Designing Graphic Presentations from First Principles

Abstract

Designing Graphic Presentations from First Principles

by

Michael Schiff
Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Robert Wilensky, Chair

This dissertation outlines a first-principles approach to automatically designing graphic presentations of information. The components of this approach include a conceptual framework for discussing how presentations encode information, algorithms for determining whether a method of presentation will be capable of presenting a given type of information, and design principles for ensuring the interpretability and perceptual effectiveness of a method of presentation.

Compared with previous approaches to automatically designing presentations, the approach outlined in this dissertation is more fine-grained and more general. It begins with an extremely general notion of how graphic presentations can encode information, then develops this into a useful framework by making a number of explicit assumptions about the types of presentations that people can use. This framework serves as a basis for analyzing the space of possible graphical languages—i.e., the space of systematic methods of presenting data. The logical adequacy of different graphical languages for different types of information, and criteria and methods for composing graphical languages for different data are also explored.

In addition to this logical emphasis, this dissertation also emphasizes the influence of psychological issues on the design of presentations. It explores factors influencing the interpretability of presentations (i.e., how easily viewers will grasp how information is encoded) and outlines some general design principles for creating interpretable presentations. It also explores perceptual issues in presentation—including perceptual organization, dimensional structure of visual stimuli, and the effectiveness of perceptual operations—and outlines design principles for guaranteeing the perceptual effectiveness of presentations.

The last emphasis of this dissertation is on operationalizing the framework and principles—i.e., on using them to create graphical languages in a relatively efficient manner. The implementation, AUTOGRAPH, demonstrates the flexibility and viability of a first-principles approach.

Acknowledgements

I have spent quite a few years working on this dissertation, and more generally in grad school. This experience would have been less fruitful and much less bearable without the support and advice of many people.

I would first like to thank my advisor, Robert Wilensky, for granting me the independence to pursue my own research interests and still managing to provide me with generous support and valuable advice. This dissertation might never have been completed, and certainly would have been weaker, without his insight. I would also like to thank the other members of my thesis committee, Marti Hearst and James Landay, whose comments and critiques greatly improved this work.

My graduate school experience was vastly enriched over the years by the various members and associates of my research group, including (in alphabetical order): Michael Braverman, Anne Fontaine, Marti Hearst, Narciso Jaramillo, Dan Jurafsky, Peter Norvig, David Palmer, Tom Phelps, Nigel Ward, Dekai Wu, Jamie Zawinski, and Jordan Zlatev. Most of these people were my officemates at one time or another. I never got much work done when they were around the office, but they nevertheless made my academic and personal experiences in grad school infinitely more rewarding.

Outside the office, other friends and colleagues in the department also helped make the experience of being a grad student not only interesting, but fun, too. Among the students and faculty members whom I had the pleasure to know were: David Cohen, Eric Enderton, Francesca Barrientos, Diane Hernek, Nina Amenta, Kim Keeton, David Parsons, Nikki Mirghafori, Terry Regier, Mike Luby, Oliver Grillmeyer, Dana Randall, Mike Clancy, and Ashu Rege.

Many other friends, near and far, also provided both diversion and support. Without the diversion, this dissertation might have been completed sooner, but without the support I'm sure it would never have been completed at all. Some of the nearby friends included Jon Levine and Susan Lin, Carrie Timko, Beth Multer, Becky Gross, Morrisa Sherman, Pete Chow, Erin Vaca, and Melanie Light. Others, further afield, included Jan Rivkin and Debbie Kadish, Ben Gillum and Heather Rayburn, Mike Rosenfeld, Tarun Khanna, Rahul Asthana, Pascale Fung, and Joanna Sadowska. Others, once near but now far, included: Ivelina Zlateva, Robert Chang, Anna Herreras, John Liechty, Toby Falk, Zenda Kuo, and Shari Cohen. A special thanks is due Kristin Conradi, who gave much support during the last year of the writing of this dissertation. And I'd like to add an extra thanks to Jon and Susan, and to Carrie, all of whom helped see me through more or less my entire grad school experience.

I've had a number of housemates over the years I've lived in Berkeley, but I've been particularly lucky in the last four or five years. My housemates made life at home fun, and always yielded the kitchen table when I needed it, even if they did sometimes leave dishes in the sink. Among the people I had the pleasure to live with were: Shari Rubin, Holly Holmquist, Roger Studley, Carla Savage, Sasa Gabarsek, Rebecca Gelman, Alex Cuthbert, and Sharon Gibson.

I have felt very fortunate during my graduate school tenure to have had the unfailing support of a very helpful Computer Science Department staff. Within the larger UCB bureaucracy, the department provided a smaller, more comfortable environment. I'd especially like to thank Kathryn Crabtree, who showed me around the day I first arrived in Evans Hall and has been of great aid in negotiating the system ever since.

My acknowledgements would not be complete without thanking the taxpayers of this country, who provided funding in the form of a National Science Foundation fellowship, through National Science Foundation grant IRI-94-11334 (part of the NSF/NASA/DARPA Digital Libraries Initiative), and through the DARPA, via the Corportation for National Research Initiative contract no. MDA 972-92-J-1029, during the writing of this dissertation.

Finally, I'd like to acknowledge the love and support of my parents Marilyn and Leonard Schiff and my sister Laura, who might have thought I'd never get out of grad school, but kept encouraging me anyway.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We live in a society awash in information. The increased use of computers over the past two decades in all types of enterprises has resulted in a vast increase of various types of data. Furthermore, the increased use of the Internet in recent years has made more information more readily accessible than ever before. Unfortunately, having more data does not lead automatically to a greater understanding of the implications of those data. If raw data are not available in comprehensible form, they may very well be useless.

People have developed various methods to cope with the problem of making information more comprehensible. Some of these methods have involved reducing data through mathematical analysis, including traditional statistics and, more recently, data-mining. One of the most enduring means of making information more useful, though, is the use of graphic presentations. By taking advantage of human perceptual abilities, graphic presentations can organize information so that it is easier to access, illuminate trends in data, or make information more memorable.

Unfortunately, like raw data itself, a graphic presentation of data is not guaranteed to be useful. A poor graphic presentation can obscure the very information it is intended to convey, and it is often difficult to create an effective one. This dissertation presents a solution to the problem of creating effective presentations: a theoretical framework and the outlines of a knowledge base for building a system which automatically finds effective ways of graphically presenting information. This is a "first principles" solution—i.e., an attempt to build a system which can design methods of presenting information with as few pre-programmed assumptions as possible. This dissertation also discusses a prototype implementation of such a system, a software tool called AUTOGRAPH.

1.1 The need for intelligent presentation design

Graphic presentations are now a standard feature of newspapers and news magazines, and a common feature of software analysis tools. Many graphic techniques have been standardized— e.g., bar graphs and pie charts—and are easy to produce with current software tools.

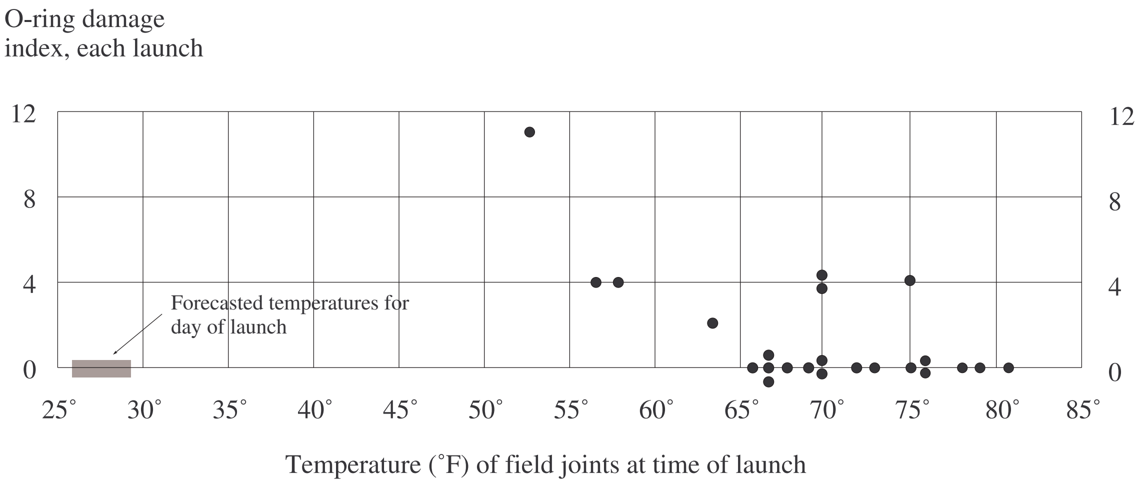


Figure 1.1: O-ring damage in shuttle test launches (redrawn from Tufte (1997))

Unfortunately, while such software tools make graphic presentations easy to produce, they are as adept at creating poor presentations as they are at creating good ones. The majority of these tools allow users to choose from a large (and ever-increasing) number of graphic formats, but provide insufficient guidance for choosing among these formats, many of which are likely to be inappropriate or ineffective for certain types of data.

The difficulty of creating a good presentation for even relatively simple data is illustrated compellingly in an incident discussed at length by Tufte (1997). Tufte examines the decision to launch the space shuttle *Challenger*, despite the concerns of engineers that the rubber O-rings might cause problems in the low temperature conditions expected on the day of the launch. It was precisely the failure of these O-rings that led to the explosion of the shuttle. The engineers presented NASA decision-makers with many (non-graphical) charts summarizing information supporting their claim that the launch should be postponed, but failed to construct the simple scatter plot shown in figure 1.1 (redrawn from Tufte (1997)), which clearly suggests a correlation between O-ring damage and temperature, and cause for concern. Tufte points out that even during congressional examinations after the disaster, some of the graphic presentations used in discussing what had happened completely obscured this correlation (see figure 1.2, from PCSSCA (1986)). Note that even re-ordering the rocket icons in figure 1.2 by temperature would have greatly alleviated this problem.

If such problems occur even with relatively simple datasets, they are likely to be even more frequent with more complex datasets, e.g., those representing data of many dimensions. Indeed, it may be quite difficult to find *any* appropriate graphic format for some complex datasets using existing graphic presentation tools. For example, consider the problem of finding a way of presenting data about Napoleon's failed Russian

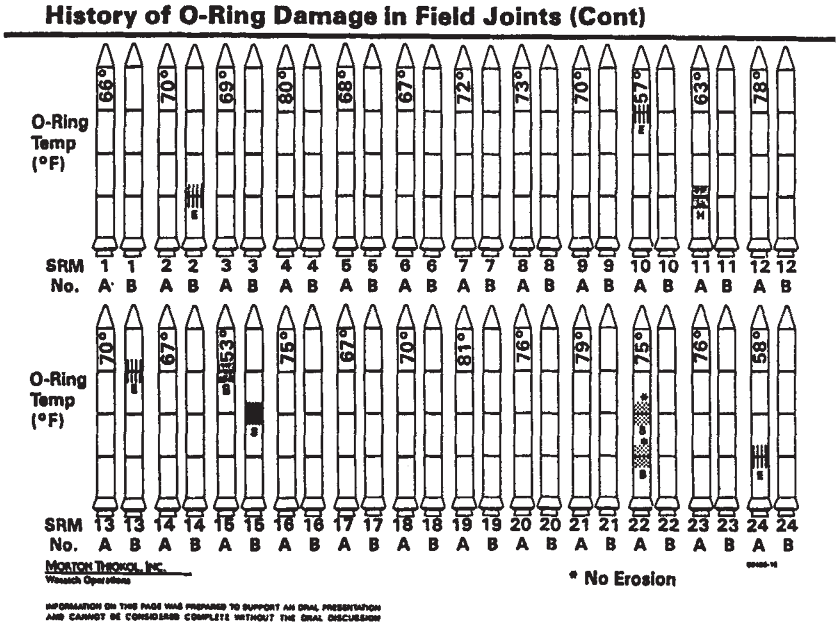


Figure 1.2: A poor graphic presentation of O-ring damage data

campaign, expressed in two datasets. The first dataset describes troop movements, expressed as tuples with the latitude and longitude of the start and end of the movements, the number of troops, and whether the movement was an advance or retreat:

```
54.8 24.0 55.0 25.4 422000 advance
55.0 25.4 54.8 26.0 400000 advance
55.0 25.4 55.5 25.4 22000  advance
54.8 26.0 55.3 26.6 50000  advance
55.3 26.6 55.2 28.3 33000  advance
54.8 26.0 55.0 29.0 250000 advance
...
```

The second dataset expresses some recorded temperatures for specific dates and places (expressed as the latitude and longitude of the place, the date, and the temperature in degrees celsius):

```
55.1 36.6 8-18-1844  0
54.7 32.0 9-9-1844  -9
54.4 31.0 9-14-1844 -21
...
```

These datasets are sufficiently complex that it might be quite difficult to find an effective means of presenting them. Together with the previous example, this example points to the need for a graphic presentation design tool— a tool which provides guidance in constructing an effective way to present information graphically, and which works for the widest possible variety of data. One approach is to create a fully automated system, which can suggest effective ways of presenting information graphically with only initial input

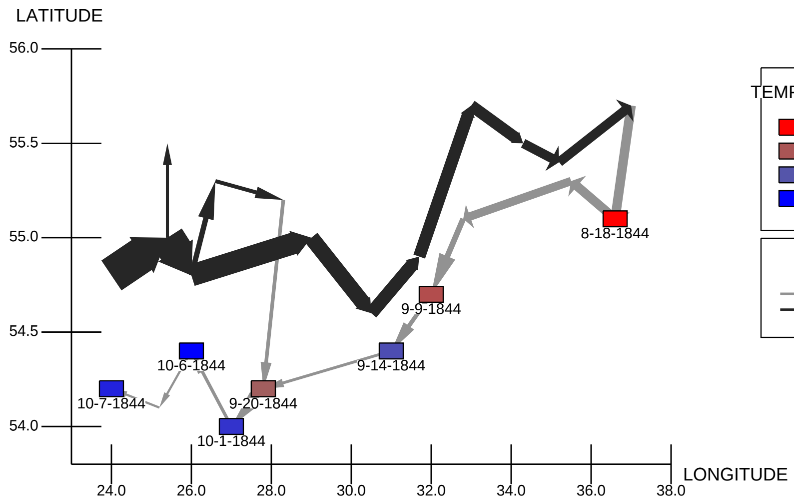


Figure 1.3: A presentation for two complex datasets

from a user. This is the goal pursued in this dissertation. AUTOGRAPH, a fully automated system based on the ideas of this dissertation (and described in chapter 7), was able to create the presentation shown in figure 1.3 for the Russian campaign datasets in under 15 seconds. This presentation, a variation of Minard's famous graph of similar data (often cited by Tufte (1983)), is discussed in more detail in chapter 7.

1.2 The graphic presentation design problem

Before discussing any solutions to the problem of automatically creating graphic presentations, it will be useful to clarify the problem I am attempting to solve, and to introduce some fundamental concepts.

The graphic presentation design problem is this: *given some data, a description of that data, and a description of goals for the use of the data, create a graphic display encoding the data and facilitating the completion of the goals.*

The input to an automatic graphic presentation system can thus be broken down into:

- The *data* to be presented. Throughout this dissertation I will assume that data are given in the form of tuples representing relations over sets of domains, possibly multiple datasets of different relations. For example, data about the number of students enrolled in a computer science department during different semesters could be given as {*enrollment(Fall94,120), enrollment(Spring95,135), enrollment(Fall95,130),*...}. When it is unambiguous, I will sometimes notate tuples without the name of the relation, e.g., {*<Fall94,120>, <Spring95,135>,*...}.[1]

[1] Describing data in terms of domains and relations requires that the data be schematized, and there may be several different ways of doing so. The approach presented in this dissertation attempts be fairly schematization-independent, however. The limitations of these attempts are discussed briefly in chapter 8.

- The *data characterization*, which describes structural properties of and possibly semantics of the relations and domains of the data sets. This characterization is intended to convey information that cannot be deduced from the data itself, such as the relationships of values in data domains (e.g. their order, if the set is ordered), the complete set of possible values for different domains (or the ranges for numeric values), and information about what sorts of tuples will occur together in datasets. For example, if *enrollment* is a relation over *semester* and *students*, the data characterization might specify that *semester* is an ordered set with the values $\{$*Fall94,Spring95,Fall95,*$\ldots\}$, that *students* is a numeric range from 90 to 170, that *students* is functionally dependent on *semester* (i.e., there will be at most one tuple in the dataset for any given semester), and that the dataset is *complete* with respect to *semester* (i.e., that there will be a tuple in the dataset for each semester).

- The *goal characterization*, which describes the goals for the use of the presentation to be generated. For example, a viewer's goal for the presentation of the enrollment data might be to spot trends in the number of students vs. the semester, which can be described as a *summary reading* of this pair of domains.

The data and goal characterization *languages* used by AUTOGRAPH, i.e., the precise syntax for characterizing data and goals, are described in chapter 7. Most of the arguments advanced in this dissertation are not dependent on the particulars of these languages, however, and I will avoid giving many details about them when such details are irrelevant. Note that in general, the greater the specificity of these languages, the more exact a system can be in choosing and evaluating methods of presentation. There is no theoretically well-defined cutoff for the degree of specificity; utility must guide the choice of characterization languages.

The most important output of an automatic graphic presentation system is a picture encoding the data, which will be referred to here as a *presentation*. For the simple enrollment dataset described above, a presentation might look like the bar graph of figure 1.4.

Solutions to the problem of automatically creating a graphic presentation from the data, data characterization, and goal characterization have also generally been constrained to solve a subproblem: finding a systematic method for encoding the data graphically, using the abstract data and goal characterizations alone, without the actual data. Such a method of encoding data graphically is referred to here, following Mackinlay (1986b), as a *graphical language*. After a graphical language has been found, it is then used to encode the data to generate a presentation. This architecture is shown in figure 1.5. Although this approach may have the disadvantage of not allowing the system to make decisions based on the exact data values to be presented (e.g., deciding that a certain type of presentation is likely to result in an overly cluttered display because the dataset to be presented is too large), it has the advantage of creating graphical languages that can be reused for different data of the same type.[2]

[2] Some of the limitations of not using the actual data when designing a presentation are discussed in chapter 8.

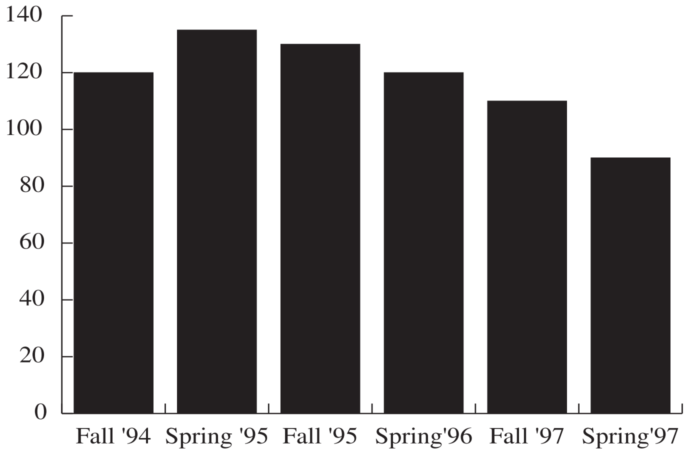


Figure 1.4: A bar graph for enrollment data

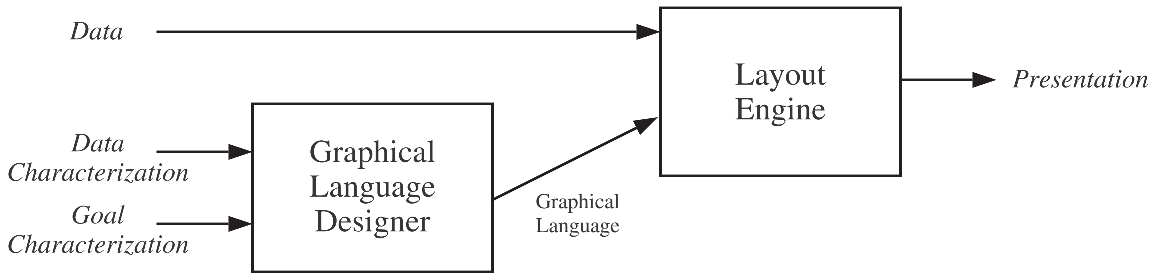


Figure 1.5: Basic system architecture

1.3 Designing graphic presentations from first principles

Any solution to the problem set forth in the previous section will necessarily involve both the implicit or explicit specification of the space of possible graphical languages that the system is capable of designing and methods for choosing among these languages. Previous research on automatic graphic presentation design has generally focused on building systems with predefined graphic techniques or languages (e.g., bar graphs, scatterplots) for representing data and methods for selecting and combining these techniques. Previous work is discussed in some detail in chapter 2. The essence of the first-principles approach described in this dissertation is to use a more fine-grained decomposition of the space of graphical languages, and to use methods for choosing languages in this space based on general knowledge about how people process visual information. While the differences between these approaches may be considered a matter of degree, the research described here attempts to be as general as possible both in the types of graphical languages to which it is applicable and in the knowledge on which it relies. It takes as a starting point human perceptual and cognitive abilities and the nature of graphic representation, rather than specific graphic presentations in common use today. The advantage of this approach is that, at least in theory, it should be capable of creating a wider variety of presentations, including novel ones not forseen during its design.

My framework starts with a basic catalog of objects that can be put into a display (e.g., circles, lines, arrows, etc.), a rudimentary description of human perceptual abilities (what properties of the graphic objects and what relationships between them people can perceive), and some basic assumptions about the systematicity of graphical languages. Given these fundamentals, *logical principles* define a space of possible graphical languages and determine which languages will be capable of representing given datasets. *Psychological principles*, which I will subcategorize into *interpretive principles* and *perceptual principles*, further determine which logically adequate graphical languages will be least subject to misinterpretation and best suited to quick and accurate achievement of the goals for a presentation. By exploring and evaluating different languages, an automatic presentation design system can choose the most suitable one.

To make such an approach work, it is necessary both to detail what the actual principles are and to determine how to combine and utilize them in an efficient manner. These are some of the main subjects of this dissertation.

Before describing the different types of principles, it will be useful to be more concrete about what is meant by a graphical language. In my framework, a graphical language consists of:

- A set of objects that will be used in the presentation (e.g., rectangles, circles, and lines).
- A correspondence between the domains of the relations of the dataset(s) and *perceptual properties* of the objects in the presentation. For example, in the bar graph of the enrollment data described earlier (figure 1.4), each tuple is encoded by a rectangle, with the horizontal position of the rectangle encoding *semester* and the height of the rectangle encoding *students*. In the network graph of figure 1.6, presenting data on flights between cities, there is a correspondence between circle labels and cities.[3]

[3] Throughout this dissertation, labels will be considered to be perceptual properties of labelled objects. This is not an integral part of my framework, but it is convenient.

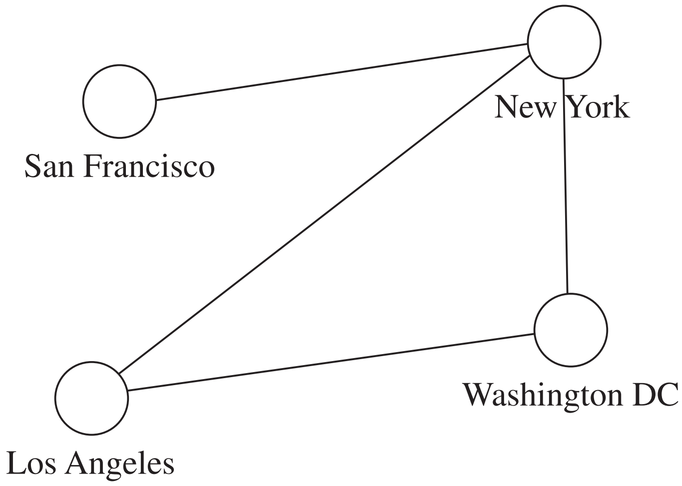


Figure 1.6: A standard network graph

- A specification of the *perceptual relationships* between objects in the presentation required to encode data relations, if multiple objects are used to encode a single data relation. For example, in figure 1.6, where a line and a pair of circles encode a flight between two cities, the line must be perceived as *touching* each of the circles.
- Additional *constraints* on objects in the presentation. For example, the bottoms of all the rectangles in figure 1.4 are constrained to be aligned.

This description is somewhat simplified. A more detailed description, motivated by logical analysis, is given in chapter 3.

It will also be useful to introduce two concepts fundamental to a discussion of graphical languages, *expressiveness* and *effectiveness*, defined by Mackinlay (1986b). Expressiveness refers to whether a particular language is logically capable of presenting a particular type of dataset as it is described by a data characterization. Effectiveness refers to how well (i.e., how quickly and easily) the language allows a viewer to achieve the desired goals for the presentation.

The three types of principles in my framework—logical, interpretive, and perceptual—arise from different sources and play different roles. I will describe each of them here, illustrated with examples, then describe the system architecture that combines them.

1.3.1 Logical principles

Logical principles define the space of conceivable graphical languages and determine which languages work with (i.e., are expressive for) which types of data. All of our notions about graphical languages must ultimately be rooted in human perceptual and cognitive abilities, but logical principles delimit what languages there can be, given a characterization of these abilities. Logical principles are themselves objective, and can be determined analytically rather than empirically, given a few assumptions.

As an example, consider the possibility of using the shape of a mark on a map to encode information about some object at the location indicated by the position on the map. If there are only n shapes available to display, one of only n possible different values can be encoded by the shape of a single mark, so no data domain of more than n values can be represented by shape alone.

Other issues of logical expressiveness concern whether entire datasets can be presented using a given language. Sometimes all the necessary objects and relationships between them cannot be displayed simultaneously, and sometimes displaying the objects and relationships needed to present certain tuples in a dataset will automatically present other tuples which may not be in the dataset. For example, if a labeled rectangle below another labeled rectangle is used to indicate that two states (encoded by the labels) border each other, it will not be possible to represent the dataset {*<Washington,Oregon>*,*<Oregon, Idaho>*, *<Idaho, Washington>*}, because all the *below* perceptual relationships cannot be satisfied simultaneously. As another example, a standard network graph representing flights between cities by lines between pairs of labeled circles (see figure 1.6) will only be appropriate if the data is symmetric—i.e., if the data characterization specifies *<city1,city2>* $\rightarrow$ *<city2,city1>*—because presenting *<city1,city2>* implicitly presents *<city2,city1>* as well. A graph using arrows instead of lines, however, would be suitable for non-symmetric data.

Logical principles are discussed in more detail chapter 3, which presents the conceptual framework underlying my approach, and in chapter 4, which operationalizes this conceptual framework with algorithms and technical details.

1.3.2 Interpretive principles

Frequently, if a graph in a newspaper or magazine seems "bad" to us, it is not because information is missing from the graph or difficult to perceive, but because the graph is confusing. With a poor graphical language it can be difficult to determine what information is being presented or how it is graphically encoded. Interpretive principles describe the qualities of a graphic presentation that make it easy to interpret correctly, without confusion.

Interpretive principles arise both from the workings of human cognition and from learned expectations about how information is presented graphically. These are the most difficult principles to determine objectively, and the ones that have been least systematically explored by previous research. The specific interpretive principles discussed in this dissertation (in chapter 5) and used in AUTOGRAPH are generally based on subjective analysis and commonsense reasoning.[4]

As an example, consider the bar graph in figure 1.7, in which different bars are arbitrarily given different shades. This is problematic because a viewer is tempted to try to find some interpretation for the

[4] Fortunately, although my analysis of these principles will be subjective, there is often a consensus among different writers on the subject of graphic presentations in their analyses of these issues, at least when it comes to specific examples.

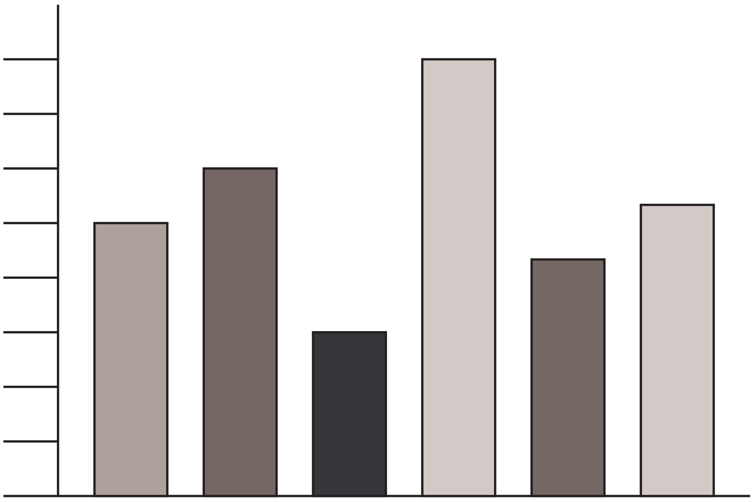


Figure 1.7: A bar graph with extraneous shading

shades, though there is none. In general, there is an expectation that a perceptually salient feature of objects in a graphic presentation will be semantically meaningful. This is the principle of *relevance*.

Consider also the graph in figure 1.8, presenting data about collaboration between students and faculty members in two departments, with students represented by circles and faculty members represented by rectangles. This graph could be made clearer if grayscale value or size were used for both circles and rectangles, rather than unnecessarily using two separate perceptual properties. This is a violation of the principle of *consistency*. There are other problems with this presentation as well, including the use of size to encode an unordered set, but I will defer a full discussion of interpretive principles until chapter 5.

In many instances, interpretive principles seem so obvious they are easy to take for granted. In automatic graphic presentation systems that use predetermined techniques for representing information, they often remain implicit or hardcoded into the layout engine. In a system that works from first principles, however, interpretive principles are essential and must be made explicit.

1.3.3 Perceptual principles

Perceptual principles, which are obviously very important to graphic presentation design, define the basic perceptual distinctions that a viewer of a graphic presentation can make and describe how effectively he or she can make them. They thus address the question of how effective a given graphical language is likely to be for a given type of data. Like interpretive principles, these are psychological in nature. Unlike interpretive principles, however, there is a large body of empirical research bearing on them. Perceptual principles are discussed in chapter 6.

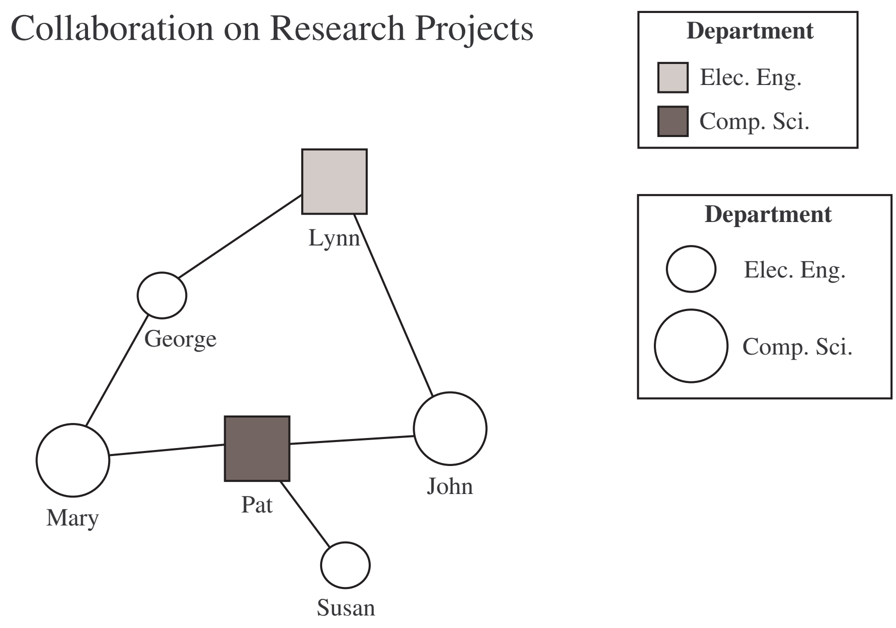
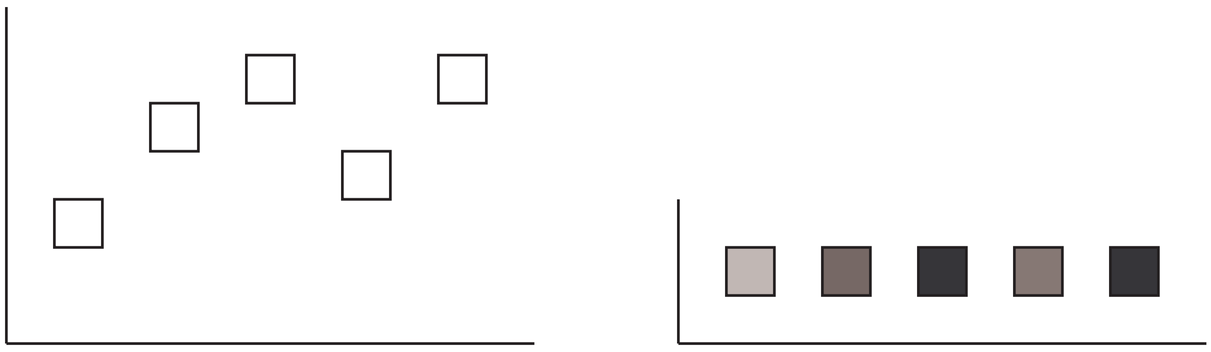


Figure 1.8: A presentation with consistency problems

Some of the most important types of perceptual principles relate to the speed and accuracy of basic perceptual abilities. Consider for example the two unlabelled graphs in figure 1.9, (a) using vertical position to encode a dependent variable and (b) using grayscale value to encode the same dependent variable. The fact that people are better at discerning position than at discerning value (i.e., how dark an object is) should lead a presentation design system to choose (a) over (b), all other things being equal.[5]

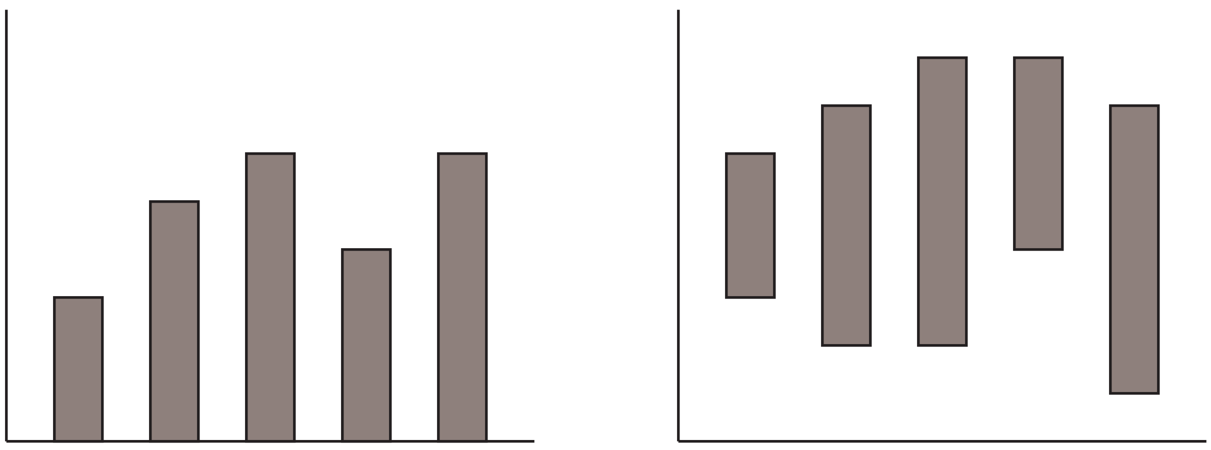
Perceptual principles are also used by AUTOGRAPH to help constrain otherwise underconstrained graphical languages. For example, it has been determined experimentally (and is probably also intuitively obvious) that aligned height of rectangles is more accurately discerned than non-aligned height (Cleveland & McGill 1984). Thus, in a language using height to convey one data value and horizontal position to convey another, AUTOGRAPH will choose to align the rectangles, whose vertical positions are otherwise unconstrained, resulting in a standard bar graph like the one in figure 1.10(a) instead of one like that of figure1.10(b). Note that such optimizations are generally built into the graphical languages used by systems that start with a set of such languages, but must be made explicit in a first-principles system.

[5]Note that 1.9(b) takes up less vertical space, and is more amenable to vertical stacking, so it is likely to have uses if there is additional information to display.



(a) (b)

Figure 1.9: Two presentations of the same data



(a) (b)

Figure 1.10: Two presentations of the same data

1.3.4 System architecture

AUTOGRAPH uses a relatively straightforward architecture to implement a first-principles approach. Working from the data characterization of a single dataset, given by a user, it examines different possible graphical language design decisions (the types of objects and properties of those objects with which to encode data, the relationships between different objects, etc.) defined by logical principles. The choices for each of these design decisions form a search tree, which is pruned by rules and heuristics based on logical, interpretive, and perceptual principles. After a set of potential languages has thus been found for one dataset, this process is repeated to find compatible languages for the other datasets. When a (possibly large) set of possible languages has been found for all of the datasets, more detailed heuristics based on perceptual and interpretive principles are used to choose the best available language. This language, which may be underconstrained, is processed by a "finalization" module to add any necessary constraints, then passed to a layout engine which uses the graphical language and the dataset(s) to create a presentation. This basic architecture is summarized in figure 1.11. The details of the AUTOGRAPH implementation are discussed in chapter 7.

1.4 An example

In order to illustrate some of the fundamental concepts brought up in this introduction, and to provide a rough sketch of the workings of AUTOGRAPH, it is useful to consider a detailed example. This example shows how a user interacts with AUTOGRAPH, reveals the steps AUTOGRAPH goes through in choosing a graphical language and more generally demonstrates how different types of principles combine in guiding the design of a language. The descriptions of design decisions made by AUTOGRAPH will be simplified here, and furthermore some assumptions which I will later make explicit will be left implicit here.

Consider the data shown in figure 1.12, organized into two datasets, describing prerequisites (either suggested or required) among a set of courses and the semester each course is offered (assuming each course is offered only one semester).

A session with AUTOGRAPH begins with the user specifying where some dataset to be presented can be found. After AUTOGRAPH reads in the dataset, it asks the user to label the individual fields of the tuples of the dataset for future reference and to describe what domains the values in these fields are drawn from, as shown in figure 1.13. It then asks the user to describe any functional dependencies among domains in the dataset, and specify whether the independent domains of the functional dependency are complete (i.e., whether a tuple will be in the dataset for every set of values for these domains), as shown in figure 1.14. After asking the user about semantic grouping among the domains (not relevant to this example), AUTOGRAPH will finally ask the user to characterize his or her goals for this dataset, in terms of *basic* or *summary* readings, as shown in figure 1.15. Several goals can be specified, if the user desires.

