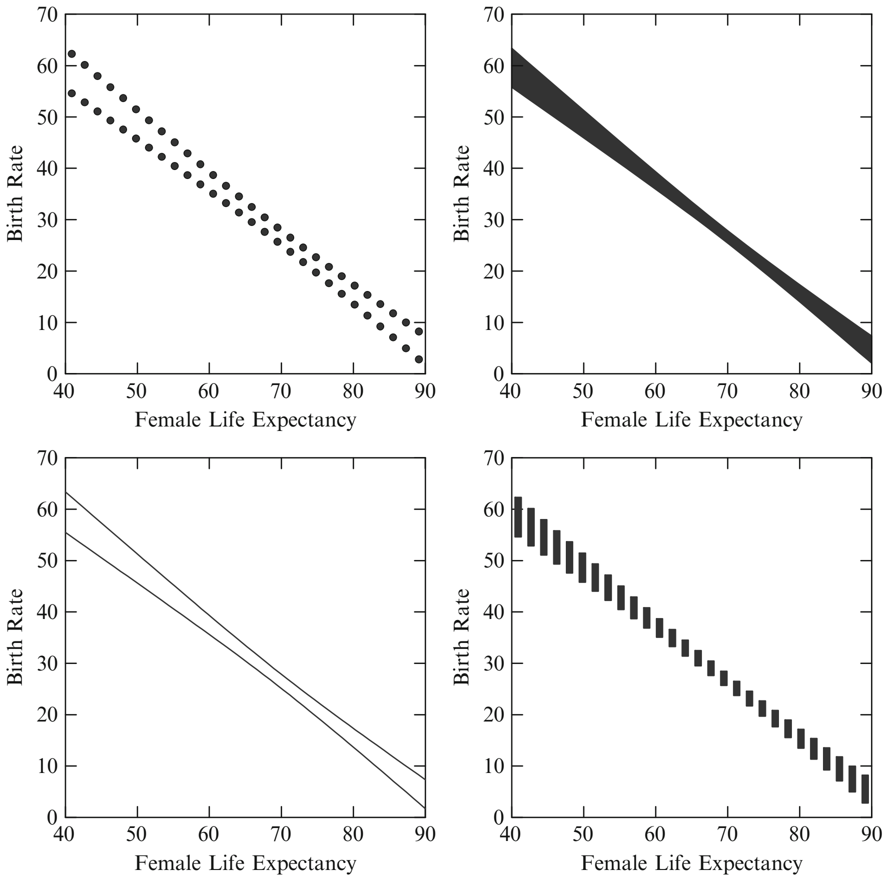

Fig. 13.9 Different graph types, same statistical method

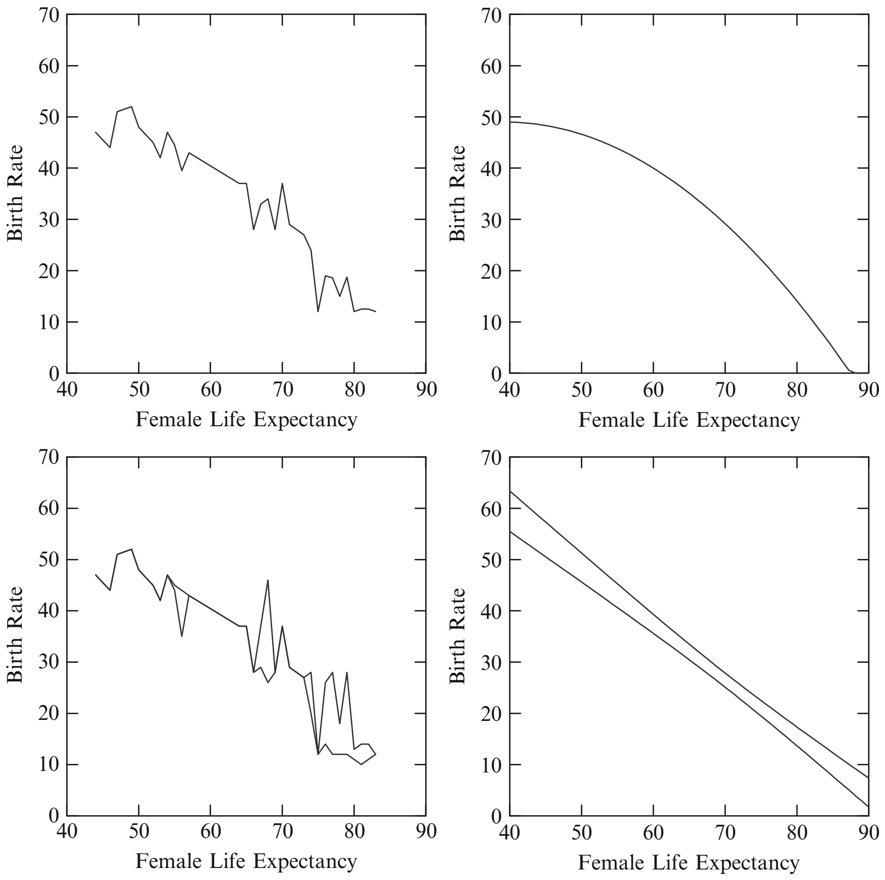

Fig. 13.10 Different statistical methods, same graph type

graph used to represent four different statistical methods – local mean, local range, quadratic regression, and linear regression confidence interval.

This exchangeability produces a rich set of graphic forms with a relatively small number of geometric graphs. Table 13.3 contains these graphing methods. The *point*() graphing function produces a geometric point, which is an n-tuple. This function can also produce a finite set of points, called a *multipoint* or a *point cloud*. The set of points produced by *point*() is called a *point* graph.

The *line*() graphing function function is a bit more complicated. Let B^m be a bounded region in $\mathbb{R}^m$. Consider the function $F : B^m \to \mathbb{R}^n$, where $n = m + 1$, with the following additional properties:

1. The image of F is bounded, and
2. $F(x) = (\mathbf{v}, f(\mathbf{v}))$, where $f : B^m \to \mathbb{R}$ and $\mathbf{v} = (x_1, \ldots, x_m) \in B^m$.