This process is repeated for any other datasets the user wishes to present simultaneously (i.e., in a single presentation). After every dataset has been described, the user finishes the data characterization by

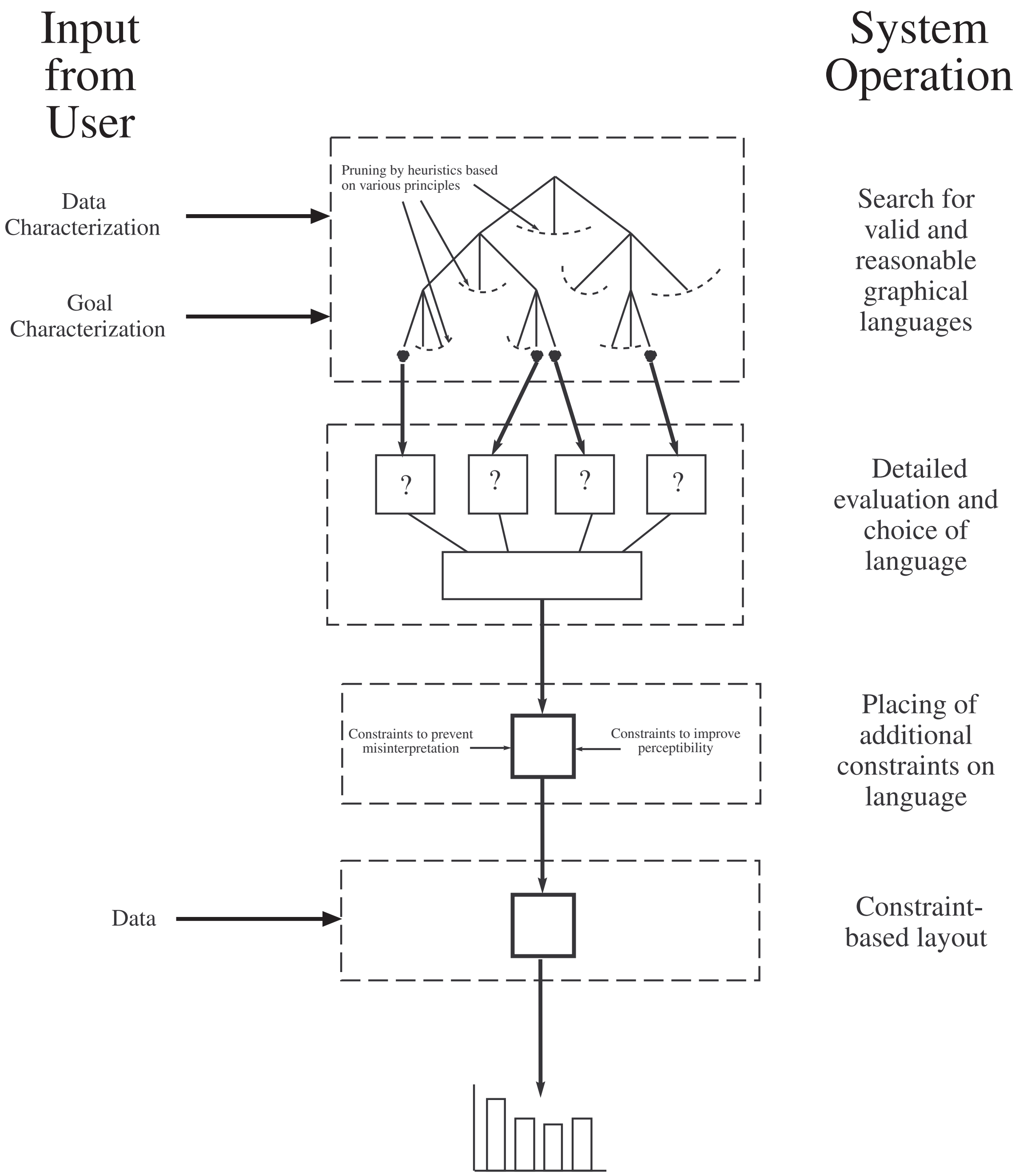


Figure 1.11: Detailed system architecture

```
          ; prerequisite dataset
CS101 CS105 suggested
CS101 CS102 required
CS105 CS205 required
CS102 CS150 suggested
CS150 CS250 required
CS250 CS255 suggested
CS205 CS305 suggested

          ; semester offered dataset
CS101 Fall95
CS102 Spring96
CS105 Fall96
CS205 Fall96
CS150 Spring97
CS250 Spring97
CS255 Fall97
CS305 Spring98
```

Figure 1.12: Datasets for course prerequisites and semester offered

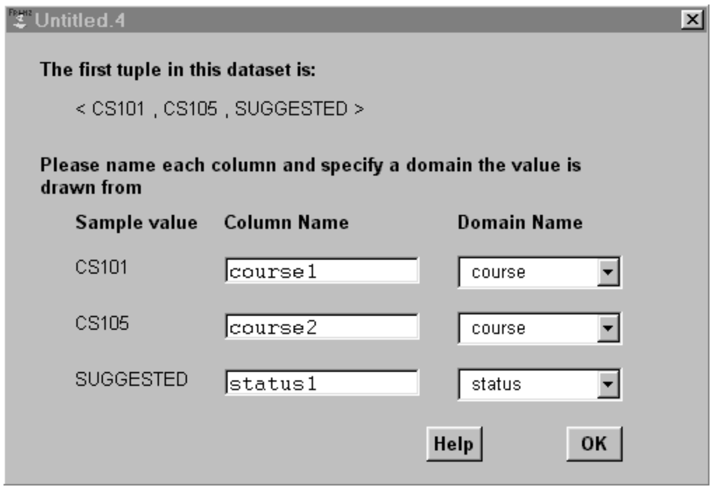


Figure 1.13: Specifying individual domains of the dataset

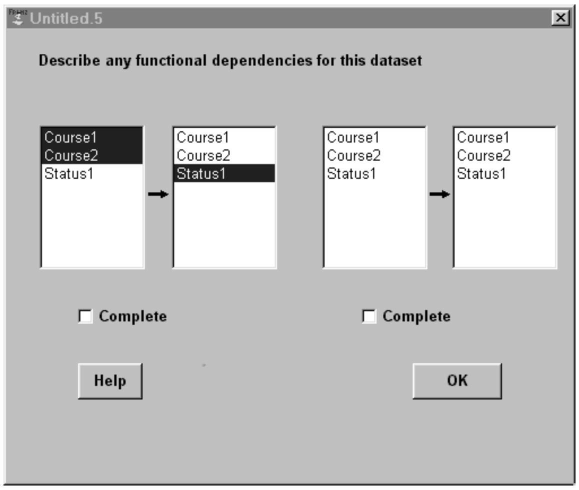


Figure 1.14: Specifying functional dependencies and completeness

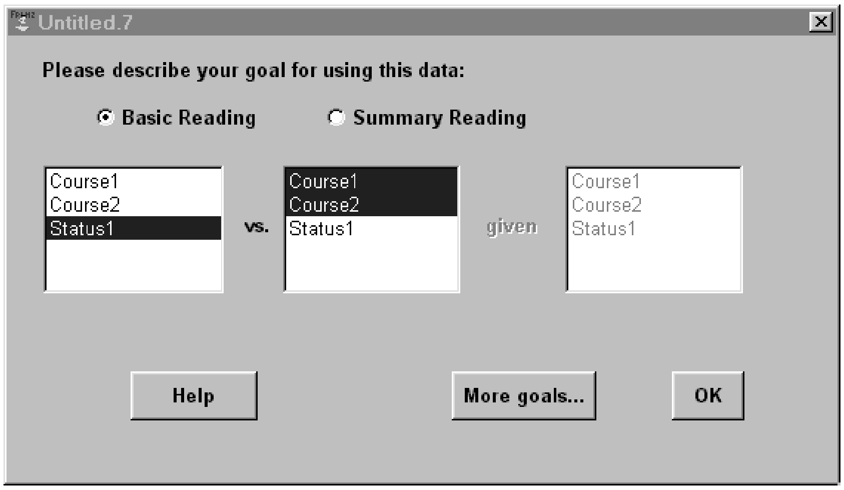


Figure 1.15: Describing goals

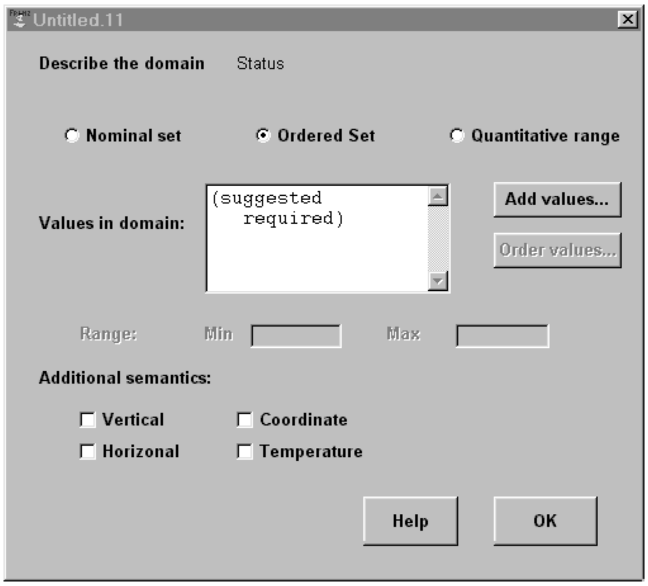


Figure 1.16: Characterizing domains

describing the individual domains used in all of the datasets, one at a time.[6] The dialog box for one of the domains for this dataset is shown in figure 1.16.

After this dialog-driven process is completed, AUTOGRAPH produces the data characterization shown in figure 1.17, and the goal characterization shown in figure 1.18.[7] These characterizations presume that the domains of the two datasets have been identified by distinct labels, with the domains of the first dataset labeled `(course1 course2 status1)` and those of the second labeled `(course3 semester1).`

The data characterization specifies that *course1*, *course2*, and *course3* are all instances of the nominal (i.e., discrete and unordered) domain *course* which contains the values {*CS101, CS102, CS105, CS150, CS205, CS250, CS255, CS305*}, that *status* is an instance of an ordered domain with the ordered set of values {*suggested*,*required*}, and that *semester* is an instance of an ordered domain containing the ordered set of values {*Fall95*, *Spring96*, *Fall96*,…}. It also specifies that *status1* is functionally dependent on *course1* and *course2* (i.e., there will be at most one *status* value for each pair of values for *course1* and *course2*), that *course1* and *course2* are independent (i.e., dependent on no other domains) and that semester is functionally dependent on *course3*.[8] Finally, it specifies that the mapping from *course3* to *semester* is complete—i.e., the dataset contains a tuple for every course specifying the semester it is offered.

The goal characterization specifies that for a given pair of courses, a viewer will want to see the status of any prerequisite relationship between them (i.e., suggested or required), and for any single course, a

[6] Performing this step last enables all of the domain values that occur in the datasets to be present when characterizing the domains.

[7] These characterizations may of course be produced by hand. Indeed, the dialog-driven front end to the system, in addition to being somewhat *ad hoc*, limits the user's description of the data in some ways. These limitations are described in chapter 7.

[8] This is slightly unrealistic, since it implies that each course is offered only once over a period of several semesters, but it is convenient for purposes of the example.

```
(setq *data-description*
      '((course1 course nil)
        (course2 course nil)
        (status1 status (course1 course2))
        (course3 course nil)
        (semester1 semester (course3))))
(setq *data-completeness-description*
      '((course3)))
(setq *data-domains*
      '((course nominal (CS101 CS102 CS105 CS205 CS150 CS250
                         CS255 CS305))
        (status ordered (suggested required))
        (semester ordered (Fall95 Spring96 Fall96 Spring97
                           Fall97 Spring98) (coordinate))))
```

Figure 1.17: Characterization of course prerequisite data

```
(setq *user-goals*
      '((basic (course1 course2) (status1))
        (basic (course3) (semester1))))
```

Figure 1.18: Characterization of goals for course prerequisite presentation

viewer will want to see which semester it is offered.

After these characterizations have been produced, AUTOGRAPH begins its search for suitable methods of presentation. AUTOGRAPH will try to find a graphical language for one of the datasets first, e.g., the prerequisite dataset, then try to find a compatible language for the other datasets (in this example, the semester-offered dataset).

AUTOGRAPH operates by exploring different ways that the domains of each tuple can be mapped onto the properties of one or more objects, and the ways that multiple objects, if they are used, can be combined to form larger entities. Taking the prerequisite dataset first (arbitrarily), AUTOGRAPH begins by considering the different ways that the tuple can be split or partitioned into groups of domains, e.g., {{*course1, course2, status*}} or {{*course1*}, {*course2*}, {*status*}}. Each group will be mapped to a separate graphic object. Some of these partitionings are rejected as leading to inconsistent (and thus difficult to interpret) languages—i.e., the search tree is pruned by a heuristic intended to assure that the interpretive principle of consistency is fulfilled. For instance, {{*course1, status*}, {*course2*}} is a rejected partitioning, because *status* is functionally dependent on both *course1* and *course2*, but grouped only with *course1*.[9] The three non-pruned partitionings are shown in figure 1.19—the domains of the tuple can be left as a single group, split into two groups (with *course1* and *course2* in one group and *status1* in the other), or split into three separate groups. Domains that are grouped together are enclosed by parentheses, as is each partitioning itself.

[9]That this is likely to yield inconsistent languages may be far from obvious at this point. The particular heuristics used to operationalize interpretive and perceptual principles in AUTOGRAPH are discussed in chapter 7.

```
Tuple: (course1 course2 status1))

   Acceptable Partitionings: (((course1 course2 status1))
                              ((course1 course2) (status1))
                              ((course1) (course2) (status1)))
```

Figure 1.19: Valid partitionings of tuples of first dataset

For each partitioning, AUTOGRAPH explores different types of objects that can be assigned to the sets of domains, with properties of the objects mapping to the data domains. It examines only properties deemed suitable for encoding the domains – i.e., those deemed logically adequate for representing the information and capable of being interpreted easily by a human. For example, hue, which is generally not considered to be perceptually ordered, is difficult to interpret when encoding an ordered domain (such as *status* or *semester*), and thus will not be considered for such domains. For the partition {{*course1, course2, status*}}, one possible mapping would represent *course1* by the horizontal position of a circle, *course2* by the vertical position of the circle, and *status* by the size of the circle. A mapping where *course1* was encoded by horizontal position, *status* by the vertical position, and *course2* by the size of a circle would be rejected as inconsistent for interpretive reasons: encoding two values from one domain differently within the same type of object is likely to be confusing. For the partition {{*course1*},{*course2*},{*status*}}, many possible mappings exist, including mapping *course1* and *course2* to the labels of circles and *status* to the thickness of lines or arrows. There are a huge number of possible correspondences between data domains and the perceptual properties of objects, so knowledge about the perceptual effectiveness of different properties is used to prune away some of the less effective ones. An excerpted view of possible correspondences is shown in figure 1.20. For each group of domains in the partitioning, corresponding object types and properties of those objects are listed. The order of the objects listed corresponds to the order of the groups of domains and the order of the properties listed with each object corresponds to the order of the domains within each group.

For partitions involving only one object, the language is complete. For those involving multiple objects, AUTOGRAPH proceeds to determine a way to connect the objects together with perceptual relations. Figure 1.21 shows this step for two set of objects, listing possible sets of perceptual relations together with the indices of the objects (in the correspondence list) that are connected by these relations. For the mapping with two circles and a line, one suitable method is for the line to touch each of the two circles. For the mapping with two circles and an arrow, the tail of the arrow could touch either circle and the head of the arrow the other circle; both possibilities are listed, enclosed by parentheses.

Included in the set of candidate graphical languages for the first dataset at this point are thus the three shown in figure 1.22.[10] Each of these languagess is checked, using the data characterization, to ensure that every possible dataset corresponding to the data characterization could be presented and that there is no possible dataset which, presented, would inadvertently represent tuples not in the dataset. This is a logical

[10] In keeping with the terminology I have been developing, the figure actually shows *presentations* of the data using the candidate languages.

```
Partitioning: ((course1 course2 status1))

   Correspondences: ((obj-prop circle hpos vpos size)
                     (obj-prop circle vpos hpos size)
                     (obj-prop rectangle hpos vpos width)
                     (obj-prop rectangle vpos hpos width)
                     (obj-prop rectangle hpos vpos height)
                     ...)

Partitioning: ((course1) (course2) (status1))

   Correspondences: (((obj-prop circle hpos)
                      (obj-prop circle hpos)
                      (obj-prop line width))
                     ...
                     ((obj-prop circle color)
                      (obj-prop circle color)
                      (obj-prop line width))
                     ((obj-prop circle hpos)
                      (obj-prop rectangle hpos)
                      (obj-prop line width))
                     ...
                     ((obj-prop circle label)
                      (obj-prop circle label)
                      (obj-prop line width)
                     ((obj-prop circle label)
                      (obj-prop circle label)
                      (obj-prop arrow width)
                     ...)
```

Figure 1.20: Some possible mappings for two different partitionings

```
Partitioning: ((course1) (course2) (status1))

Correspondence: ((obj-prop circle label) (obj-prop circle label)
                 (obj-prop line width))

          ; 3 is index of line, 1 and 2 are indices of circles
  Perceptual Relations: (((touches 3 1) (touches 3 2)))

Partitioning: ((course1) (course2) (status1))

Correspondence: ((obj-prop circle label) (obj-prop circle label)
                 (obj-prop arrow width))

  Perceptual Relations: (((head-touches 3 1) (tail-touches 3 2))
                         ((tail-touches 3 1) (head-touches 3 2)))
```

Figure 1.21: Possible perceptual relations for two different correspondences

expressiveness check. This check would eliminate the language with the line between the circles shown in figure 1.22.b, because a presentation of any dataset including e.g., <*CS101,CS102,required*> would automatically also represent the tuple <*CS102,CS101,required*> whether or not it were in the dataset. Note that if the data characterization had specified that any dataset including a tuple <*course1,course2,status1*> would also include the tuple <*course2,course1,status1*> (which would of course be nonsensical, given the semantics of this dataset), the language of figure 1.22.b would not have been rejected.

After finding a set of graphical languages for the first dataset, AUTOGRAPH next tries to find compatible languages for the next dataset. The process is similar, but each new language is checked to make sure it is consistent with the language chosen for the first dataset. If it uses objects that are also used in the first language, it must use them in a similar way. For example, representing *course3* with the label of a circle and *semester* with the horizontal position of a circle is compatible with the language using two labeled circles and an arrow, but not with the language in which horizontal position is used to represent a course. Similarly, the new language couldn't use horizontal position to represent *course3* and label to represent the *semester* with the circle/arrow language, because label is already being used to represent something else.[11]

When one or more compatible graphical languages have been created for each of the earlier chosen languages (if possible), the system evaluates each set of languages to determine the one it expects to be most perceptually effective. In this example, assuming AUTOGRAPH only found compatible languages for the circle/arrow language, one where circle position (horizontal or vertical) was used to encode semester (so that the horizontal or vertical positions of the circles in the graph are constrained) would be deemed superior to one where the size of the circle encoded semester, as in figure 1.23.[12]

[11] The actual mechanisms of determining when two graphical languages are consistent/compatible are rather involved. They are discussed in detail in chapter 4.

[12] This latter language would actually be rejected for interpretive reasons as well, if the data characterization specified that semester, while being an ordered set, was not a *quantity*, which is what size is typically assumed to represent.

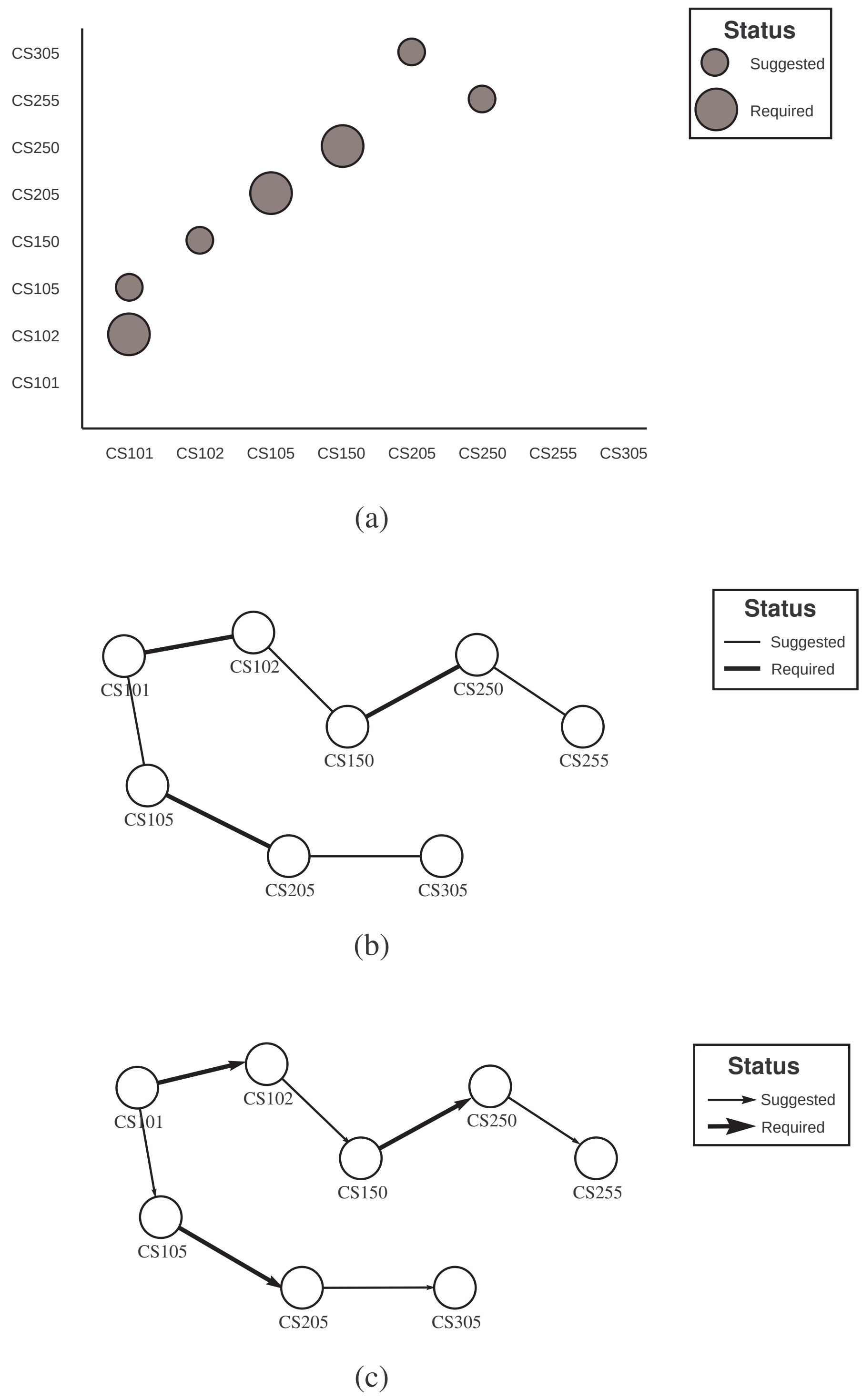


Figure 1.22: Three potential graphical languages for the course prerequisites dataset

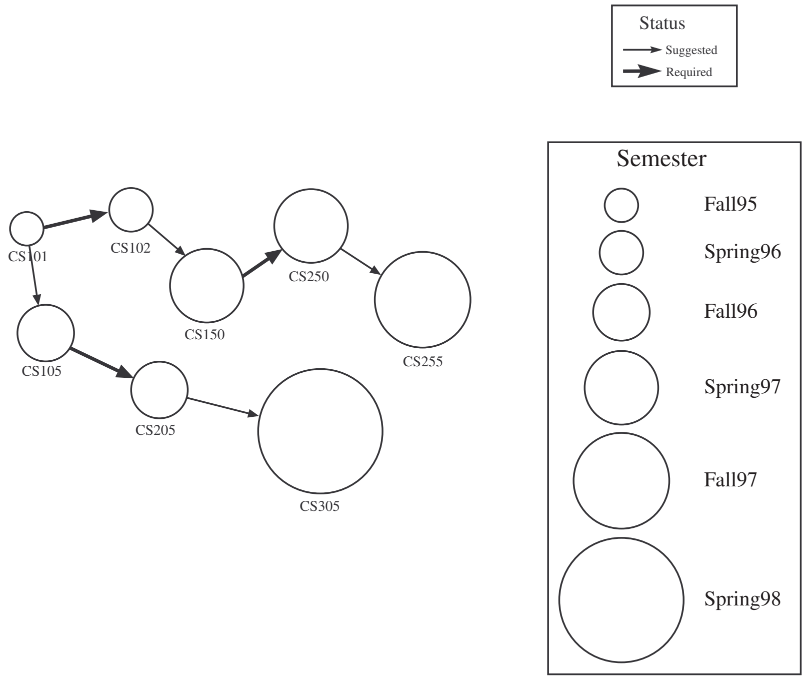


Figure 1.23: Using size to encode semester, a sub-optimal presentation

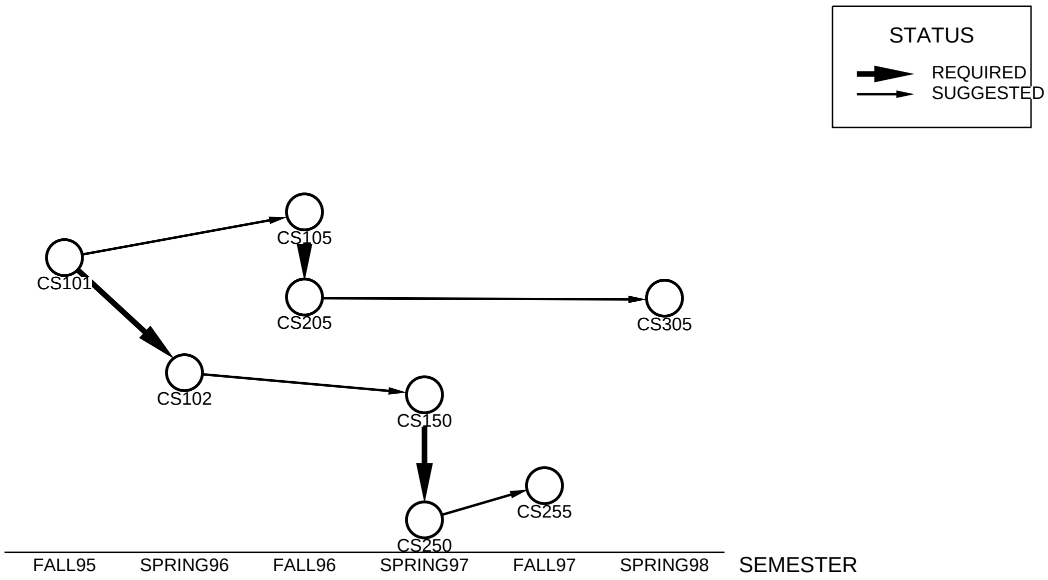


Figure 1.24: A constrained network graph

When the final set of languages has been chosen, it is passed to the finalization module to make sure it is fully constrained and optimized in any way possible. In this case, the finalization module would ensure that the non-semantic perceptual properties of the objects in the presentation are kept uniform–e.g., all circles will be the same size, same shade, etc. After finalization, the set of languages is passed together with the actual data to the layout engine, which produces the display shown in figure 1.24.[13]

1.5 Guide to remainder of dissertation

The remainder of this dissertation is organized as follows:

- Chapter 2 discusses relevant previous research in a number of different fields, including psychology, statistics, and computer science. It describes both prior attempts to build graphic presentation design systems, work that only generally discusses graphic representation, and work that is not directly concerned with graphic representation *per se* but is nevertheless relevant.
- Chapter 3 develops the conceptual framework of my research. This framework defines graphical languages in general terms, and discusses some aspects of a first-principles solution to the problem of finding a graphical language for a presentation. Rather than detailing specific algorithms or knowledge, this chapter sets out a template to be filled in by later chapters.
- Chapter 4 takes the general framework of chapter 3 and makes it operational by providing concrete algorithms and more detailed analyses of particular problems.
- Chapter 5 discusses sources of difficulty in the interpretation of graphic presentations, and presents a small set of broadly applicable design principles for effective interpretation, along with specific ways in which they can be applied.
- Chapter 6 discusses aspects of visual perception relevant to graphic presentation and presenting a number of design principles for perceptual effectiveness.
- Chapter 7 discusses the implementation, AUTOGRAPH, and also addresses general issues in operationalizing the framework and principles described by the other chapters.
- Chapter 8 concludes by highlighting some of the contributions of the dissertation, pointing out some of its limitations, and discussing possible future work.

[13] This figure was actually generated by AUTOGRAPH. Other figures for this example were generated by hand.

Chapter 2

Related Work

AUTOGRAPH is far from the first attempt to create a system for automatically designing graphic presentations. It builds on other systems embodying somewhat different approaches to this problem. In addition, in attempting a first-principles approach, it draws on other research efforts in a number of fields including psychology, philosophy, and statistics. In this chapter, I will describe work that is relevant to designing graphic presentations automatically and work relevant to analyzing and designing presentations from first principles.

The next section of this chapter describes a number of previous efforts to build automatic presentation design systems, as well as other relevant work in the field of computer science. Section 2.2 briefly summarizes some of the limitations of this work, arguing for a first-principles approach. Finally, section 2.3 describes work from a number of other fields that I draw from in attempting such an approach.

2.1 Automatic presentation design systems and related work

Within the field of computer science, there have been a number of efforts to develop complete systems that automatically design graphic presentations of relational data, addressing precisely the problem addressed by this dissertation. There has also been research on systems that address only particular aspects of this problem, and work which is only partially relevant. I will focus here on the most relevant research first, emphasizing the most significant efforts, then describe more briefly some of the other work, including work on scientific visualization and graph drawing.

2.1.1 APT

Mackinlay's APT system (1986a, 1986b, 1988, 1991) was one of the first significant and theoretically well-grounded systems for automatically designing graphic presentations. It introduced the idea of treating graphic presentations as "sentences in graphical languages", which has been used in most subsequent systems. For this reason, it is worth describing in some detail.

APT designs graphical presentations of relational information, given in the form of relation tuples—e.g., `Price(Honda Accord, 5799)`—with additional information specified about the domains of the relation. In particular, domains are characterized as quantitative ranges, ordered sets, or unordered (nominal) sets. Functional dependencies are also noted. APT takes as input this information, a description of the information to display, and a description of the information that can be ignored, and from this input designs a graphic presentation.

APT's framework treats graphical presentations as sentences of graphical languages with precise syntactic and semantic definitions. Mackinlay outlines a set of primitive graphical languages (not all of which are implemented in APT) that make use of various encoding techniques. Organized by encoding technique, these languages are:

- **Single-Position**—horizontal axis, vertical axis
- **Apposed-position**—line chart, bar chart, plot chart
- **Retinal-list**—color, shape, size, saturation, texture, orientation
- **Map** —road map, topographic map
- **Connection** —tree, acyclic graph, network
- **Misc (angle, contain, . . .)**—Pie chart, Venn diagram, . . .

APT uses these primitive languages as part of a composition algebra, which also includes *composition operators* for creating new graphical languages by combining existing ones. As a simple example, an apposed position language such as a plot chart can be combined with a retinal-list language such as color to form a new language where the plotted symbols on a plot chart are colored, expressing some other dimension of the data. This works because the languages APT can combine are "orthogonal" in their conventions. Generally, APT's set of primitive languages and composition operators is not intended to be complete, though it is sufficient for generating a wide range of common graphic presentations.

All graphical languages in APT are characterized according to two sets of criteria: *expressiveness* and *effectiveness*. Expressiveness criteria are used to determine whether a language is capable of expressing some given information. For example, bar graphs can be used to display binary relations $r(x, y)$, representing y using the length of a bar, only when y is functionally dependent on x and is not a nominal domain (since length is normally interpreted as encoding ordered values). Effectiveness criteria are used to decide which sufficiently expressive language is *best* for presenting that information. In APT, languages are ranked based on a ranking of perceptual tasks from most effective to least effective for each of three types of data sets: quantitative, ordinal, and nominal. These rankings reflect human cognitive and perceptual limitations (e.g., how well people can discriminate colors based on hue) as well as the influence of convention.

APT's algorithm for generating presentations is effectively a depth-first search with backtracking. The search is through possible partitionings of the relations to be displayed (i.e., into lower-arity relations), languages to express the relations, and finally possible compositions of the individual languages into a unified presentation.

APT has a number of limitations, including its minimal capabilities for data and goal characterization, which restrict the system's ability to choose appropriate presentations. A more significant limitation of the framework, however, is that it uses a small set of primitive techniques and composition operators not intended to be complete. Indeed, the classification of primitive languages used in Mackinlay's analysis are based more on commonly used techniques than on a bottom-up analysis of what techniques are possible. Furthermore, the framework is not intended to extend to "unconventional" languages, and would be hampered in doing so partly because it is restricted to using primitive languages that are known to be easily interpreted, rather than addressing issues of interpretability in general.

2.1.2 SAGE

A number of related systems both for designing graphic presentations automatically or interactively and for general exploratory data analysis have been designed by the SAGE group at Carnegie Mellon University, led by Steve Roth. Their work has explored various issues in presentation design and use, including integrating text and graphics (Mittal *et al.* 1995; Roth *et al.* 1991), interactively designing graphical languages by drawing (SageBrush (Roth *et al.* 1997; Goldstein & Roth 1994)), interactively designing presentations by recalling appropriate past presentations (SageBook (Chuah *et al.* 1995a)), coordinating the use of multiple interactive presentations (Visage (Kolojejchick *et al.* 1997; Roth *et al.* 1996)) and allowing users to modify three-dimensional presentations interactively (SDM (Chuah *et al.* 1995b; Chuah *et al.* 1995c)). I will discuss only the most relevant research here—the original SAGE system for automatically designing graphic presentations (Roth *et al.* 1991; Roth & Mattis 1991; Roth & Mattis 1990)—though parts of the other work are also somewhat relevant.

In many ways, SAGE is fairly similar to APT. Its fundamental framework is the same: primitive graphical languages are chosen to represent relations and combined to produce complete presentations. SAGE uses a larger number of primitive techniques and composition operators than APT, but its main improvements come in the areas of data and goal characterization.

In addition to distinguishing between domains based on their set ordering (i.e., nominal vs. ordinal vs. quantitative), SAGE makes a distinction between quantities and coordinates. This distinction is important in helping to avoid graphing project dates on a bar chart, for example, since the lengths of bars in bar charts imply not just ordered values, but that a larger value means *more* of something. Relations between data domains are also characterized in greater detail, capturing distinctions between one-to-one mappings and one-to-many mappings, for example, as well as other distinctions. SAGE also allows relationships among relations to be expressed. For example, the start date, end date, and duration of an activity are related. This sort of knowledge, necessary for designing certain types of graphs, isn't utilized in APT. SAGE has a set of complex data-types which can be used to characterize such relationships.

Goal characterization is also more sophisticated in SAGE than in APT. For example, a user who is more interested in accurately determining individual values than getting an overview of a set of values might prefer a table over a graph. SAGE has a small set of goal types, and also allows the specification of goal-relatedness information, which is used when segmenting information to create multiple presentations.

Despite its improvements over APT, SAGE retains some of the same fundamental limitations—i.e., a reliance on predefined techniques for presenting information and the lack of a framework for analyzing unconventional or novel methods of presentation. Recent research by the SAGE group has focused on extending the ways the framework can be used, rather than on improving the underlying framework.

2.1.3 BOZ

The central notion underlying Casner's BOZ system (1991, 1990) is that without knowing the purpose for which a graphic presentation will be used, it is impossible to determine how useful the presentation will be. That is, a characterization of the data to be displayed is not sufficient for determining how to display it; the designer or design program must know in detail the goals of the user.

While goal characterizations are used by SAGE to choose effective presentations, they are fundamental in Casner's system. BOZ works primarily by translating a detailed description of the user's task, expressed as a series of "logical operators", into a corresponding series of equivalence classes of visual operators. It then decides how best to represent the data visually (i.e., which graphical languages to use) in order to facilitate the use of these visual operators. For example, determining from a chart showing airline flights the layover between two flights is a logical "difference" operator (i.e., a subtraction of one time from another), which can be translated into any of number of different visual difference operators, such as determining the horizontal distance between two objects. A user's task in BOZ can be as complex as, "Find a pair of connecting flights that travel from Pittsburgh to Mexico City, with a layover of no more than four hours. Both flights must be available, and the combined cost of the flights cannot exceed $500." Such a task would be expressed as a long series of related logical operators, and would be translated into a corresponding series of visual operators. The advantage of this method is that it theoretically allows very precise estimates of the effectiveness of different presentations, as long as the sequence of logical operators and effectiveness of the visual operators are modeled accurately.

Like APT and SAGE, BOZ uses a set of primitive graphical languages. Although these are simpler than the languages used in APT (i.e., they represent a more detailed breakdown of graphic conventions), BOZ's graphical languages can only be combined in limited ways. Another drawback of BOZ is that it requires a very explicit description of not only the user's goals, but also of the operations the user will use to achieve these goals. This description, which greatly resembles a procedural computer program, must be generated by hand.

Later work by Casner (1993) addresses some of these issues by proposing a more complete framework for composing graphic elements. This framework is not integrated with the framework used for BOZ, however, and ignores some issues of logical expressiveness. Neither this work nor Casner's earlier work addresses the subject of interpretability of graphical languages.

2.1.4 ANDD

A number of papers by Joe Marks and various co-authors (Dengler *et al.* 1993; Kosak *et al.* 1994; Ryall *et al.* 1996; Ryall *et al.* 1997) address problems related to automatically and/or interactively drawing network graphs with specified *visual organization features* (VOFs), exploring different algorithms and interaction techniques within the context of a system called ANDD. Different VOFs (e.g., symmetry, alignment, hub-shape, T-shape, enclosure) can be applied to different groups of nodes in their system, producing very well organized and aesthetically appealing network graphs that are capable of visually encoding various semantic aspects of the data being presented. Most of the work on ANDD is not concerned with the problem of automatically mapping semantic distinctions to VOFs, or with evaluating the effectiveness of different types of graphs. However, its general framework of mapping semantics to visual features is similar in spirit to part of my framework, and the various implementations go far beyond the current capabilities of AUTOGRAPH for creating network graphs.

2.1.5 AVE

AVE (Golovchinsky *et al.* 1995; Reichenberger *et al.* 1995; Kamps *et al.* 1995) is an automatic presentation system for semantic networks. Unlike most of the automatic presentation systems described above, it uses basic components to represent information rather than a predefined set of graphical languages. In this way, it somewhat more similar to the approach described in this dissertation. However, it is restricted to representing combinations of binary relations, rather than relations of any arity, and designs presentations by matching a fixed set of mathematically defined relation types (e.g., irreflexive order relations and symmetric relations) to similarly mathematically characterized graphic relations, where graphic relations include both lines and arrows between objects (seemingly only rectangles) and visual properties of the objects. AVE's framework does not address issues of completeness in relation types or in methods of graphic presentation—i.e., it does not attempt to be fully general—and does not address general issues of language expressiveness.

2.1.6 IMPROVISE

IMPROVISE (Zhou & Feiner 1996; Zhou & Feiner 1997; Zhou & Feiner 1998) is a system that automates visual discourse generation, including graphic presentations of information as well as visual explanations. The foci of the research include data characterization and task characterization (for translating high-level intents into visual tasks and visual tasks into low-level visual techniques). Although this work addresses a number of different issues and different types of presentations, including three-dimensional diagrams of real objects, it seems to attempt to be complete by virtue of large and complex hierarchies for describing data, tasks, etc., rather than by a substantial analysis of the space of possible graphical languages.

2.1.7 Scientific visualization research

Over the last decade or so, the field of scientific visualization has blossomed. Although all of the work in this area is relevant to the subject of visual representation in general, much of it is not of particular relevance to this dissertation, for a variety of reasons. First, much of the research, especially in recent years, is domain or application-specific. Furthermore, in many scientific visualization problems, there is a looser relationship between what is represented and the details of how it is represented. A fair amount of research concerns transformations of data or novel rendering techniques; such research lies outside the range of concerns of this thesis. Finally, most work in scientific visualization presumes three-dimensional and animated or interactive output, which places the representations it develops beyond the scope of this dissertation in some ways. Nevertheless, some of the work in this field bears mentioning for its relevance to the problem addressed here.

On the framework side, Wehrend and Lewis (Wehrend & Lewis 1990; Wehrend 1990) provide a classification of visualization methods based on task and data type. Although this classification is not developed enough to be the framework for a completely automatic visualization system, the taxonomy of goals is fairly sophisticated. A less precise classification of visual representations is also described in Lohse *et al.* (1991).

Many systems for scientific visualization provide modular environments for visualization, which implicitly define a space of possible graphical languages. Although most of these systems define this space using standard high-level techniques for scientific visualization, and most do not involve automatic design, a number are of particular interest because they provide user guidance based on perceptual or other principles. The GADGET system (Fujishiro *et al.* 1997) uses the Wehrend taxonomy of visualization techniques to suggest appropriate visualizations (i.e., predefined visualizations from appropriate categories) given user goals and the type of data being represented. Some extensions to IBM's Data Explorer (Bergman *et al.* 1995; Rogowitz & Treinish 1993) use rules based on perceptual knowledge to help guide users in selecting ways to map data onto visual dimensions. For example, the system can help the user to design colormaps which exaggerate certain details or which ensure isomorphic mappings of the data, depending on the user's task.

The VISTA system (Senay & Ignatius 1994; Senay & Ignatius 1991) goes even further than the work on Data Explorer in some ways, suggesting ways of extending Mackinlay's framework to three-dimensional scientific visualization, using a large variety of rules for guidance. These rules can also be used for interactive selection and combination of visualization techniques. While VISTA brings new techniques to Mackinlay's framework, and new means of analyzing effectiveness, it does not address questions of how to systematically analyze the space of possible presentation methods.

2.1.8 Graph-drawing research

A large body of research addresses the problem of graph drawing—i.e., of producing a presentation given a basic design for the presentation and the data to be presented. Most of this work is concerned only with drawing standard undirected network graphs, much of it building on the spring model of graph layout first described by Eades (1984). However, some graph drawing work (such as Frick *et al.* (1996),He & Marriott (1996) and Wang & Miyamoto (1995)) extends to drawing network graphs with constraints on the absolute or relative positions of the nodes or the angles of the lines connecting the nodes. The work on ANDD and on AVE described earlier also addresses this problem.

Although graph-drawing research is not concerned with the central problem of this thesis—i.e., designing methods of presenting data—it does relate to a subproblem that must be solved by any system for automatic presentation design: laying out the presentations it has designed. The systems that incorporate constraints into a spring model layout of network graphs were an influence on the design of AUTOGRAPH's layout manager, described in chapter 7.

2.2 Limitations of previous work

Although previous work has addressed many aspects of the automatic graphic presentation design problem, a number of issues have not been addressed in a sufficiently systematic and thorough manner.

First, no work has attempted to present a detailed framework capable of describing anything close to the complete space of possible graphic presentations. Most of the systems for automatically designing graphic presentations start with a set of predefined graphic techniques and ways of combining these techniques. Those that do not start with predefined techniques tend to be *ad hoc*, having been designed from the top-down to recreate conventional presentations, rather than analyzing the space of possible presentations from the bottom up. The work of Casner (1990, 1991, 1993) comes closest to providing a framework for a complete, fine-grained analysis of graphic presentations, but even it is limited in the way it analyzes the use of compound objects in presentations.

Secondly, although many frameworks have addressed perceptual issues in the effectiveness of graphic presentations, no framework has adequately addressed the issue of interpretability for graphic presentations—i.e., how to determine if a graphic presentation will be confusing. They have either relied on the known interpretability of their predefined graphic techniques or ignored the issue entirely.

If the ultimate goal of work on automatic presentation design is to create a system capable of creating the widest possible range of effective presentations, both of these issues must be addressed. It does not seem possible to address these issues simply by extending previous approaches. Rather, it is necessary to address issues of graphic presentation from the ground up—i.e., from first principles—both logically and psychologically. This is the approach pursued in this dissertation.

2.3 Frameworks and principles for graphic presentation

To approach the automatic presentation design problem in a first-principles manner, we will ultimately need both a general framework for systematically analyzing and describing presentations and a set of principles to flesh out the framework. A fair amount of work in a variety of fields has touched on one or the other of these problems. Although it is not always easy to pigeonhole this work, it can be roughly divided into two main categories: work on logical or psychological frameworks for describing presentations and work directly or indirectly describing interpretive or perceptual design principles. In sections 2.3.1 and 2.3.2, I will describe work on various logical and psychological frameworks, respectively. In section 2.3.3, I will describe research into design principles. Finally, in section 2.3.4, I will briefly discuss the work of Edward Tufte, an important writer on graphic presentation.

2.3.1 Logical frameworks

The work of Bertin (1981, 1983) is a systematic and thorough attempt to provide a theoretical framework for graphic presentation. Bertin discusses graphs, charts, and maps in terms of spatial and retinal variables—where retinal variables include size, value, texture, color, orientation and shape— similar in concept to the perceptual properties of my framework. He analyzes the utility of different variables for presenting different types of data domains, distinguishing between nominal, ordered, and quantitative domains, and also discusses the concept of different levels of readings of presentations (e.g., basic and summary). These distinctions are incorporated in the framework of Mackinlay and others, including my framework. Bertin's work is partially limited by his reliance on intuition rather than on empirical evidence from psychology, and his analysis seems more suited to simple graphs such as scatterplots and bar graph than to graphs where multiple objects combine to represent complex information. Nevertheless, it has greatly influenced most systems for the automatic design of graphic presentations, including the one described in this dissertation.