Table 13.3 Geometric graphs

Relations	Summaries	Partitions	Networks
Point	*Schema*	*Tile*	*Path*
Line (*surface*)		*Contour*	*Link*
Area (*volume*)			
Bar (*interval*)			
Histobar			

If $m = 1$, this function maps an interval to a functional curve on a bounded plane. And if $m = 2$, it maps a bounded region to a functional surface in a bounded 3D space. The *line*() graphing function produces these graphs. Like *point*(), *line*() can produce a finite set of lines. A set of lines is called a multiline. We need this capability for representing multimodal smoothers, confidence intervals on regression lines, and other multifunctional lines.

The *area*() graphing function produces a graph containing all points within the region under the *line* graph. The *bar*() graphing function produces a set of closed intervals. An interval has two ends. Ordinarily, however, bars are used to denote a single value through the location of one end. The other end is anchored at a common reference point (usually zero). The *histobar*() graphing function produces a histogram element. This element behaves like a bar except a value maps to the area of a histobar rather than to its extent. Also, histobars are glued to each other. They cover an interval or region, unlike bars.

A *schema* is a diagram that includes both general and particular features in order to represent a distribution. We have taken this usage from Tukey (1977), who invented the schematic plot, which has come to be known as the box plot because of its physical appearance. The *schema*() graphing function produces a collection of one or more points and intervals.

The *tile*() graphing function tiles a surface or space. A *tile* graph covers and partitions the bounded region defined by a frame; there can be no gaps or overlaps between tiles. The Latinate *tessellation* (for tiling) is often used to describe the appearance of the tile graphic.

A *contour*() graphing function produces contours, or level curves. A *contour* graph is used frequently in weather and topographic maps. Contours can be used to delineate any continuous surface.

The *network*() graphing function joins points with line segments (edges). Networks are representations that resemble the edges in diagrams of theoretic graphs. Although networks join points, a point graph is not needed in a frame in order for a network graphic to be visible.

Finally, the *path*() graphing function produces a *path* that connects points such that each point touches no more than two line segments. Thus, a path visits every point in a collection of points only once. If a path is closed (every point touches two line segments), we call it a circuit. Paths often look like lines. There are several important differences between the two, however. First, lines are functional; there can be only one point on a line for any value in the domain. Paths may loop, zigzag,

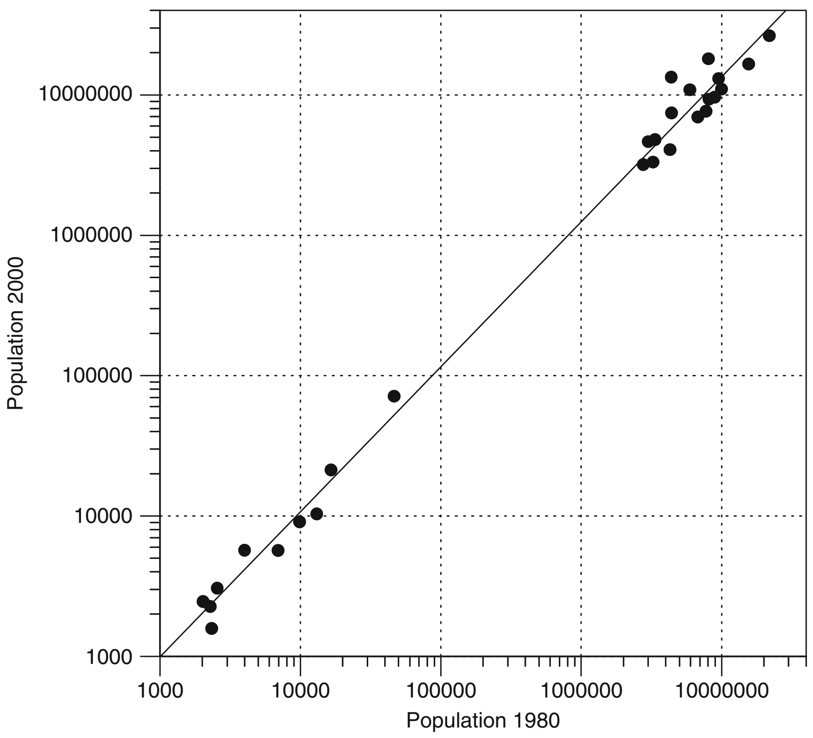

Fig. 13.11 `pop1980` $*$ {`pop2000`, *estimate*(`pop2000`)}, *xlog*, *ylog*, {*point*, *line*}

and even cross themselves inside a frame. Second, paths consist of segments that correspond to edges, or links between nodes. This means that a variable may be used to determine an attribute of every segment of a path.

Figure 13.11 contains two geometric objects for representing the regression we computed in Fig. 13.8. We use a *point* for representing the data and a *line* for representing the regression line.

13.7 Coordinates

The most popular types of charts employ Cartesian coordinates. The same real tuples in the graphs underlying these graphics can be embedded in many other coordinate systems, however. There are many reasons for displaying graphics in different coordinate systems. One reason is to simplify. For example, coordinate transformations can change some curvilinear graphics to linear. Another reason is to reshape graphics so that important variation or covariation is more salient or accurately perceived. For example, a pie chart is generally better for judging proportions of wholes than is a bar chart (Simkin and Hastie 1987). Yet another

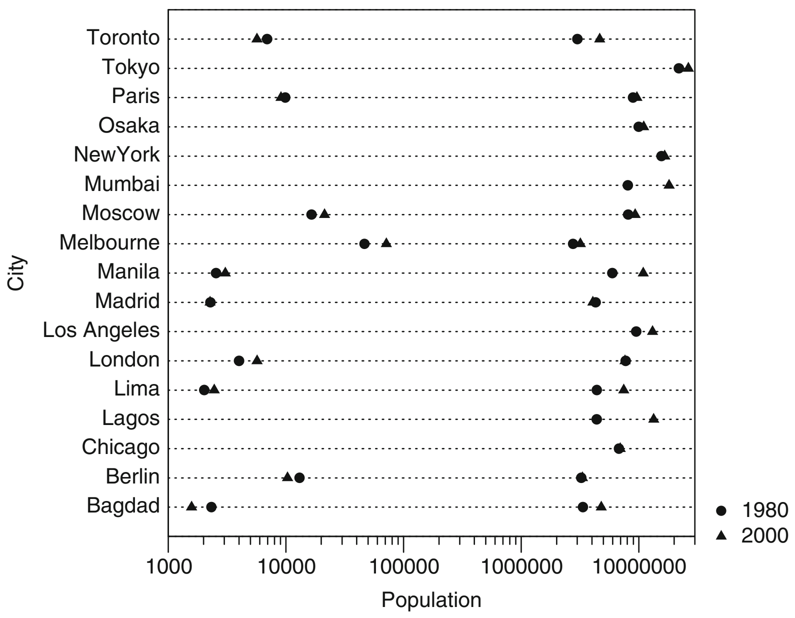

Fig. 13.12 *transpose*(`city` ∗ (`pop1980` + `pop2000`)), *ylog*

reason is to match the form of a graphic to theory or reality. For example, we might map a variable to the left-closed and right-open interval $[0,1)$ on a line or to the interval $[0,2\pi)$ on the circumference of a circle. If our variable measures defects within a track of a computer disk drive in terms of rotational angle, it is usually better to stay within the domain of a circle for our graphic. Another reason is to make detail visible. For example, we may have a cloud with many points in a local region. Viewing those points may be facilitated by zooming in (enlarging a region of the graphic) or smoothly distorting the local area so that the points are more separated in the local region.

Wilkinson (1999) contains many examples of ordinary charts rendered in different coordinate systems. A simple example suffices for the data in this chapter. Figure 13.12 shows a transposed version of Fig. 13.7. The result of this coordinate transformation (a rotation composed with a reflection) is to make the city names more readable.

13.8 Aesthetics

The term aesthetics derives from a Greek word that means perception. The derivative modern meanings of beauty, taste, and artistic criteria arose in the eighteenth century. We have chosen the name *aesthetics* to describe the class of functions that turn theoretical graphs into perceivable graphics because of its original connotations and because the modern word *perception* is subjective rather than objective;

Table 13.4 Aesthetics

Form	Surface	Motion	Sound	Text
Position	*Color*	*Direction*	*Tone*	*Label*
Size	*Texture*	*Speed*	*Volume*	
Shape	*Blur*	*Acceleration*	*Rhythm*	
Rotation	*Transparency*		*Voice*	

perception refers to the perceiver rather than the object. Aesthetics turn graphs into graphics so that they are perceivable, but they are not the perceptions themselves. A modern psychologist would most likely call aesthetics in this sense stimuli, aspects, or features, but these words are less germane to our purpose.

Table 13.4 summarizes these aesthetic attributes. We have grouped these attributes in five categories: *form*, *surface*, *motion*, *sound*, and *text*. This is not intended to be an exhaustive list; other attributes, such as *odor*, can be devised. The *color* aesthetic has three components: *hue*, *brightness*, and *saturation* (other color components are possible). The *texture* aesthetic includes components of *pattern*, *granularity*, and *orientation*.

Seven of these attributes are derived from the *visual variables* of Bertin (1967): *position* (position), *size* (taille), *shape* (forme), *orientation* (orientation), *brightness* (valeur), *color* (couleur), and *granularity* (grain). Bertin's *grain* is often translated as *texture*, but he really means granularity (as in the granularity of a photograph). Granularity in this sense is also related to the spatial frequency of a texture.

These aesthetic attributes do not represent the aspects of perception investigated by psychologists. This lack of fit often underlies the difficulty graphic designers and computer specialists have in understanding psychological research relevant to graphics and the corresponding difficulty psychologists have with questions asked by designers. Furthermore, these attributes are not ones customarily used in computer graphics to create realistic scenes. They are not even sufficient for a semblance of realism. Notice, for example, that *pattern*, *granularity*, and *orientation* are not sufficient for representing most of the textures needed for representing real objects. Instead, these attributes are chosen in a tradeoff between the psychological dimensions they elicit and the types of routines that can be implemented in a rendering system. Specifically:

- An attribute must be capable of representing both continuous and categorical variables.
- When representing a continuous variable, an attribute must vary primarily on one psychophysical dimension. In order to use multidimensional attributes such as color, we must scale them on a single dimension such as hue or brightness, or compute linear or nonlinear combinations of these components to create a unidimensional scale.
- An attribute does not imply a linear perceptual scale. In fact, few aesthetic attributes scale linearly. Some attributes such as hue scale along curvilinear segments in two- or three-dimensional space. All linear scales are unidimensional but not all unidimensional scales are linear.

- A perceiver must be able to report a value of a variable relatively accurately and effortlessly when observing an instance of that attribute representing that variable.
- A perceiver must be able to report values on each of two variables relatively accurately upon observing a graphic instantiating two attributes. This task usually, but not necessarily, requires selective attention. This criterion probably isn't achievable for all of our attributes and may not even be achievable for any pair of them. But any attribute that is clearly non-separable with another should be rejected for our system. It is too much to expect, of course, that higher order interactions among attributes be non-existent. Much of the skill in graphic design is knowing what combinations of attributes to avoid.
- Each attribute must name a distinct feature in a rendering system. We cannot implement an attribute that does not uniquely refer to a drawable (or otherwise perceivable) feature. An attribute cannot be mapped to a miscellaneous collection of widgets or controls, for example.

We have attempted to classify aesthetics so that they are orthogonal in a design sense. One must not assume that this implies they are uncorrelated in our perceptual mechanisms, however. Orthogonalization in design means making every dimension of variation that is available to one object available to another. How these variations are perceived is another matter. Many aesthetic attributes, even ones such as *size* or *position* that are usually considered visual, need not be perceived visually. There is nothing in the definition of a graphic to limit it to vision. Provided we use devices other than computer screens and printers, we can develop graphical environments for non-sighted people or for those unable to attend to a visual channel because, perhaps, they are busy, restrained, or multiprocessing. Touch, hearing, even smell can be used to convey information with as much detail and sensitivity as can vision.

Every one of the figures in this chapter incorporates several aesthetics. Without aesthetic functions, they would not be visible. Consequently, we will not add a figure to illustrate other aesthetics, particularly since we are constrained in publishing format. Note, however, that in addition to using the *position* aesthetic function in every graphic, we have employed *shape* to differentiate symbols. Note, also, that *position* aesthetics are usually referenced by axes and *shape* and other aesthetics are usually referenced by legends.