Another notable attempt to provide a logical framework for describing graphical presentations is Goodman's *Languages of Art* (1968). Goodman lays out a framework for discussing visual representations, intended to be even more general than the framework presented here. Although the framework is correspondingly more abstract and thus insufficient for my purposes without modification, it in many ways influences the framework for describing graphic presentations which I present in chapter 3. Goodman's work is described in more detail there.

2.3.2 Psychological frameworks

Approaching graphic presentations from a different angle, cognitive psychologists and other cognitive scientists have theorized about how people understand graphs and other displays of information. Several psychologists and cognitive scientists have attempted to describe general psychological frameworks for how people understand and extract information from graphic presentations.

Pinker (1990) presents an information-processing theory of graph understanding that provides a good general framework, similar in many respects to the framework used in this dissertation. In this model, early visual processes construct an initial description of a presentation which is transformed by encoding processes into a more sophisticated description in short-term memory. This description is matched with *graph schemas* in long-term memory, which describe procedures for decoding information from different graph types, and an appropriate schema is used to extract information from the graph. Pinker makes the important observation that *configural properties*—i.e., gestalt properties of a presentation—can often be seen as encoding abstract properties of the data. This notion of configural properties influenced the development of the framework presented in this dissertation. I will discuss configural properties in some detail in chapter 6. One key difference between Pinker's model and mine, which will become more clear in chapter 5, is that his model does little to explain how unconventional graphs can be interpreted, relying on a fairly vague notion of a "general graph schema" rather than attempting to describe general principles for interpretation.

Lohse (1991b, 1991a, 1993) also provides a model—similar to Pinker's, but much more explicit—of extracting information from graphs, implemented in a computer program called UCIE that predicts the effectiveness of different presentations. Along with the work of Casner (described in section 2.1.3), this research attempts the most fine-grained analysis to date of how graphic presentations are used. It is limited primarily by its reliance on graph schemas similar to those used in Pinker's model, restricting its predictions to a few basic types of graphs.

Eschewing a specific model of graphic comprehension, Kosslyn (1989) provides a general framework for describing presentations and a broad analysis of psychological (and other) principles underlying successful and unsuccessful graphs. Although his framework is very different from the framework presented in this dissertation—making distinctions between background, framework (i.e., axes and legends), and specifiers (the objects of the presentation actually encoding information) and between graphs, diagrams, and maps—it is similiar in that it emphasizes the logical and psychological nature of constraints on graphic presentation, and different levels of analysis for presentations.

In addition to the value of his framework, many of Kosslyn's principles for analyzing presentations are generally useful. Some are similar to the ones described in chapters 5 and 6, and Kosslyn's analysis of the factors underlying various principles influenced my analyses in those chapters. Kosslyn's principles are stated and organized in rather abstract terms, however, and are thus more useful for human designers than as a complete basis for an automatic system for designing presentations. In other work, Kosslyn (1994) provides much more specific design guidelines for specific types of graphs in specific situations, but also suggests some general design principles for constructing perceptually effective and easily interpreted graphs.

2.3.3 Research relevant to establishing perceptual and interpretive design principles

The framework used in this dissertation divides psychological design principles into two classes: interpretive and perceptual principles. Interpretive principles address the question of how easily a viewer can determine the way information is encoded in a presentation. Perceptual principles are primarily concerned with the speed and accuracy with which a person can make the visual discriminations needed to extract information from the presentation. I will briefly discuss here research relevant to each of these types of principles.

Research on representational ability

I am not aware of any research directly addressing issues of interpretability in graphic presentations. However, some research into the development in children of representational ability is highly suggestive.

diSessa *et al.* (1991) studied a group of sixth-graders given the problem of finding a visual representation for a complex event—a car driving along quickly in the desert, slowing down to stop at a cactus, then driving off slowly. Over a period of days, the children came up with a number of representations, demonstrating in the process what the researcherers refer to as “meta-representational expertise”. As analyzed by the researchers, the criteria the students used in evaluating potential representations included *transparency* (i.e., the ability to be used with little explanation), *homogeneity* (consistency in representing motion and stops) and *economy* (lack of redundancy). Some of the criteria are very suggestive as possible design principles for interpretability.

Tversky (1995) summarizes some other research on children’s development of graphic representations, comparing it with the historical development of certain types of visual representations. She makes a number of observations about the use of of spatial grouping and delineation to convey category and the use of spatial arrangements to convey order, and also suggests reasons underlying certain conventions. These reasons also underly some of the interpretive principles discussed in chapter 5.

Perceptual research

At a general level, a vast amount of psychological research has been done on visual perception. While this body of research as a whole is not directly applicable to graphic presentation understanding, it describes important constraints on what people are capable of perceiving and how well different perceptual distinctions can be made, both critical factors in designing graphic presentations. I will defer a discussion of the most relevant work of this sort until chapter 6, which discusses perceptual principles.

Some of the best empirical evidence concerning the effectiveness of different types of perceptual properties for presenting data is provided by the work discussed in Cleveland (1985) and Cleveland & McGill (1984). This work explores various issues relating to graphing data, emphasizing the importance of what the authors term “graphical perception”. Most importantly, for purposes of developing an automatic presentation design system, they report on a series of experiments testing the accuracy with which people perform various perceptual tasks necessary to read different types of graphs. These experiments form part of the basis for my rankings of perceptual operations, described in chapter 7. The results of the experiments themselves are discussed briefly in chapter 6.

Healey (Healey 1996) summarizes research on preattentive processing, extends this research, and discusses ways of using preattentive processing to help design effective visualizations. His discussions include techniques and guidelines for choosing colors and for using hue and orientation simultaneously to represent data. While these techniques and guidelines have not been incorporated into the current version of AUTOGRAPH, they could be valuable in a later version.

2.3.4 The work of Edward Tufte

A discussion of work on graphic presentation would not be complete without mentioning the work of one of the best-known writers on the subject of data graphics, Edward Tufte. His three books on the subject (Tufte 1983; Tufte 1990; Tufte 1997) contain a large number of examples of both the best and worst in graphic presentations, along with trenchant analyses of what makes them good or bad, categorizations of presentations into different types, and maxims for design.

While Tufte's work is generally insightful, it has shortcomings as a basis for implementing an automatic system for designing presentations, since it doesn't provide a general framework for graphic presentations, favoring instead loosely constructed categories and compelling examples. In addition, some of Tufte's principles, while reasonable, tend to be heuristic rather than theoretical. For example, his oft-cited "data-ink principle" suggests that the ratio of data presented to the ink used to present it is a good indicator of how effective a presentation will be. While this might be true, generally, it is likely to be so only to the extent that using less ink enhances the effectiveness of the perceptual operations a viewer undertakes in extracting information from the presentation.

Nevertheless, Tufte's work serves as an excellent reference on the variety of visual representations and on their history, as well as making compelling arguments for their usefulness.

2.4 Summary

This chapter described previous work related to the goals of this dissertation. There have been a number of previous attempts to create systems for automatically designing graphic presentations. Most attempts have started with some notion of predefined techniques (or graphical languages) for presenting data. None of these research efforts have really attempted to provide a framework for systematically analyzing the complete range of possible presentations in any detailed way. Furthermore, no research effort has sufficiently addressed the issue of the interpretability of graphic presentations.

In attempting an alternative, first-principles approach, this dissertation draws on a number of logical and psychological frameworks for analyzing presentations and how people use them. It also draws on psychological research and other writings on graphic presentation.

Chapter 3

A framework for a first-principles approach to presentation design

In the last chapter, I discussed some of the limitations of presentation design systems that are based on selecting and combining predefined techniques (graphical languages), and argued for a first-principles approach. I did so without clarifying what such an approach would look like, and what precisely I meant by "first principles". These are the issues addressed by this chapter.

It might be possible to build a first principles automatic graphic presentation design system in an *ad hoc* fashion. After examining a large enough number of existing graphic presentations, one could infer a set of design decisions the system would have to make to construct new presentations, and by examining the advice available in books on graphic presentation, one could derive an unstructured set of principles that could guide these decisions. This approach is unlikely to be satisfactory, however. First, a system designed in this haphazard manner is bound to have "gaps" in the space of presentations it will be capable of creating, and these gaps will be arbitrary and unpredictable. Second, the principles that can be gleaned from books and common sense may be difficult to apply to all the designs the system creates, either because they are too general (e.g., "be consistent in the way you encode data") or because they are too specific (e.g., applying only to the choice of a bar graph vs. a line chart in particular situations, but not addressing all applications and types of presentations). Finally, if these principles fail to guide the system in making good choices for how to present data, it may be difficult to determine precisely what is wrong with them.

Instead of an *ad hoc* approach, it is desirable to be able to discuss *systematically* the principles governing graphic presentations and their use. To enable this discussion, this chapter develops a framework for discussing graphic representation consisting of a concrete and very general notion of the space of all possible graphical languages, and a simple but specific model of how graphic presentations are perceived and interpreted. This framework will allow me to clarify and delimit both the technical problems that a first-principles presentation design system must solve and the types of knowledge it must use. I will not discuss algorithms here, or details of psychological knowledge; rather, I will use the framework to provide a logical outline to be filled in by later chapters. I will further argue that this outline is complete—i.e., that the technical solutions and knowledge specified here, if supplied, will be sufficient to allow the construction of a working system.

Before developing the framework, it will be useful to be more precise about some desiderata which will shape it:

- **Generality** The framework should be capable, in theory, of describing *any* type of graphic presentation, including both conventional presentations such as bar charts and scatter plots, as well as novel presentations. In practice, of course, it is unlikely than a single working implementation could generate any presentation imaginable. But the limitations of an implementation should be the result of limitations of the knowledge and computational resources available to the system, rather than those of the underlying theoretical framework.

- **Knowledge demarcation** The framework should be capable of accommodating various knowledge and design guidelines already determined by psychologists, statisticians, graphic designers, etc. in their analyses of graphic presentations, or of explaining these guidelines in terms of more general principles. Furthermore the framework should provide a means of *organizing* such diverse knowledge and accumulated wisdom, as well as *delimiting* the knowledge that is needed to make decisions about presentations.

- **Operationalizability** The framework should suggest a means of exploring the space of graphic presentations and finding an appropriate one for some data or class of data to be presented. That is, it should suggest a means of implementing an actual presentation design system.

In the next section of this chapter I will develop a conceptual framework for describing graphic presentations and the ways they are used, ending with a precise definition of what a graphical language is in this framework. In section 3.2, I explain how this framework can be used to describe an explicitly-defined space of possible languages. Finally, in section 3.3, I analyze briefly the logical and psychological issues that affect the usability and appropriateness of different languages for different types of data, thus outlining the issues that will be addressed in future chapters.

3.1 A framework for graphic representation

In this section I develop a framework for discussing graphic representation out of the union of psychological and formal principles. Graphic representation is fundamentally a process of encoding information with marks on paper or on a computer screen, so the primary purpose of the framework will be to allow a description of what such encodings are and how they work. Because the relationships between information and the way it is displayed can vary widely, I develop the framework in several stages. First, I describe a simple model of how people process graphic presentations. This model will develop a vocabulary with which to discuss presentations at three different levels of description, and a conceptual reference point for analyzing encodings. Using this model, I then develop the notion of *encoding conventions*, systematic descriptions of how information can be mapped onto perceptual features of a presentation which can then be mapped onto an objective graphic description sufficient for drawing the presentation. I then refine this notion by making some further assumptions about the sorts of encoding conventions that will be usable by humans. Finally, I use the notion of encoding conventions to define precisely what a graphical language is in the framework.

3.1.1 A simple psychological model of graphic comprehension

Clearly, no full account of how humans extract information from graphic presentations can be given without specific psychological knowledge of human perceptual and cognitive capabilities, gleaned from both empirical research and general observation. Before discussing particular psychological knowledge, however, it is necessary to make some fundamental assumptions about the outlines of the processes involved. These assumptions will lead to a vocabulary and a framework for discussing presentations and graphical languages.

One natural starting point for a model of how people extract information from graphic presentations is the division between perception and interpretation. It is necessary both to perceive the information in the form it is encoded in a graphic presentation and to interpret these perceptions correctly, and these processes seem conceptually separable.[1] I will assume, therefore, a general model of viewing and extracting information from a graphic presentation that looks like figure 3.1. Perception builds an intermediate description of the perceptual distinctions that can be made in the graphic presentation; interpretation transforms this perceptual description into a cognitive-level description of the information encoded in the graph. This model is similar to those of Pinker (1990) and Lohse (1993), both discussed in chapter 2. Although it is simplistic—among other ways in its assumption that perception and interpretation are purely bottom-up, independent processes—it will nevertheless be a useful starting point.

The graphic level

Since graphic presentations must ultimately be drawn on paper or a computer screen, they must be describable in some theory-neutral way. The details of the graphic-level description are not very important in any theoretical sense. The graphic level will serve as the output for a presentation design system, and also as a conceptual reference point for a discussion of perception, so I will choose a language for description based on its convenience for these purposes, without loss of generality.

Although the graphic level could be thought of as a bitmap, I will assume that it is a set of statements in a language more or less equivalent to a subset of PostScript, describing discrete marks on a page. This is convenient for describing virtually all conventional graphic presentations. This dissertation will be minimally concerned with media issues (e.g., the resolution of the screen or printer), so it will not be important to distinguish between an abstract model of what is drawn and what is actually drawn. While this distinction has

[1] The assumption that perception can be described in terms of information-processing runs counter to the theories of some psychological theorists (notably Gibson (1979)), but it is consistent with the central tenets of cognitive science.

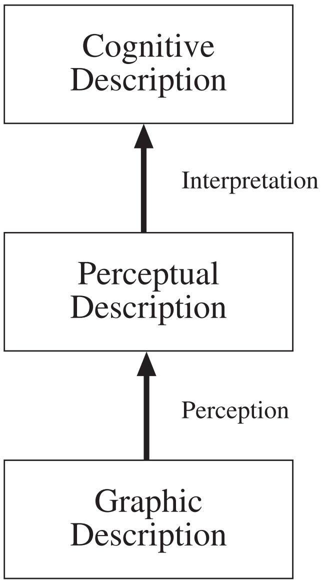


Figure 3.1: A basic psychological model of graphic presentation understanding

been made by some researchers (e.g., Mackinlay (1986b)) and has the potential to be useful—e.g., restricting the use of color when designing for a black and white printer—I believe that most of its value is practical, rather than theoretical, and can be achieved without making it an explicit part of the framework.

To be more concrete, I will assume that the description of a presentation at this level consists of *graphic objects* which are instances of different types of *graphic primitives* with different *parameter* values. In the current version of the AUTOGRAPH implementation, the set of possible primitives consists of *lines*, *arrows*, *rectangles*, and *circles*.[2] Each primitive has a fixed set of orthogonal parameters, and each instance must have a value for each parameter, which must either be a scalar value in a given range or a choice of one value from a discrete set (e.g., the set of colors *red*, *blue*,...), depending on the parameter.[3] For example, the graph of figure 3.2 can be described as a set of circles and lines, the circles having the set of parameters {*x,y,radius,label,red,green,blue*} and the lines having the parameters {*ep1hpos, ep1vpos, ep2hpos, ep2vpos, thickness, red, green, blue*}. The parameters *ep1hpos*... *ep2vpos* refer to the horizontal and vertical positions of the two endpoints. *Red*, *green*, and *blue* refer to the components of a color—fill color for a circle, line color for a line—with 0,0,0 indicating black and 255,255,255 indicating white. Such a description would look like:

circle(300,400,1.0,"Paris",255,255,255)
circle(500,480,1.0,"New York",255,255,255)
...
line(330,410,480,460,3.0,0,0,0)

[2] I will refer to the AUTOGRAPH implementation on occasion as one concrete example of how the framework outlined in this chapter might be used. The framework has been designed to be more general than this particular implementation, however.

[3] I will consider textual object labels to be parameters of some graphic objects, for convenience, and consider their values to be drawn from a very large discrete set (e.g., strings of characters under a certain length). The practical advantages of this outweigh the theoretical inelegance.

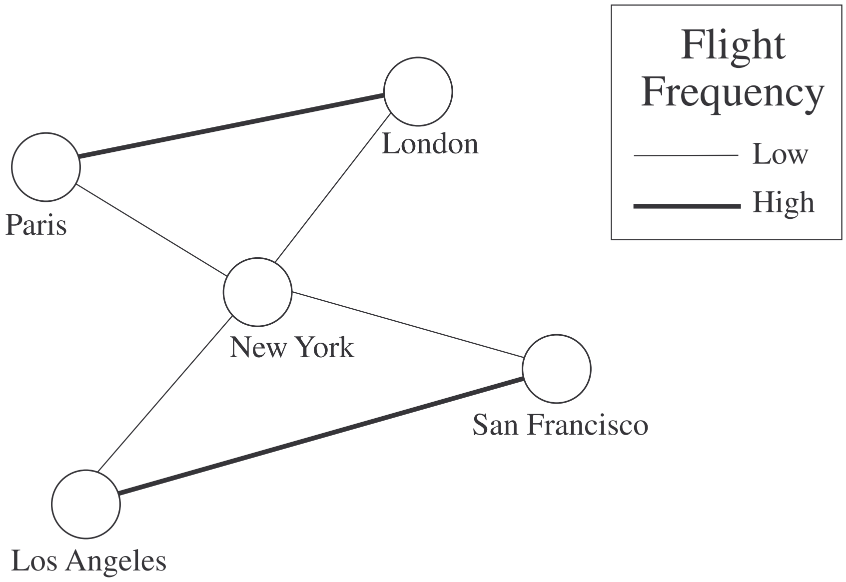


Figure 3.2: A network graph

...

The perceptual level

For purposes of understanding how graphic presentations are used, an objective description of a presentation is not nearly as useful as subjective ones—descriptions of what perceptual distinctions a human viewer will make when looking at the presentation. Understanding the potential perceptual distinctions a viewer is capable of making is crucial to understanding the potential range of methods for visually presenting data. If faced with a group of circles on a piece of paper, for example, a viewer will be able to estimate the positions of the circles on the page, the colors and sizes of the circles, the distances between the circles, which circles are largest and smallest, which are lightest and darkest, which circles, if any, group together, etc. Conceivably, any of these features could be used to encode information. If it is important to be able to describe the largest possible variety of encoding conventions, the perceptual-level description language should be capable of noting any perceptual distinctions a viewer could make.

The perceptual level describes presentations in terms of *perceived objects* which are instances of different *object classes*. Each object class has a set of *perceptual properties*, monadic functions which map from the domain of objects to values in *perceptual dimensions* (scalar values in a given range or elements from a finite set of values). These features of perceptual-level descriptions are analogous to the entities for description at the graphic level. Unlike the graphic level, however, the properties of perceptual level object classes are not constrained to be orthogonal. For example, the properties of an arrow can include the length and direction of the arrow as well as the horizontal and vertical positions of its head and tail, even though these properties are interrelated and overdetermine the underlying graphic parameters of the object. Thus, a presentation involving arrows may be described in multiple ways, with different descriptions being appropriate or convenient depending on which perceptual features are being used to encode information.

Since relationships between pairs or groups of objects might also be semantically meaningful, the perceptual level needs additional entities for description. *Perceptual relations* are predicates over a pair or group of objects (e.g., specifying that a line touches a circle or that one rectangle contains another). *Perceptual functions* map multiple objects to values in perceptual dimensions (e.g., specifying that the distance between two circles is 1.5 inches).

The perceptual level also allows groups to be described as *compound objects*, made up of multiple *component* objects (some of which may be part of other compound objects). All objects that are not compound will be referred to as *atomic*. For example, it is possible to talk about a compound object composed of a pair of circles and a line touching each of them. The value of talking about compound objects is that it will allow an analysis of the compositional nature of graphical languages—i.e., how different elements of a presentation combine to express information. The class of a compound object will be implicitly defined by the classes of its component objects and the perceptual relations between these components; perceptual functions between components of a compound object are analogous to properties of that compound object (e.g., the distance between a pair of circles, considered as a compound object, can be considered a property of that pair of circles).[4]

Since I am allowing class and perceptual properties to describe all aspects of single objects, and using perceptual relations and perceptual functions to describe aspects of multiple objects, this set of entities—classes, properties, perceptual functions, and perceptual relations—should be sufficient for allowing the expression of any relevant features of a graphical language. Note that the notion of compound objects with properties even allows a discussion gestalt or *configural* properties of an entire presentation, such as the shape and density of a point cloud in a scatter plot. I will defer most of the discussion of configural properties until chapter 6, however.

For the purpose of discussing encoding conventions, I will typically use the perceptual level to describe only perceptually salient and/or semantically relevant features of a presentation, as opposed to every possible perceptual distinction a viewer might be able to make while looking at the presentation. For example, in the graph of figure 3.2, a convenient perceptual-level description might look like:

circle(c1,label = Paris)
circle(c2,label = New York)
...
line(l1, thickness = 3)
...
touches(l1,c1)
touches(l1,c2)
...

[4] I will not make much of this analogy, but it is useful to bear in mind because compound objects can often be used in a way similar to atomic objects.

compound-object([l1,c1,c2])
...

In this description, circles and lines are perceptual object classes, *c1*, *c2*, and *l1* denoted instances of these classes, and *label* and *thickness* are perceptual properties of circles and lines, respectively.

Since the positions of the lines are not meaningful (as long as they touch the relevant circles), it is not necessary to describe the positions of the endpoints of the lines, nor their orientations and lengths. If the encoding conventions were different—e.g., if the lengths of the lines encoded distances between the cities—the perceptual-level description would include the necessary information.

The object classes used in the current version of AUTOGRAPH—*circle*, *rectangle*, *line*, and *arrow*—correspond precisely to the classes at the graphic level, though this need not generally be the case.

For the moment, I will refrain from discussing the types of perceptual dimensions, relations, functions, and the way different properties of an object can be interrelated. These will be discussed in detail in section 3.3.1.

The cognitive level

The *cognitive* level describes a mental representation of the information encoded in the graph. I will make minimal assumptions about the exact psychological nature of this representation, merely treating it as equivalent to the information that can be extracted from the graphic presentation. This will enable a discussion of how this information corresponds to features of the perceptual-level description.

Primarily, then, the cognitive level of description includes the data given as input to the presentation design system: a set of relation tuples $R = \{R_1, R_2, \ldots, R_n\}$, where R_i is a tuple of arity k is defined over some fixed set of domains $D_1 \ldots D_k$, not necessarily unique (i.e., D_i may equal D_j, $i \neq j$), and where each domain is a discrete set or a continuous range of values (e.g., the set of cities {New York, San Francisco, Los Angeles, Boise} or the range of temperatures from 20 to 60 degrees Fahrenheit). In the graph of figure 3.2, the cognitive-level description would consist of the dataset encoded by the graph: {*flight-frequency(Paris, London, high), flight-frequency(London, Paris, high), flight-frequency(Paris, New York, low), flight-frequency(New York, Paris, low),*...}. When it will not be confusing, I will sometimes notate such a dataset in the form: {<*Paris, London, high*>, <*London, Paris, high*, ...>}.

Abstractions over the information in the tuples may also be represented at this level (e.g., information about trends or clusters in the data). I will generally not try to characterize the content of this information explicitly, since in graphic presentations this information is not usually encoded explicitly, but rather emerges through the presentation of other information. Nevertheless, this idea is useful for understanding the importance of the configural properties of a presentation. For example, one can talk about the shape of a point cloud in a scatterplot representing the correlation between two domains of a dataset.

Some of the entities used in describing the different levels of description are summarized in table 3.1.

Level	Entity/Component	Example
Cognitive Level	tuples/relations	$<$ *Paris,New York* $>$
	Domains	*City*
Perceptual Level	Perceptual Objects	*circle*
	Properties	*hpos, vpos, size*
	Perceptual Relations	*touches, above*
	Perceptual Functions	*distance*
Graphic Level	Graphic Objects	*circle*
	Parameters	*x,y,radius*

Table 3.1: Entities for graphic, perceptual, and cognitive levels of description

Limitations of the model

The simple model described above overlooks a number of issues. Two of these are particularly notable:

- **The temporal nature of perception** The perceptual level represents an abstraction of what a viewer can perceive in the graph over time, rather than reflecting the sequence of perceptual operations a viewer must execute in order to extract information. These issues, which are central in Casner (1991) and also in Lohse (1993), will be treated in my framework only as they are relevant to perceptual effectiveness, and will be discussed in chapter 6.

- **Inference** In the discussion above, I have assumed a process where all information is extracted from a graphic presentation through interpretation of what is perceived in the graph. Although I have not yet discussed the process of interpretation, as I will develop this concept it will consist of a fairly direct decoding of a perceptual-level description, not including arbitrary cognitive processing. Such cognitive processing can be important in interpreting certain graphic presentations, however. For example, if two bars are stacked one on top of the other in a graphic presentation, and the length of the bars encodes some numerical data values, the sum of the values will have an explicit perceptual representation in the length of the composite bar. In my framework, this is a property of a compound perceptual object (see fig 3.3.a). If the two bars are placed side by side, however, the sum is not represented explicitly. It can only be calculated by cognitive processing of the values extracted from the graph (see fig 3.3.b). While I will in general not address this type of cognitive processing, issues relative to inference seem largely orthogonal to the issues addressed here, and could be introduced into the framework at a later date.[5] The result of this limitation is that my framework will be more restrictive in the set of potential methods of representation that it will consider for a given dataset.

[5] For the beginnings of an approach to inference, see Casner (1991). Pinker (1990) also discusses inference, briefly.

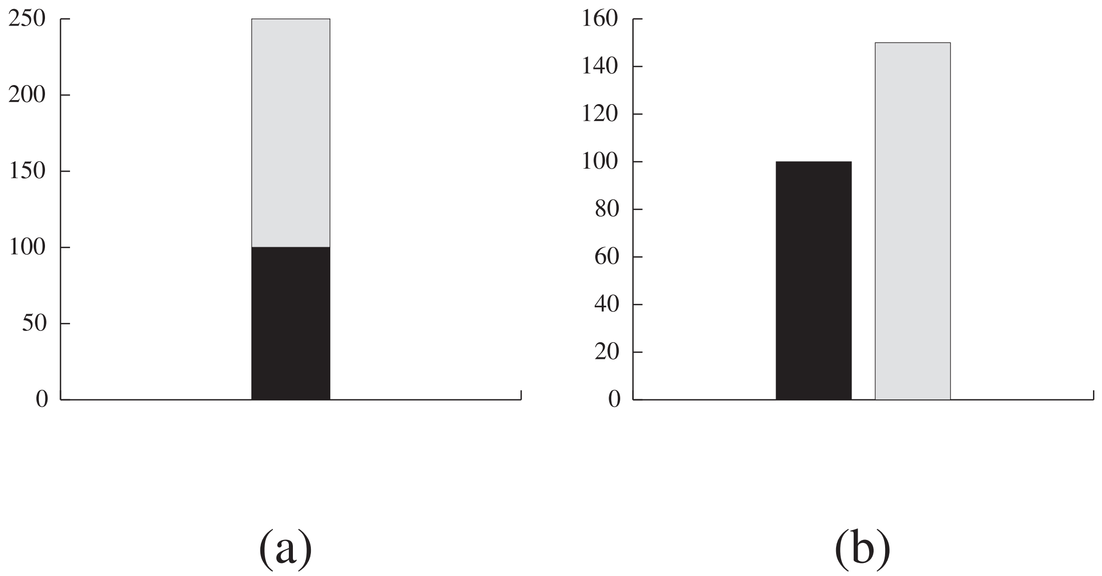


Figure 3.3: Perception vs. inference

3.1.2 From presentations to graphical languages

If a particular presentation can be described at several levels, as I have just discussed, it is clear that there must be some correspondence between the different descriptions. That is, for any particular presentation, graphic objects must correspond to perceptual objects (not necessarily in a one-to-one manner) and perceptual objects must somehow correspond to the information that can be extracted from the presentation. The goal now, however, is to describe *encoding conventions*—systematic methods of graphically encoding data—so it is necessary to go beyond describing how different levels of description map onto each other for individual presentations. I want to describe, for a given *type* of dataset, how *any* dataset of that type can be presented graphically.

Rather than describing direct mappings of information to graphic objects, it will be convenient to make use of the perceptual level to divide this problem into two smaller problems: *representation* and *layout*. Representation is the problem of how to map cognitive-level descriptions of information onto perceptual descriptions specifying semantically relevant perceptual features of a presentation, so that the information can be extracted by a viewer. It is this problem I will be most concerned with in this dissertation. The layout problem—that of mapping the perceptual-level description onto the graphic level (i.e., determining what particular graphic objects and parameter values can be used to guarantee that a viewer perceives certain perceptual properties and relationships)—is less central to the focus of this thesis, and will be discussed only in chapter 7, which describes the AUTOGRAPH implementation.

My approach in this section will be to first discuss how presentations encode information in an extremely general manner. I will then gradually restrict this general approach, paving the way for a more

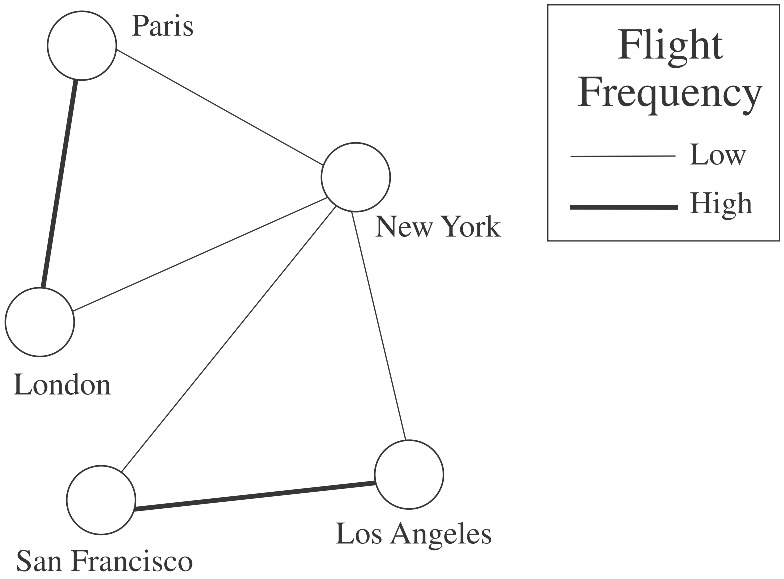


Figure 3.4: An alternate layout for the network graph of figure 3.2

structured definition of a graphical language in section 3.1.3.

Encoding conventions and equivalence classes

If one suspends normal preconceptions about how graphic presentations can represent information, it quickly becomes apparent that there are very few absolute, logical constraints on such representations. Nearly any combination of perceptual features of a presentation can encode nearly any information. As a baseline, then, it will be convenient to have a means of discussing how arbitrary information can map onto arbitrary combinations of perceptual features. In this section, I will develop the concepts and vocabulary needed to enable such a discussion.

I have formulated representation as a specification of the *semantically relevant* features of a perceptual description. Implicit in this formulation is the notion that other features of a perceptual-level description can be semantically *irrelevant*. It is this notion that makes the concept of layout meaningful. The perceptual-level description should describe an equivalence class of possible presentations. For example, the graph in figure 3.4 could have a perceptual-level description identical to that of figure 3.2—i.e., a description that doesn't include information about the absolute positions of the circles or lines, only the relationships among them—if the positions of the objects are not semantic.

Just as there are equivalence classes for entire presentations, there can be equivalence classes for pieces of presentations. For example, in the network graph of figure 3.4, every tuple is encoded by a pair of labeled circles with a line between them (i.e., a compound object), every city is encoded by a single labeled circle, and every frequency value is encoded by the thickness of some line. Only the labels, the thicknesses of the lines, and the relationships among the objects (i.e., which lines touch which circles) are semantically relevant. The fact that all the circles are the same size and color can even be considered a fact about the layout. If they vary, they might make the presentation more difficult to understand, but would not change the information being presented. Furthermore, there is implicitly a set of equivalence classes not only for the representation of tuples in the dataset, but for other potential tuples as well. For example, if the dataset included the additional tuples <*Barcelona,Madrid,high*> and <*Madrid,Barcelona,high*>, a pair of appropriately labeled circles connected by a thick line could be added to the presentation, at any position (barring layout considerations).

Specifying the equivalence classes at different levels of detail, along with their meanings, is the essence of specifying an encoding convention. That is, for any dataset, the encoding convention must specify how the dataset, each tuple in the dataset, or every part of every tuple in the dataset maps to some perceptual objects with certain characteristics.

Following Goodman (1968), I will refer to perceptual objects or groups of objects which can be considered equivalent for the purposes of interpretation as *character classes*. I will refer to the semantics of these character classes, at the cognitive level of description, as *compliance classes*.[6]

In theory, a character class can be defined by any meaningful combination of perceptual distinctions, consisting of, for example:

- A description of a single perceptual object of a given type, with particular values or ranges of values for some property—e.g., “any circle with color green and label ‘New York’”, or “any rectangle between 1 and 2 inches from the left edge of the page, with height of 1.5 inches and width of one inch”. Other properties of the object are thus irrelevant—i.e., do not affect the character class and thus the semantics of the object—such as the position of the circle or the color of the rectangle in the above examples.

- A disjunctive set of such object descriptions—e.g., “any red circle with radius 1 inch or green circle with radius 2 inches” or “any rectangle labeled ‘New York’ or blue triangle”. That is, two completely different objects can have the same meaning under some set of encoding conventions.

- Multiple objects, with given property values or ranges of values for the individual objects and given relations and functions between them—e.g., “any pair of circles, one labeled ‘New York’ and one labeled ‘San Francisco’, with a line touching each of them” or “two rectangles at a vertical position of two inches from the bottom of the paper, separated by a distance of two inches”.

  I will refer to character classes describing multiple objects, at least one of which is itself a member of some other character class, as *compound* character classes. The group of objects will be considered a compound object at the perceptual level.

  As an example of a compound character class, “a pair of labeled circles with an arrow between them” encoding (i.e., having the compliance class) the relation $< x_1, x_2 >$ might incorporate a labeled circle

[6] Many of the concepts and much of the vocabulary used in this section are due to Goodman, though he does not address the types of systematic encoding conventions used in graphic presentations. I will note other differences as they arise.

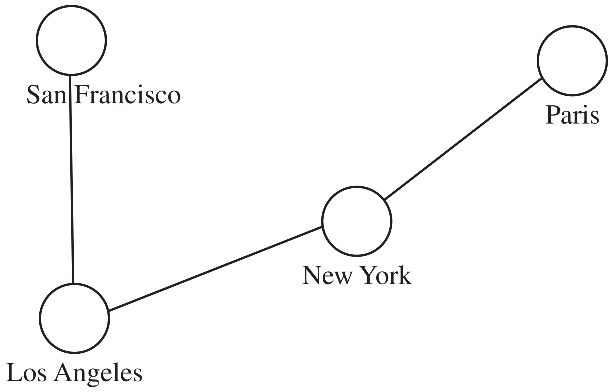


Figure 3.5: A simple network graph

Cognitive Level	Compliance Class Composite Prime	Semantic equivalence class Incorporating semantics of component objects Not incorporating semantics of component objects
Perceptual Level	Character Class Compound Intermediate objects	Equivalence class of perceptual objects Incorporating members of other character classes Non-semantic components of compound objects

Table 3.2: Vocabulary for describing encoding conventions

encoding x_1 and a labeled circle encoding x_2. If the compliance class of a compound character class includes the compliance classes of the incorporated character classes (as $< x_1, x_2 >$ includes x_1 and x_2), I will refer to the compliance class as *composite*; otherwise, it will be referred to as *prime*.[7]

It is important to note that while a compound object incorporates objects with their own character classes by definition, and these objects possess compliance classes, it may also incorporate objects without semantics of their own, and will generally have a compliance class with semantics greater than the sum of its parts. For example, a line between two circles in a network graph such as figure 3.5 can be ascribed no semantics in terms of individual domain values; i.e., outside the context of the compliance class of the compound object consisting of two circles and a line, the line has no meaning. I will refer to such non-semantic objects as *intermediate* objects.

Since information in my framework is given in the form of tuples, compliance classes must correspond to individual tuples, parts of tuples (i.e., individual data values or groups of data values), or sets of tuples. For example, the character class "any circle labeled 'New York'" might correspond to a domain value "New York" in the domain *cities*, and the character class "any circle labeled 'New York' and circle

[7]I am departing slightly from Goodman's use of these terms here. Using his conceptual framework, a composite compliance class must not have any semantics apart from the semantics of its components. This is not very useful for analyzing graphic presentations that are the focus of this dissertation.

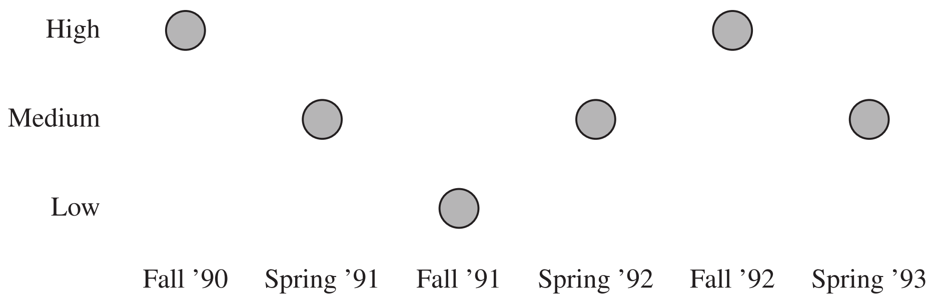


Figure 3.6: A simple graph

labeled ‘San Francisco’ with a line between them” might correspond to a compliance class {*flight(New York, San Francisco), flight(San Francisco, New York)*}— i.e., a pair of tuples, one indicating that there is flight between New York and San Francisco and one indicating that there is a flight in the reverse direction.

Encoding conventions must specify a correspondence between character and compliance classes for every possible dataset that might be presented using these conventions. This specification can be at different levels of detail: a set of character classes might correspond to entire potential datasets, potential tuples, groups of domain values, or individual domain values (with tuples corresponding to compound character classes). Some of the vocabulary just introduced is summarized in table 3.2.

Some examples will help to clarify how an encoding convention works. For example, for the graph of figure 3.6, the individual circles encode the tuples {*<Fall ’90, high>, <Spring ’91, medium>, <Fall ’91, low>,* … }, but the full method of presentation is captured by a larger set of character and compliance classes, describing every potential piece of data (i.e., tuple) that can be represented, as shown in figure 3.7. This figure shows how a space of possible perceptual features can be divided into character classes mapping onto a similarly divided space of character classes (i.e., possible tuples). Given a set of tuples, an encoding convention describes how a set of perceptual objects with certain perceptual properties can visually present the dataset.

For a bar graph like that of figure 3.8, encoding average enrollment over the course of a year, there is a similar set of character classes, defined by the horizontal positions and heights of the bars.[8] Since one dimension (i.e., height) represents a continuous domain here, however, there are an infinite number of tuples that can potentially be represented, and the set of character classes (and corresponding compliance classes) is actually infinite.[9]

As another example, consider the network graph of figure 3.2, which I have already discussed, encoding the dataset {*<Paris,New York,Low>,<New York,Paris,Low>,*… }). For this graph, there is a set of character classes for labeled circles, with compliance classes corresponding to the cities named by the labels.

[8] There is another possibility: that the vertical position of the top edge of the bar determines its character class. Both encoding conventions can be made equivalent with the use of additional constraints on the bars. The notion of additional constraints will be discussed shortly.