Our discussion of the seven primary GOG components ends here. But there are several important topics remaining. We will first examine issues in graphics layout, and then conclude with a discussion of the relation between graphics algebra and statistical design models.

13.9 Layout

Chart algebra does not determine the physical appearance of charts plotted on a screen or paper. It simply produces a set of tuples $(x_1, \dots, x_p)$ that can be rendered using geometric primitives and a layout interpreter. If we have 2-tuples, then we can render them directly on a computer screen or piece of paper. If we have 3-tuples, then we can use a perspective projection to render them on the plane. Higher-order tuples require a layout scheme to embed all dimensions in the plane. Various layout schemes are attempts to solve a graphic representation problem: how to transform a p-dimensional vector space to a 2-dimensional space so that we can perceive structures in the higher dimensional space by examining the 2-dimensional space. We will discuss several approaches in this section.

Projection

One scheme is to employ a linear or nonlinear projection from p-dimensions to two. This may cause loss of information because a projection onto a subspace is many-to-one. Also, projection is most suitable for displaying points or $\{V, E\}$ graphs. It is less suitable for many geometric chart types such as bars and pies. Nevertheless, some 2D projections have been designed to capture structures contained in subspaces, such as manifolds, simplices, or clusters (Friedman 1987). Other popular projection methods are principal components and multidimensional scaling (Hastie et al. 2001).

Sets of Functions

A second possibility is to map a set of n points in $\mathbb{R}^p$ one-to-one to a set of n functions in $\mathbb{R}^2$. A particularly useful class of functions is formed by taking the first p terms in a Fourier series as coefficients for $(x_1, \ldots, x_p)$ (Andrews 1972). Another useful class is the set of Chebysheff orthogonal polynomials. A popular class is the set of $p - 1$ piecewise linear functions with $(x_1, \ldots, x_p)$ as knots, often called *parallel coordinates* (Inselberg 1984; Wegman 1985).

An advantage of function space representations is that there is no loss of information, since the set of all possible functions for each of these types in $\mathbb{R}^2$ is infinite. Orthogonal functions (such as Fourier and Chebysheff) are useful because zero inner products are evidence of linear independence. Parallel coordinates are useful because it is relatively easy to decode values on particular variables. A disadvantage of functional representations is that manifolds, solids, and distances are difficult to discern.

Recursive Partitioning

A third possibility is recursive partitioning. We choose an interval $[u_1, u_2]$ and partition the first dimension of $\mathbb{R}^p$ into a set of connected intervals of size $(u_2 - u_1)$, in the same manner as histogram binning. This yields a set of rectangular subspaces of $\mathbb{R}^p$. We then partition the second dimension of $\mathbb{R}^p$ similarly. This second partition produces a set of rectangular subspaces within each of the previous subspaces. We continue choosing intervals and partitioning until we finish the last dimension. We then plot each subspace in an ordering that follows the ancestry of the partitioning. Recursive partitioning layout schemes have appeared in many guises: $\mathbb{R}^p \mapsto \mathbb{R}^3$

(Feiner and Beshers 1990), $\mathbb{R}^p \mapsto \mathbb{R}^2$ (Mihalisin et al. 1991), $\mathbb{R}^4 \mapsto \mathbb{R}^2$ (Becker et al. 1996).

There are several modifications we may make to this scheme. First, if a dimension is based on a categorical variable, then we assume $(u_2 - u_1) = 1$, which assures one partition per category. Second, we need not partition a dimension into equal intervals; instead, we can make $[u_1, u_2]$ adaptive to the density of the data (Wilkinson 1999, page 186). Third, we can choose a variety of layouts for displaying the nodes of the partitioning tree. We can display the cells as an n-ary tree, which is the method used by popular decision-tree programs. Or, we can alternate odd/even dimensions by plotting horizontally/vertically. This display produces a 2D nested table, which has been variously named a *mosaic* (Hartigan and Kleiner 1981) or *treemap* (Johnson and Schneiderman 1991). We use this latter scheme for the figures in this article.

This rectangular partitioning resembles a 2D rectangular fractal generator. Like simple projection, this method can cause loss of information because aggregation occurs within cells. Nevertheless, it yields an interpretable 2D plot that is familiar to readers of tables.

Because recursive partitioning works with either continuous or categorical variables, there is no display distinction between a table and a chart. This equivalence between tables and graphs has been noted in other contexts (Pedersen et al. 2002; Shoshani 1997). With recursive partitioning, we can display tables of charts and charts of tables.

13.10 Analytics

If conclusions based on statistical charts are to be useful, we must identify and interpret the statistical models underlying charts. A statistical model determines how the location of a representation element (point, line, ...) in a frame (a measurable region of representation) is computed from the values of a variable. Statistical models usually (but not necessarily) incorporate error terms and help us to articulate the domains of generalizations and inferences we make from examining a chart. Glymour et al. (1996) summarize these issues from a data mining context. Because chart algebra is based on statistical design algebras, it can be used to generate statistical models for visual data mining or predictive analytics.

This section presents the statistical model equivalents of chart algebra expressions. In each subsection, we show the chart algebra notation on the left of each equivalence expression and the statistical model notation on the right. The terms on the left comprise varsets and the terms on the right comprise variables. Note that some symbols (e.g., +) are common to both notations but have different meanings. The general linear statistical models presented in this section are due to (Fisher 1925, 1935). More recent introductions to the design notation used for statistical models are (Heiberger 1989) and (Kutner et al. 1996).

13.10.1 Statistical Model Equivalents

In the following subsections, we assume a functional model $Z = f(X, Y)$, where Z is a (possibly multivariate) variable. Z corresponds to a varset Z, which itself might be produced from a chart algebra expression. In statistical terms, we sometimes call Z a *dependent* variable and X and Y *independent* variables. In this section, we ignore Z and focus on expressions involving X and Y. These expressions are used to construct statistical models that help to predict or estimate Z.

Cross

$$\mathrm{X} * \mathrm{Y} \sim \mathcal{C} + X + Y + XY$$

The cross operator corresponds to a fully factorial experimental design specification. This design employs a product set that includes every combination of levels of a set of experimental factors or treatments. The terms on the right of the similarity represent the linear model for fitting fully factorial designs. The terms in the model are:

$$\begin{aligned} \mathcal{C} &: \text{constant term (grand mean)} \\ X &: \text{levels of factor X (X main effect)} \\ Y &: \text{levels of factor Y (Y main effect)} \\ XY &: \text{product of factors X and Y(interactions)} \end{aligned}$$

We could use boldface for the variables on the right because the most general form of the model includes factors (multidimensional categorical variables) having more than one level. These multivariate terms consist of sets of binary categorical variables whose values denote presence or absence of each level in the factor. Alternatively, terms based on continuous variables are called *covariates*.

An example of a two-way factorial design would be the basis for a study of how teaching method and class size affect the job satisfaction of teachers. In such a design, each teaching method (factor X) is paired with each class size (factor Y) and teachers and students in a school are randomly assigned to the combinations.

Nest

$$\mathrm{X/Y} \sim \mathcal{C} + Y + X(Y)$$

The terms on the right of the similarity are:

$$\begin{aligned} \mathcal{C} &: \text{constant term} \\ Y &: \text{levels of factor Y (Y main effect)} \end{aligned}$$

$$X(Y) : \text{X levels nested within levels of Y}$$

The term $X(Y)$ represents the series $X \mid (Y = Y_1) + X \mid (Y = Y_2) + \ldots$

Notice that there is no interaction term involving X and Y because X is nested within Y. Not all combinations of the levels of X and Y are defined. An example of a nested design would be the basis for a study of the effectiveness of different teachers and schools in raising reading scores. Teachers are nested within schools when no teacher in the study can teach at more than one school. With nesting, two teachers with the same name in different schools are different people. With crossing, two teachers with the same name in different schools may be the same person.

Blend

$$\mathrm{X} + \mathrm{Y} \sim \mathcal{C} + F_{XY}$$

The terms on the right of the similarity are:

$$\mathcal{C} : \text{constant term}$$

$$F_{XY} : \text{function of } X \text{ and } Y \text{ (e.g., } X - Y)$$

The blend operator usually corresponds to a time series design. In such a design, we predict using functions of a time series. When the blend involves dependent variables, this is often called a repeated measures design. The simplest case is a prediction based on first differences of a series. Time is not the only possible dimension for ordering variables, of course. Other multivariate functional models can be used to analyze the results of blends (Ramsay and Silverman 1997).

An example of a repeated measures design would be the basis for a study of improvement in reading scores under different curricula. Students are randomly assigned to curriculum and the comparison of interest involves differences between pre-test and post-test reading scores.

It would appear that analytics have little to do with the process of building a chart. If visualization is at the end of a data-flow pipeline, then statistics is simply a form of pre-processing. In our model, however, analytics are an intrinsic part of chart construction. Through chart algebra, the structure of a graph implies a statistical model. Given this model, we can employ likelihood, information, or goodness-of-fit measures to identify parsimonious models. We will explore some graphic uses of statistical models in this section.

13.10.2 Subset Model Fitting

The factorial structure of most chart algebra expressions can produce rather complex models. We need to consider strategies for selecting subset models that are adequate

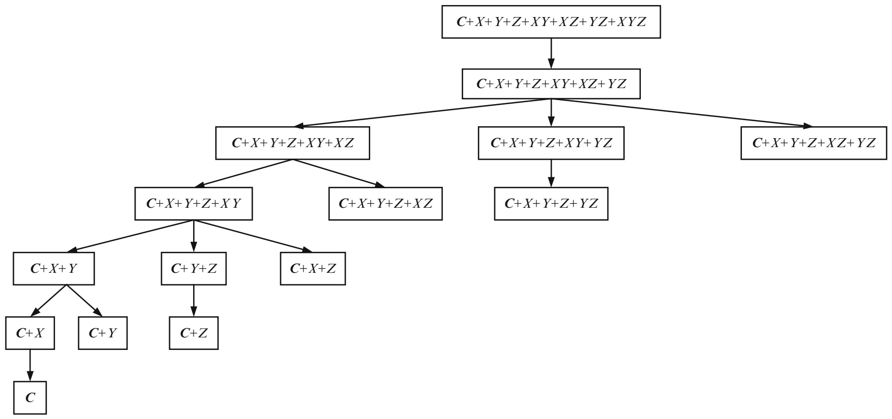

Fig. 13.13 Model subset tree

fits to the data. We will discuss one simple approach in this section. This approach involves eliminating interactions (products of factors) in factorial designs.

Interactions are often regarded as nuisances because they are difficult to interpret. Comprehending interactions requires thinking about partial derivatives. A three-way interaction XYZ, for example, means that the relation between X and Y depends on the level of Z. And the relation between X and Z depends on the level of Y. And the relation between Y and Z depends on the level of X. Without any interaction, we can speak about these simple relations unconditionally. Thus, one strategy for fitting useful subset models is to search for subsets with as few interactions as possible. In this search, we require that any variables in an interaction be present as a main-effect in the model.

Figure 13.13 shows a submodel tree for the three-way crossing $X * Y * Z$. Not all possible submodels are included in the tree, because the convention in modeling factorial designs is to include main effects for every variable appearing in an interaction. This reduces the search space for plausible submodels. By using branch-and-bound methods, we can reduce the search even further. Mosteller and Parunak (1985) and Linhart and Zucchini (1986) cover this area in more detail.

13.10.3 *Lack of Fit*

Statistical modeling and data mining focus on regularity: averages, summaries, smooths, and rules that account for the significant variation in a dataset. Often, however, the main interest of statistical graphics is in locating aspects that are discrepant, surprising, or unusual: under-performing sales people, aberrant voting precincts, defective parts.

An *outlier* is a case whose data value and fitted value (using some model) are highly discrepant relative to the other data-fitted discrepancies in the dataset. (Barnett and Lewis 1994). Casewise discrepancies are called *residuals* by statisticians. Outliers can be flagged in a display by highlighting (e.g., coloring) large residuals in the frame. Outliers are only one of several indicators of a poorly fit model, however. Relative badness-of-fit can occur in one or more cells of a table, for example. We can use subset modeling to highlight such cells in a display. Sarawagi et al. (1998) do this for log-linear models. Also, we can use autocorrelation and cross-correlation diagnostic methods to identify dependencies in the residuals and highlight areas in the display where this occurs.