[9] I will discuss infinite sets like this shortly, in section 3.1.2

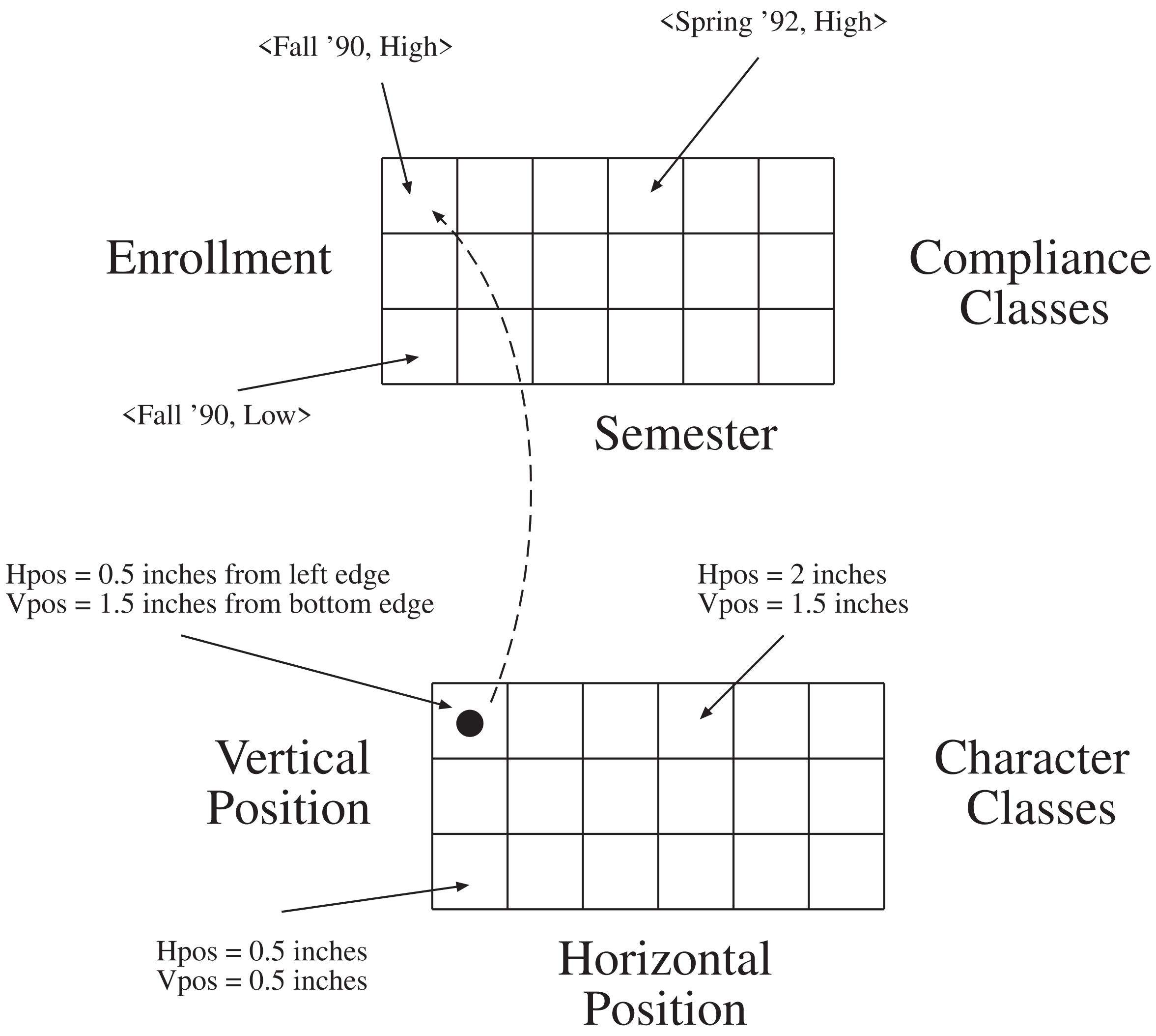


Figure 3.7: Character and compliance classes for the graph of figure 3.6

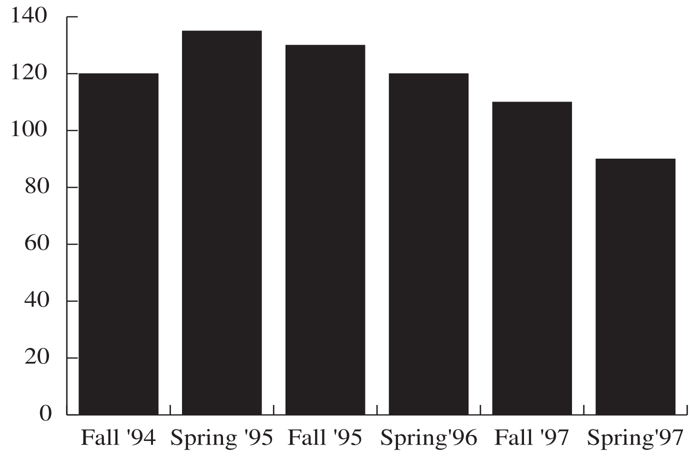


Figure 3.8: A bar graph

There is also a set of character classes for thick and thin lines, corresponding to compliance classes for high and low frequency of flights. Finally, there is a set of compound character classes describing every potential pair of circles with a line (either thick or thin) touching each of them, with compliance classes corresponding to pairs of tuples encoding high-frequency or low-frequency flights (in both directions) between the cities named by the circle labels.

The bar graph example makes apparent a limitation of describing a method of representation through sets of syntactic and semantic equivalence classes. If the heights (i.e., vertical extents) of the bars determine their meanings, the fact that their bottoms are aligned is only a fact about the layout. Encoding conventions as I have so far defined them greatly underconstrain the way presentations will be generated. This was also true in the previous example; the character classes didn't specify the sizes or colors of the circles in figure 3.6.

Another limitation of encoding conventions as I have developed them thus far arises in the network graph example. When tuples are encoded by compound objects, something must specify whether individual component objects (encoding a subset of the domain values of the tuple) may be included in multiple compound objects. For example, in figure 3.2, there is only one instance of a labeled circle encoding *Paris*, but there are many lines in the character class encoding *high* frequency. It would also be possible to create a graph with multiple circles labeled "Paris", as in figure 3.9, with precisely the character and compliance classes as the graph of figure 3.2. I will refer to component objects for which there will be only one object instance for any character class as *shared*, because the single object instance will be part of any compound object incorporating a component of that character class. I will refer to component objects for which there

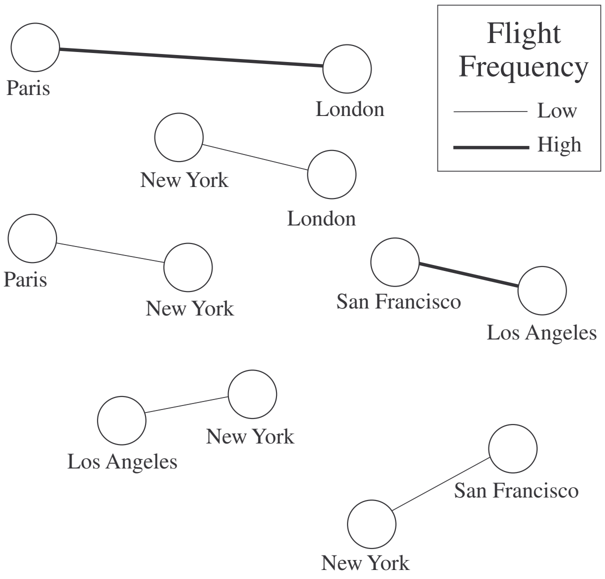


Figure 3.9: A network graph with non-shared nodes

may be many instances encoding the same sets of domain values as *non-shared*. This concept will be further clarified later in this section.

Rather than refining the current notion of encoding conventions, I will eventually use it at the core of a less general but more thorough description of a method of graphic representation—i.e., a specification of what a graphical language is in my framework.

It is convenient to note here that the notion of systematic encoding conventions for presentations allows a refinement of the psychological model of graphic comprehension developed earlier. While I will still eschew giving a detailed temporal account of how graphic presentations are perceived and interpreted, it will be useful to distinguish two phases: *identification* and *extraction*.[10]

In the identification phase, a viewer determines the encoding conventions of the graphical language —i.e., which objects, properties and perceptual functions encode which data domains, and how the individual values in the data domains map onto different values in perceptual dimensions.[11] In the extraction phase, a viewer uses the knowledge of the encoding conventions to actually decide which objects to attend to and which properties of these objects to inspect, thus extracting information from the graph.

While these two processes are not completely separate (i.e., it is clear that people will often need to refer to axes and legends, reminding themselves of the encoding conventions, while extracting information from objects in a presentation) the distinction between them is useful because it will allow a discussion of

[10] The term *identification* and the general notion of these phases are due to Bertin (1983), though his analysis is somewhat different than mine.

[11] While the identification phase is clearly influenced by the axes and legends in a graphic presentation, I will focus in this dissertation on other issues affecting identification, such as the choice of properties and perceptual functions, and the nature of the mappings used. Designing effective axes and legends seems to be a largely orthogonal problem.

different issues pertaining to each.

Fundamental logical requirements for encoding conventions

The example encoding conventions described in the previous section satisfy (and all encoding conventions generally must satisfy) two fundamental logical requirements, observed by Goodman:

- **Disjointedness** Each possible perceptual object should fall into only one character class, and each character class should map onto only one compliance class. Were this not true, some perceptual objects would be ambiguous—i.e., they would have competing meanings.[12]

- **Differentiation** If the difference between two character classes is theoretically imperceptible (e.g., one class includes all lines less than or equal to one inch long and another class includes all lines greater than 1 inch long, so an infinitely small change in the length of a line could change its character class), the difference in semantics between these two character classes should be effectively nil. That is, character classes should either be distinct (as in the encoding conventions illustrated by figure 3.7) or should describe a continuum (e.g., the height of a rectangle in inches $\times$ 200 = profits in dollars). The essence of this requirement is that it must be possible to determine which marks fall into which character classes and which character classes map onto which compliance classes. For continuous sets of classes, although it is not possible to determine the precise character class into which an object falls, it is possible to determine the character and compliance class within some margin of error.[13]

Systematicity assumptions

Using the concepts of character and compliance classes, and the three levels of description, it is possible to precisely describe *any* conceivable encoding convention. This flexibility is more a hindrance than a help, however, permitting presentations to be constructed that no human could possibly use. The defining characteristic of the encoding conventions with which humans are familiar, as opposed to the much more general set the framework is now capable of describing, is *systematicity*. That is, the character classes and compliance classes of an encoding convention are generally related to each other in regular, systematic ways, not in a completely arbitrary fashion.

Previous work on automatic graphic presentation has relied on using known graphical languages—i.e., using existing systematic conventions for representing data. In my first-principles framework, systematicity will be guaranteed with a set of assumptions about, or limitations on, what an encoding convention will look like. These assumptions, which I will refer to as *systematicity assumptions*, are principles about interpretability that are so fundamental as to be made an integral part of the framework. They are the following:

[12] This requirement does not prevent perceptual objects from having different meanings in different contexts; in these cases, they can be thought of as belonging to compound objects with prime compliance classes.

[13] These requirements are pretty substantially different from Goodman's original conception, though based on Goodman's ideas. In particular, Goodman distinguishes between syntactic and semantic disjointedness and differentiation and doesn't allow for mappings along a continuum. Disregarding the differences between syntactic and semantic requirements is convenient for my purposes, and the extension of differentiation to handle mappings along a continuum is needed to discuss typical graphic presentations of quantitative data.

- **Discrete encoding** *A dataset will be encoded by a set of perceived objects (either compound or atomic), with each object encoding either one tuple or a set of tuples sharing all of the same domain values.*[14]

  This assumption restricts the notion of graphical languages in a significant way, disallowing a variety of types of presentations in which the objects encoding tuples cannot be neatly separated. I will discuss some of the limitations imposed by this and the other systematicity assumptions in chapter 8.

- **Uniform object encoding** *If a dataset is encoded by a set of atomic objects, they must all be instances of the same perceptual object class. If it is encoded by a set of compound objects, each such object must be made up of the same types of atomic component objects as the others, and these components must be connected by the same types of perceptual relations.*

  For example, in the network graph of figure 3.2, each tuple is encoded by exactly two circles and one line, and the line touches each of the two circles. No tuple of the same type is represented using two squares and an arrow, or with a line tangent to two circles, etc.

- **Systematic domain mapping** *Atomic objects encode domain values with certain of their property values, with individual properties corresponding to individual domains and particular property values or ranges of values encoding particular domain values. All atomic objects of the same type use the same encoding scheme.*

  For example, if one rectangle in a bar graph encodes a value from the domain *Semester* with its horizontal position and a value from the domain *Students* with its height, all the other rectangles in the presentation will similarly encode values for *Semester* and *Students*, with different heights corresponding to different values for *Students* and different horizontal positions corresponding to different values for *Semester*. The encoding conventions for the graph of figure 3.6, described by figure 3.7, illustrate this concept.

- **Compositional semantics** *The compliance classes of a compound object will be based on those of its component parts and possibly the values of perceptual functions between those parts. More specifically, if a component object encodes certain domain values, the compound object will also encode those domain values, although it may order them differently in the context of representing a tuple. That is, its compliance classes must be composite.* For example, in the graph of figure 3.2, the labeled circles encode particular city values (e.g., *Paris*, *New York*), the lines encode values for flight frequency (either *low* or *high*), and the compound objects consisting of pairs of circles with a single line between them encode tuples incorporating both city values and the flight frequency value, to represent a pair of tuples (e.g., {*<Paris,New York,Low>,<New York,Paris,Low>*}).

- **Systematic component sharing** *Each component of a compound object will either be shared by all*

[14] The latter possibility allows objects to represent sets of tuples that are permutations of each other—e.g., {< *Paris, New York, Low* >, < *New York, Paris, Low* >}.

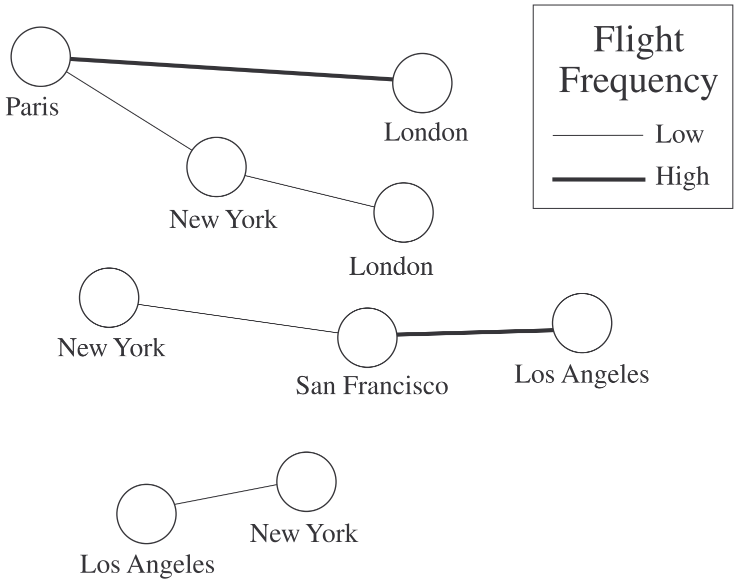


Figure 3.10: A network graph with semi-shared nodes

compound objects incorporating an object of that character class, or duplicated as necessary except where obvious redundancies will occur.

By redundancies, I am referring to creating two objects with exactly the same semantic properties and participating in exactly the same perceptual relations and functions with other objects. This assumption, which makes use of the distinction between *shared* and *non-shared* components noted earlier, disallows graphs like that of figure 3.10, in which certain components are shared by multiple compound objects, but other components of the same type are duplicated.

The issue of redundancies will be discussed in detail in chapter 4. For now, it will be convenient to treat shared objects as potentially being part of multiple compound objects and non-shared objects as being part of only one compound object. More precisely, non-shared objects will only participate in the perceptual relations needed to describe a single compound object.

These assumptions limit the notion of graphic presentations to a manageable space of possibilities, but are general enough to describe a vast number of conventional and unconventional methods of presentation, as will be demonstrated in chapter 7. I will discuss some of the limitations imposed by the systematicity assumptions in chapter 8.

3.1.3 A definition of a graphical language

It is now possible to define precisely what is meant by a graphical language in my framework, incorporating the systematicity assumptions just described. At the core of this definition is the notion of encoding conventions as developed throughout this section, but this notion has been extended to address the issues of object sharing and non-semantic constraints on the objects of a presentation.

Definition 1 *A graphical language is a specification of a systematic method of encoding a set of data tuples by a set of perceptual objects, atomic or compound, consisting of:*

1. *A specification of the type(s) of object(s) used to encode each tuple.*
2. *A correspondence between each data domain of a tuple and either some property of an atomic (component) object used to encode the tuple or a perceptual function between multiple component objects, so that individual data domain values correspond to individual perceptual dimension values or ranges of values.*
3. *A specification of how the components of compound objects are linked by perceptual relations, if multiple objects are used for each tuple.*
4. *A specification of how component objects are shared by different compound objects—i.e., whether each class of component objects is shared or non-shared.*
5. *A specification of additional constraints on the objects of the presentation.*

As an example, in the standard bar graph of figure 3.8, the five specifications are as follows:

1. Tuples correspond to rectangles.
2. *Semester* corresponds to the horizontal position and *Students* to the height of the rectangles.
3. Not applicable.
4. Not applicable.
5. The bottoms of the rectangles are constrained to be aligned.

As another example, in the network graph of figure 3.5:

1. Tuples correspond to a pair of circles and a line.
2. *City1* is encoded by the label of the first circle, *City2* is encoded by the label of the second circle.
3. The line must touch both circles.
4. Circles are shared. Lines are not.
5. There are no other constraints.

3.2 The space of possible graphical languages

The definition of a graphical language I have just given enables a concrete discussion of the space of possible graphical languages. A graphical language consists of five specifications, and the different choices for each specification yield a combinatorial set of possibilities for presentation methods. Conceiving of possible methods of presentation as a space will enable us to explore different design decisions in order to generate different graphical languages.

Rather than structuring the space as a set of decisions explicitly corresponding to the definition of a graphical language, I will structure it as a related set of design decisions which will be easier to analyze and more convenient for the purpose of generating graphical languages.[15] I will discuss the relationship between these design decisions and the specifications of graphical languages shortly.

The design decisions are, roughly:

- How to partition the tuple into sets of domains to be represented by separate perceptual objects.
- How to assign the sets of domains to classes of objects and individual domains to particular properties of those objects.
- How (generally) to connect components to form compound objects.
- How (more specifically) to connect components to form compound objects.
- How to share components of compound objects.

I will now discuss each of these decisions in more detail, including analyses of how many possible choices each decision entails:

A. *How to partition the tuple* By the definition given earlier, a graphical language must include a specification of how properties of one or more objects (i.e., atomic objects or components of compound objects) encode domains of tuples. Thus, a graphical language must divide the domains of a tuple into subsets, with different subsets corresponding to different objects. If all the domains are put into a single set, an atomic object will be used to encode the tuple. If they are separated into multiple subsets, each subset will be represented by the properties of a different component of a compound object. I will refer to the individual subsets of domains as *subtuples*, and the set of groups as a *partitioning* into subtuples. For example, in the network graph of figure 3.2, a tuple of the form <*City1, City2, Flight Frequency1*> is partitioned into subtuples of the forms <*City1*>, <*City2*> and <*Flight Frequency1*>. For the bar graph of figure 3.8 the tuples are not partitioned at all; the domains are kept together in one group and represented by atomic, rather than compound, objects.

For tuple of arity n, the total number of possible partionings can be analyzed as follows:

$$p(n) = \sum_{i=1}^{n} p'(i, n)$$

[15] The discussion here will assume that single dataset is being presented. I will address issues related to multiple datasets in chapter 4.

where $p'(i,n)$ = the number of possible partitionings into i groups for a tuple of n domains.

This value, p', can be defined: :

$$p'(i,j) = \begin{cases} 0 & i > j \\ 1 & i = j \text{ or } i = 1 \\ p'(i-1, j-1) + i(p'(i, j-1)) & \text{otherwise} \end{cases}$$

The terms in the general case are derived from the fact that an element of the tuple can go in its own partition (in which case the rest of the tuple must be divided into $i-1$ groups) or into one of the i other partitions (if the rest of the tuple can be partitioned into i groups).

The number of possible partitionings $p(n)$ grows rapidly with the size of the tuple, but most tuples that can be presented with static, two-dimensional presentations will be of relatively small arity. As an example of the number of partitionings with reasonable values of n, $p(3) = 5$, $p(4) = 15$, and $p(5) = 52$.

B. *How to assign the subtuples to classes of objects and the domains of the subtuples to properties of those objects* Once the domains have been grouped together into subtuples, assigning these subtuples to perceptual object classes and individual domains to properties of the objects completes the first two specifications of a graphical language (unless intermediate objects are introduced when making the topology concrete). It is also possible to assign subtuples with only one domain to perceptual functions, which are analogous to the properties of compound objects. For example, if a tuple of the form <*City1, City2, Distance*> has partitioned into subtuples of the forms <*City1*>, <*City2*> and <*Distance*>, the first two subtuples could be assigned to the labels of circles, and the last could be assigned to the distance between two circles. Note that it is only possible to assign two subtuples the same representation because they encode the same domain; the systematicity assumptions prevent assigning multiple meanings to a single perceptual object class.[16]

To estimate the number of possible mappings, one can make the simplifying assumption that there will be a closed set of m perceptual object classes, each with p properties, and ignore mappings to perceptual functions. Doing so, the number of possible mappings is, for a partitioning into s subtuples with d_i domains per subtuple:

$$\prod_{i=0}^{s-1} \left((m-i) \prod_{j=0}^{d_i - 1} (p-j) \right)$$

The terms in this equation are based on the calculation that for a partitioning into s subtuples, there will be $\prod_{i=0}^{s-1}(m-i)$ ways of assigning subtuples to objects, and for each subtuple of size d_i, there will be $\prod_{j=0}^{d_i-1}(p-j)$ ways of assigning domains to properties.

[16] Note also that the use of the perceptual function *distance* is only possible for data satisfying certain mathematical constraints. I will address such issues later in this chapter.

The number of possible mappings grows very rapidly, and can be large even with a small number of objects and properties. For example, with $m = 3$, $p = 4$, and a partitioning into subtuples of sizes 2, 2, and 1, there will be 3456 possible mappings, and with m increased to 4 and p increased to 5 there will be 48,000 mappings. Fortunately, there turn out to be many ways of pruning mappings while generating them, which reduces the total number of mappings to a still large but manageable number. I will discuss logical and psychological constraints on mappings in section 3.3.

C. *How to assign a topology to the components of a compound object* Compound objects are defined by their components and by the relationships between the components. The most important components, of course, are the ones that encode domain values. One way of quantifying possible arrangements of components is to consider first the topology of the semantic components—i.e., which of them are connected to which others—before considering the specific perceptual relations that instantiate this topology.[17] For example, if there are 3 components (e.g., two circles and a line) referred to as a, b, and c, the topology could specify that a is connected to b and c, or that a is connected to b and b is also connected to c, or that a and b are both connected to c. It is the last topology that would lead to a network graph (i.e., each circle is connected to the line).

I will assume that there are only binary connections between objects— i.e., that all perceptual relations are between pairs of objects, rather than among larger groups. Making this assumption, a loose upper bound of $2^{\frac{n(n-1)}{2}}$ can be placed on the number of possible topologies, where n is the number of semantic component objects. This bound is derived from the fact that every pair of components may or may not be connected, and connections are symmetric.

The AUTOGRAPH implementation furthermore assumes that all topologies will be connected graphs—i.e., that compound objects will be formed of groups of perceptually related components.[18] This reduces the number of possible topologies, especially for small values of n. Given this assumption, with three objects, there are four possible topologies (three with perceptual relations between two pairs of objects, one with perceptual relations between every pair of objects). With four objects, there are 41 distinct connected topologies.

D. *How to make the topology concrete* Having determined the information each component encodes and the topology of a compound object, it is then necessary to determine the way the topology is instantiated. Each link between a pair of semantic components can be instantiated by a perceptual relation—e.g., if a compound object has been determined to consist of a connected line and circle, they could touch each other. Alternately, it can be instantiated by a chain of perceptual relations and intermediate objects. For example, if the semantic components are two circles labeled with the names

[17] I am thus assuming that intermediate (non-semantic) objects in compound objects can be defined relative to semantic components.

[18] This assumption underlies the example just cited. It is worthwhile to note, however, that it is an assumption rather than a logical requirement. For example, it would be possible to represent the distances between a set of cities with the distances between a set of circles on a page. In this graphical language, every possible pair of circles would be considered a compound object, with distance a perceptual function between the components and no perceptual relations used to form the compound objects. Nevertheless, situations in which languages like this can be used are sufficiently rare that I will choose to ignore the possibility.

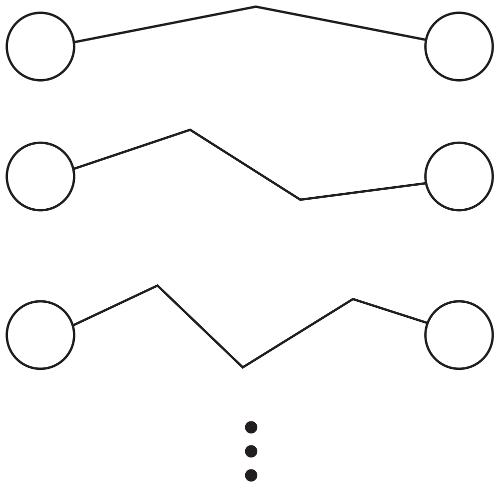


Figure 3.11: Chains of non-semantic objects

of cities, a line can be introduced, touching each of them to represent a flight between the cities (i.e., a tuple $< city_1, city_2 >$).

Clearly, there are an infinite number of possible chains between pairs of objects. For example, two circles could be connected by a pair of lines whose endpoints touch, a trio of such lines, four lines, etc. (see figure 3.11). I will assume, however, that there will be no more than one non-semantic (i.e., intermediate) object between any pair of semantic objects. Thus, the number of possible realizations of a topology can be analyzed as follows.

If there are r possible perceptual relations between each pair of objects, a total of m perceptual object classes with o of them already used as semantic objects, and n abstract links in the topology, then there will be at most

$$n \times (r + (m - o)r^2)$$

possible instantiations of the topology. That is, each of n abstract links can be implemented with one of r possible perceptual relations or with one of $(m - o)$ types of intermediate objects (since the o semantic objects cannot be used as intermediate objects) connected to each of the two semantic objects with one of r perceptual relations.

E. *Object sharing* As described earlier, each semantic component can be shared or non-shared. Non-semantic (intermediate) objects will always be non-shared, since there is no logical basis for incorporating such objects in multiple compound objects. Thus, for a partitioning into n subtuples, there will be 2^n possible choices for sharing, or fewer if some of the subtuples are represented with perceptual functions.

Extra non-semantic constraints (specification 5 of the definition of a graphical language) seem to

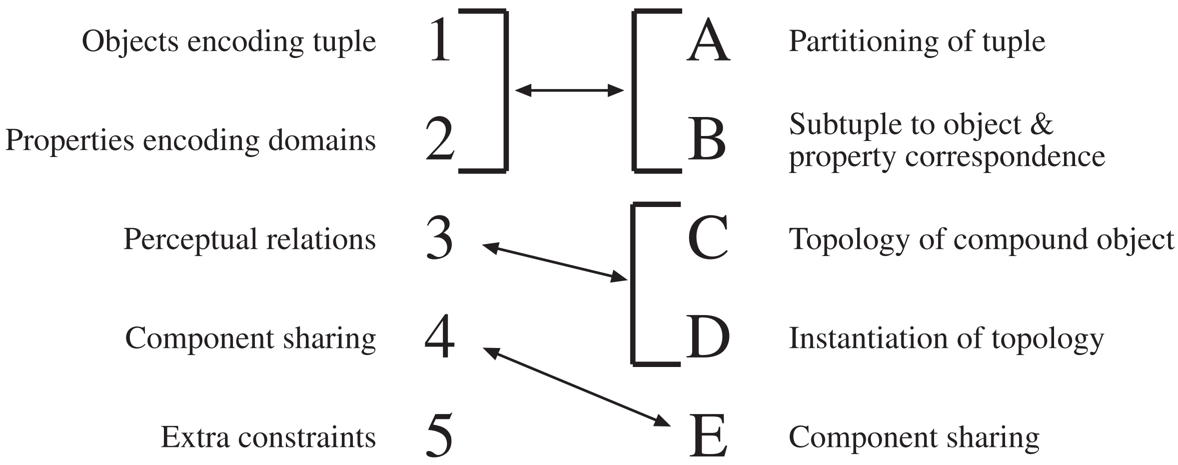


Figure 3.12: The correspondence between the graphical language definition and design decisions

be the most difficult part of a graphical to analyze quantitatively. While there are in theory an infinite number of possibities for extra constraints, in practice there seem to be only a few that are actually useful. In AUTOGRAPH, these are actually determined after the other choices are made for a language, in a deterministic fashion, and thus are not considered part of the space.

These five decisions plus the extra non-semantic constraints determine a graphical language. The rough correspondence between the specifications required for a graphical language and the set of design decisions used to structure the space is shown in figure 3.12. The objects for specification 1 of the definition are primarily determined by decisions A and B, though intermediate (non-semantic) objects are determined by decision D. The mappings for specification 2 are also determined by stages A and B, which divide the domains into subsets and then assign objects and properties to these subsets. The way compound objects are formed, specification 3, is determined by the topology of decision C and the instantiation of decision D. Finally, sharing is specified by decision E. As I have stated, extra non-semantic constraints are not part of the space.

As an example of how this set of decisions can determine a graphical language, consider again the bar graph of figure 3.8. The choices leading to this language are:

A. The domains of a tuple of the form <*Semester, Students*> are kept in a single set.

B. The set is assigned to the perceptual object class *rectangle*, with *Semester* assigned to the horizontal position of the rectangle and *Students* assigned to the height of the rectangle.

C. Not applicable. (No topology is needed for a single component).

D. Not applicable. (No topology is needed for a single component).

E. Not applicable. (Sharing is also irrelevant, for tuples encoded by atomic objects.

For another example, with the network graph of figure 3.5:

A. The domains of a tuple of the form <*City1,City2*> are divided into two sets.

B. Each set is assigned to the perceptual object class *circle* (which is possible only because the sets contain the same type of domain), and each domain is assigned to the property *label* of the circle.

C. Only one topology is possible with two components: they must be connected to each other.

D. This topology is instantiated by introducing an intermediate object, a line that touches each circle.

E. Circles are shared.

A different graphical language could be created for the same data by keeping the domains together in a single subtuple and representing this subtuple with the horizontal and vertical positions of a circle.

All of these design decisions can be considered branches in a search tree. It is clear that even with the assumptions described above, the number of leaves of this tree is very large and grows extremely quickly with the number of object classes, properties, perceptual relations and perceptual functions available to the system. Nevertheless, several factors make a first-principles approach viable. First, the actual number of object classes, properties, etc. needed to generate the bulk of common types of presentations (and also some novel ones) is quite small. The current version of AUTOGRAPH, described in chapter 7, uses four perceptual object classes (*circle*, *rectangle*, *line*, and *arrow*) with an average of six to seven properties each, uses only three perceptual relations (*touches* for lines and circles/rectangles, *head-touches* and *tail-touches* for arrows and circles), and doesn't use perceptual functions at all. It is still capable of generating a wide range of presentations. Second, even if there were a larger perceptual "vocabulary," both logical and psychological factors allow substantial pruning of the search tree, and also ordering the branches for heuristic search. The AUTOGRAPH implementation performs such a search to generate graphical languages.

3.3 Limitations on the space of graphical languages

The space of possible graphical languages as I have just described it is a theoretical abstraction. In the quantitative analysis above, I have implicitly assumed that all properties and perceptual functions are equivalent and capable of representing any data domains, and that perceptual relations are similarly interchangeable. Perceptual dimensions and relations have varying characteristics, though, as do datasets. As a result, different graphical languages will be appropriate for different types of data, both in terms of expressiveness (i.e., whether they are capable of representing the data) and effectiveness (i.e., how well they facilitate a viewer's goals). In this section, I will first describe some of the characteristics of perceptual dimensions and relations. Then, in section 3.3.2, I will outline some logical restrictions on the space of possible graphical languages due to these differing characteristics, and categorize the interpretative and perceptual issues that affect the expressiveness and effectiveness of graphical languages. In the course of doing so, I will also point out the aspects of a data characterization that will be necessary for making sound decisions about graphical languages. The goal for this discussion is not to address logical, interpretive, and perceptual issues in detail, but to provide an outline to be filled in by later chapters.

3.3.1 Characteristics of perceptual dimensions and relations

To this point, I have only spoken of perceptual properties and functions as mapping to abstract perceptual dimensions, and have spoken about perceptual relations only as abstract "links" between the components of compound objects. These abstract descriptions overlook several important issues, however. In this section, I will examine a number of distinctions that can be applied to perceptual dimensions and perceptual relations; future sections in this chapter and in later ones will reveal the value of these distinctions.

One important issue, considering how properties and perceptual functions map objects onto values in perceptual dimensions, is that different properties may map onto the *same* dimensions, or onto *similar* ones. The horizontal positions of a circle and a rectangle are directly comparable, for example—i.e., both properties map onto the same perceptual dimension. The horizontal and vertical position of a circle are also comparable, as are the height and width of a rectangle, as are the length of a line and the distance between two circles, although each of these comparisons is somewhat more difficult than comparing two horizontal positions. There are other pairs of perceptual dimensions which are still similar, but less so, such as the area of a circle and that of a rectangle. Area is a mathematically well-defined concept, but estimating the area of a rectangle and estimating the area of a circle are perceptually different tasks, and the areas of different objects are thus difficult to compare accurately.

Looking at perceptual dimensions in more detail, it is possible to characterize them in some general ways. The most important distinctions seem to be:

- **Continuous vs. discrete**—Some perceptual dimensions, such as size, are best thought of as ranges of scalar values, while others such as shape, if shape is a choice among a closed set of possibilities (e.g., circle or rectangle), are discrete sets of values.

- **Ordered / non-ordered / locally ordered**—Some dimensions, such as length and value (i.e., gray-scale value) are perceptually ordered, with pairs of property values supporting *less-than* / *equal-to* / *greater-than* comparisons. With other dimensions, such as shape, pairs of values can only be compared as *equal* / *not-equal*.[19] Still other dimensions (e.g., orientation) are locally ordered, but lack a fixed endpoint for absolute comparison.

- **Quantitative vs. non-quantitative**—Some ordered dimensions, such as length, support quantitative comparisons (i.e., rectangle A is 15% larger than rectangle B), while others (e.g., value) do not.

- **Amount vs. coordinate**—Some quantitative perceptual dimensions, such as the length of a bar, connote size or amount in some way. Others, such as horizontal position, do not.

[19] Note that for value, the assignment of black and white to minimum and maximum is somewhat arbitrary. Nevertheless, making this assignment defines an order.

Individual properties and perceptual functions may also have their own more specific semantic associations. For example, horizontal position is typically used to represent longitude and vertical position, latitude, and there are standard meanings associated with certain colors (e.g., red equals danger) and color schemes (e.g., those commonly used in weather maps).

The principle issue in characterizing perceptual relations is that due to underlying mathematical considerations, certain combinations of perceptual relations imply other perceptual relations or are incompatible with other perceptual relations. For example, if circle B is *below* circle A and circle C is *below* circle B, then circle C will also be *below* circle A. Furthermore, it is impossible to create a pair of circles such that A is *below* circle B, which is *below* circle C, which is *below* A. This contrasts with perceptual relations such *intersects*; if line D intersects line E, line E must also intersect line D.

In general, I will assume that the properties of perceptual relations can be described by statements in first-order logic. For example:

$$\mathit{contains}(x,y) \land \mathit{contains}(y,z) \rightarrow \mathit{contains}(x,z)$$
$$\mathit{contains}(x,y) \rightarrow \neg\, \mathit{contains}(y,x)$$

Like perceptual dimensions, some individual perceptual relations also have particular semantic associations (e.g., *above* can connote superiority, at least in western cultures.)

It is also important to note that the properties of an object can be interrelated. For example, the positions of the endpoints of a line determine its length. Similarly, properties of an object and perceptual relations the object participates in can be related, as can properties and perceptual functions an object participates in, etc. For example, *below* is inversely related to *above*, and both relations are related to vertical position.

3.3.2 Logical and psychological constraints on the space of graphical languages

The particular characteristics of individual perceptual dimensions and relations determine what types of data they can be used to represent, due to both logical expressiveness considerations and psychological expressiveness and effectiveness considerations. I will survey these considerations here, and also make note of the information a data characterization must provide in order for a system to make use of them.[20]

It would be valuable to structure the discussion of these issues in some way that could help ensure its thoroughness—i.e., by following some self-evidently complete organization. One method of doing this would be to follow the five specifications that comprise a graphical language. That is, it would be possible to discuss issues relating to the choice of objects, issues relating to the mappings of data domains to perceptual properties and functions, etc. It would similarly be possible to structure the discussion along the lines suggested by the structuring of the space of possible encoding conventions. Neither of these possible divisions turn out to be very convenient, however, so I will use yet another one.

[20] Note that some of the logical issues raised here will be discussed in more detail in the next chapter. The psychological issues raised here will be addressed in more detail in chapters 5 and 6, which discuss interpretive and perceptual issues, respectively.

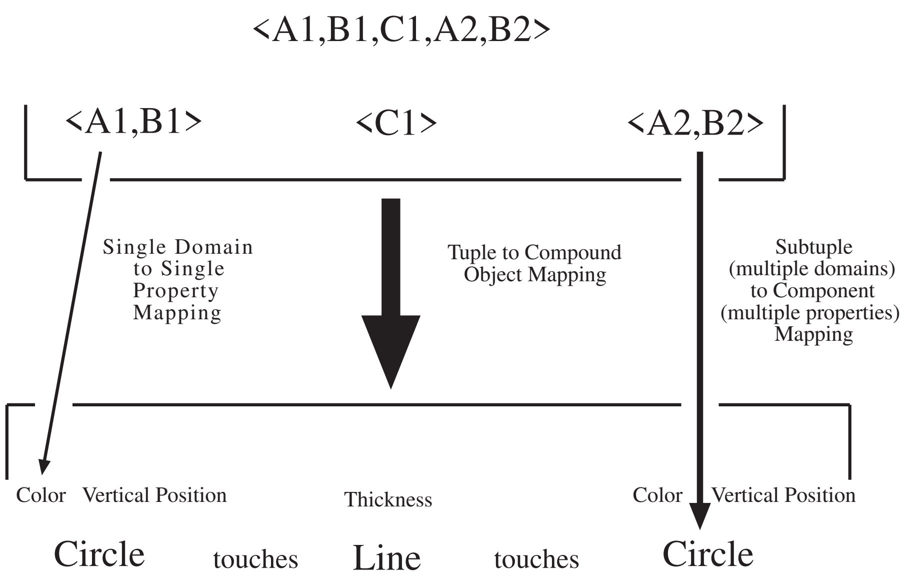


Figure 3.13: Three levels of mapping

In the general case, a set of tuples is represented graphically by a set of compound objects, with individual tuples corresponding to individual compound objects, subtuples corresponding to components of these compound objects, and individual domains corresponding to individual properties of these component objects. This hierarchically organized set of correspondences is illustrated by figure 3.13. I will begin with the most detailed issues, considering first those related to mapping individual data domains onto perceptual dimensions. I will then consider issues relating to mapping multiple data domains onto properties of a single object. Finally, I will examine the mappings of tuples onto compound objects, simultaneously considering the mapping of datasets onto sets of compound objects—i.e., I will consider mappings relating to the use of multiple objects in a presentation. This structure is complete and will help to guarantee that this discussion will be thorough. The discussion will also address, throughout, the two phases of identification and extraction that comprise viewing and understanding the presentation.

Constraints on individual property and perceptual function mappings

From a purely logical point of view, there are two restrictions on the use of individual properties and perceptual functions to represent individual data domains arising from the fundamental requirements of disjointedness and differentiation discussed in section 3.1.2:

- *A discrete data domain of* n *values can only be represented by a discrete or continuous perceptual dimension with at least* n *perceptually distinct values.* For example, if a system is restricted to using a small palette of colors, it will not be able to use color to represent a data domain with a large number of values.

- *A continuous data domain (i.e., a numeric range) can only be represented by a continuous perceptual dimension.* For example, if there is a class of objects called marks with the property of shape, and there are only three shapes (e.g., circle, square, triangle), shape cannot be used to represent temperature. Furthermore, a continuous data domain must be represented in a continuous way.

Any mapping meeting these criteria will be adequate from a purely logical standpoint, but may be deficient from psychological one, either for interpretive or perceptual reasons. Considering the two stages of using a presentation—identifying the encoding conventions of the graphical language, then making use of these conventions to extract information—it is possible point to several guidelines to help guarantee the effectiveness of identifying individual mappings:

- *Semantic perceptual dimensions should be perceptually salient—i.e., a viewer must be able to identify which dimensions actually encode information.*

- *The mapping should be "natural"—i.e., easy to grasp and remember. This is true both at the level of entire dimensions and at the level of individual values.* For example, using the size of the circle to encode a data domain that isn't an amount (e.g., the date of an event) is likely to be confusing, since it is natural to see size as encoding an amount. Furthermore, with respect to individual values, if size is used to encode some domain that is thought of as an amount, larger sizes should encode larger amounts, and in a manner such that the relative proportions of different values can be estimated by a viewer.

Once the encoding conventions have been understood, the main issue affecting effectiveness is:

- *A viewer should be able to quickly and accurately identify values for a given property or perceptual function.* Different perceptual dimensions can be estimated with different degrees of speed and accuracy (e.g., the position of the circle can be estimated much more accurately than its value), so more effective perceptual dimensions should be used wherever possible. Furthermore, the individual values in these dimensions should be chosen for maximum discriminability.

In order to take these issues into account when designing graphical languages, an automatic presentation system must have access to psychological knowledge concerning the perceptual salience of different perceptual dimensions, the factors that make mappings difficult or easy to interpret, and the effectiveness of discriminating different perceptual dimensions.

It is also possible discern certain features of the data characterization that will be useful to an automatic presentation system:

- The data characterization should describe individual domains as discrete or continuous and as ordered or unordered, specify the number of values in a discrete domain and the number and order of values in an ordered domain, and provide other semantic information about domains if possible (e.g., whether a domain describes an amount or a coordinate).

Constraints on sets of properties of a single objects

With regard to mapping a *set* of domains and domain values onto the properties of a single object or perceptual functions that object participates in, there are two major logical issues.

The first issue relates to the use of non-orthogonal properties:

- *If multiple perceptual properties of some object being used to encode data are interrelated, the data domains they encode must be related in a similar way.* For example, population density, area of a country, and total population can be mapped to the height, width, and area of a rectangle, but these perceptual properties could not be used to represent three independent domain values, because they would overdetermine the underlying parameters of the graphic object.[21]

The second important logical issue in mapping a set of domain values to a set of property values relates to the perceptual equivalence of objects with different perceptual property values.

- *If an object may appear precisely the same with different perceptual property values, distinguishing between the identical-seeming property values must be semantically irrelevant.* Although in general two objects with different property values will be perceptually distinguishable, there are cases where this is not true. For example, if the two endpoints of a line are used to encode two different pairs of data values of an arity-4 tuple, it will be impossible to know which endpoint is which, and any such line could potentially encode either of two tuples, or both. This is likely to violate the logical requirement of differentiation, and must be avoided. Note that if the data to be presented is such that both tuples will be in the dataset if either one is, this will not be problematic in and of itself. The nature of this problem is discussed more extensively in the next section.

The relevant perceptual issues at this level of mapping are similar to those at the previous level, only related to the influence of different perceptual dimensions on each other. With regard to identification, if multiple properties of a set of objects vary, different perceptual dimensions may appear salient. For example, if the lengths and widths of a set of rectangles are both varied, the areas of these rectangles may become more salient than either their lengths or their widths. With regard to extraction, the discriminability of some perceptual dimensions may be affected by other perceptual dimensions. For example, it is easier to estimate the relative heights of two rectangles if the bottoms of the rectangles are aligned, and it is easier to represent the relative sizes of two circles if they are near each other.