13.10.4 Scalability

Subset design modeling is most suited for deep and narrow (many rows, few columns) data tables or low-dimensional data cubes. Other data mining methods are designed for wide data tables or high-dimensional cubes (Hand et al. 2001; Hastie et al. 2001). Subset design modeling makes sense for visualization applications because the design space in these applications does not tend to be high-dimensional. Visual data exploration works best in a few dimensions. Higher-dimensional applications work best under the guidance of other data mining algorithms.

Estimating design models requires $O(n)$ computations with regard to cases, because only one pass through the cases is needed to compute the statistics for estimating the model. Although computing design models can be worse-case $O(p^2)$ in the number of dimensions, sparse matrix methods can be used to reduce this overhead because many of the covariance terms are usually zero.

13.10.5 An Example

Smoothing data reveals systematic structure. Tukey (1977) used the word in a specific sense, by pairing the two equations

$$data = fit + residual$$

$$data = smooth + rough$$

Tukey's use of the word is different from other mathematical meanings, such as functions having many derivatives.

We smooth data in graphics to highlight selected patterns in order to make inferences. We present an example involving injury to the heads of dummies in government frontal crash tests. Figure 13.14 shows NHTSA crash test results for selected vehicles tested before 1999. The dependent variable shown on the

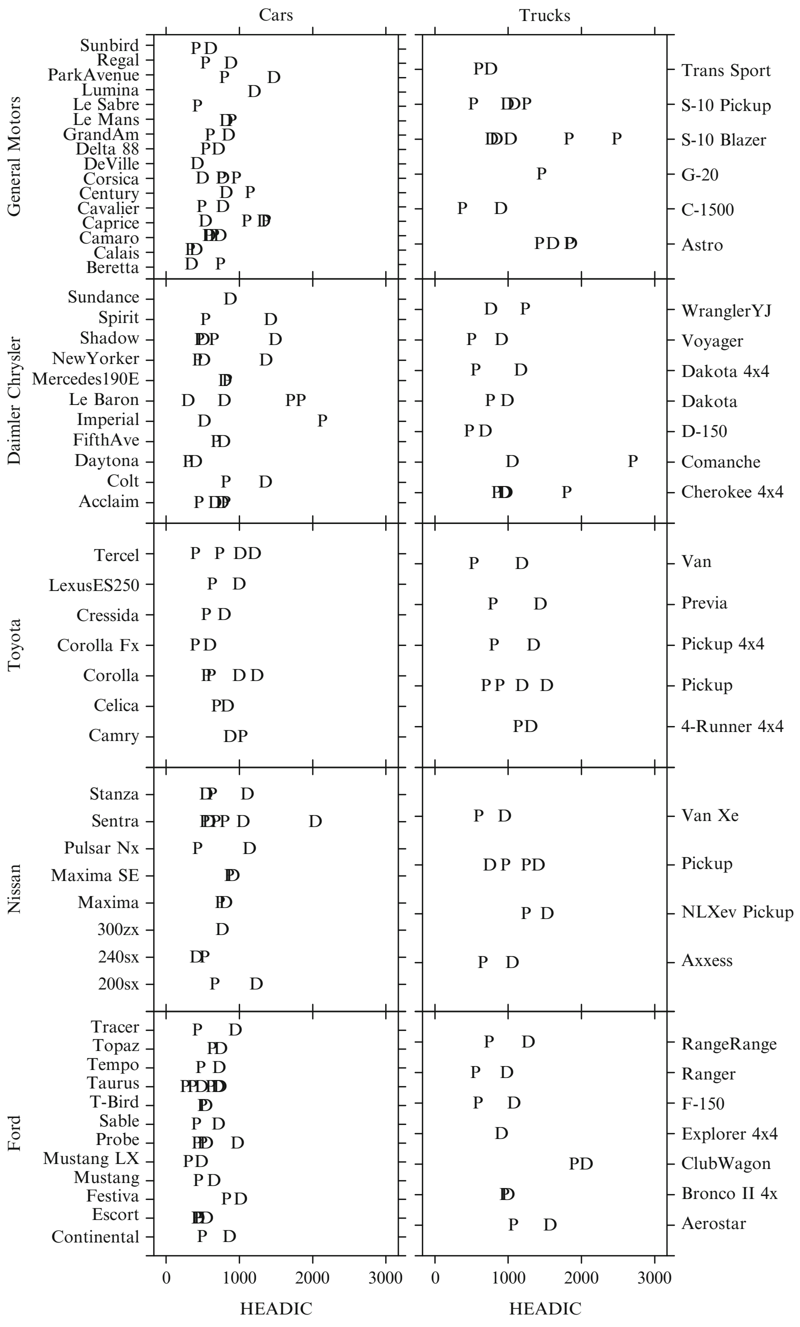

Fig. 13.14 Crash data

horizontal axis of the chart is the Head Injury Index computed by the agency. The full model is generated by the chart algebra $H * T/(M * V) * O$. This expression corresponds to the model:

$$H = \mathcal{C} + M + V + O + T(MV) + MV + MO + VO + OT(MV) + MVO$$

where the symbols are:

$$\begin{aligned} H &: \text{Head Injury Index} \\ \mathcal{C} &: \text{constant term (grand mean)} \\ M &: \text{Manufacturer} \\ V &: \text{Vehicle (car/truck)} \\ O &: \text{Occupant (driver/passenger)} \\ T &: \text{Model} \end{aligned}$$

This display is difficult to interpret. We need to fit a model and order the display to reveal the results of the model fit. Figure 13.15 charts fitted values from the following subset model:

$$H = \mathcal{C} + V + O + T(MV)$$

Figure 13.15 has several notable features. First, the models are sorted according to the estimated Head Injury Index. This makes it easier to compare different cells. Second, some values have been estimated for vehicles with missing values (e.g., GM G-20 driver data). Third, the trends are smoother than the raw data. This is the result of fitting a subset model. We conclude that passengers receive more head injuries than drivers, occupants of trucks and SUVs (sports utility vehicles) receive more head injuries than occupants of cars, and occupants of some models receive more injuries than occupants of others.

13.11 Software

Four systems have so far implemented the algebra described in this chapter. Wilkinson et al. (2000) developed a system called nViZn, which used the algebra to present interactive graphics in a Web environment. Norton et al. (2001) used the algebra to structure and display data in a real-time streaming environment; their system is called Dancer. Wills (2002) developed a server-based presentation graphics system with an XML schema for GOG; this system is called. ViZml. Stolte et al. (2002) used the algebra to develop a visualization front-end for an OLAP; their system is called Polaris.

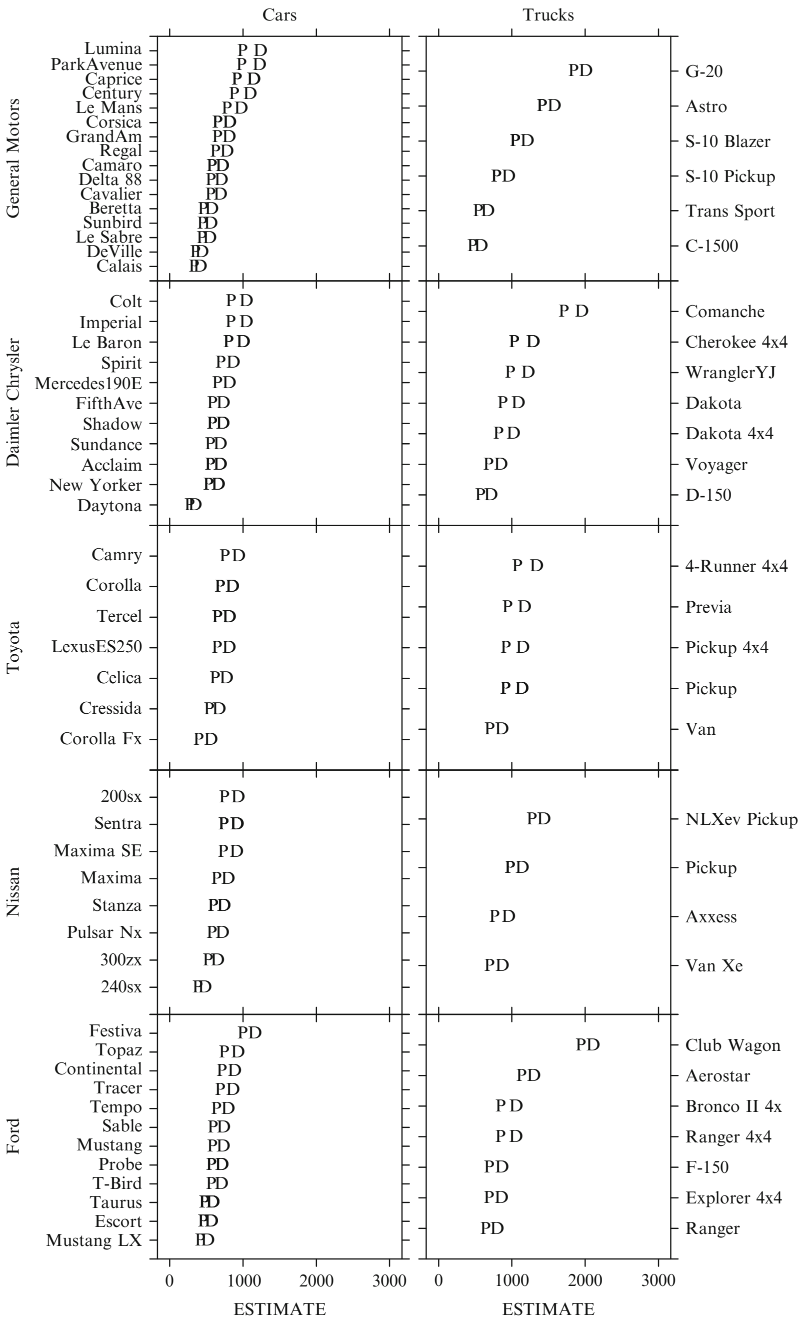

Fig. 13.15 Smoothed crash data

13.12 Conclusion

Many of the pieces that motivate graphics algebra have been lying around for decades: experimental design algebras, relational algebras, table algebras. These algebras emerged from separate disciplines, so that in most instances, researchers have been unaware of developments in the other disciplines. What is new about chart algebra is the explicit and formal equivalence between the data structures needed for statistical models and the methods for displaying them in charts. In a general sense, this equivalence allows us to think about manipulating data by manipulating statistical representation elements of a chart.

The GOG project has had several purposes. One, of course, is to develop statistical graphics systems that are exceptionally powerful and flexible. Another is to understand the steps we all use when we generate charts and graphs. This understanding leads to a formalization of the problem that helps to integrate the miscellaneous set of techniques that have comprised the field of statistical graphics over several centuries. Another purpose is to develop, ultimately, intelligent systems that can 1) graph data without human specification and 2) read already published statistical graphics to recover data and interpret relationships. Finally, we hope to define problem specification and user interaction in a way that enables graphics systems to be understood by ordinary people as well as by statisticians. By formally relating display structure and statistical models, we can produce environments in which users interact with data, receive guidance, and draw conclusions that are based on appropriate statistical inferences.

References

Abiteboul, S., Fischer, P.C., Schek, H.J.: Nested Relations and Complex Objects in Databases. Springer, New York (1989)

Agrawal, R., Gupta, A., Sarawagi, S.: Modeling multidimensional databases. In: Gray, A., Larson, P.-Å(eds.) Proceedings of 13th International Conference of Data Engineering (ICDE), IEEE Computer Society, pp. 232–243 (1997)

Andrews, D.: Plots of high dimensional data. Biometrics **28**, 125–136 (1972)

Barnett, V., Lewis, T.: Outliers in Statistical Data. Wiley, New York (1994)

Becker, R.A., Cleveland, W.S., Shyu, M.-J.: The design and control of trellis display. J. Comput. Stat. Graphics **5**, 123–155 (1996)

Bertin, J.: Smiologie Graphique. Editions Gauthier-Villars, Paris (1967)

Date, C.: What is a domain? In: Date, C.: (eds.) Relational Database Writings 1985–1989, pp. 213–313. Addison-Wesley, Reading, MA (1990)