Finally, it is possible to suggest another guideline for the use of multiple properties of a single object, to help ensure ease of interpretability:

- *Where possible, when multiple properties of an object are used to encode information, the mappings they employ should be consistent with one another.* For example, if two properties of an object are used to represent two values from one domain, they should ideally be similar properties (e.g., the height and

[21] For convenience, I will speak about properties of perceptual objects, but in many cases this should be taken more generally to include perceptual functions that the object participates in. This should be clear from context.

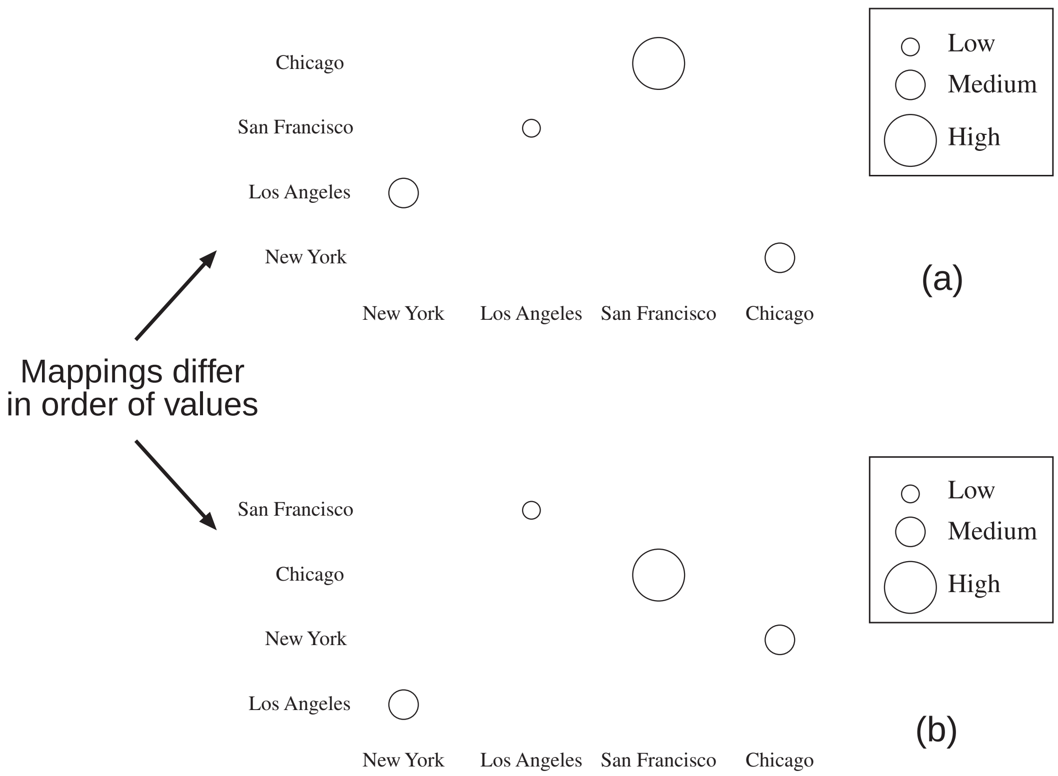


Figure 3.14: Mapping consistency: (a) Consistent (b) Less-consistent use of horizontal and vertical position

width of a rectangle, or the horizontal and vertical position of a circle), and the mappings of individual data values to property values should be similar as well (compare figure 3.14.a with figure 3.14.b). This is likely to make the mapping easier to grasp and to remember. Note however that while using the same mappings may expedite the identification of the encoding conventions, using slightly different mappings (e.g., reordering the rows of a dot plot) may make certain trends more perceptible in some datasets. This is effectively a trade-off between interpretability and perceptual effectiveness.

To take into account all of these issues, an automatic presentation design system should have access to psychological knowledge concerning how perceptual salience and the discriminability of values in perceptual dimensions are influenced by variation in other perceptual dimensions. It should also have knowledge about which pairs of perceptual properties can be considered similar.

To permit sound decision-making concerning multiple properties, the data characterization should:

- Describe mathematical relationships between the different values of tuples (e.g., *duration* = *end-day* – *start-day*), if non-orthogonal perceptual properties are to be used.
- Describe pairs and groups of tuples that occur together in datasets (e.g., $< a_1, b_1, a_2, b_2 >$ in the dataset implies that $< a_2, b_2, a_1, b_1 >$ is also in the dataset) in order to permit the use of objects which can be perceptually equivalent with different perceptual property values.

Constraints on multiple objects

If individual mappings of properties and perceptual functions to data domains in a graphical language are acceptable, and all the property mappings for a single object are compatible, it is still possible for a graphical language to fail to be logically expressive for certain types of data because of the way compound objects are formed. Consider the network graph in figure 3.5, using a graphical language where cities are encoded by the labels of circles, these circles are shared (i.e., there is only one circle for each city), and flights between cities map to the compound objects formed by a line touching a pair of labeled circles. If the dataset to be represented is $\{$*flight(Paris, New York)*, *flight(New York, Los Angeles)*, *flight(San Francisco, Los Angeles)*$\}$, the graph in figure 3.5 and the graphical language it uses are inadequate, because it might also be thought to represent the relations *flight(Paris,New York)*, *flight(New York, Los Angeles)*, and *flight(San Francisco, Los Angeles)*. This graphical language—using labeled circles with a line between them—is only expressive for relations that are symmetric (i.e., for datasets in which any tuple $< a_i, a_j >$ in a dataset is accompanied by a tuple $< a_j, a_i >$).

For other datasets, this language is not expressive, because it violates the logical requirement of disjointedness. For such datasets, a character class describing a pair of circles with a line between them might be used to present any of three separate compliance classes—i.e., classes for each of the two possible tuples that the single compound object might represent, and a class for both tuples. For the graphical language to be considered expressive, it must be used to present datasets in which there can be a single character class for each compound mark, corresponding to a single compliance class representing a pair of tuples. In

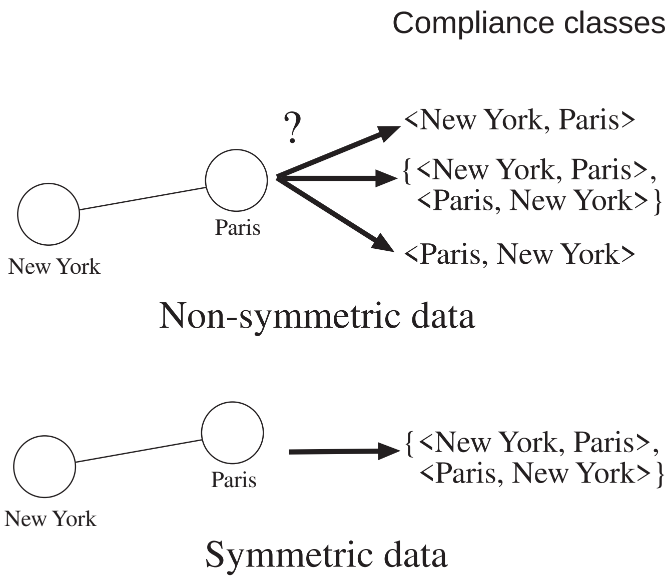


Figure 3.15: A disjointedness violation leads to ambiguity for non-symmetric data

the example above, two circles labeled *New York* and *Paris* with a line between them can encode the *pair* of tuples {<*New York*, *Paris*>, <*Paris*, *New York*>}. As long as tuples only occur in symmetric pairs, decoding the presentation will be possible. This is illustrated by figure 3.15.

Graphical languages that are inexpressive for certain datasets because they may implicitly represent tuples not in the datasets violate what Mackinlay (1986b) has referred to as the *strict condition* of expressiveness. Note that strict expressiveness problems may manifest themselves when presenting single tuples in some languages and only groups of tuples in other languages. Strict expressiveness problems also subsume the problem of perceptual equivalence described in the last section; a perceptual object which can appear identical with different property values will only be problematic if it violates the strict expressiveness condition.

The other condition of expressiveness, also noted by Mackinlay, is that of *intuitive expressiveness*. A given graphical language meets this condition for a given type of data if and only if any set of tuples which is legal according to the data characterization can be expressed using the language.

This condition will be violated if the properties of the objects and the perceptual relations and perceptual functions they participate in can overconstrain the underlying parameters of the graphic objects in some way. This can occur with perceptual relations alone, or through a combination of properties and perceptual relations. For example, the graphical language of figure 3.16—in which one labeled rectangle containing another is used to indicate a relation (*part-of*, here) between the two domain values represented by the rectangles—cannot be used for any data characterization that permits a dataset with both $< a_i, a_j >$ and $< a_j, a_i >$, because is impossible for a pair of rectangles to contain each other. As another example,

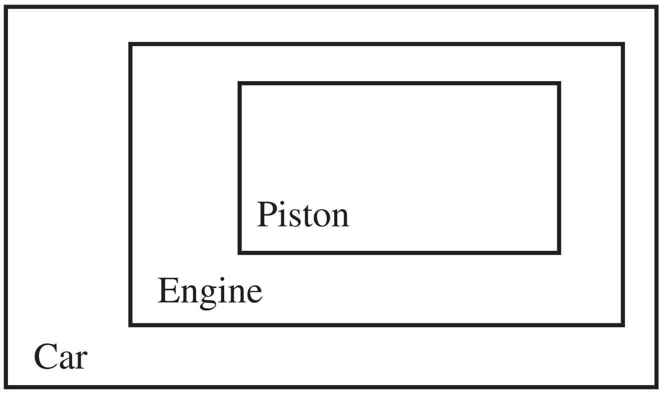


Figure 3.16: A graphical language with logical restrictions

a language using the vertical positions of a pair of circles to represent some data values and simultaneously constraining one circle to be below the other one will only be expressive for a class of data with a characterization specifying $< a_i, a_j > \rightarrow a_i < a_j$ (assuming that the domain A is a numeric range, and that higher (greater) position indicates larger values).

The principle logical constraint on graphical languages which becomes evident at this level of analysis, then, is:

- *A graphical language must conform to the strict and intuitive conditions of expressiveness for the type of data it will be used to present.*

I discuss both intuitive and strict expressiveness at length in the next chapter.

A number of psychological issues pertinent to the expressiveness and effectiveness of a graphical language also arise when considering compound objects. During the identification phase :

- *A presentation should make it easy for a viewer to distinguish semantic objects from non-semantic objects, and also determine which semantic objects encode which domains.*

- *A presentation should make it easy for a viewer to determine which perceptual dimensions encode which data domains.* Although this issue arises even when considering atomic objects, when compound objects are considered consistency issues become very important. For example, while a graphical language using the horizontal position of a circle to encode one domain and the horizontal position of a rectangle to encode another domain may be logically valid, and does not violate the systematicity assumptions, it is likely to be confusing to human viewers. For example, in figure 3.17, tuples of the form <*Semester*, *Students*> (e.g., < *Fall '90, 100* >) are encoded by the horizontal positions of a circle and rectangle, encoding the domain values for *Semester* and *Students*, respectively. The inconsistent use of space makes this presentation confusing.

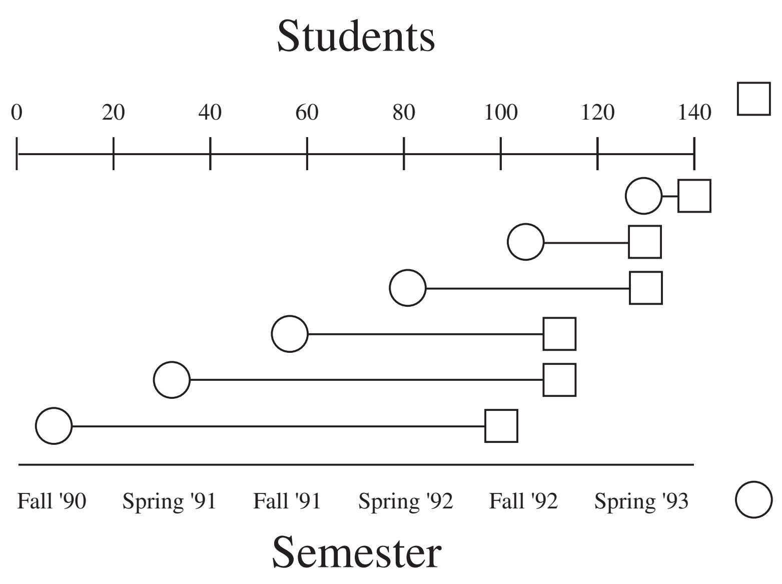


Figure 3.17: An inconsistent presentation

- *A presentation should make it easy to determine which perceptual relations are used to form compound objects*

During the extraction phase, the main psychological issue is the speed of determining perceptual relations. The presence of multiple objects in a presentation with various perceptual relations between them may also affect perceptual discrimination.

To deal with both identification and extraction issues, any psychological knowledge bearing on the expectations viewers carry for how compound objects encode information in graphic presentations, or bearing on perceptual issues affecting the determination of perceptual relations and compound objects, will thus be relevant to an automatic presentation system.

The logical constraints on the expressiveness of the mapping place an important requirement on the data characterization:

- The data characterization should specify which *sets* of tuples are legal. This specification might include, for example, information about the number of tuples a domain value can occur in (e.g., for a binary relation where one domain value is functionally dependent on another, each of the values of the independent domain will occur in at most one tuple) and also logical formulas characterizing legal sets of tuples (e.g., $\forall a_i, a_j, a_k [< a_i, a_j > \wedge < a_j, a_k > \longrightarrow < a_i, a_k >]$). Note that this is basically a refinement of an aspect of the data characterization I discussed relative to constraints on sets of properties of single objects, that groups of tuples which always occur together be characterized as such.

3.4 Summary

This chapter developed a framework for looking at graphic presentations and for designing them automatically from first principles. It began by outlining a few assumptions about how graphic presentations are used and positing *graphic*, *perceptual*, and *cognitive* levels at which any graphic presentation can be described. It also described processes of *perception* and *interpretation* relative to these levels. Having made these psychological assumptions, the chapter then explored a logical framework for looking at the encoding conventions of graphic presentations. This analysis, strongly influenced by the work of Goodman (1968), described how the encoding conventions of presentation can be understood in terms of syntatic (*character*) and semantic (*compliance*) equivalence classes.

In order to make this extremely general framework more useful, the chapter described several explicit assumptions about the systematicity of graphic presentations usable by humans. These *systematicity assumptions* allow the specification of the definition of a graphical language. This definition was then used as the basis for describing a complete space of graphical languages consistent with the systematicity assumptions.

Having described a space of possible languages, the chapter explored limitations on the uses of different languages for different datasets. It gave a rough overview of criteria for ensuring the logical adequacy and perceptual and interpretive effectiveness of graphical languages. These criteria will be explored in much greater detail in the next three chapters of this dissertation.

Chapter 4

Logical expressiveness of simple and composite graphical languages

In the last chapter, I introduced informally the notion of logical expressiveness, that certain graphical languages may not be suitable for certain types of datasets. I further introduced the two conditions required for a graphical language to be expressive for a given type of dataset: *intuitive expressiveness*, which specifies that any set of tuples comprising a dataset of the specified type must be presentable using the language; and *strict expressiveness*, which specifies that any dataset of that type must be presentable without implicitly presenting some tuple not in the dataset. These are necessary and sufficient conditions for a graphical language to be logically expressive.

In this chapter, I formalize these conditions, and describe algorithms for determining whether they are met by a given graphical language for a given type of dataset (i.e., a given dataset characterization) under a set of limiting assumptions. I begin in section 4.1 by discussing intuitive expressiveness and how it can be determined for a characterization of a single dataset. In section 4.2, I similarly discuss strict expressiveness. Finally, in section 4.3, I extend the discussion to cover the problem of presenting multiple types of datasets simultaneously, addressing both general issues that arise and the determination of intuitive and strict expressiveness.

4.1 Intuitive expressiveness

Informally, the condition of intuitive expressiveness will be satisfied by a graphical language for a given type of data when any dataset of the given type can be presented using the language. A graphical language specifies how a dataset will be encoded in terms of a perceptual-level description, but the description of objects with the appropriate property values and participating in the appropriate perceptual functions and perceptual relations must correspond to some realizable graphic-level description. A graphical language that cannot guarantee this for a given class of datasets violates the intuitive expressiveness condition. For example, if some graphical language specifies that presenting a tuple $< a_i, a_j >$ requires placing a circle labeled a_i *above* a circle labeled a_j, the dataset $\{< a_1, a_2 >, < a_2, a_1 >\}$ will not be presentable, because no parameter values for drawing two circles will allow each to be perceived above the other. Thus, this graphical language violates the intuitive expressiveness condition for general datasets, although it may be expressive for datasets with particular characterizations.

To formalize the intuitive expressiveness condition, we must first have a way of describing data characterizations—characterizations specifying which datasets a graphical language should be capable of presenting. This specification might include, for example, information about *functional dependencies* (i.e., that all tuples with certain values for certain domains will have the same values for other domains) and *completeness* (i.e., that there will be a tuple in the database for every value of certain domains or groups of domains). For example, a set of tuples describing revenues for some corporation over a period of years $\{< 1995, 100M >, < 1996, 110M >, < 1997, 150M >, < 1998, 50M >\}$ might be characterized as having a functional dependency of *revenues* on *year* and being complete for the domain *year* (where *year* = $\{$1995, 1996, 1997, 1998$\}$). A data characterization might also include information about pairs or groups of tuples that will always occur together—e.g., $\forall a_i, a_j\, [< a_i, a_j > \longrightarrow < a_j, a_j >]$.[1] I will refer to potential datasets (i.e., sets of tuples) using the letter T, and use the predicate $Legal_C(T_0)$ to indicate that a dataset T_0 is acceptable according to a given characterization C—i.e., is described accurately by the statements in the characterization.[2]

I will refer to the objects, perceptual relations, and perceptual functions in a perceptual-level description as O, R, and F, respectively, and their combination as Φ (i.e., $\Phi = < O, R, F >$). The encoding of a dataset with a set of objects, perceptual relations, and perceptual functions according to the conventions of a particular language L will be described using the notation $T \overset{encodes(L)}{\longrightarrow} \Phi$ and the extraction of information from a set of objects with given properties and perceptual relations and functions using the notation $\Phi \overset{decodes(L)}{\longrightarrow} T$. The nature of encoding is as described in the previous chapter—i.e., a graphical language precisely specifies the perceptual-level description needed to encode any dataset. For convenience, I will refer to the encoding process as *creating* objects and creating or *asserting* perceptual relations and functions between objects. Note however that this discussion will not be concerned with the dynamic aspects of encoding; the central focus will be what can and what cannot be encoded.

Perceptual-level descriptions corresponding to realizable graphic-level descriptions will be described by the predicate *consistent*.

It is possible to formalize the intuitive expressiveness condition for a graphical language L and a data characterization C, then, in a straightforward way:

$$\forall T_0 \left[Legal_C(T_0) \longrightarrow \exists \Phi_0 \left[T_0 \overset{encodes(L)}{\longrightarrow} \Phi_0 \wedge \mathit{consistent}(\Phi_0) \right] \right] \tag{4.1}$$

[1] This notation indicates that if some tuple $< a_1, a_2 >$ is in the dataset then the tuple $< a_2, a_1 >$ will also be in the dataset.

[2] I will make more specific assumptions about data characterization later, when discussing particular algorithms. The general concepts I am discussing apply to all possible data characterization languages.

Intuitive expressiveness thus clearly hinges on two questions: whether perceptual-level descriptions will exist for legal datasets and whether these perceptual-level descriptions will correspond to realizable graphic-level ones. Given that the definition of a graphical language describes how to map datasets onto perceptual-level descriptions, the first question can *almost* always be answered affirmatively. The exceptions arise when the compound objects used to represent tuples contain multiple components of the same type and these components are not guaranteed to be distinct. For example, consider a language in which relations between two data values drawn from the same domain (e.g., a pair of cities) are encoded by compound objects consisting of arrows between a pair of shared, labeled circles encoding the data values. This is a standard language for representing directed graphs. If a relation between two identical values (e.g., the tuple $< Paris, Paris >$) is to be encoded, however, this language does not define a consistent way of creating a compound object encoding that relation. Compound objects in this language consist of three distinct components (two circles and an arrow), and encoding a tuple with identical values could only create two distinct components (one circle and an arrow). Thus this language will be inexpressive if the data characterization permits datasets with such tuples.

The second question, whether a perceptual-level description will correspond to a realizable graphic-level one, requires more discussion. Generally speaking, a set of perceptual objects with certain properties, perceptual relations, and perceptual functions at the perceptual level will be described at the graphic level as a set of graphic objects with particular constraints on their parameters. For example, the perceptual-level specification:

rectangle(r1,width = 1 in)
rectangle(r2,width = 2 in)
above(r1,r2)

might be described at the graphic level as:

rectangle(R1,x0,y0,x1,y1)
rectangle(R2,x2,y2,x3,y3)

x1 – x0 = 1 inch
x3 – x2 = 2 inch
y3 > y0 ;*above(R1,R2)*

with the additional implicit constraints $y1 > y0$ and $y3 > y2$, specifying that the top of a rectangle is above the bottom of a rectangle.

In this example, there is a one-to-one correspondence between the rectangle of the perceptual-level description and that of the graphic-level one. The constraints on the property values (widths) of the perceptual-level translate into simple constraints on the parameters of single rectangle at the graphic level, and the perceptual relation *above* translates into a simple inequality constraint on the parameters of a pair of rectangles. Even if the correspondence between these levels were not so simple, however, it would be reasonable to assume that the perceptual-level description would be translatable into some set of constraints on the parameters of some set of graphic objects. That is, some subspace of the possible space of graphic parameters of a set of graphic objects would correspond to the perceptual-level description. I will generally assume, furthermore, that the appropriate subspace of parameters can be described analytically—i.e., with a set of mathematical constraints. I will make further assumptions when describing particular algorithms for determining expressiveness.

The problem of determining intuitive expressiveness is thus largely one of determining constraint satisfiability. Note that if we were concerned with determining whether the perceptual encoding of some *particular* dataset were consistent, this task could be accomplished by giving the appropriate graphic-level constraints to a constraint solver. If the constraint solver couldn't find a solution, the graphical language could be considered inexpressive for that dataset.[3]

The task at hand, however, is more difficult. It is necessary to determine if a set of constraints on a graphic-level description will be satisfiable for any member of a given *class* of datasets—i.e., all those that satisfy the predicate *legal*. In order to be able to do this, it will be necessary to make certain assumptions about both the data characterization and the types of perceptual relations and perceptual functions available for use in describing graphical languages. I will present here a solution for one particular set of assumptions, then very briefly discuss other possible assumptions. The solution presented here corresponds to the one implemented in the current version of AUTOGRAPH.

Note that if an algorithm checking for intuitive expressiveness is over-conservative—i.e., if it sometimes determines that a graphical language is inexpressive for a given type of dataset when the graphical language actually *is* expressive—a system using it will still be able to function. This assumes, of course, that the algorithm doesn't *always* reject languages as inexpressive. Generally, over-conservative algorithms merely limit the space of graphical languages usable by the system. This idea will apply equally well to algorithms used to determine strict expressiveness.

4.1.1 Solution: Mutually independent perceptual relations and independent data values

In order to describe a specific algorithm, it is necessary to make some specific assumptions about data characterizations and perceptual-level descriptions. For this algorithm, I will assume:

- The data characterization consists of:
  - Statements about functional dependencies—e.g., that the domain *enrollment* is functionally dependent on the domain *year*.
  - Statements about completeness—e.g., that there is a tuple for every value of the domain *year* in the dataset.

[3] I am assuming here that the solver would find a solution if one existed. However, if a solution did exist but could not be determined by a constraint solver, the graphical language might as well be considered inexpressive, since a presentation employing it could not be rendered.

– Logical *implication* rules specifying conditions under which certain tuples must be in the dataset in order for the dataset to be consistent, expressed in the form:

$$t_0 \wedge t_1 \wedge \ldots \wedge t_n \longrightarrow t'$$

where each t refers to a specification of a tuple with variables, and all the variables used in t' also occur on the left side of the implication. For example $< city_i, city_j > \longrightarrow < city_j, city_i >$ could be used to indicate that a dataset of flights between cities is symmetric, in that if there is tuple representing a flight between a pair of cities in the datset, there will also be a tuple representing a flight in the reverse direction.

– Logical *restriction* rules specifying conditions under which certain tuples *cannot* be in the dataset, expressed in the form:

$$\neg t_0 \vee \neg t_1 \vee \ldots \vee \neg t_n \vee equality_0 \vee equality_1 \vee \ldots \vee equality_m$$

where each t refers to a specification of a tuple with variables and each *equality* specifies that certain variable values must be equal.

For example, a simple functional dependency could be expressed as:

$$\neg < a, b_1 > \vee \neg < a, b_2 > \vee (b_1 = b_2)$$

This rule specifies that two distinct tuples with the same value for a (and different values for b) cannot be in a legal dataset together, which is equivalent to saying that b is functionally dependent on a. Note that while restriction rules subsume statements about functional dependencies, it is often convenient to talk specifically about functional dependencies, apart from the restriction rules. For convenience, I will assume that functional dependencies will be noted explicitly as dependencies *and* as restriction rules, since it is possible to convert explicitly-noted functional dependencies into restriction rules.

- The data characterization includes no mathematical terms (i.e., terms describing relationships other than equalities and disequalities between tuple values).

- Individual objects at the perceptual level of description correspond to individual objects at the graphic level, and perceptual property values correspond to equality constraints or combinations of inequality constraints on the parameters of the graphic objects—i.e, each perceptual property value corresponds to one or more graphic parameter values or ranges of graphic parameter values.

- All the perceptual relations that can be expressed at the perceptual level are *mutually independent*—i.e., no perceptual relation between two objects is logically inconsistent with any other perceptual relation between any other two objects, or with the lack of a particular perceptual relation between any pair of objects. This excludes perceptual relations such as *above*, *below*, *contains*, *same-color*, etc. but still allows for perceptual relations such as *touches* (e.g., a line touches a circle).

Given these assumptions, it is possible to determine if a given graphical language will satisfy the intuitive expressiveness condition for a given type of dataset by: (1) checking whether the data characterization permits datasets with tuples that will lead to compound objects with insufficient (i.e., indistinct or identical) components and (2) doing a simplified degree-of-freedom analysis on the properties, perceptual functions, and perceptual relations used by the language.

To accomplish the first check, it is necessary to determine if the language encodes information with compound objects, and whether these compound objects include multiple components of the same type. If so, and if these components are shared, there is potential problem. The problem may arise because, given the graphical language definition of the previous chapter, two objects of the same type must encode values from the same domains. Thus, the objects may be identical. To determine whether there will actually be an expressiveness problem, it is necessary to consider whether a legal dataset can contain tuples that will lead to creating compound objects with identical components. This can be accomplished by checking specifications of such tuples against the logical restriction rules in the data characterization.

As an example, consider using a standard network graph language to represent data about traffic between cities, where the data is given by tuples in the form $< City_1, City_2, Traffic >$. Each tuple is supposed to be represented by a compound object consisting of a pair of (shared) circles labeled with names of the cities and a line connecting them whose thickness encodes the traffic between the cities. Since the compound object contains two circles, and they are shared, there is a potential problem: any tuple of the form $< city_i, city_i, \_ >$ cannot be represented, because this would lead to creating a compound object with only a single circle. If such tuples are forbidden by the data characterization, (i.e., by a simple restriction rule such as $\neg < city_i, city_i, traffic_j >$), the language will be expressive. Otherwise, it will not be.

It should be fairly clear that it is possible to determine specifications for tuples that will lead to compound objects with identical components. A more detailed analysis of *legality checking* (i.e., checking whether these tuple specifications are consistent with the data characterization) is presented in section 4.2.1.

The second part of the checking procedure, the degree-of-freedom analysis, can be accomplished as follows. For each component object of the compound object encoding a tuple, or for the single atomic object encoding the tuple:

1. *Determine which perceptual properties of the object are used to encode data values.* Such properties will constrain the parameters of the underlying graphic object to which the perceptual object corresponds.

2. *Determine the perceptual relations and perceptual functions that the component participates in within a single compound object.* This step can be skipped for atomic objects.

3. *Determine if there is a limit to the number of instances of each relation or function the component may participate in, considering both the number of compound objects it may be a part of and the roles it can have in different compound objects.*[4]

[4]By *roles*, I am referring to the different instances of a type of component object within a compound object. For example, if a compound object consists of two circles and an arrow between them, and a given circle can be at the head of one arrow and the tail of

As discussed in the previous chapter, component objects may be either shared or non-shared. As specified by the *systematic component sharing* systematicity assumption of the previous chapter, non-shared components will only participate in the perceptual relations and functions required for a single compound object. Shared components may participate in many compound objects and thus the limit can only be determined by analyzing the data characterization to see what combinations of tuples may be presented.

As an example, consider a graphical language presenting tuples of the form $< A, B >$ with compound objects consisting of shared labeled circles encoding A touching non-shared colored lines encoding B. Since the lines are non-shared, each line will be limited to only one *touches* relation. Each circle, however, will participate in as many *touches* relations as there are tuples with the corresponding value for A. This might be limited by a restriction rule describing a functional dependency of B on A, or specifying that there can be only n distinct tuples incorporating any value a_i—e.g., (specifying a limit of two distinct tuples):

$$\neg < a_i, b_i > \vee \neg < a_i, b_j > \vee \neg < a_i, b_k > \vee (b_i = b_j) \vee (b_i = b_k) \vee (b_j = b_k)$$

If there are no limiting restriction rules, a circle may participate in as many *touches* relations as there are values in B. That is, the circle labeled "A1" may touch one line for each of the tuples $< a_1, b_1 >$, $< a_1, b_2 >$, $< a_1, b_3 >$, etc. In general, the number of compound objects a shared object participates in will be a function of the number of tuples in the dataset incorporating the domain values encoded by that object. This will be limited only by restriction rules or by the number of values in the domains of the tuple.

For practical purposes, if a perceptual relation that some perceptual object participates in will constrain the parameters of the underlying graphic object in any way, more than two or three instances of this perceptual relation will overconstrain the parameters. For example, a line can be made to have its endpoints touch each of two circles, but not three circles. Thus, if the limit of how many perceptual relations the object can participate in is not quite low, it is effectively unlimited for purposes of a degree-of-freedom analysis. Given this stipulation, the limit can be determined by checking whether two tuples that will create the perceptual relation or perceptual function for some object can be in a legal dataset together, whether three such tuples can be in a legal dataset, etc. up to some small number of tuples (e.g., four).

Performing this check again involves a problem of checking tuple specifications (in this case, pairs or groups of tuples) against a data characterization, to see if any sets of tuples meeting the specifications can be in a legal dataset together. Performing this legality check is slightly complicated. Since a very similar problem aries in analyzing strict expressiveness, and since the discussion of this problem fits more naturally into that section (section 4.2.1), I will defer a detailed discussion of legality checking until then.

another, that circle is playing different roles in different instances of the compound object.

Note that this step can be skipped for atomic objects.

4. *Tally the maximum number and type of constraints on the parameters of the underlying graphic object corresponding to the component, assuming for each perceptual relation or function that it will only constrain the parameters of one of its objects, and the same object each time.*

   As an example of the mentioned assumption, the algorithm might assume that a perceptual relation *touches* between a line and a circle will always constrain the position of one of the endpoints of the line, rather than the circle.

5. *Determine if any of the graphic objects will be overconstrained.* Given the number and type of constraints on the parameters of a graphic object, it is relatively simple to determine if the object is overconstrained, for any fixed set of object types and constraints. For use in an algorithm where speed is important, combinations of properties and perceptual relations which result in overconstraining an object can be determined in advance and stored for each perceptual object type.

A couple of examples will serve both to clarify the concept of intuitive expressiveness and to illustrate how a such degree-of-freedom analysis works. Consider first a graphical language where the vertical positions of the top and bottom of a rectangle and also the height of the rectangle are used to encode data values, and the rectangles aren't part of larger compound objects. Although the perceptual properties being used are the only source of constraints on the underlying graphic parameters of the rectangles, there are only two underlying graphic parameters specifying the vertical extent and position of the rectangle (*y1* and *y2*), and three independent property values constraining them (*top*, *bottom*, and *height*), so the parameters must be overconstrained. Note that if the data characterization language allowed the specification of mathematical relationships between data values, e.g.,

$$\forall A_1, B_1, C_1 \,[< A_1, B_1, C_1 > \longrightarrow C_1 = B_1 - A_1]$$

this graphical language might be usable for certain types of datasets. For example, the vertical position of the top of a rectangle could be used to encode the *start time* for a meeting, the vertical position of the bottom of the rectangle the *end time* of a meeting, and the height of the rectangle the *duration* of the meeting. However, a somewhat more complicated expressiveness-checking algorithm would be needed to make this determination.

Consider also another graphical language in which shared lines of different colors encode different airlines, and are made to touch two (shared) labeled circles to indicate flights between two cities offered by these airlines.[5] Following the procedure described above, the intuitive expressiveness of this language will be analyzed as follows (combining the analysis of both object classes at every stage):

1. The colors of the lines and the labels of the circles encode data.

2. Circles may touch lines, and vice-versa.

[5] Note that in a normal network graph, lines are non-shared —i.e., a new line is created for each symmetric pair of tuples. In the language being discussed, there will be only a single line of each color.

3. Without logical restriction rules, circles may participate in as many compound objects as there can be tuples involving that city in the dataset—a function of the numbers of cities and airlines. Lines may participate in as many compound objects as there can be tuples involving the relevant airlines in the dataset—a function of the number of cities.

4. Assume that the perceptual relation *touches* for lines and circles constrains lines rather than circles. For circles, then, only their labels will be constrained. For lines, in addition to their color being constrained, their positions may be constrained by three or more *touches* relations. That is, there is nothing preventing a legal dataset from containing two tuples involving the same airline and different cities.

5. Lines will be overconstrained by the touches relations they can participate in. A line cannot touch three or more independently-placed circles.[6] This language will thus be overconstrained, and inexpressive.

Note that if the lines were not shared in the above example, new lines would be created for every flight and would participate in only two *touches* relations each—i.e., touch only two circles. The positions of the lines would not be overconstrained and the language would satisfy the intuitive expressiveness condition.

4.1.2 Limitations and possible alternative algorithms

The algorithm just presented is over-conservative, in the sense described earlier, due to the assumption that perceptual relations and functions must be ensured by constraining the parameters of only one object. This assumption is convenient for building a layout engine, but it is overly restrictive. Percepual relations and functions should really constrain the parameters of two objects relative to each other. If a line is constrained to touch a circle, it should be possible to satisfy this constraint by changing the parameter values of (i.e., moving) the line *or* by changing the parameter values of the circle. In a graphical language using compound objects composed of lines between positioned circles, having the positions of the circles determine the placement of the lines is appropriate. In a language using the position of a circle, the length and orientation of a line touching the circle, and the color of a rectangle touching the line to encode information, the constraints should be solved differently. The algorithm described above will not consider this possibility.

It seems clear that this limitation can be addressed, at least partially. Clearly, it should be possible to modify the algorithm to perform a more flexible degree-of-freedom analysis, assuming that constraints used to ensure perceptual relations may be propagated in different directions to present different combinations of perceptual relations of a compound object. However, to be maximally flexible, an algorithm should be able to consider propagating constraints differently on an instance-by-instance basis. That is, for some graphical languages, it might be necessary to ensure "line touches circle" perceptual relations in certain compound objects by constraining the lines and in others by constraining the circles. Such an analysis would need to be significantly more complicated than the analysis described above, in order to proceed from knowledge of the data characterization alone, rather than the actual data.

[6] Note that by referring to a line *touching* a circle, I mean that an endpoint of the line touches the circle.

The algorithm just described has additional limitations as well. It is limited by its inability to make use of data characterizations expressing mathematical relationships between data values in tuples. It is also limited by its inability to deal with perceptual relations that are not mutually independent, such as containment.

These limitations could also be addressed, conceivably. It is easy to envision an algorithm that could make use of inter-dependent perceptual properties of a single object, comparing the mathematical relationships of these properties to a limited set of mathematical data characterizations. Similarly, it is possible to imagine creating an algorithm for presenting binary relations with a limited set of data characterizations. Such an algorithm might easily determine where non-mutually independent perceptual relations could be used. For example, containment could be used for *anti-symmetric* tuples— i.e., datasets where tuples imply a partial order of their values.[7]

4.2 Strict expressiveness

In order to be considered expressive, a graphical language must satisfy both the intuitive condition of expressiveness just described and also the strict condition of expressiveness. The strict condition of expressiveness will be satisfied for a given type of data when all of the tuples in any dataset of that type can be encoded without implicitly presenting any other tuples *not* in the dataset. In my framework, this means that the perceptual-level description of the encoding of some legal dataset in terms of objects, perceptual relations, and perceptual functions, must not either contain or imply a perceptual-level description that can be decoded to yield some other dataset, or the strict condition will be violated.

For example, consider a standard network graph in which flights between cities are encoded by lines between labeled circles. The perceptual-level description of the presentation created for *flight(New York, Paris)* would look something like:

circle(c1,label = "New York")
circle(c2,label = "Paris")
line(l1)
touches(l1,c1)
touches(l1,c2)

Due to symmetries in the graphical language, this description contains all the objects and perceptual relations needed to present the relation *flight(Paris, New York)* as well as the relation *flight(New York, Paris)*. That is, these two compliance classes with different semantics map to the same character class—a pair of labeled circles with a line between them. Thus, a tuple of the form $< city_i, city_j >$ will always be presented implicitly if $< city_j, city_i >$ is encoded explicitly. This will violate the strict expressiveness condition unless the graphical language is only used to represent datasets in which tuples always occur in symmetric pairs.

[7] Such an approach is taken in the AVE system (Golovchinsky *et al.* 1995; Reichenberger *et al.* 1995) described in chapter 2.

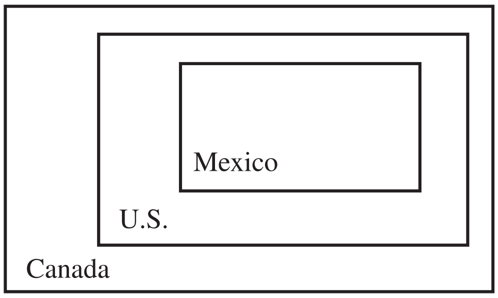


Figure 4.1: A language with implicit encoding problems

Consider also a graphical language in which facts about countries bordering each other are encoded by a pair of labeled rectangles, one of which contains the other. The perceptual-level description of the dataset $\{< \mathit{Canada}, \mathit{US} >, < \mathit{US}, \mathit{Mexico} >\}$ would look like:

rectangle (r1,label = "Canada")
rectangle (r2,label = "US")
rectangle (r3,label = "Mexico")
contains(r1,r2)
contains (r2,r3)

This perceptual-level description implies another perceptual relation, however—*contains(r1,r3)*—because containment is inherently transitive. Thus the presentation shown in figure 4.1 includes all the perceptual objects and relations necessary to implicitly encode the tuple $< \mathit{Canada}, \mathit{Mexico} >$. Again, this is a violation of the strict expressiveness condition, though a graphical language of this type could be expressive for a more restrictive data characterization.

Informally, a graphical language will satisfy the strict expressiveness condition for a given data characterization when no legal dataset, when encoded, can be decoded to yield some other dataset. Formally, it will be more instructive to describe the conditions under which the strict expressiveness condition is violated. Using the same terms I used when discussing intuitive expressiveness, and in addition the notation $\Phi_1 \stackrel{implies}{\longrightarrow} \Phi_2$ to describe how some perceptual-level description can imply another with additional perceptual relations, the narrowest set of conditions under which the strict expressiveness condition will be violated for a graphical language L and data characterization C can be formalized as follows:

$$\exists T_0, T_1 \left[legal_C(T_0) \wedge \left[T_0 \stackrel{encodes(L)}{\longrightarrow} \Phi \stackrel{implies}{\longrightarrow} \Phi^* \stackrel{decodes(L)}{\longrightarrow} T_1 \right] \wedge \exists t_0 \left[t_0 \in T_1 \wedge t_0 \notin T_0 \right] \right] \tag{4.2}$$

with the additional stipulation that the perceptual objects and relations needed to represent T_1 in the above formula include all the perceptual objects and relations of Φ^*—i.e., if $T_1 \stackrel{encodes(L)}{\longrightarrow} \Phi'$, then $\forall \phi \in \Phi^* \; [\phi \in \Phi']$.[8]

[8]Note that in my analysis, I am assuming that objects cannot be implied, which is not an insignificant assumption. For example,

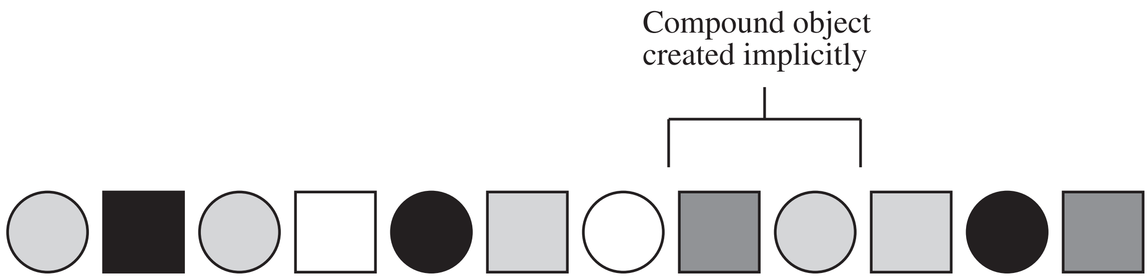


Figure 4.2: A globally unambiguous presentation

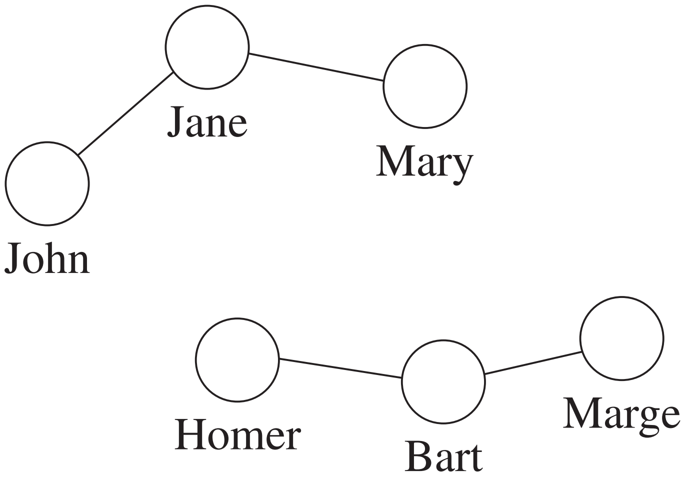


Figure 4.3: A locally unambiguous presentation

This additional stipulation specifies there must not be any objects or perceptual relations in the perceptual description that do not seem to have meaning in the incorrect interpretation of Φ^*, which could indicate that the interpretation/decoding of the presentation as T_1 is incorrect. This permits graphical languages that create presentations that cannot be locally disambiguated (i.e., which cannot be decoded to yield a unique dataset based on local considerations alone) to be considered expressive. For example, if the grayscale values of an adjacent square and circle represent a tuple of arity 2 (with one domain of the tuple represented by the value of the square and one by the value of the circle), and the graphical language specifies that each square and circle may be part of only one compound object (i.e., that they are non-shared), then the presentation in figure 4.2 is unambiguous, since the circles and squares can be segregated into a unique set of adjacent pairs. This cannot be done without examining up to half the row of objects, however.

An alternative formulation of the strict expressiveness that might be considered, the broadest def-

this excludes the possibility that a configuration of lines could implicitly create a rectangle. Still, this assumption will hold under most conditions.

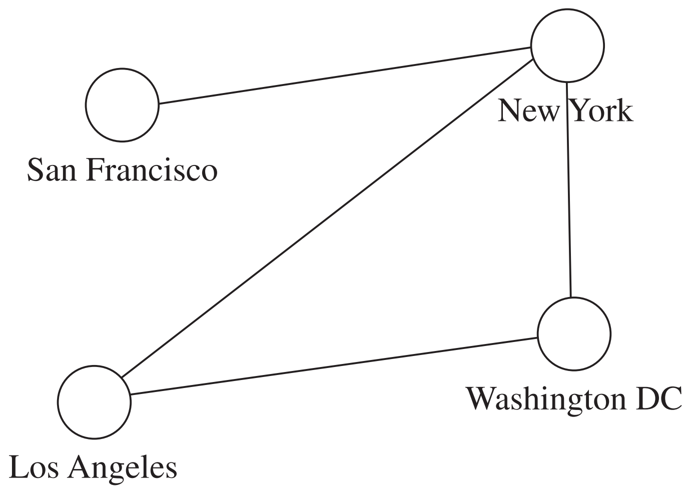


Figure 4.4: A simple network graph

inition, forbids such graphical languages by eliminating the additional stipulation. With this formulation, any occurrence in the perceptual-level description of a compound object representing a tuple that was not explicitly encoded would result in ambiguity, even if context might allow for disambiguation. Still other formulations that might be considered would allow for limited local disambiguation. For example, to present parent/child relationships for a group of schoolchildren, knowing that each child had two parents and each parent exactly one child, it would be possible to use a pair of circles and a line to represent the parent/child relationship so that the dataset $\{< John, Jane >, < Mary, Jane >, < Homer, Bart >, < Marge, Bart >\}$ would be presented as in figure 4.3. Which circles represent parents and which represent children can be determined by local perceptual context (i.e., a circle that has two lines touching it represents a child), even though individual compound objects (each pair of circles connected by a line) would themselves be ambiguous.

For the remainder of this chapter, I will use the broadest definition of strict expressiveness condition violation when considering graphical languages. That is, I will deem graphical languages to violate the strict expressiveness condition whenever the perceptual-level description of some legal dataset would contain or imply a compound object encoding a tuple not in the dataset, regardless of whether the context of that compound object would allow a viewer to determine that it did not actually encode a tuple.[9]

Note that using this formulation, graphical languages which in the course of explicitly representing certain tuples implicitly represent other tuples may be expressive for certain classes of datasets—i.e., for certain data characterizations—while not being expressive in general. This is true of the graphical languages discussed at the beginning of this section. A network graph language such as the one in figure 4.4 is expressive

[9] It might seem preferable to view these varying definitions of the strict expressiveness condition as the same formal definition, modified by a parameter for computational effort. I prefer to view it as a formal, distinction because the broadest definition seems to most closely parallel the notion of the strict expressiveness present in previous work such as Mackinlay (1986b).

for a set of data tuples characterized as symmetric (e.g., one in which *flight (city1,city2)* implies *flight(city2, city1)*) and a containment language is expressive for a set of data tuples characterized as transitive and anti-symmetric.

A graphical language will be expressive for a given data characterization if the data characterization limits the datasets which can be *encoded* by the graphical language to those which can be unambiguously decoded. Note that in terms of the concepts of character and compliance classes described in the previous chapter, the set of possible compliance classes needed to represent such datasets can be limited so that there is a one-to-one mapping between character and compliance classes.

As with intuitive expressiveness, I will describe a solution to the problem of detecting possible strict expressiveness violations under particular assumptions about data characterizations and the perceptual relations that may be used to form compound objects.

4.2.1 Solution: Mutually independent perceptual relations

Consider again the assumptions about the perceptual-level descriptions that we made in describing solution 1 for intuitive expressiveness checking in section 4.1.1. These same assumptions will hold for this strict expressiveness checking algorithm. I will additionally assume in this section that perceptual functions are not part of the perceptual-level description, and that perceptual equivalence (i.e., when an object may appear identical to another object with different perceptual property values, as described in the last chapter) will never occur.

Given the mutual independence assumption of section 4.1.1, which states that all combinations of asserted perceptual relations and the absence of such relations are possible, it is possible to rewrite formula 4.2 in the formalization of implicit presentation, eliminating implication:

$$\exists T_0, T_1 \left[legal_C(T_0) \wedge \left[T_0 \overset{encoding(L)}{\longrightarrow} \Phi \overset{decoding(L)}{\longrightarrow} T_1 \right] \wedge \exists t_0 \left[t_0 \in T_1 \wedge t_0 \notin T_0 \right] \right] \tag{4.3}$$

If the only objects and perceptual relations in Φ are those explicitly created and asserted to encode valid tuples, the question of whether the strict expressiveness condition may be violated is a question of whether a set of objects and perceptual relations created to encode a legal set of tuples can be reorganized in such a way that at least one compound object encoding a tuple not in T_0 is present.[10]

Consider for example the presentation of a set of tuples of the form $< A, B, C >$ (i.e., a set of arity-3 tuples with values from the domains A, B, and C) by a set of compound objects composed of labeled circles representing values from domain A, labeled triangles representing values from B, and labeled squares representing values from C, with all components shared. For each compound object, there is a (non-shared) line from the circle to the triangle, and from the triangle to the square. Thus, the dataset

[10] I am assuming that any perceptual relations not explicitly asserted in the perceptual-level description to encode information will not hold inadvertently. For example, if the perceptual-level description of a presentation specifies that a particular line touches two circles, it should be assumed that it does not touch any other circles. That this is possible is guaranteed given the mutual independence assumption.

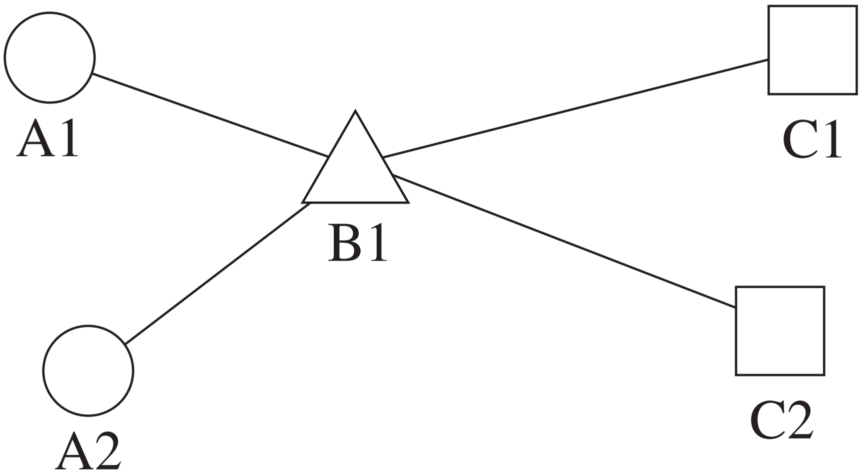


Figure 4.5: A presentation with implicit representation

$\{< a_1, b_1, c_1 >, < a_2, b_1, c_2 >\}$ would be encoded by a presentation such as the one shown in figure 4.5, with $< a_1, b_1, c_2 >$ and $< a_2, b_1, c_1 >$ implicitly encoded.

Now consider the question of how an arbitrary tuple, say $< a_1, b_1, c_1 >$, might be presented implicitly by encoding some other set of tuples with this language. The set of tuples must be encoded by a set of perceptual objects, with certain perceptual relations between them, as follows:

circle(c1,label =“A1”)
triangle(t1,label = “B1”)
line(l1)
touches(l1,c1)
touches(l1,t1)
square (s1, label = “C1”)
line(l2)
touches(l2,t1)
touches(l2,s1)

Any tuple of the form $< a_1, \_, \_ >$ (i.e., any tuple with the value a_1 and any other values for domains B and C) will when encoded create a circle labeled “A1” and a line touching the circle. But in order for a line to be created that touches both a circle labeled “A1” and a triangle labeled “B1”, as in the perceptual-level description above, some tuple of the form $< a_1, b_1, \_ >$ must be encoded. Similarly, for the line between the triangle and the square needed to present $< a_1, b_1, c_1 >$ to be created, some tuple of the form $< \_, b_1, c_1 >$ must be encoded.

The question of whether $< a_1, b_1, c_1 >$ can be encoded implicitly without being encoded explicitly, then, can be reduced to a question of whether two tuples of the forms $< a_1, b_1, \_ >$ and $< \_, b_1, c_1 >$ can be in a legal dataset without $< a_1, b_1, c_1 >$ also being in the dataset.[11]

If the domains A and C are finite, one way of answering this question would be to plug in all possible values for these domains, checking each pair of tuples created to see if they can be in a dataset together, or if, given the data characterization, they imply that $< a_1, b_1, c_1 >$ must also be in the dataset (in which case it would be irrelevant that $< a_1, b_1, c_1 >$ is presented implicitly, because it must be presented).[12] Clearly, though, the strategy of exhaustively checking all possible actual values will be highly inefficient if the data characterization does not refer to specific values. Without specific values, the only determining factors for the legality of a dataset are structural—i.e., whether particular domain values of certain tuples are equal to or not equal to particular domain values of other tuples. Thus, if the dataset $\{< a_1, b_1, c_2 >, < a_2, b_1, c_1 >\}$ is legal and doesn't imply $< a_1, b_1, c_1 >$, the dataset $\{< a_1, b_1, c_3 >, < a_3, b_1, c_1 >\}$, the dataset $\{< a_1, b_1, c_4 >, < a_4, b_1, c_1 >\}$, etc. must also be legal.

Without some sort of restrictive data characterization, the graphical language I have been describing will not be strictly expressive. Encoding the legal dataset $\{< a_1, b_1, c_2 >, < a_2, b_1, c_1 >\}$, for example, will cause $< a_1, b_1, c_1 >$ to be presented implicitly. However, a data characterization specifying that $< a_i, b_j, c_k > \wedge < a_l, b_j, c_m > \longrightarrow < a_i, b_j, c_m >$, or specifying that A and C are functionally dependent on B (i.e., that no pair of tuples in the dataset will have the same value for domain B) will make the graphical language expressive. In both of these cases, there can be no legal dataset with a pair of tuples of the form $\{< a_1, b_1, \_ >, < \_, b_1, c_1 >\}$ that doesn't also contain $< a_1, b_1, c_1 >$, in the first because the dataset will imply $< a_1, b_1, c_1 >$ and in the second because unless both of the pair of tuples are equal to $< a_1, b_1, c_1 >$, the dataset will violate the functional dependency described in the data characterization.

The general strategy used above—assume that there is some tuple presented implicitly without being encoded explicitly, then try to determine whether this is actually possible, given the graphical language and data characterization—can be applied to any graphical language and data characterization, and summarized (with some simplification) as follows:

1. Choose an arbitrary tuple t_0. In the example just discussed, the arbitrary tuple would be $< a_1, b_1, c_1 >$.

2. Segregate the corresponding compound object (i.e., an object that would encode t_0) into *clusters*—groups of objects and perceptual relations that must be created to encode a single tuple. In the example above, there would be two clusters: one composed of a circle labeled A_1, a triangle labeled B_1, and a line between them; the other composed of a triangle labeled B_1, a square labeled C_1, and a line between them.

3. Find *tuple fragments* for each cluster specifying classes of tuples which will create the cluster when encoded. In the example above, the tuple fragments would be $< a_1, b_1, \_ >$ and $< \_, b_1, c_1 >$.

4. Try to find tuples instantiating the tuple fragments which can be in a dataset together, legally, and which do not imply t_0 (i.e., that t_0 is in the dataset), according to the data characterization. In the

[11] In relation to formula 4.3, a dataset described by $\{< a_1, b_1, \_ >, < \_, b_1, c_1 >\}$ is a subset of T_0, and $< a_1, b_1, c_1 >$ is t_0.
[12] I will discuss the nature of this check shortly.

example just discussed, this could be the set $\{< a_1, b_1, c_2 >, < a_2, b_1, c_1 >\}$, at least for certain data characterizations.

If such tuples can be found, they prove that the graphical language can violate the strict expressiveness condition for some legal datasets. If they cannot be, the language is strictly expressive.

I will discuss each of these steps in turn.

Choose an arbitrary tuple

As mentioned above, the strict expressiveness checking procedure begins by assuming that there is some tuple t_0 presented implicitly without being encoded explicitly. It then tries to determine whether this is actually possible, given the graphical language and data characterization. For this purpose, one tuple is as good as another, unless the two tuples are somehow structurally different—i.e., unless the equalities and disequalities among the domain values are different—because the data characterization cannot describe specific domain values, only relationships among domain values. For example, if there is no way of presenting $< a_1, b_1, c_1 >$ implicitly, there cannot be a way of presenting $< a_1, b_2, c_2 >$ or $< a_2, b_2, c_2 >$ or $< a_6, b_4, c_5 >$ implicitly. If the tuples in the dataset contain multiple domain values of the same type, structural differences are possible—e.g., $< a_1, a_1 >$ is different than $< a_1, a_2 >$. In these cases the algorithm must be run multiple times—once for each structurally different possibility, with a different arbitrary tuple each time.

Segregate the compound object into clusters

In order for a compound object to be created implicitly, all of the components of the object and all of the requisite perceptual relations between these components must be created in the course of encoding other tuples. When a tuple is presented explicitly, all of the components and the perceptual relations between them are created and asserted simultaneously. These relations may be created without being asserted simultaneously, however. Presenting one tuple may create some objects and establish certain relations; presenting another tuple may create other objects and establish relations between these and the previously created ones. In order for perceptual relations that are asserted in the course of explicitly presenting different tuples to refer to the same object, however, this object must be shared. The use of non-shared objects necessitates that all of the perceptual relations pertaining to a non-shared object be asserted in the course of encoding a single tuple, meaning that the non-shared objects cannot be the intersections of compound objects.

For example, consider the graphical language of figure 4.5, modified so that triangles are never shared. With this new language, the set of tuples $\{< a_1, b_1, c_2 > , < a_2, b_1, c_1 >\}$ will be presented as in figure 4.6, and no tuples will be represented implicitly. Note that intermediate objects (i.e., objects that don't encode any information individually, such as the lines in figures 4.5 and 4.6) are always non-shared.

I will define a *cluster* as a group of objects and perceptual relations that can only be created and asserted in the course of encoding a single tuple. A compound object will be created if and only if every

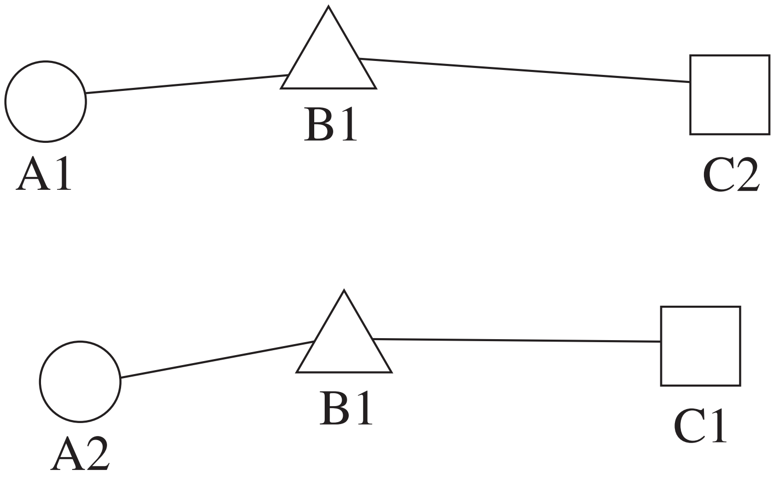


Figure 4.6: A graph without implicit representation

cluster in the object is created, possibly each by a separate tuple. The clusters in a compound object will be:

- Every pair of shared objects connected by some perceptual relation, and the perceptual relation that connects them.
- Every set of connected non-shared objects (not connected through shared objects), the shared objects they connect to, and the perceptual relations that connect them.

For example, in figure 4.5, there are two clusters: the shared circle and shared triangle connected by a non-shared line, and the shared triangle and shared square connected by a non-shared line. In figure 4.6, the shared circle and shared square are connected by a sequence of three non-shared objects (two lines and a triangle) and thus form a single cluster. This difference is illustrated by figure 4.7.

Find tuple fragments corresponding to each cluster

For any cluster, there may be a very large set of tuples that will, when encoded, establish the objects and perceptual relations of the cluster. If the cluster comprises a compound object encoding an entire tuple, only a small set of tuples will establish it (the original tuple and possibly tuples formed by permuting the values of the original tuple). For example, in a standard network graph encoding flight traffic between different cities using non-shared lines of differing thickness between shared circles labeled with the names of the cities, the compound object representing *traffic(Paris, New York, high)* could also be created by encoding *traffic(New York, Paris, high)*, but not by encoding any other tuples.

If the cluster is only *part* of the original compound object, however, there may be a very large or even infinite set of tuples that might establish it when encoded, since only some domain values of the tuple will be encoded by the objects of the cluster and the other domain values can be arbitrary. For example, in the

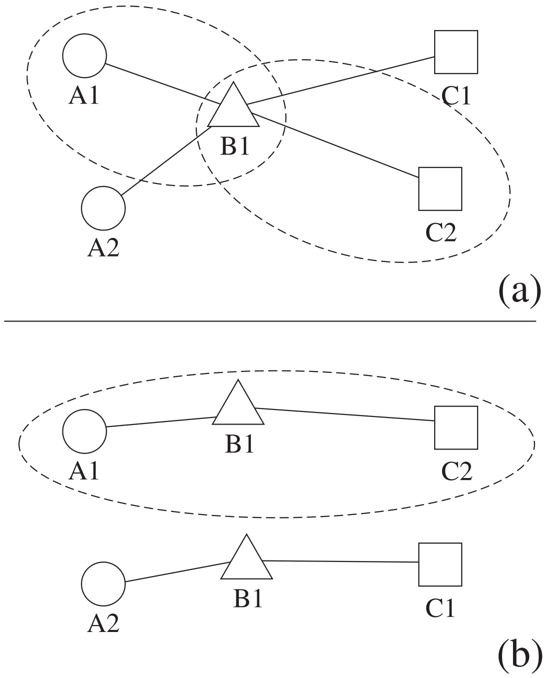


Figure 4.7: (a) Two clusters per compound object with shared triangles (b) One cluster per compound object with non-shared triangles.

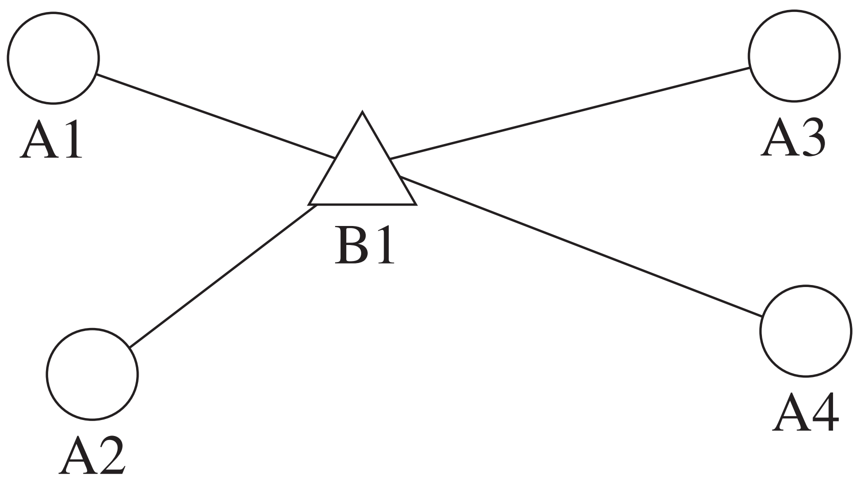


Figure 4.8: A graph with multiple ways of establishing an A–B cluster

graphical language of figure 4.5, any tuple of the form $< a_1, b_1, \_ >$ (e.g., $< a_1, b_1, c_2 >$, $< a_1, b_1, c_3 >$,...) will create the cluster of a circle labeled "A1" connected by a line to a triangle labeled "B1". If a similar graphical language encoded tuples of the form $< A, B, A >$ with (shared) circles encoding A values and (shared) triangles encoding B values, as in figure 4.8, any tuple of the form $< a_1, b_1, \_ >$ *or* of the form $< \_, b_1, a_1 >$ would establish a cluster composed of a circle labeled "A1" connected by a line to a triangle labeled "B1".

The set of tuples establishing a cluster can be described concisely by a small set of *tuple fragments*—tuples with only some values instantiated. The tuple fragment $< a_1, b_1, \_ >$ denotes the (possibly infinite) set of tuples with column 1 = a_1, column 2 = b_1, and column 3 equal to any legal domain value.

Finding the tuple fragments for a cluster can be accomplished directly by examining the compound object for instances of the cluster (i.e., groups of objects of the appropriate types and with the appropriate perceptual relations between them) and setting the columns of some tuple equal to the domain values represented in that instance of the cluster. Note that clusters with inherent symmetry (e.g., two circles connected by a line, but not two circles connected by an arrow) will have multiple corresponding tuple fragments.

Finding a problematic legal tuple set

Each cluster corresponds to one or more tuple fragments, and each tuple fragment describes a (possibly infinite) set of tuples. A dataset including one tuple drawn from each of the sets of tuples corresponding to a cluster, when encoded, will create a compound object encoding t_0, the arbitrary tuple chosen in step 1.[13] This will be problematic—i.e., encoding such a dataset will implicitly represent t_0—unless the dataset will necessarily be illegal, or unless the data characterization implies that t_0 must also be in the dataset. The strict expressiveness violation can thus be re-stated:

$$\exists T_0 \, [T_0 = \{t_1, t_2, \ldots, t_n\} \wedge t_1 \in F_1, t_2 \in F_2, \ldots, t_n \in F_n \wedge legal_C(T_0) \wedge T_0 \not\rightarrow t_0] \qquad (4.4)$$

where F_i refers to the (possibly infinite) set of tuples instantiating one of the set of tuple fragments establishing cluster i, and where t_0 is the arbitrary tuple chosen in step 1.

As discussed earlier, if the data domains of the tuple are all finite, an algorithm could exhaustively generate such a set of tuples. Fortunately, this is not necessary. Since the data characterization cannot describe individual domain values, any consistent substitution of domain values in a set of tuples which preserves equalities and disequalities will preserve the legality or illegality of T_0. Thus, an algorithm only need examine a relatively small set of structurally different tuples. This set can be generated in the following manner.

For each possible choice of n tuple fragments, one from each of the sets of tuple fragments that establish each cluster:

- Construct a set of possible values for each domain of each tuple as follows:

[13] Note that a tuple can be a member of multiple sets. In fact, t_0 itself is a member of every set, since it necessarily contains the values needed to create each cluster.

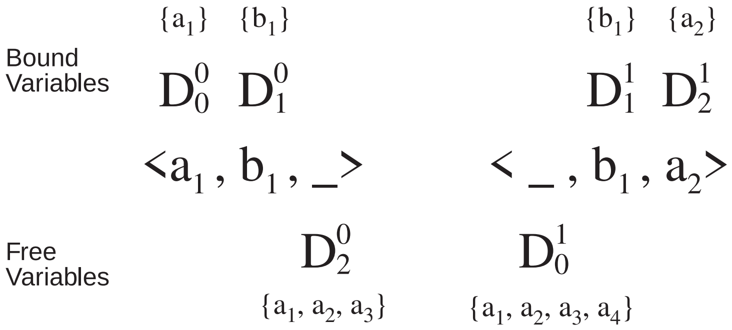


Figure 4.9: Possible values for each column of the tuple fragments

If there are n tuple fragments, and the tuples are of arity m, label the set of possible values for each column in each tuple $D_0^0, D_1^0, \ldots, D_m^0, D_0^1, D_1^1, \ldots, D_m^n$. Sets corresponding to bound variables in the tuple fragments contain only one value, the value of the bound variable. Create the other sets serially (in the order above), making each set contain every possible value created for that domain thus far (including the values of all the bound variables of that domain) plus one new value. Having constructed the sets in this matter, it is possible to choose the values from each set so that every pair of values from the same domain may be either equal or not equal. However, there will be only one set of values given any structure of equalities and disequalities, for any given ordering of the tuple fragments.[14] For example, with the graphical language of figure 4.8, and the initially chosen arbitrary tuple $t_0 = <a_1, b_1, a_2>$ with tuple fragments $<a_1, b_1, \_>$ and $<\_, b_1, a_2>$, the sets will be $D_0^0 = \{a_1\}$, $D_1^0 = \{b_1\}$, $D_2^0 = \{a_1, a_2, a_3\}$, $D_0^1 = \{a_1, a_2, a_3, a_4\}$, $D_1^1 = \{b_1\}$, and $D_2^1 = \{a_2\}$. This example is illustrated by figure 4.9.

If all the domains of the tuples are the same, there can be $O((nm)!)$ possible value assignments, since there will be $O(mn)$ sets of unbound variables, each set containing one more value than the previous one. If all are different, there can be only $O(n!m)$ possible value assignments, since the sets of unbound variables will be divided into m groups (one set per domain of the tuple).

- For each possible choice of domain values, add the resulting tuples to the set.

Each possible value assignment for a set of fragments must be checked for legality, and for not implying t_0. To see how this can be accomplished for a value assignment, a set of tuples which I will refer to for convenience as T_1, consider first data characterizations involving no completeness descriptions. Using the implication rules of the data characterization, the transitive closure of T_1, T^* can be computed. If T^* contains t_0, or if it violates any of the restriction rules of the characterization, T_1 must be rejected—i.e., it is not part of a legal dataset for which the graphical language will violate the strict expressiveness condition.

[14] There may be redundancies, if the tuple fragments are reordered, but this small potential inefficiency should not be of too much concern.

To see how the process can be extended to handle completeness characterizations, note that it can be implemented by a resolution theorem prover, with implication and restriction rules and $\neg t_0$ as axioms and the theorem to be proved the statement that the tuples of T_1 are not in a legal dataset. Seen in this light, completeness rules can easily be implemented as additional axioms, and the process can proceed in an equivalent manner. For simplicity, I will ignore completeness in the remainder of this discussion, however.

To estimate the efficiency of this legality-checking stage, note that in the worst case, the transitive closure T^* may contain all possible tuples which can be constructed using the values of T_1, since implication rules can permute and combine the values of multiple tuples to imply new tuples. For example, if T_1 contains the tuples $< a_1, a_2, b_2 >$ and $< a_3, a_2, b_1 >$, T^* might conceivably contain 18 tuples which can be summarized: $< (a_1|a_2|a_3), (a_1|a_2|a_3), (b_1|b_2) >$ (i.e., each of the 18 tuples corresponds to one choice of domain values). The worst-case combinatorics will occur when all the domains of the tuples are the same and thus have interchangeable values. In this case, if the n tuples of T_1 are of arity m and contain nm different values, there may be up to $(nm)^m$ different tuples in T^*. If all the domains of the tuples are different, n tuples of arity m composed of n different values for each domain might lead to n^m different tuples in T^*.[15]

Assuming that there are k restriction rules of l terms, and the number of tuples in T^* is x, using these rules to check legality will require $k\prod_{i=0}^{l-1}(x-i)$ operations of plugging tuples into the restriction rules, since for each rule, the algorithm will have to plug in a set of l different tuples drawn from the x tuples of T^*.

Since this legality checking operation must be performed for many possible values of T_1, the combinatorial properties of this whole algorithm may make it seem impractical. Multiplying the upper bounds for the different stages, it is possible to calculate an upper bound for the number of operations of plugging tuples into restriction rules, which will be considered primitive operations: $(nm)!k\prod_{i=0}^{l-1}((nm)^m - i)$. However, this worst-case analysis is potentially misleading. First, the values of k, l, m, and n in the above equations will tend to be quite small—all of them except m (the arity of the tuples) are likely to be ≤ 3. Furthermore, as already mentioned, this analysis overstates the worst case, which only occurs when all of the domains of the tuple are the same. More importantly, the worst-case performance is not at all reflective of the average-case performance of the algorithm. Implication rules are unlikely to generate such a large set of new tuples, and for most values of T_1, implication rules will generate t_0 or restriction rules will determine T^* to be illegal without having to go through the entire legality checking process. Furthermore, the checking will terminate when some legal T_1 is found, since this dataset will demonstrate that the graphical language violates the strict expressiveness condition.

Note that in the common case of a data characterization without any implication or restriction rules (i.e., the case where any possible dataset is considered legal), legality checking will not be necessary at all, because all possible datasets will be legal and none will imply any tuples not explicitly given. In cases with small numbers of restriction and implication rules, this algorithm as implemented in AUTOGRAPH generally runs in a reasonable amount of time. It does not seem to account for an overlarge portion of the

[15] Note that the cases just described cannot actually occur, since the nature of how T_1 is constructed ensures that pairs of tuples must have some values in common—i.e., some of the values of t_0—so there cannot possibly be nm different values if all the domains are the same, or n different values if all the domains are different. However, these cases are useful as an upper bound.

overall execution time when AUTOGRAPH finds possible presentations for some input, where execution times are generally under one minute. Further discussion of the performance of AUTOGRAPH occurs in chapter 7.

The legality checking procedure just described also plays multiple roles in the intuitive expressiveness checking algorithm described earlier. It is used both for determining whether any individual tuples of a certain form may exist in a dataset to be presented (to check that compound objects will not have too few distinct components) and in determining the number of compound objects a component may be part of (step three of the degree-of-freedom analysis). The application of the procedure to these problems is straightforward.

4.2.2 Limitations and possible alternative algorithms

The main limitation of this algorithm is its restriction to mutually independent perceptual relations. This is a significant limitation, since it prevents a system from using perceptual relations such as *contains*, *above*, etc. to form compound objects. While it is not immediately obvious how to extend the current algorithm to handle these perceptual relations using the same type of data characterization and tuples, it seems likely that for more restrictive types of tuples (e.g., binary tuples) and data characterizations, a simple algorithm could match perceptual relations to data characterizations.[16]

Another limitation of this algorithm is its inability to deal with perceptual functions and with perceptual properties that can give rise to perceptual equivalence (e.g., the endpoint positions of a line, though the endpoint positions of an arrow are usable). It seems likely that these capabilities could be added by making minor changes to the algorithm.

4.3 Logical expressiveness for multiple datasets

The previous discussions and analyses have generally assumed that an automatic presentation system will be most concerned with presenting single datasets—i.e., single sets of tuples, each tuple consisting of domain values drawn from the same sets of domains. In this section, I will discuss the problems involved in presenting multiple datasets simultaneously, when different datasets may contain different types of tuples. The discussion will include an analysis of how to extend the algorithms for determining intuitive and strict expressiveness. First, however, I will discuss more general issues involved in determining whether multiple graphical languages can be used together to present multiple datasets simultaneously.

4.3.1 Composition

Consider the two presentations shown in figure 4.10, each of which presents a different dataset. Although it is possible to present two datasets in this manner—as two separate presentations on the same

[16] As has been mentioned, this is similar to what is done in AVE (Golovchinsky *et al.* 1995; Reichenberger *et al.* 1995).

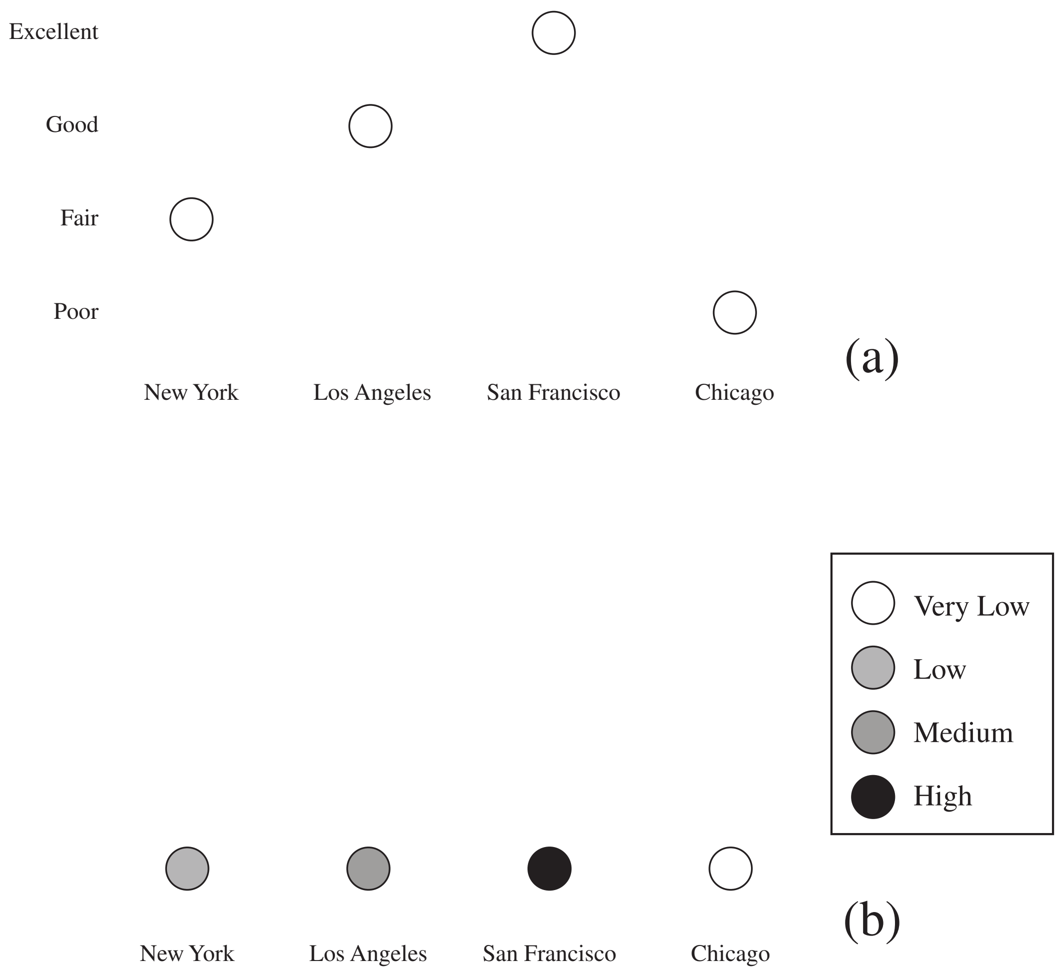


Figure 4.10: Two related but separate presentations

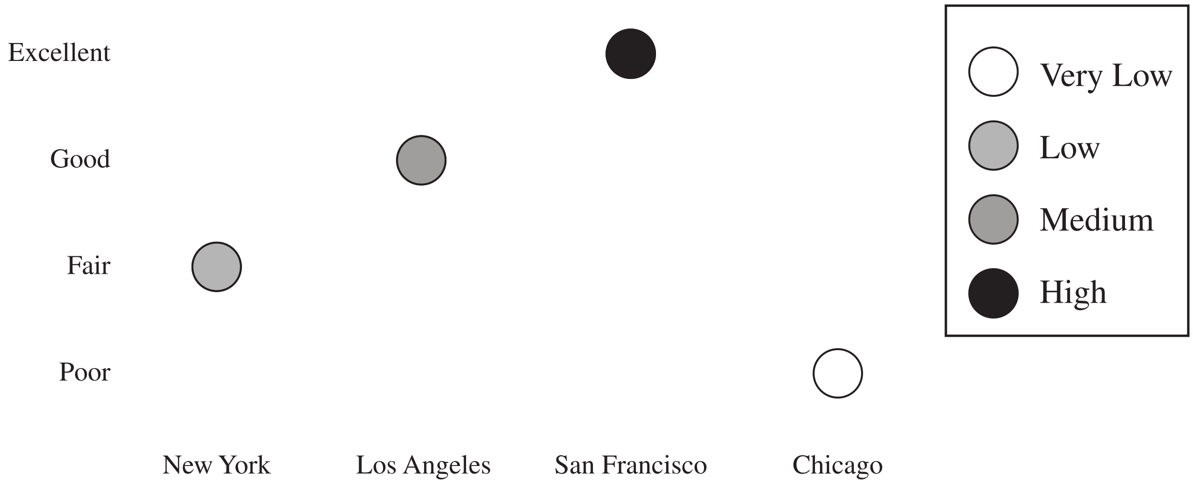


Figure 4.11: Two related, composed presentations

page—it is often preferable to combine them, as in figure 4.11.[17]

The combining of two or more methods of presentation has been referred to by Mackinlay (1986b) as the *composition* of graphical languages. Mackinlay describes several particular methods of composition ("composition operators") suitable for a fixed set of "orthogonal" graphical languages. These methods include *double axis composition*, which superimposes two presentations (e.g., two scatterplots) when their semantics are compatible, *single axis composition*, which stacks and aligns presentations (e.g., puts one bar graph above another) when their semantics are compatible, and *mark composition*, which merges the elements of two presentations, as in figure 4.11. The first two composition operators are effectively capable of modifying a graphical language, or at least the default way it is rendered, in the course of the composition. For example, when merging two scatter plots, double axis composition can change the shapes used by one language so that the objects used for each plot will not be confused. That is, it introduces a new semantic feature, shape, to distinguish the presentation of different datasets. Stacking presentations, too, can be seen as modifying a graphical language, since it introduces a new semantic feature (horizontal or vertical position) to distinguish different datasets.

Rather than restricting ourselves to a specific set of composition operators, I will attempt to treat composition in a more unified manner. The approach outlined here does not address modifying graphical languages when composing them. However, it provides a general way of looking at composition as well as a detailed analysis of when composition will be possible in my framework.[18]

As I will define it, the composition of graphical languages simply involves using two or more sets of encoding conventions simultaneously, where different encoding conventions are used for the presentation

[17] I will defer explanation of the reasons this may be preferable for chapter 5.

[18] It is worthwhile noting that the automatic presentation systems created by Mackinlay and most other researchers rely extensively on graphical languages for presenting binary tuples, thus making the composition necessary in some situations which in my framework can be covered by simple (i.e., non-composed) presentations.

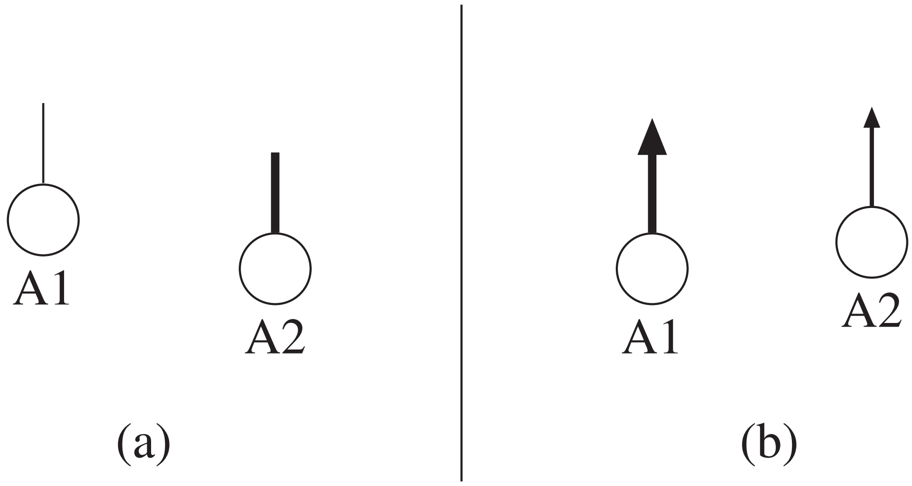


Figure 4.12: Two separate presentations

of different datasets. For purposes of discussion, it will also be convenient to talk about the composition of two or more *presentations*, where objects of the individual presentations may be merged—e.g., the shaded and vertically-positioned circles in figure 4.11 result from merging the positioned circles of figure 4.10.a and the shaded circles of figure 4.10.b.

Generally, of course, the presentation of multiple datasets has much in common with the presentation of a single dataset: atomic objects represent sets of domain values with their property values and perceptual relations combine groups of atomic objects into compound objects which represent tuples, with some domains of the tuple possibly being represented by perceptual functions between components. It is possible to imagine an approach to representing multiple datasets analogous to the approach described in chapter 3 to representing a single dataset, in which partitions would be found for the domains of multiple tuples, mapped to objects, and topologies would be chosen considering all objects equally. I believe that reducing the problem to finding graphical languages for the individual datasets and then merging them is likely to be more efficient, however. In addition, there are scenarios in which adding information to an existing presentation is desirable, making composition a useful technique.