Date, C.J., Darwen, H.: Relation-valued attributes. In: Date, C.J., Darwen, H. (eds.) Relational Database: Writings 1989-1991, pp. 75–98. Addison-Wesley, Reading, MA (1992)

Feiner, S., Beshers, C.: Worlds within worlds: Metaphors for exploring n-dimensional virtual worlds. In: Proceedings of ACM UIST '90. pp. 76–83. ACM Press, New York, NY (1990)

Fisher, R.: Statistical Methods for Research Workers. Oliver and Boyd, Edinburgh (1925)

Fisher, R.: The Design of Experiments. Oliver and Boyd, Edinburgh (1935)

Friedman, J.: Exploratory projection pursuit. J. Am. Stat. Assoc. **82**, 249–266 (1987)

Glymour, C., Madigan, D., Pregibon, D., Smyth, P.: Statistical inference and data mining. Comm. ACM **39**(11), 35–41 (1996)

Gyssens, M., Lakshmanan, L.V.S., Subramanian, I.N.: Tables as a paradigm for querying and restructuring (extended abstract). In: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp. 93–103. ACM Press, Montreal, Quebec, Canada (1996)

Hand, D.J., Mannila, H., Smyth, P.: Principles of Data Mining: Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA (2001)

Hartigan, J., Kleiner, B.: Mosaics for contingency tables. In: Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface, pp. 268–273 (1981)

Hastie, T., Tibshirani, R., Friedman, J.H.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, New York (2001)

Heiberger, R.M.: Computation for the Analysis of Designed Experiments. Wiley, New York (1989)

Inselberg, A.: The plane with parallel coordinates. Visual Comput. **1**, 69–91 (1984)

Johnson, B., Schneiderman, B.: Treemaps: A space-filling app. roach to the visualization of hierarchical information structures. In: Proceedings of the IEEE Information Visualization '91, pp. 275–282 (1991)

Kutner, M.H., Nachtschiem, C.J., Wasserman, W., Neter, J.: App.lied Linear Statistical Models, Richard D. Irwin, Incorporation, Homewood, IL (1996)

Linhart, H., Zucchini, W.: Model Selection. Wiley, New York (1986)

Mackinlay, J.: Automating the design of graphical presentations of relational information. ACM Trans. Graph. (TOG) **5**(2), 110–141 (1986)

Makinouchi, A.: A consideration on normal form of not-necessarily-normalized relation in the relational model. In: Proceedings of the Third International Conference on Very Large Data Bases. pp. 447–453 (1977)

Mendelssohn, R.C.: The bureau of labor statistic's table producing language (TPL). In: Proceedings of the 1974 annual conference, Bureau of Labor Statistics, pp. 116–122. Washington, DC (1974)

Mihalisin, T., Timlin, J., Schwegler, J.: Visualizing multivariate functions, data, and distributions. IEEE Computer Graphics and Applications. **11**(13), 28–35 (1991)

Mosteller, F., Parunak, A.: Identifying extreme cells in a sizable contingency table: Probabilistic and exploratory app. roaches. In: Hoaglin, D.C., Mosteller, F., Tukey, J.W. (eds.) Exploring Data Tables, Trends, and Shapes, pp. 189–224. Wiley, New York (1985)

Nelder, J.A.: The analysis of randomised experiments with orthogonal block structure (Parts I and II). Proc. Roy. Soc. Lond. Series A **283**, 147–178 (1965)

Norton, A., Rubin, M., Wilkinson, L.: Streaming graphics. Stat. Comput. Graph. Newsletter **12**(1), 11–14 (2001)

Pedersen, D., Riis, K., Pedersen, T.B.: A powerful and SQL-compatible data model and query language for OLAP. In: Proceedings of the thirteenth Australasian conference on Database technologies, Australian Computer Society, Incorporation, pp. 121–130. Melbourne, Victoria, Australia (2002)

Ramsay, J., Silverman, B.: Functional Data Analysis. Springer, New York (1997)

Roth, S.F., Kolojejchick, J., Mattis, J., Chuah, M.C., Goldstein, J., Juarez, O.: SAGE tools: a knowledge-based environment for designing and perusing data visualizations. In: Proceedings of the CHI '94 conference companion on Human factors in computing systems, pp. 27–28. ACM Press, Boston, Massachusetts, USA (1994)

Sarawagi, S., Agrawal, R., Megiddo, N.: Discovery-driven exploration of OLAP data cubes. In: Proceedings of the Sixth International Conference on Extending Database Technology (EDBT) (1998)

Shoshani, A.: OLAP and statistical databases: similarities and differences. In: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp. 185–196. ACM Press, Tucson, Arizona, USA (1997)

Simkin, D., Hastie, R.: An information processing analysis of graph perception. J. Am. Stat. Assoc. **82**, 454–465 (1987)

Stevens, S.S.: On the theory of scales of measurement. Science **103**, 677–680 (1946)

Stolte, C., Tang, D., Hanrahan, P.: Polaris: A system for query, analysis, and visualization of multidimensional relational databases. IEEE Trans. Visual. Comput. Graph. 8 **1**, 52–65 (2002)

Taylor, B.N.: The international system of units (SI). In: Special Publication 330, pp. 171–182. NIST, Washington, DC (1997)

Tukey, J.W.: Exploratory Data Analysis. Addison-Wesley Publishing Company, Reading, MA (1977)

Wegman, E.J.: Hyperdimensional data analysis using parallel coordinates. J. Am. Stat. Assoc. **85**, 664–675 (1985)

Wilkinson, G.N., Rogers, C.E.: Symbolic description of factorial models for analysis of variance. J. Roy. Stat. Soc. Series C **22**, 392–399 (1973)

Wilkinson, L.: A graph algebra. In: Computing Science and Statistics: Proceedings of the 28th Symposium on the Interface, pp. 341–351 (1996)

Wilkinson, L.: The Grammar of Graphics. Springer, New York (1999)

Wilkinson, L., Rope, D., Carr, D., Rubin, M.: The language of graphics. J. Comput. Graph. Stat. **9**, 530–543 (2000)

Wills, G.: Using the java 2 platform for advanced information visualization. Unpublished paper presented at the Annual Java One conference (2002)