It is worth noting the limitations of the notion of composition that will be discussed here. The graph in figure 4.12.a and the graph in figure 4.12.b—representing information with the labels of circles and the thicknesses of lines (in figure 4.12.a) or arrows (in figure 4.12.b)—can only be composed to yield the presentation of figure 4.13.a, not that of figure 4.13.b. Figure 4.13.b follows the general pattern of breaking tuples into subtuples, representing subtuples by different objects, and forming compound objects representing whole tuples using perceptual relations. However, unlike figure 4.13.a, the compound objects in figure 4.13.b are formed with perceptual relations between components encoding parts of different tuples, thus modifying the encoding conventions of the original presentations. As a result, this presentation cannot be generated by

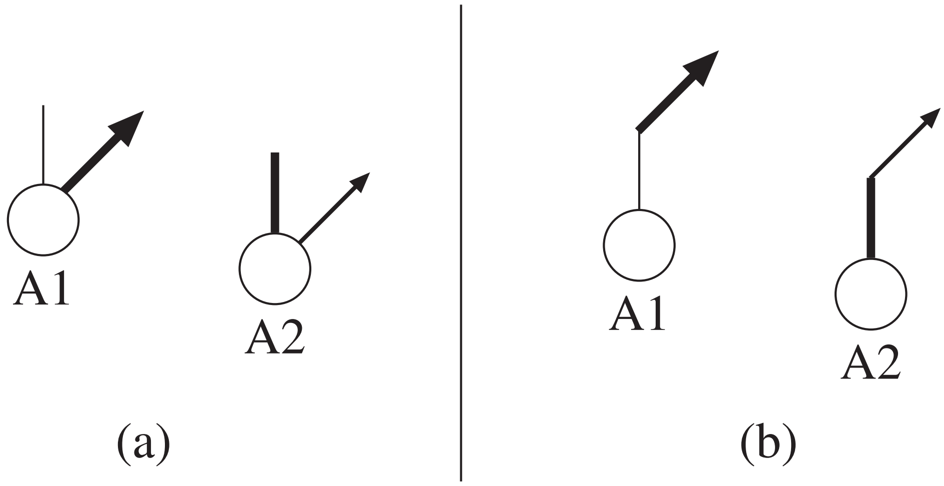


Figure 4.13: (a) Composition covered by the framework (b) Composition outside the limitations of the framework

simply composing two existing presentations.

While the notion of composition presented here is thus limited in some ways, it is still fairly flexible and useful. Future work may provide even more general solutions to this problem.

The concerns of intuitive and strict expressiveness are as applicable to composite graphical languages as they are to simple ones. If multiple encoding conventions are applied simultaneously, there are likely to be more constraints on the underlying parameters of the graphic objects of the presentation and thus more possibilities for violations of the intuitive expressiveness condition. Similarly, there may be more possibilities for compound objects representing tuples to be created implicitly, resulting in violations of the strict expressiveness condition. Before these issues are considered, however, it is necessary to consider more generally the issue of when graphical languages for two datasets can be composed. In my analysis, I will only be concerned with two languages, but my discussion will extend in a straightforward way to more than two languages.

Some of the potential complexities of composition can be illustrated with a few examples. Consider the network graph of figure 4.14.a, showing the volume of traffic on the roads between three cities using the thicknesses of lines between circles labeled with names of the cities. This presentation can be composed with the network graph of figure 4.14.b, which shows which roads are safe and which have hazardous conditions using dashing patterns of lines (i.e., whether they are dashed or solid). The composite presentation is shown in figure 4.14.c. This composition seems like a natural one, but it is only possible because the datasets have information for precisely the same pairs of cities. If the dataset for the graph of figure 4.14.a were missing tuples describing traffic between El Dorado and Atlantis City, for example, it would not be possible to determine the appropriate thickness of the line between the circles with those labels in a composite graph. In order to determine composability from data characterization, then, the data characterizations of these

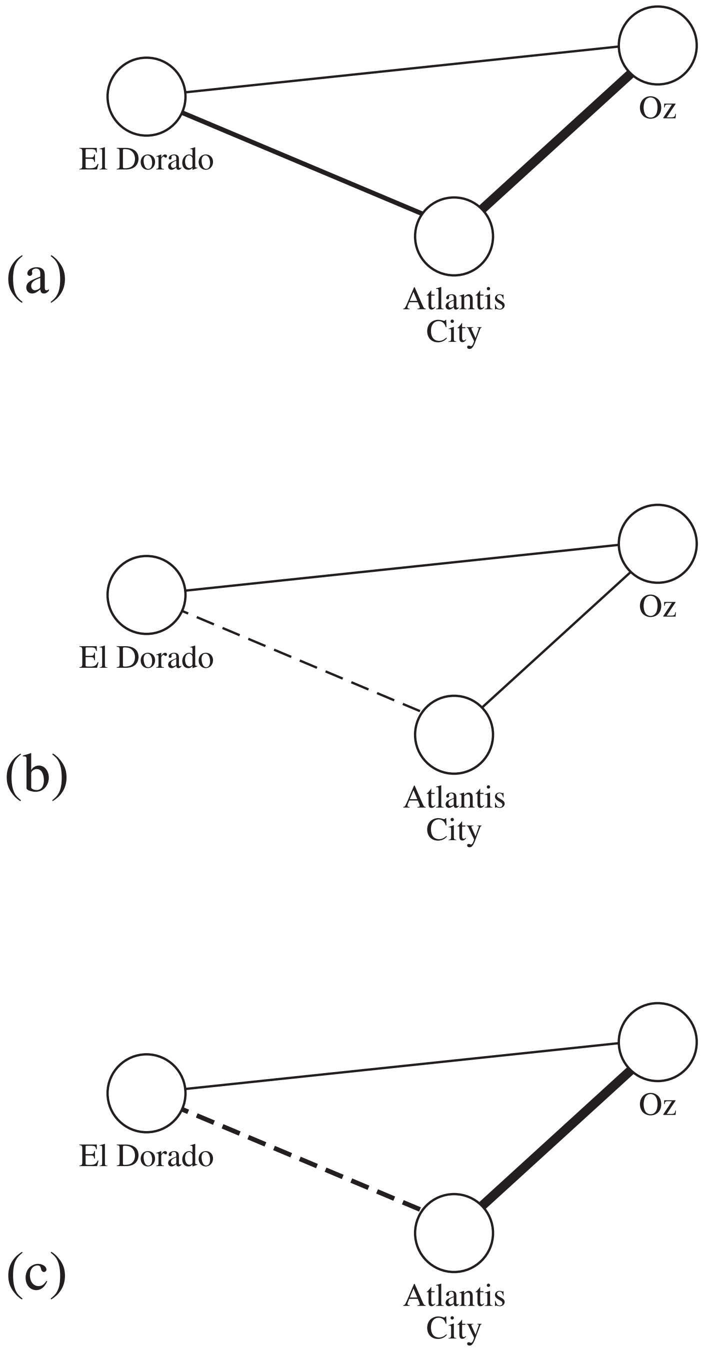


Figure 4.14: (a) Traffic between cities (b) Safety of roads (c) Composite presentation

datasets must somehow indicate that data is available for the same pairs of cities.

This same basic problem crops up in the composition of the presentations of figure 4.15.a (presenting the locations of a set of cities using the positions of a set of circles) and 4.15.b (presenting the dominant industries of a set of cities using the grayscale values of a set of circles). These presentations cannot be composed because no data is available for Oceanview in the presentation of 4.15.b. Note however that the presentation of figure 4.15.a can be composed with that of figure 4.15.c even with the same missing data, yielding the presentation in figure 4.16.

In addition to the normal concerns of intuitive and strict expressiveness, there are two interrelated issues affecting the composability of graphical languages. The first is the composition of the encoding conventions of the atomic objects used by the languages. If in one graphical language a certain set of properties of an object are used to represent a set of domains, and in another graphical language another set of properties of the same type of object is used to represent another set of domains, the languages may not be composable, because no set of encoding conventions can be found merging the two encoding conventions for that object type. That is, there may be no way to construct a consistent mapping of properties to domains for the merged objects. The other issue concerns composing *instances* of perceptual objects. If two graphical languages employ a common type of component object to encode information, instances of this type of object must play multiple roles in the composite presentation, perhaps forming part of multiple classes of compound objects. Determining which instances from each original presentation get 'merged' in the composite presentation may be complicated.

The problem of determining whether composite encoding conventions will exist is relatively simple. Consider the composition of two graphical languages. In one, the properties of some perceptual object class (e.g., *circle*) are used to represent some set of domains. In another graphical language, some other set of properties of the same object class, possibly overlapping with the first set of properties, is used to represent a possibly overlapping set of domains. The restriction on composing encoding conventions is that in the composite character class, no property can be used to represent more than one domain. For example, if the horizontal position of a circle represents *year* in graphical language 1 and *city* in graphical language 2, the languages will not be composable.[19] It is possible, however, to merge two encoding conventions where properties represent the same domains in the same ways, as long as the particular instances that are merged encode the same values for these domains. For example, if the population of a city is represented by the size of a labeled circle in one language and a city's per capita income is represented by the grayscale value of a labeled circle in another language, these encoding conventions can be composed without difficulty if the label encodes the city in either case.

The problem of composing instances (i.e., specific individual objects) is a bit more complicated. If there is a logical one-to-one correspondence between instances of some type of object in each of two graphical languages, composition is conceptually simpler and composability is easy to guarantee. I will term

[19] Using a property to represent two domains simultaneously is not completely inconceivable if the values for the two domains can be paired exactly. This is rarely likely to occur, however, and is furthermore something that has been outlawed for individual graphical languages with the systematicity assumptions given in the previous chapter. As a result, it will similarly be forbidden for composite languages.

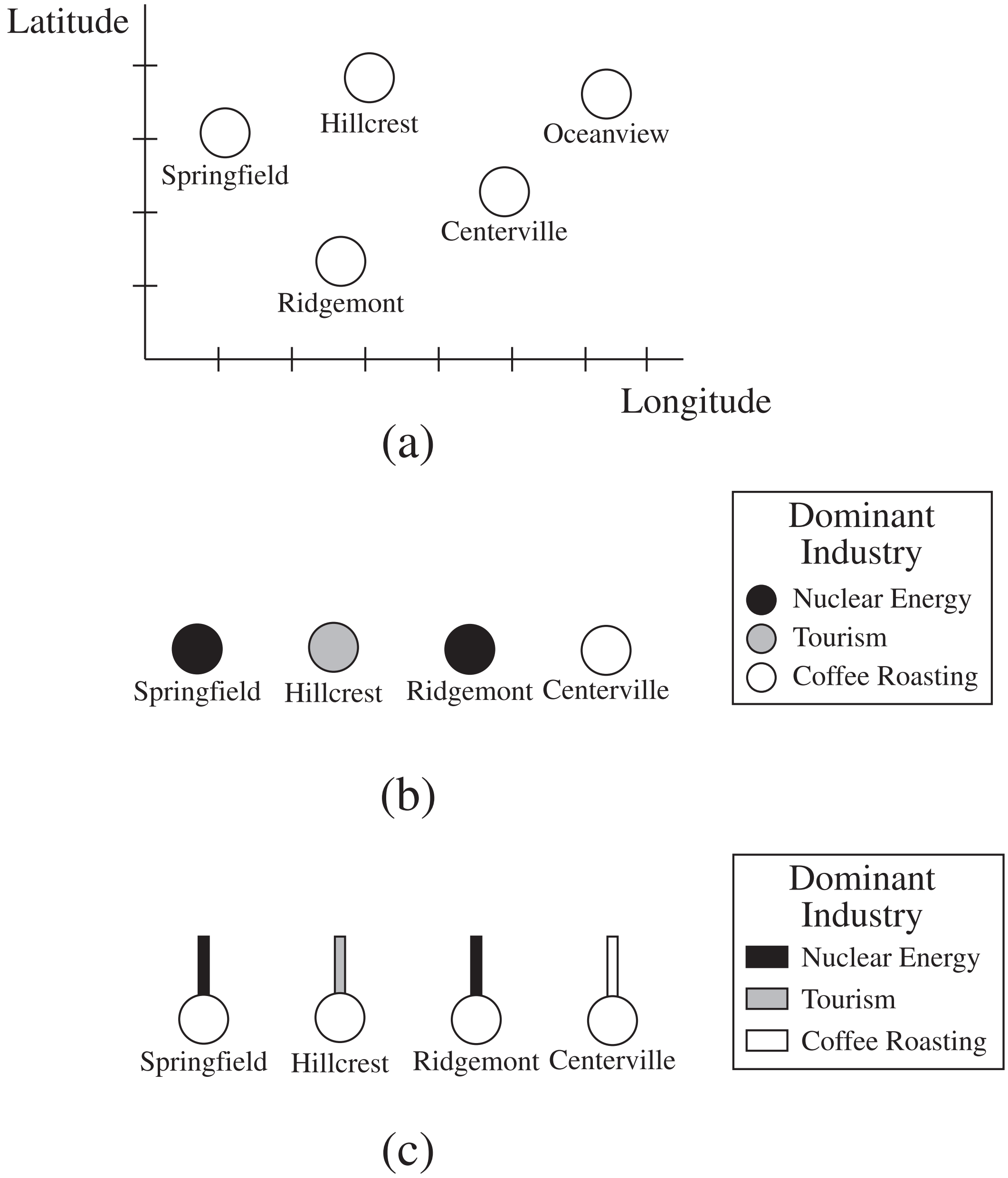


Figure 4.15: (a) Locations of cities (b) Dominant industries of cities represented by value (c) Alternate version of (b)

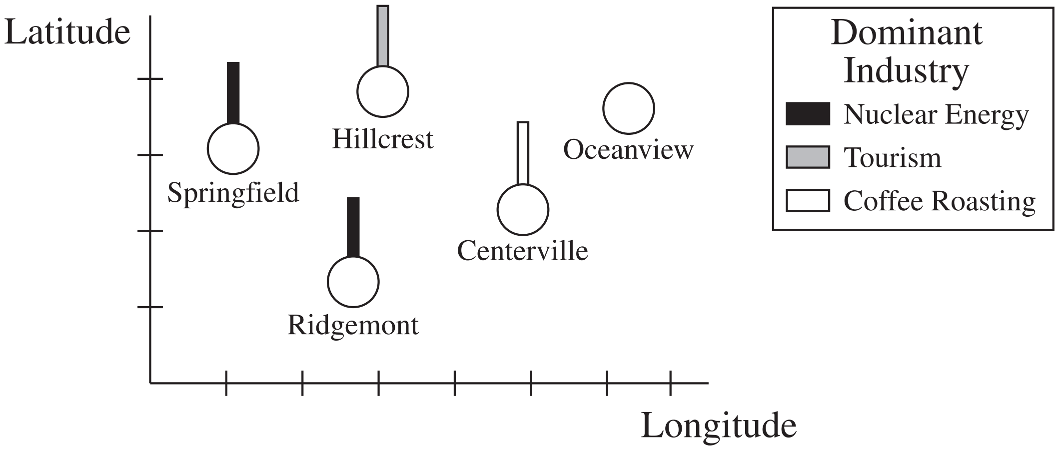


Figure 4.16: A composed presentation

this *exact* composition. Composition is also possible without this condition being satisfied; I will term this *inexact* composition. I will discuss each of these types of composition in turn.

Exact composition

For exact composition to be possible, there must be a guaranteed one-to-one mapping between instances of some object type for different presentations. Two aspects of this mapping are crucial for merging objects to create a composite presentation:

- The mapping must be logical—i.e., there must be some rational basis for pairing object instances.
- The mapping must be complete, so that each object in one presentation can be merged with some object in the other.

Both of these are true in the case of figure 4.10, for example—the objects in each presentation correspond to single cities, so there is a way of determining which objects to merge, and there is a circle for every city in each presentation, so each object can be paired with one in the other presentation.

This example is representative of a large class of situations in which exact composition is possible—situations where a *logical basis of generation* (or *generation basis*) can be established independently for the objects in each presentation to be merged, based on the individual graphical languages and the data characterizations of the datasets being presented, and where the data characterization guarantees that this generation basis will be completely realized. In these situations, it is possible to determine the precise number of instances of each object type that will be created, and a basis for pairing these objects with objects in another presentation with which they can be merged.

For shared objects there will by definition always be a one-to-one correspondence between the subtuples represented by some object type that occur in the dataset and the instances representing the subtuples, so calculating the number of instances of the object type that will occur in a presentation can be accomplished by determining the number of subtuples. Using the data characterization language described in the solution to the intuitive expressiveness checking problem presented in section 4.1.1, two factors enable the number of subtuples to be determined conveniently in some circumstances: *completeness* and *functional dependency*. If a set of subtuples is composed of values from a single domain and the domain is represented completely in the dataset (i.e., there is at least one tuple for each value of the domain) there will be one subtuple for each value in the domain. If two domains are represented by a subtuple, the number of subtuples of this type can be determined easily if either domain is complete and the other is functionally dependent on it, or the two domains are *jointly complete* (i.e., if the data characterization specifies that there is a tuple in the dataset for each possible *pair* of domain values). In the first case, there will be one subtuple for each value of the complete domain. In the second case, there will be one subtuple for each possible pair of values.

Extending this logic to three or more domains, a set of subtuples will be of a size determinable from the data characterization if and only if some set of domains x of the subtuple are characterized as jointly complete and the other domains in the subtuple y are directly or indirectly functionally dependent on domains in x. That is, the domains in y may be functionally dependent on domains in x, functionally dependent on other domains in y that are dependent on domains in x, functionally dependent on domains in y that are dependent on other domains in y that are dependent on domains in x, etc. Completeness guarantees that there will be at least one subtuple for each potential set of values for x. Functional dependency guarantees that there will be no more than one set of values for y for each set of values for x. In such cases, the generation basis of these objects will be considered to be the set x, since the product of the sizes of the domains in x determines the number of objects that will be created. To distinguish between the concepts of generation basis and completeness, the generation basis of a set of shared objects will more generally be considered to be the set of domains of the subtuple not dependent on other domains of the subtuple, whether or not these domains are characterized as complete. This distinction will be useful when I consider inexact composition shortly.[20]

As a simple illustration of the concept of generation basis, consider representing a dataset of profits by year and quarter (with tuples of the form $< \mathit{Year}, \mathit{Quarter}, \mathit{Profits} >$) with a set of shared circles, where the horizontal position of a circle represents *year*, the vertical position *quarter* and the size of the circle *profits*. The generation basis of these circles is $\{$*year, quarter*$\}$, because presumably profits are functionally dependent on year and quarter. If the dataset is characterized as jointly complete for *year* and *quarter* (i.e., if there is a profit value recorded for each quarter in each year), and spans a period of 10 years, there will be exactly forty circles in the resulting graph.

The above example relies on the fact that the circles are shared. For non-shared objects, the correspondence between subtuples and object instances is not normally one-to-one. Generally, new non-shared

[20] It should be clear that completeness and functional dependency are only one way of characterizing datasets. Alternative methods of characterizing are also possible, if they make clear the same facts about the number of subtuples needed. For example, Roth & Mattis (1990) propose characterizing data in terms of *cardinality*, *coverage* and *uniqueness*. Cardinality allows data to be characterized as having n values of some domain for every value of some other domain (e.g., every *child* has two *parents*). While the analysis of composition presented here is therefore somewhat dependent on the data characterization capabilities described earlier, the outlines of the analysis should generalize to alternate methods of characterization.

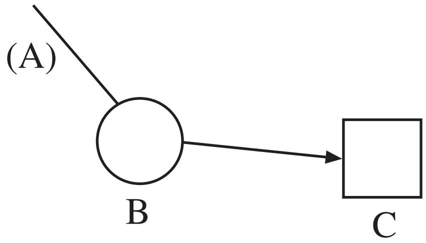


Figure 4.17: A compound object structure

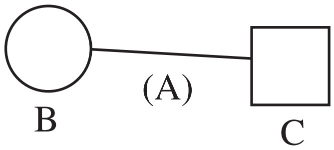


Figure 4.18: A compound object structure

objects of a given type may be created for every tuple in the dataset, even if there are a much smaller number of subtuples represented by that object type. For example, in a graph representing the amount of traffic between cities using the thicknesses of lines between circles, as in figure 4.14.a, there may be only three possible domain values for *traffic* and three corresponding thicknesses, but many lines of each thickness. In some cases, however, it is still possible to calculate a generation basis for non-shared objects in a manner analogous to that in which the generation basis of shared objects can be determined. Rather than presenting a complete analysis of the generation bases of non-shared objects, I will restrict my analysis to these cases, which seem to make up the majority of expressive and easily-interpreted presentations.

The description of *non-shared* in the previous chapter specified that new non-shared objects will be created for every tuple, except when this will introduce pointless redundancies. In order to understand the generation bases of non-shared objects, it is useful to consider when these redundancies will occur. Consider for example figure 4.17, in which an arity-three tuple $< A, B, C >$ is represented by a line (non-shared), a circle (shared), and a square (shared), with the circle and square connected by a (non-shared and non-semantic) arrow.[21] If A is functionally dependent on B, and B is shared, there will never be a need to create more than one line per circle, even if the circle is connected by arrows to multiple squares. If multiple lines were created they would encode precisely the same value and have precisely the same relationships with other objects—i.e., they would be redundant. Similarly, in figure 4.18, if A is functionally dependent on B and C, which are represented by shared objects, there will never be a need to create more than one line for every pair

[21] In this figure and in figure 4.18, I will use the convention of labeling non-shared objects with the names of domains in parentheses, and shared objects with the names of domains not in parentheses. These domains are presumed to be represented by any properties of the objects, including labels, colors, sizes, etc.

of B and C values.

In these examples, the number of instances of the non-shared objects in the cluster will be determinable from the data characterization if the generation basis domains of the shared objects of the cluster are complete. If B is complete in figure 4.17, or if B and C are jointly complete in figure 4.18—there will be a number of instances equal to the product of the sizes of the independent domains of the shared objects of the cluster. In figure 4.17, there will be $|B|$ lines. In figure 4.18, there will be $|B||C|$ lines. As with the calculation of the generation basis of shared objects described above, completeness guarantees that there will be at least one set of non-shared objects for every set of values of the generation basis domains, and functional dependency guarantees that there will be no more than one such set of non-shared objects

To summarize, it is possible to establish the generation basis of the non-shared objects of a cluster when:

- The generation bases of the shared objects of the cluster are all characterized as jointly complete.
- The non-shared objects of a cluster, if they are used to encode information, encode domain values dependent only on domains encoded by the shared objects of the cluster.

In these cases, the generation basis for the non-shared object type will be the union of the generation bases of the shared objects of the cluster. As with the generation basis of shared objects, it is possible to use generation basis plus completeness to determine if two sets of objects can be merged.

As an example of calculating the generation basis for a non-shared object, consider again presenting a dataset of profits by year and quarter with tuples of the form $< \mathit{Year}, \mathit{Quarter}, \mathit{Profits} >$, only this time with a graphical language using compound objects. Each year could be presented using an appropriately labeled shared circle, each quarter with an appropriately labeled shared rectangle, and profits with the thickness of a (non-shared) line between the relevant circle and rectangle. If *year* and *quarter* are again assumed to be characterized as jointly complete, and *profits* is functionally dependent on both domains, the generation basis of the set of lines will be the set $\{\mathit{year, quarter}\}$. That is, there will be a number of lines equal to the number of years times the number of quarters (four, presumably), and each line will correspond to a particular year and quarter.

The calculation of generation basis just described provides one convenient way of determining when two graphical languages will be composable with exact composition. Other ways are also possible. If data characterizations specifying relationships between different datasets could be specified, e.g.,

$$\forall A_1, A_2\, [R_2(A_1, A_2, B_1) \longrightarrow \exists C_1\, [R_1(A_1, A_2, C_1)]]$$

exact composition would be possible in other circumstances. For example, this might allow exact composition of presentations like the one in figure 4.14 even if there aren't roads between every pair of cities. Future work may address this possibility; I will not discuss it here.

Note that the example of figure 4.14 relies on exact composition, as does the example of figure 4.11.

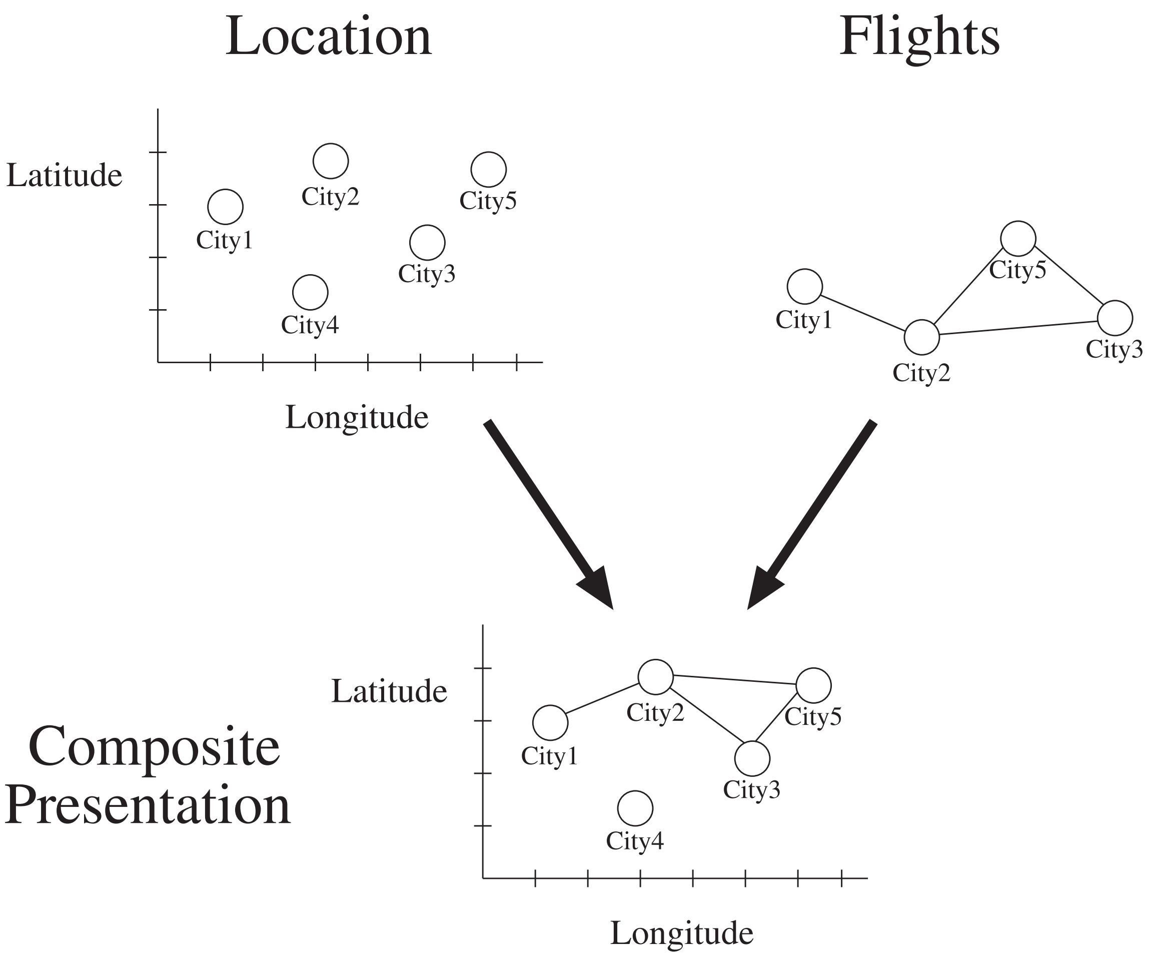


Figure 4.19: Inexact composition: Leaving non-merged instances as is

Inexact composition

The conditions just explored for exact composition are intended to satisfy two requirements about the correspondence between object instances used by two different graphical languages: that there is a logical basis for establishing a one-to-one correspondence between instances of objects created for the different languages, and that the correspondence is complete, in the sense that for every object instance in one presentation there is guaranteed to be an object instance in the other presentation. For example, in figure 4.14, there is a natural correspondence between the lines to be composed, determined by the circles they connect. Furthermore, lines to be composed exist for the same pairs of circles for both presentations. If either of these requirements is violated, exact composition will not be possible, but in certain circumstances other types of composition will be possible. There are at least three methods of inexact composition, each appropriate under particular conditions, which are worthy of brief discussion:

1. *Merge the available instances, leaving non-merged instances as is.* There are cases when there is an obvious one-to-one correspondence between object instances from different presentations, but the correspondence is incomplete. In some situations, notably when non-merged objects will have a completely specified set of property values, this may not cause problems. This will specifically occur when the subtuples encoded by some type of object from each graphical language are identical (i.e., composed of the same types of domains) or when one set of subtuples A corresponds to a subset of the other set B, and the domains of A are a subset of the domains of B. For example, consider a set of tuples of the latitudes and longitudes of a set of cities (e.g., *Location(City*$_1$*, 40, 50)*) that is represented by the positions of a set of circles, and is complete for some set of cities. If another dataset of flights between pairs of cities (e.g., *flight(City*$_1$*, City*$_2$*)*), which isn't complete in any way, is presented by lines between labeled circles, the presentations can be composed without difficulty, as shown in figure 4.19. Fewer circles may need to be created for the second presentation, but no additional properties are contributed by this presentation when merging. Note that figure 4.16 is created from the individual presentations of figure 4.15.a and 4.15.c using this method of composition.

2. *Merge available instances, introducing distinguished property values encoding "null" when necessary to give non-merged objects complete sets of property values.* In conditions similar to where unmerged object instances can be left "as is" (i.e., where there is an incomplete one-to-one correspondence) but where some unmerged instances may not have complete sets of property values needed to render them, it may be possible to introduce property values indicating that there is no value for the domain normally encoded by that property. This would allow exact composition to take place. For example, if in one presentation, safety levels of some activities were encoded using the colors (e.g., green, yellow and red) of a set of labelled circles, and in another presentation some other data about the activities were encoded using the positions of labelled circles, it would be possible to merge the presentations even if the safety data were incomplete by introducing a new color (e.g., white) to indicate that no safety data were available.

   Null values seem appropriate in some cases, but are clearly easier to find for some perceptual dimensions than for others. This seems to relate in part to the "markedness" of particular values in a perceptual dimension. White (or black) seems quite different from saturated red, yellow and green colors, for example, but no particular horizontal position is likely to be as distinct.[22]

3. *Duplicate object instances from one presentation before merging.* There are some cases where there is a clear correspondence between object instances from different graphical languages but this correspondence is many-to-one. For example, imagine merging a presentation of city populations (where the size of a circle encodes the population of the city indicated by the circle's horizontal position) with a presentation of supermarket chains operating in a city (where the horizontal position of a circle again encodes the city, and the vertical position of the circle encodes the supermarket chain). Assuming that

[22] Of course, in a dot plot of discrete values, a horizontal position encoding "no data available" is certainly conceivable. However, it has the potential to be confusing. Interpretability issues are discussed in chapter 5.

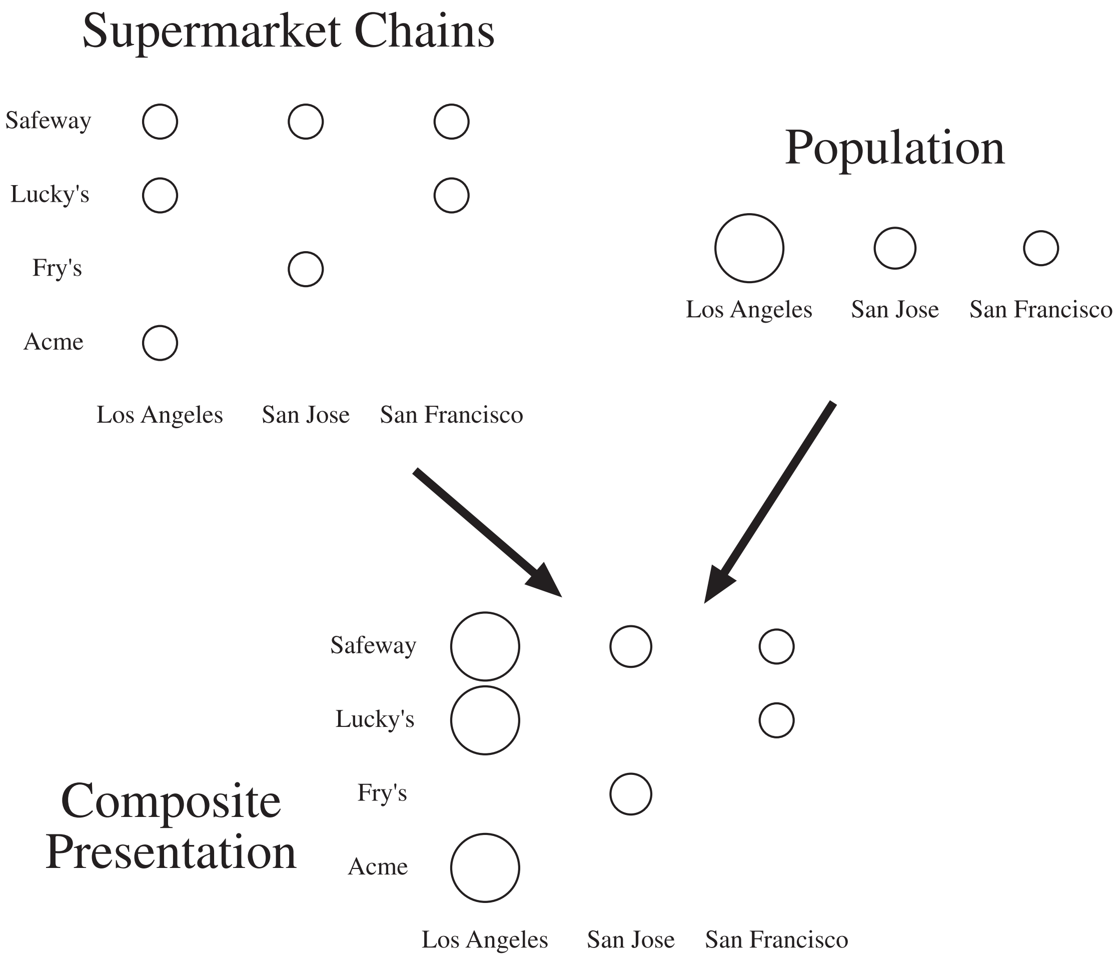


Figure 4.20: Inexact composition: duplicating object instances before merging

both presentations are complete for the domain *city*, these two presentations can be composed as shown in figure 4.20.

In cases like this, merging multiple objects with single objects becomes possible, making for the redundant encoding of information. In general, this is likely to be confusing, for reasons to be discussed in chapter 5. In some cases it seems more reasonable than in others, however. In the example of figure 4.20, it seems fairly reasonable. If supermarket chain presentation were being merged with a network graph showing flights between cities, it would seem less so, because many extra lines would need to be introduced.

Other methods of inexact composition are also conceivable. For example, when merging objects with incomplete one-to-one mappings, in situations similar to method 2, it might sometimes be possible to introduce arbitrary values instead of "null " values without creating any complete compound objects encoding information not in the dataset being presented. However, composite presentations created in this manner seem unlikely to be easily interpreted.

The current AUTOGRAPH implementation permits the composition of graphical languages where non-merged object instances may be left as is (method 1 described above). It does not make use of the other techniques, though they may be added at some future date.

4.3.2 Intuitive and strict expressiveness checking with composite presentations

Even if two composable graphical languages are individually logically expressive, the composite language may fail to be so. In this section I briefly discuss the logical expressiveness of composite languages and how to extend the algorithms of section 4.1.1 and 4.2.1 to determine whether composite languages meet the intuitive and strict expressiveness conditions.

A composite presentation represents two or more types of relations (i.e., tuples) simultaneously. Individual objects in the presentation may thus be used for representing tuples from any of the datasets being presented, or from multiple types of datasets simultaneously (if they are merged). The intuitive expressiveness condition may thus be violated in a composite language because using perceptual objects for presenting two or more datasets simultaneously places more constraints on the underlying parameters of the corresponding graphic objects. If this results in overconstraining those parameters, the composite presentation will be impossible to create. For example, if a graphical language constraining the tops and bottoms of a set of rectangles is composed with a language constraining the heights of a set of rectangles, the parameters of the rectangles will be overconstrained, barring a particular mathematical relationship between the two datasets.

The strict expressiveness condition may be violated in a composite language because clusters from the compound object presenting one type of tuple may be created by encoding some other type of tuple. Thus, tuples may be presented implicitly in new ways. For example, a graphical language representing flights between pairs of cities on American Airlines using lines between labeled circles may be strictly expressive. Similarly, a graphical language representing flights between cities on United Airlines in the same manner

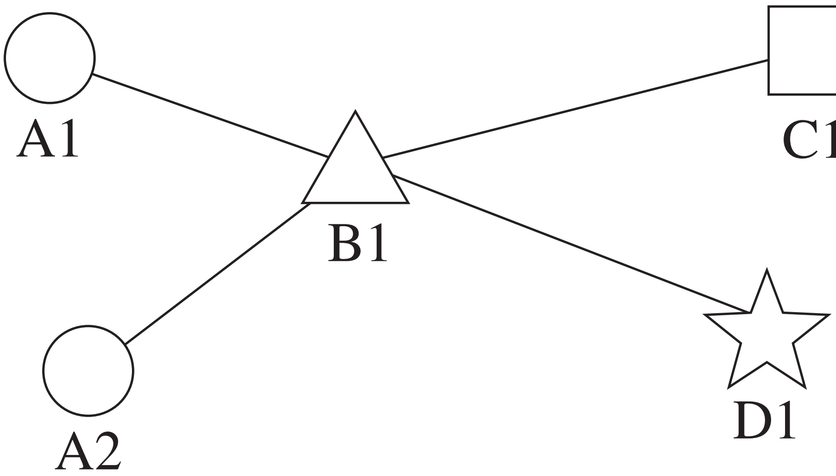


Figure 4.21: A violation of the strict expressiveness condition

may also be strictly expressive. But if these two languages are composed— using inexact composition, since there aren't lines between every pair of circles—the composite language will violate the strict expressiveness condition, for obvious reasons. Encoding flights between cities on American Airlines implicitly presents flights on United Airlines, and vice-versa.[23]

Violation of the strict expressiveness condition in composite presentations can also occur even when the graphical languages being composed do not use identical encoding conventions. For example, consider again the circle/triangle/square language of figure 4.5, presenting a set of tuples of the form $< A, B, C >$. If this language were composed with a graphical language presenting tuples of the form $< A, B, D >$ using similar encoding conventions (but representing domain D with some other labeled shape, such as a star), encoding $< a_1, b_1, c_1 >$ and $< a_2, b_1, d_1 >$ would implicitly present both $< a_2, b_1, c_1 >$ and $< a_1, b_1, d_1 >$ (see figure 4.21), resulting in a violation of the strict expressiveness condition. If C and D were functionally dependent on B in the individual datasets, each of the individual languages being composed would themselves be expressive. However, the expressiveness problem could still arise in the composite language.

Extending the intuitive expressiveness checking algorithm to cover composite graphical languages is straightforward. The original algorithm presented in section 4.1.1 determines all of the constraints placed on the parameters of the underlying graphic objects in a presentation due to properties, perceptual relations, and perceptual functions, then determines if any of these objects may be overconstrained. To extend this to composite languages, it is simply necessary to consider constraints due to the considerations of all the graphical languages being composed.[24]

[23] Note that one reasonably obvious way of composing these languages would be to use different colors to represent different airlines. Effectively, this can be viewed as creating subclasses of the perceptual object class *line*. The current version of AUTOGRAPH is not capable of treating lines of different colors as different classes of objects, or of modifying languages in order to compose them, but there is no reason that a future version could not do so. This is fairly similar to changing shapes during double-axis composition as is done in Mackinlay's (1986b) APT system, as discussed near the beginning of section 4.3.1.

[24] This presumes that the individual languages meet the first condition of intuitive expressiveness—i.e., that composite objects with

The algorithm for strict expressiveness checking presented in section 4.2.1 requires a bit more modification to analyze composite languages. The central idea remains the same: the algorithm tries to find a legal set of tuples that when presented will implicitly present some tuple not in the set. However, the modified algorithm must check for the potential implicit encoding of any of the types of tuples being presented, and must consider all the different ways the clusters presenting a tuple can get created. As the example of figure 4.21 illustrates, multiple types of tuples may be encoded using some of the same types of clusters. The algorithm must also take composition into account; if objects of the presentation are merged and different presentations use different properties of these objects, clusters may only be created when tuples of multiple types are presented. For example, in figure 4.14, a cluster presenting a tuple of the form $< \mathit{El\ Dorado}, \mathit{Oz}, \_ >$ in the traffic presentation will always be accompanied by a tuple of the form $< \mathit{El\ Dorado}, \mathit{Oz}, \_ >$ in the safety presentation, if exact composition is to be possible. This will affect legality checking.

The algorithm can be modified in a few straightforward ways to accommodate these complications. I will assume in this description that two languages are being composed, but the modified algorithm can be extended in a straightforward way to handle more than two languages.

1. *Pick an arbitrary tuples for each dataset*

   The strict expressiveness algorithm must now check for the implicit presentation of tuples from any of the datasets being presented. That is, the algorithm must be run for each of the types of datasets.

2. *Divide the corresponding compound objects into clusters*

   This is done exactly as it was with a single dataset.

3. *Find sets of tuple fragments for each cluster*

   For a simple graphical language (i.e., a graphical language presenting a single dataset), there will be at least one tuple fragment (and possibly more than one) corresponding to each cluster. For a composite graphical language, there may be—in addition to the tuple fragments from the original dataset—one or more tuple fragments from other datasets corresponding to the cluster. These describe tuples from other datasets that when presented normally (i.e., in a simple presentation) will create a cluster of the same types of objects as the cluster the algorithm is trying to present implicitly. For example, for the languages described earlier in relation to figure 4.21, a cluster consisting of a circle connected by a line to a triangle may be created by presenting a tuple from either the $< A, B, C >$ dataset or the $< A, B, D >$ dataset.

   As noted earlier, in cases where objects of the cluster are used in presenting multiple types of tuples, and must be merged to produce a full set of property values, the creation of these objects will only occur when tuples of multiple types are encoded. For each tuple fragment, in these cases, whether that tuple fragment comes from dataset 1 or from dataset 2, there will be a corresponding set of possible tuples from the other dataset that must be presented simultaneously in order to *co-create* these objects.

too few distinct components will never be created.

This set of tuples can again be described by a set of tuple fragments. Note that these tuple fragments only need co-create the objects of the cluster, not the perceptual relations.

To illustrate the idea of co-creation, consider how the algorithm would deal with the example of figure 4.14 (traffic between cities composed with road conditions between cities). If the chosen arbitrary tuple were *Traffic(Atlantis City, El Dorado, Low)* the corresponding cluster would be a circle labeled *Atlantis City* and a circle label *El Dorado* and a thin line touching both the circles. The tuple fragments from the same dataset that would create this cluster—entire tuples in this case —would be *Traffic(Atlantis City, El Dorado, Low)* and *Traffic(El Dorado, Atlantis City, Low*). These tuples by themselves would create the cluster, but must necessarily be presented along with some tuple of the form *Conditions(Atlantis City, El Dorado, _)* or *Conditions(El Dorado, Atlantis City, _)* from dataset 2 specifying the road conditions between the cities and thus the dash pattern of the line between two circles. These are co-creating tuple fragments, and must be considered when determining the tuple fragments that will create a cluster.

Another way this cluster could be created would be to start with the tuple fragments from dataset 2 that create the objects of the cluster (*Conditions(Atlantis City, El Dorado, _)* and *Conditions(El Dorado, Atlantis City, _)*) and look to dataset 1 for the co-creating tuple fragments *Traffic(Atlantis City, El Dorado, Low)* and *Traffic(El Dorado, Atlantis City, Low)*. Note that although in this case there are single co-creating tuple fragments corresponding to the tuple fragments creating the cluster (i.e., there are several possible co-creating tuple fragments, but only one of these is needed to co-create the cluster), in some cases there may be several co-creating tuple fragments, each establishing the properties of one object of the cluster. Note also that while in this example, the tuple fragments of dataset 2 establishing the objects of the cluster are identical to the co-creating tuple fragments needed by dataset 1 and vice-versa, this need not always be the case, because the co-creating tuple fragments only need to create the objects of the types used by the cluster, not the perceptual relations of the cluster.

The general structure of the tuple fragments that will be found in this stage of the algorithm is shown by the tree in figure 4.22, annotated to show how the structure relates to the example just cited.

4. *Find a problematic legal tuple set*

   As with the checking algorithm for a single dataset, the final step of the algorithm involves looking for possible instantiations of one set of tuple fragments that is legal and doesn't imply the original tuple t_0. Only the structure of the sets of tuple fragments that must be checked is different.

4.4 Summary

This chapter explored criteria for evaluating the logical adequacy of graphical languages for different types of datasets. It analyzed these criteria generally, and also gave specific algorithms for determining

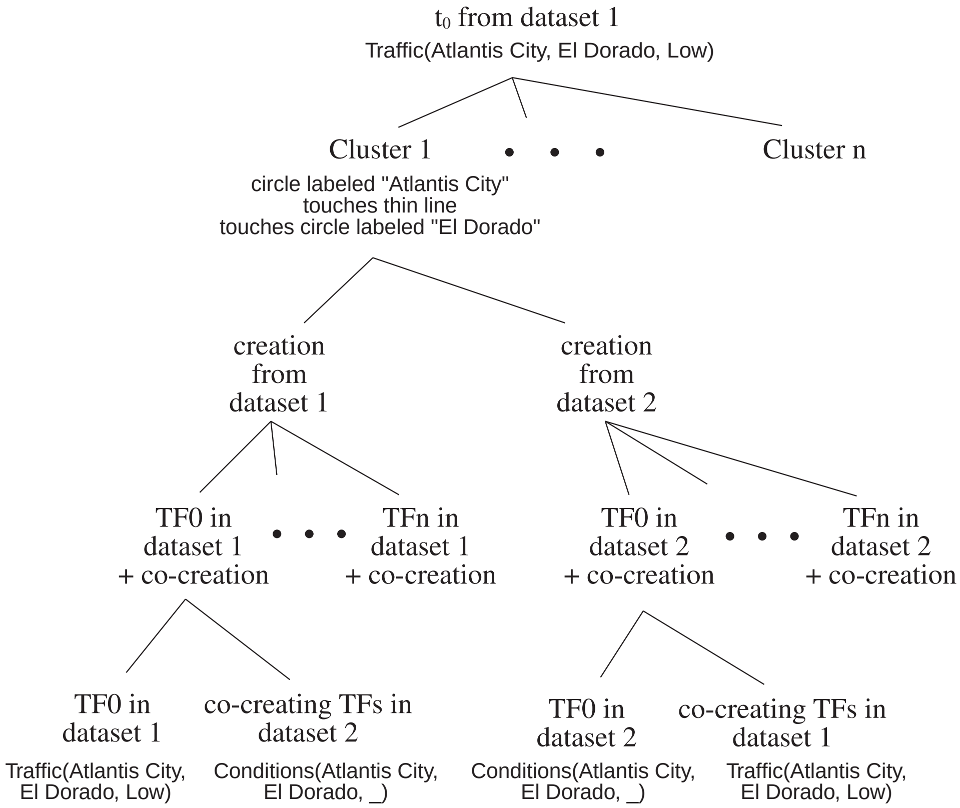


Figure 4.22: The structure of a tuple fragment set

logical expressiveness under particular assumptions about perceptual-level descriptions and dataset characterizations. It also addressed the issues involved with presenting multiple datasets simultaneously through the use of composite graphical languages.

The first conditions for logical expressiveness explored was the *intuitive* conditions, which requires that any dataset of a given type must be expressible in a given language for the language to be considered expressive for that type of dataset. The algorithm given, for specific assumptions about perceptual-level descriptions and data characterization, consisted mainly of a degree-of-freedom analysis. If all of the components needed to create a compound object encoding a tuple can be created and the parameters of all of the underlying graphic objects cannot be overconstrained, a language will meet the intuitive expressiveness condition.

The second condition for logical expressiveness, the *strict* condition, requires that any dataset of a given type must be expressible in a language without implicitly expressing any tuples not in the dataset, if the language is to be considered expressive for that type of data. The algorithm given, again under specific assumptions, can be thought of as a search for a legal dataset which when presented will implicitly present an additional tuple. If such a dataset does not exist, the strict condition will be met.

Having described conditions under which graphical languages for single datasets will be expressive, the chapter then discussed how graphical languages for presenting multiple datasets can be created by composing the languages for the individual datasets. One method of composition, *exact composition* was described at length. Another class of methods of composition, referred to as *inexact* composition, was also described. Finally, the chapter discussed how the analyses and algorithms for checking the two expressiveness conditions could be extended to composite languages.

Chapter 5

Interpretive Principles

Having analyzed logical principles relating to graphic presentation in the last chapter, I now turn to psychological principles—i.e., principles of interpretation and perception. In this chapter, I will discuss issues relating to the interpretation of graphic presentations; perceptual issues are discussed in chapter 6.

5.1 Introduction

A number of principles related to interpretation have already been incorporated into the definition of a graphical language given in chapter 3. These are the systematicity assumptions, also discussed in that chapter. In this chapter I will discuss interpretive principles taking the already systematic notion of a graphical language as a starting point.

Some models of graphic comprehension (e.g., Pinker (1990), Lohse (1993)) have taken the view that the process of interpreting graphic presentations can best be thought of as a process of determining an appropriate graph type or schema from a set of known ones and using the schema to interpret relevant perceptions. Some automatic presentation design systems (e.g., APT (Mackinlay 1986b) and SAGE (Roth & Mattis 1990; Roth & Mattis 1991; Roth *et al.* 1991)), while not explicitly based on this view, nevertheless take as a starting point graphical languages which are known to be interpretable, without analyzing what makes them interpretable.

Clearly, though, there are general principles at work in graphic interpretation, and also conventions more general than the level of complete graph types. This notion of general principles is present implicitly in most of the graphic presentation design systems that use predetermined graphical languages, since they combine these languages to form new ones, assuming the new ones will still be interpretable. Some research by diSessa *et al.* (1991), described in chapter 2, has supported the claim that even children have meta-representational expertise—i.e., knowledge of general principles underlying the interpretability of graphic presentations. Finally, there is compelling evidence for general principles in the fact that new variations of old types of presentations and even completely novel presentations are introduced all the time, and some are easier to adjust to than others. Thus, even if it is the case that the only presentations that can be understood *instantly* belong to a closed set of types, there must be principles that explain which new types of graphs will be usable given a little time. In this chapter, I will analyze such principles, the general factors that make presentations easy or difficult to interpret.

There are two fundamental difficulties that must be addressed in a discussion of these factors. The first is that it is difficult to say precisely what is meant by a presentation being difficult to interpret. Although one is likely to have an intuitive notion of what this means, it is not obvious how to quantify it. Ideally, a graphic presentation should be readily interpretable by all its intended viewers; they should be able to identify the encoding conventions of the presentation and extract information quickly and without making errors (i.e., misinterpreting the encoding conventions). But there are several ways in which presentations can fall short of this ideal:

- If a method of graphic representation is novel, it is likely that most viewers will need a lot of time to understand the encoding conventions before being able to interpret the presentation at all. That is, the *identification* phase, introduced in chapter 3, may be time-consuming.
- If a presentation method is superficially similar to conventional methods, but different in some subtle but important way (e.g., a bar graph where the area of rectangles, in addition to their heights, encodes information) there is a possibility that some viewers will misinterpret it, using incorrectly determined encoding conventions.
- Even once their encoding conventions are well understood, some presentations may be difficult to interpret quickly—even if it is not difficult to make the relevant perceptual distinctions. For example, in a map such as the one in figure 5.1, employing lots of symbols with no intuitive correspondences to the values they encode, a viewer will be unlikely to remember the meanings of all the symbols, and thus need to consult the legend repeatedly. Thus, the *extraction* phase can also be affected by interpretive difficulties.

A second difficulty in talking about the interpretive effectiveness of presentations is the lack of direct empirical evidence bearing on the issues.[1] Although there have been some experiments measuring perceptual effectiveness, few if any have directly measured *interpretive* effectiveness for the sort of presentations with which this dissertation is concerned. Much of the anecdotal knowledge available, too, in the form of suggestions from statisticians, psychologists and graphic designers, concerns specific choices about specific types of presentations. These are not always generalizable enough to be useful.

Given these difficulties, the goals of this chapter will be accordingly modest. I will not attempt to provide enough knowledge to make quantitative predictions—e.g., about the probability of interpreting a presentation accurately, or about the time that will be required to understand the encoding conventions of

[1] I will refer throughout this chapter to interpretive *effectiveness*, but this really subsumes both effectiveness and expressiveness considerations. With respect to interpretation, the effectiveness/expressiveness distinction is really a performance/competence distinction, as made by Chomsky (1965). I will choose to regard a presentation that is extremely difficult to "decode" as expressive (i.e., well-formed) but ineffective (i.e., difficult to parse).

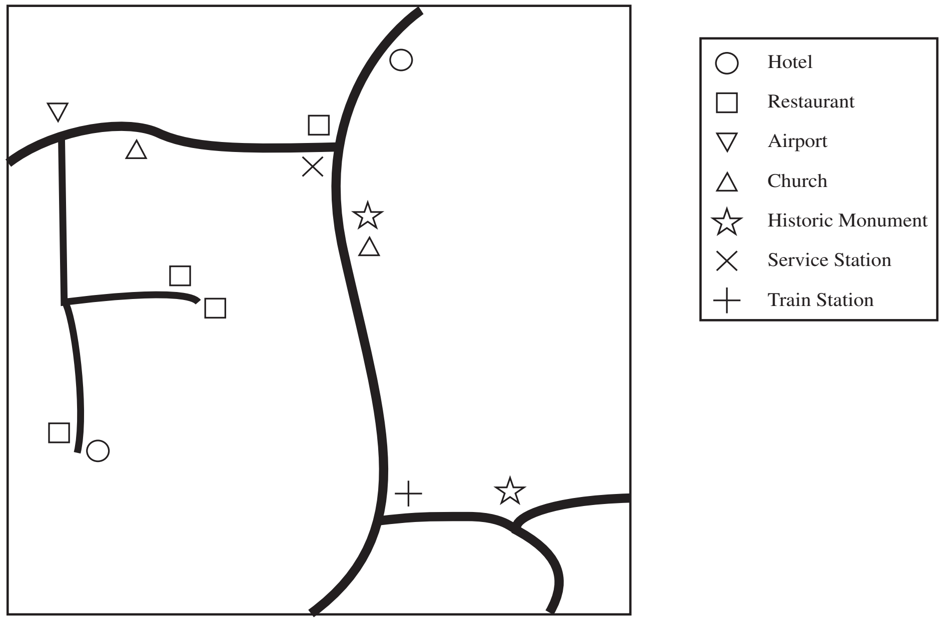


Figure 5.1: A confusing map

a particular method of representation. Instead, I will merely provide design guidelines for creating presentations that are reasonably likely to be interpreted quickly and accurately, and for making decisions about whether presentations will be more or less difficult to interpret accurately. That is, this chapter will provide a basis for heuristics for pruning and for ordering branches in the search tree of possible graphical languages.

Given the lack of systematic empirical evidence for different interpretive principles, in arguing for these principles I will rely on both appeals to intuition and on consensus of opinion, where it exists, among various writers on graphic presentation.[2] A third form of informal evidence for the principles will come in the form of the implementation, AUTOGRAPH, discussed in chapter 7. To the extent that an implementation based in part on the principles given here is successful in generating presentations that are easy to interpret, the principles can be judged reasonable. It is of course also possible that future research could empirically evaluate these principles.

In the next section of this paper, I analyze the sources of the tendencies and difficulties viewers have in interpreting graphic presentations. In section 5.3, I distill from these tendencies and difficulties a small set of design principles for graphic presentations, and discuss applications of these principles to various aspects of the design of such presentations.

[2]Most valuable for these purposes are Tufte (1983, 1990, 1997) and Kosslyn (1989, 1994).

5.2 Sources of interpretive tendencies and difficulties

It is often difficult to tease apart the sources of tendencies and difficulties in interpreting graphic presentations, because their effects are often similar. For example, graphic presentations that are difficult to interpret due to innate cognitive limitations are also likely to be difficult to interpret because they will be unconventional, their cognitive difficulties having discouraged their use in the past. However, the sources can be roughly divided, *a priori*, into three categories: cognitive limitations, constraints due to the nature of graphic communication, and graphic conventions. I will discuss these sources generally in this section, and use them as a basis for design principles in the next section.

5.2.1 Cognitive limitations

Some of the most important and most general factors affecting the interpretability of graphical languages are the cognitive limitations of human viewers. These can be divided into two categories: processing limitations and memory limitations.

Without an explicit model of the process of interpretation, it is difficult to predict in detail how processing limitations affect interpretability. If one makes the assumption that viewers can only identify the encoding conventions of unfamiliar types of presentations *serially*, however—i.e., when first encountering such a presentation, they examine each axis and legend in turn to comprehend how each domain is encoded—it is at least possible to make the prediction that graphical languages employing fewer encoding conventions will be easier to interpret than one employing a larger number of distinct conventions. Using the same types of mappings of perceptual features to information repeatedly—e.g., using horizontal position to encode the domains *cities*, then using vertical position to encode the domain *cities* in a similar manner—should be more effective than using multiple different mappings to encode such information.[3]

We might also predict that if the information being presented is structured in some way, graphical languages that somehow preserve this structure will be easier to interpret than graphical languages that do not, since cognitive processing might be required to rearrange the structure. For example, a graphical language using the positioned endpoints of a line to represent the starting and ending latitude and longitude of some movement would be easier to interpret if one endpoint encoded the latitude and longitude of the start and the other endpoint encoding the latitude and longitude of the end, rather than each endpoint encoding mixed coordinates from start and end.

More direct information is available concerning memory limitations. Most important for purposes of designing graphic presentations are the limitations of short-term memory. Short term or ‘working’ memory seems to be limited to holding approximately 7 items for a small number of a seconds (though what an ‘item’ consists of seems highly task-dependent) (Miller 1956).

Like processing limitations, memory limitations will generally make “simple” presentations easier to interpret than more complex ones. More specifically, one can predict, intuitively, that a graphical language

[3] Bear in mind that I am discussing the *identification* phase now, not the extraction phase. Information may be extracted in parallel.

employing a larger number of distinct mappings will place more of a burden on a viewer's memory. The presentation will thus require more *refreshing*, i.e., looking at legends and axes to refresh short-term memory, which will make extraction slower. It can also be predicted that individual mappings that place less of a demand on short-term memory, either because they are stored in long-term memory (e.g., using different recognizable icons, or a familiar color scheme) or because they are perceptually-supported (e.g., using larger objects to represent greater values) will require less time to understand and less time spent refreshing.[4]

5.2.2 Constraints due to the nature of graphical communication

Graphic presentations are created to convey information. Viewers of presentations are aware of this, which is bound to influence interpretation, just as verbal communication is influenced by a similar understanding. In his analysis of language, Grice (1975, 1978) postulated a cooperative principle for communication and four specific maxims comprising this principle: *Quality* ("try to tell the truth"), *Quantity* ("be as informative as necessary, but not more so than necessary"), *Relevance* ("make relevant contributions"), and *Manner* ("be perspicuous"). These maxims are employed not only by speakers, but also by listeners making inferences about what a speaker meant.

The maxim most valuable for present purposes is *relevance*; viewers of graphic presentations have an expectation that the different pieces of the presentations will be meaningful, as will relevant perceptual features. For example, if some points in a scatter plot are colored blue, and some are colored red, viewers will assume that the color distinction means something.[5]

5.2.3 Graphic conventions

Graphic presentations have been around for centuries, and conventions have developed for their use. The violation of one of these conventions can make a presentation very confusing. An ignored convention can force a user to spend extra time understanding the encoding conventions, while a flouted one can lead to a misunderstanding of the information the presentation is intended to convey.[6]

Many conventions are based on metaphors. For example, positions of greater power are typically conceived of as "higher" rank or position in western cultures. As a result, one might expect that a tree showing all of a company hierarchy will be interpreted more easily if the CEO is placed at the top, rather than the bottom of the page. Similarly, if being a subclass is typically conceptualized as containment, a presentation using containment to show subclasses will probably be understood quickly and correctly.

[4] Lohse (1991b) makes this point.

[5] Other Gricean maxims undoubtedly have importance for graphic presentations as well, though generally not at the level of the present analysis. That is, these maxims tend to be most important when analyzing issues of pragmatics, which often go beyond my treatment of graphic communication as a process of encoding and decoding information. For example, imagine a network graph so complex and tangled that it is effectively incomprehensible. This graph, flouting the maxim of manner, might be used to convey how difficult to comprehend the data is, but such an analysis does not fit into my framework. More generally, pragmatic issues such as presenting the "appropriate" type and amount of information for a given user and task are beyond the scope of this dissertation. They are discussed by Kosslyn (1989) to a limited degree, however.

[6] Note that I am using the word convention in two distinct ways. When I talk about the *encoding conventions*, I am referring to any encoding schema used by a graphical language. When I talk about conventions more generally, I am talking about accepted conventions developed over time.

Other graphic conventions are based on more general cultural factors. For example, Americans and Europeans virtually always represent time as a progression from left to right. There is evidence to suggest that this is related to the conventions of written language. In cultures that read and write from right to left, children are more likely to represent temporal increases in that direction when creating representations than in cultures that write from left to right (Tversky 1995).

Other conventions, while quite possibly originating in cognitive limitations or cultural associations, have roots which can at best be hypothesized. The use of color in weather maps may well be influenced by semantic associations such as the coolness of water (which can be blue) and the colors exhibited by metal when it is heated (i.e., red and yellow). However, it is probably best treated simply as a convention, since it cannot easily be generalized.

Pinker (1990) has argued that knowledge of the meanings of particular configural properties as they occur in different types of presentations is extremely valuable for understanding graphs involving such properties. This also argues for the use of "conventional" encodings. For example, linear encodings of position are generally more familar than logarithmic ones, at least among the general populace. Thus, it would be easier for the average viewer to understand the correlation between two sets of values as encoded by the shape of a point cloud plotted on linear axes than it would be for such a viewer to understand the meaning of a point cloud plotted on logarithmic axes.

5.3 Design principles for interpretability

Having discussed the principal reasons that some graphic presentations may be more difficult to interpret than others, it is possible to distill a set of design principles, a set of qualities that a presentation should posess in order to ensure quick and accurate interpretation:

- **Simplicity** Presentations should minimize the complexity of their encoding conventions.
- **Consistency** Presentations should use, when possible, consistent mappings of perceptual features to information.
- **Compatibility** Presentations should, if possible, use mappings compatible with perceptual judgments and semantic associations of perceptual dimensions and values.
- **Congruence** Presentations should not confuse any cognitive structure implicit in the information being presented.
- **Relevance** Salient perceptual features in a presentation should be semantically meaningful.
- **Conventionality** Conventions for presenting data should be used, where possible, all other things being equal.

These principles overlap in some ways, and may not be complete. Nevertheless, they seem to capture most of the more specific design principles for guaranteeing interpretability described in various sources (e.g., Kosslyn (1994)). Various instantiations of these principles also seem to be successful in helping the AUTOGRAPH implementation, described in chapter 7, to design easily-interpreted presentations.

Having described these six general principles for interpretive effectiveness, they can be applied in a number of specific ways. In the remainder of this section, I further explain each of the principles in turn, and present a detailed survey of the ways they can be used. Implicit in the survey will be two divisions described in chapter 3: the division between the identification and extraction phases of the use of a presentation and the division between the three levels of perceptual feature to information mapping (i.e., single properties, multiple properties of a single object, and multiple objects in a presentation). The principles cited above are applicable to both phases, and at all levels of mapping. As I argued in chapter 3, these divisions provide a structure that will help guarantee the thoroughness of the discussion.

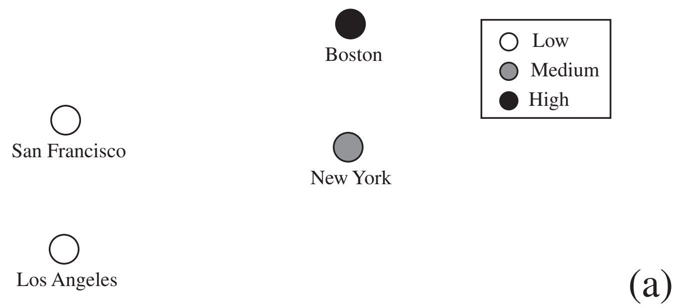
There are also two general issues to take into account when considering all of these principles: the level of sophistication of the intended audience and the context of the presentation. By level of sophistication, I mean familiarity with different encoding conventions. More sophisticated viewers will be able to identify a larger set of potential encoding conventions quickly. By context, I mean whether the presentation method will be used once, say in a newspaper, or repeatedly for varying data sets. In the former case, it is important that the encoding conventions be understood quickly; in the latter case, viewers may be expected to develop a facility with the graphical language over time, even if it is initially difficult to interpret.

5.3.1 Simplicity

For cognitive reasons, as discussed in section 5.2.1, a "simpler" presentation will generally allow viewers to identify encoding conventions more quickly and to use them with less need to refresh their memories than a more complex presentation would require. There are several different possible measures of the simplicity of a graphical language: the number of different types of objects, the number of different types of properties used to encode information, etc.

For identification to proceed quickly and accurately, the simplicity principle would suggest using as few objects as possible—i.e., using a graphical language with a partitioning into a smaller number of groups. This will result in fewer objects whose meanings must be determined. For example, in figure 5.2, (a) should be somewhat easier to understand than (b). It is impossible to change the number of data domain to perceptual dimension mappings, since each data domain is mapped to some perceptual dimension, but it is possible to reduce the number of *distinct* mappings by re-using them. For example, encoding the domain *cities* with both vertical and horizontal position (as in figure 5.3.a) should make the encoding conventions easier to remember than if cities were also encoded with grayscale values (as in figure 5.3.b).[7]

[7] Note that this principle can also be thought of as an example of the consistency principle.



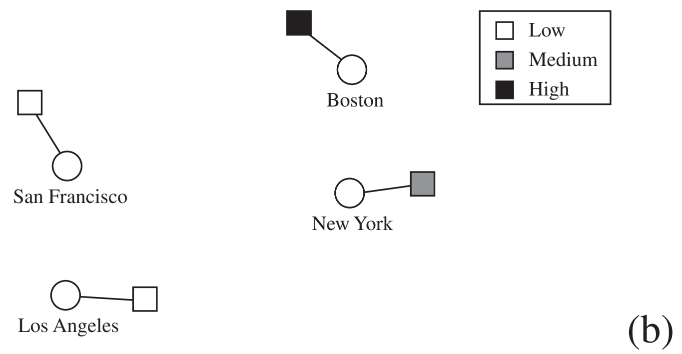


Figure 5.2: Using fewer objects leads to more easily interpreted presentations

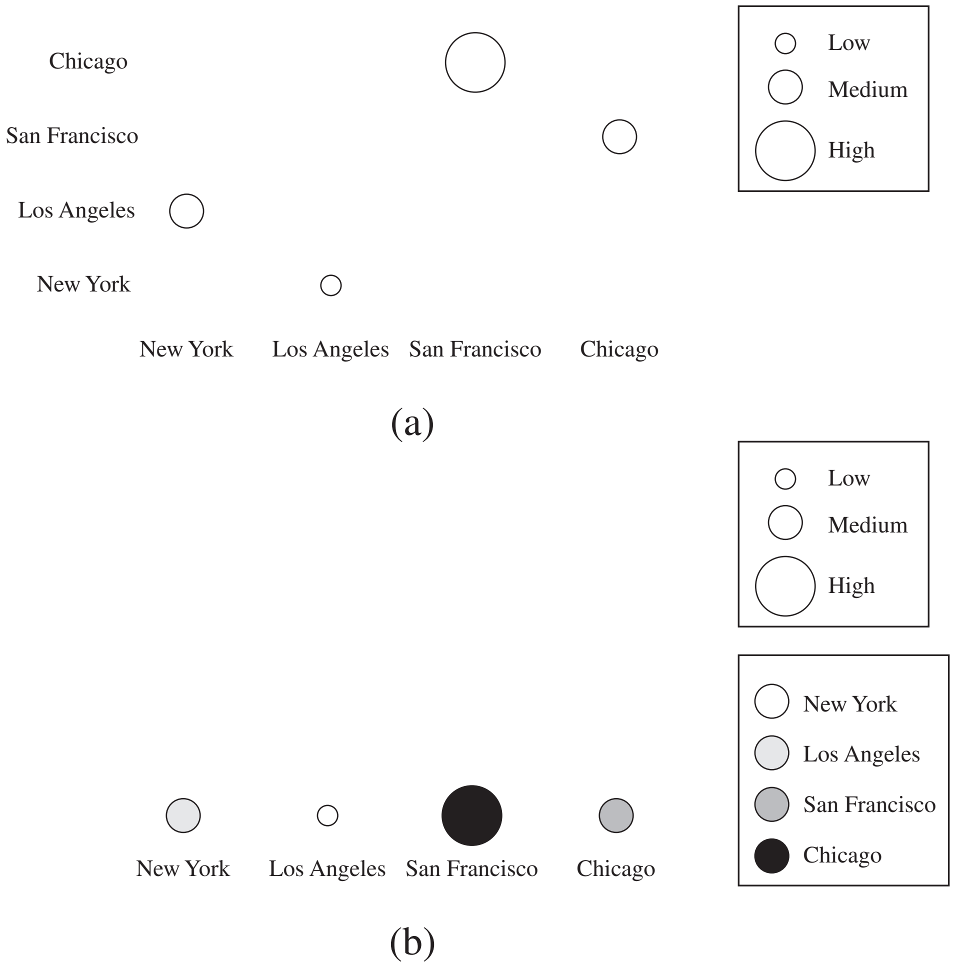


Figure 5.3: (a) Reusing mappings (b) Using two distinct mappings

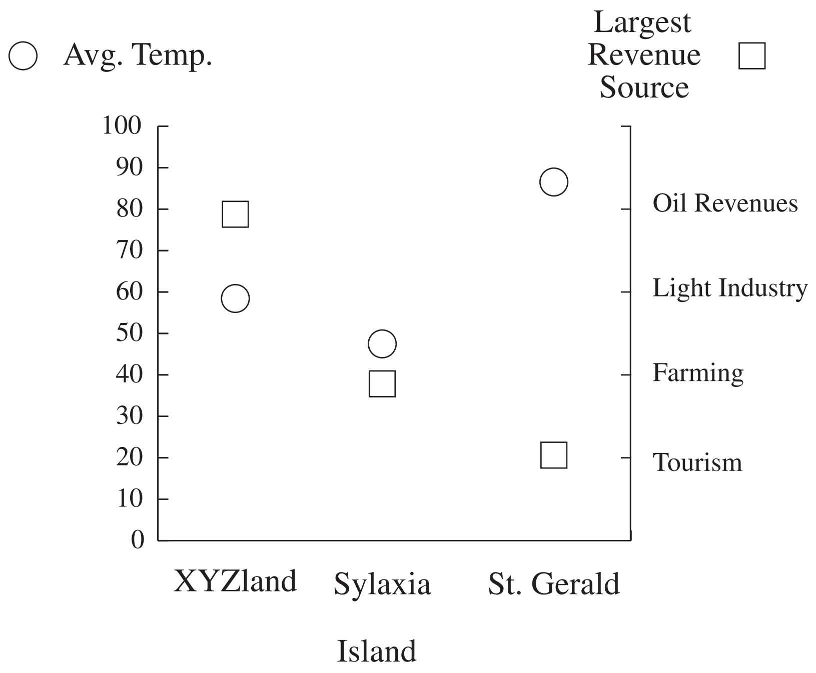


Figure 5.4: A graph with inconsistent use of space

5.3.2 Consistency

Consistency is the principle that representations of similar information should be perceptually similar, and vice-versa. Like simplicity, consistency makes the encoding conventions of a graphic presentation easier to identify and to remember. In many instances, consistency overlaps with simplicity; making a graphical language more consistent by using similar encoding conventions multiple times will also make it simpler. For example, encoding the two instances of the domain *city* by the horizontal and vertical positions of a circle (as in figure 5.3.a) rather than by the horizontal position and value of the circle (as in figure 5.3.b) makes the presentation both more consistent and simpler. In other instances, however, inconsistency is problematic even where simplicity is not an issue. For example, a presentation using the vertical positions of different types of objects within the same space to represent different domains, as in figure 5.4, is likely to be somewhat confusing, probably because people may misapply the encoding conventions of one set of objects to the other set. Using space in a semantically consistent way seems to be particularly important. Although graphs that use space in multiple ways are fairly common, they should be more difficult to interpret that presentations that use space in a uniform way.[8]

Consistency applies at the level of subtuples to object mappings, too. For example, the graph in

[8] As Kosslyn (1994) has pointed out, it is sometimes useful to superimpose two graphs using different encoding conventions—e.g., line charts of crime rate and unemployment—when only the trends of the presentations are important, and comparing these trends is desired. In these cases, the inconsistent use of space is less of a problem because the specific data encoded (inconsistently) by individual objects is less important than the trends, which might be thought of as being encoded consistently.

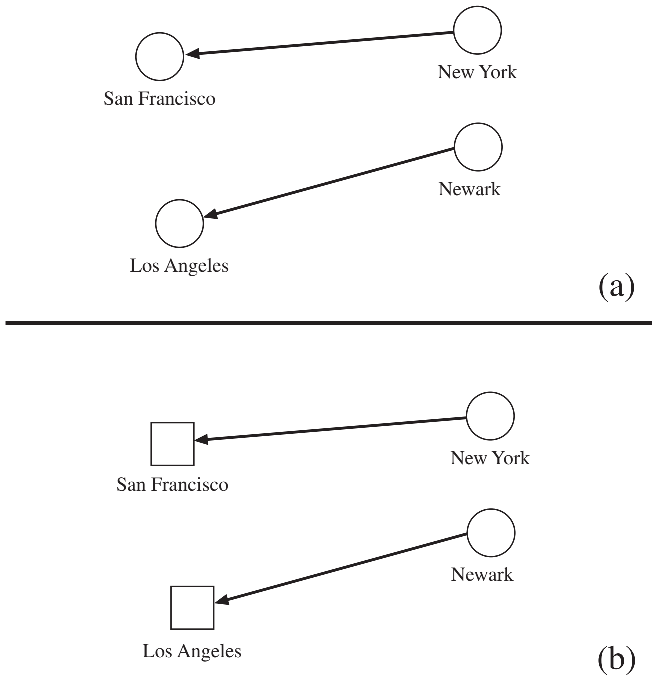


Figure 5.5: (a) Consistent use of object types (b) Inconsistent use of object types

figure 5.5.a uses arrows between two circles to show some relationship between pairs of cities. Figure 5.5.b, which for each tuple represents one city with a circle and one city with a square (e.g., for $<city_1, city_2>$, represents $city_1$ with a circle and $city_2$ with a square), is likely to be slightly more difficult to understand.[9] Note that the complement of this application of the consistency principle, the notion that *different* subtuples should be represented by *different* types of objects, is built into the systematicity assumptions described in chapter 3, which prevent any single class of perceptual object from being used to encode multiple types of information.

5.3.3 Compatibility

Certain perceptual dimensions have either innate or learned semantic associations. A graphical language making use of such associations is likely to be easier to interpret than one that ignores them, and much easier to interpret than one that flouts them. I will use a term, *compatible* (suggested by Kosslyn (1994)), to describe mappings that respect these associations. As suggested in section 5.2.1, part of the reason compatibility may be useful is that it lessens the demand on short-term memory.

It is possible to distinguish at least two types of compatibility: *perceptual* and *metaphoric*. Perceptual compatibility is the principle that where possible, perceptual differences should correspond to semantic

[9] This is also predicted by the relevance principle, to be discussed shortly; the difference in object types suggests a difference in the data encoded by the objects.

Figure 5.6: Two reasonable mappings and one questionable one

ones. This is particularly important in mapping perceptual dimensions to domains. At a minimum, if a perceptual dimension that is perceptually ordered represents an ordered domain, successively larger domain values should map to either successively higher or lower perceptual dimension values, as illustrated in figure 5.6.[10] Beyond that, it is possible to distinguish several subclasses of compatibility which might be thought of as principles in their own right. *Iconicity* is the principle that larger domain values should map on to larger sizes and other "greater" perceptual values. *Proportionality* is the principle that perceived ratios between perceptual values should correspond to ratios between represented semantic values, if possible. Proportionality is generally violated in graphs with truncated axes, often leading to misinterpretations. For example, the bar graph in figure 5.7 exaggerates the increase in profits, because the relative sizes of the bars do not correspond to the true proportions of the profit values.[11]

Metaphoric compatibility is the principle that mappings should be consistent with any semantic associations of perceptual objects, relations, and values. It is applicable at many levels of mapping:

- If multiple classes of objects are used in a presentation, ideally their semantics should be related to the semantics of the tuples they represent. For example, a set of tuples giving the latitudes and longitudes of airports, $\{$*airport(25.2,31.3), airport(30.1,18.4),* $\ldots\}$, might be represented with a number of positioned airplane icons, but probably not with positioned automobile icons.

- Perceptual relations should be used in a way that respects their semantic associations. For example, a chart showing the power hierarchy of an organization should place people in superior positions *above* people in inferior positions, rather than *below* them.

- Perceptual dimensions can have semantic associations with different types of domains; these should be respected. For example, latitude should generally be represented as vertical position, and longitude as horizontal position.

[10] For certain perceptual dimensions (e.g., grayscale value), it is difficult to say which end of the continuum of values is low and which is high.

[11] Of course, sometimes misinterpretations like this may be intended. For a general discussion of graphical integrity, see Tufte (1983).

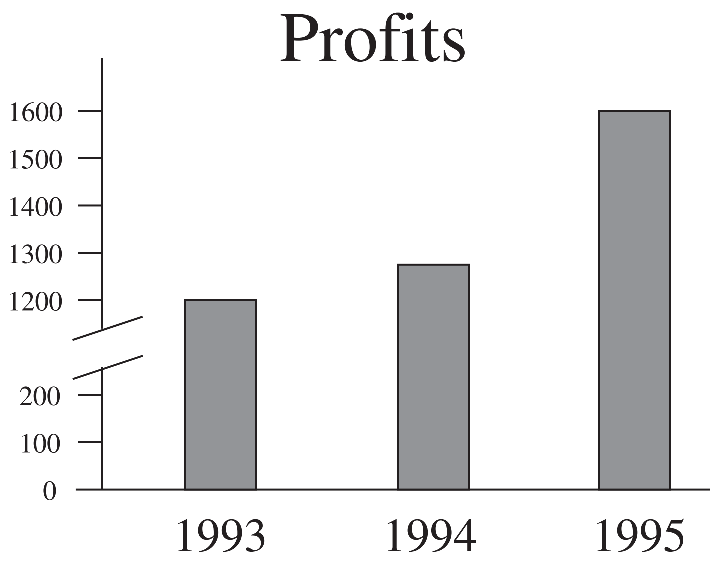


Figure 5.7: A violation of the principle of proportionality

- Individual perceptual values should encode the domain values with which they are most semantically associated. For example, red should be used to represent danger and green to represent safety, rather than the reverse.

There is a great deal of overlap between the notions of metaphoric compatibility and conventionality described here, since very general conventions may often be based on metaphor. I will not be overly concerned with determining the line between the two concepts, however, because of principles will suggest similar choices when designing presentations.

5.3.4 Congruence

Sometimes, some structure may underlie the domains of a tuple, so that certain domains are semantically grouped with other domains. For example, if a set of arity-4 tuples represents troop movements with the latitudes and longitudes of the start and end of the movement, the latitude and longitude of the start are semantically grouped together, as are the latitude and longitude of the end. Structure is also often present in the perceptual-level description of a tuple, in the way that different domains may be represented by different objects, and in some perceptual groupings of the properties of individual objects. As an example of the latter type of structure, a line can be described by the horizontal and vertical positions of both its endpoints; the horizontal and vertical position of each endpoint group together, in a way that the horizontal position of one endpoint does not group with the vertical position of the other endpoint. If the structure implicit in a viewer's conceptualization of some information is incompatible with the structure present in the presentation of the

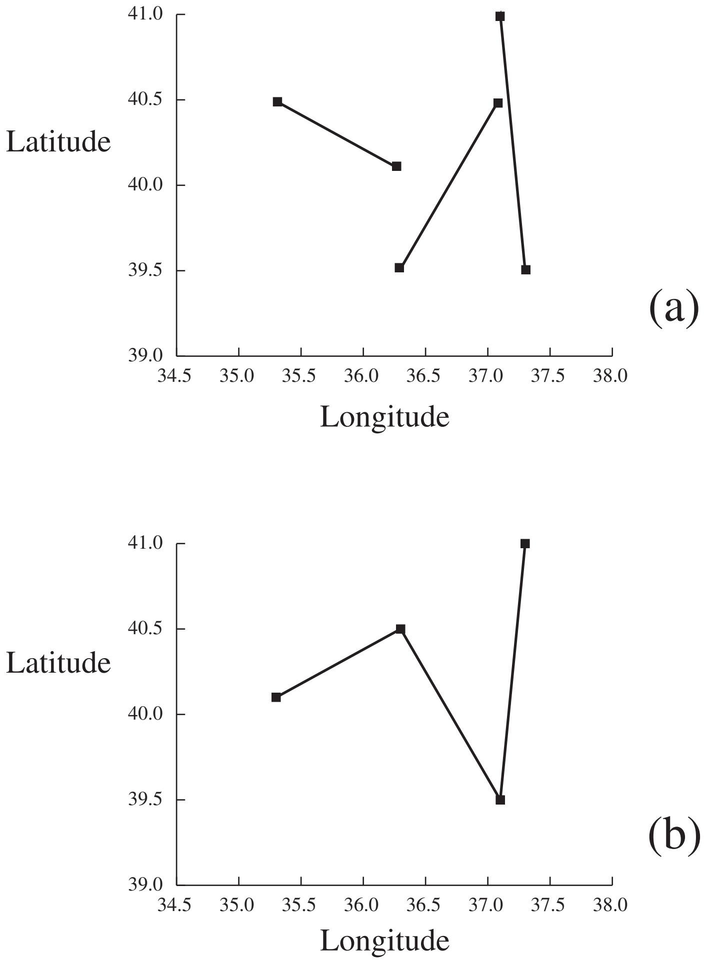


Figure 5.8: (a) A graph violating the principle of congruence (b) A graph respecting the principle of congruence

information, confusion may result, since extra cognitive processing may be necessary to convert from one structure to the other.[12]

For example, consider presenting troop movements represented by the following dataset (with latitude and longitude paired as just described):

$\{< 40.1, 35.3, 40.5, 36.3 >, < 40.5, 36.3, 39.5, 37.1 >, < 39.5, 37.1, 41.0, 36.5 >\}$

In the presentation shown in figure 5.8.a, the data values for each tuple are represented by the endpoint positions of a line. That is, the horizontal position of each endpoint encodes a longitude value and the vertical position of each endpoint encodes a latitude value. However, the way latitude and longitude values are paired is incorrect. As a result, this presentation is much more confusing than that of figure 5.8.b, in which perceptual grouping mirrors semantic grouping.[13]

[12] This reasoning is speculative. The value of congruence as a design principle, however, should be intuitively obvious.

[13] At first glance, it may be extremely difficult to see how figure 5.8.a encodes the data at all, demonstrating the importance of this

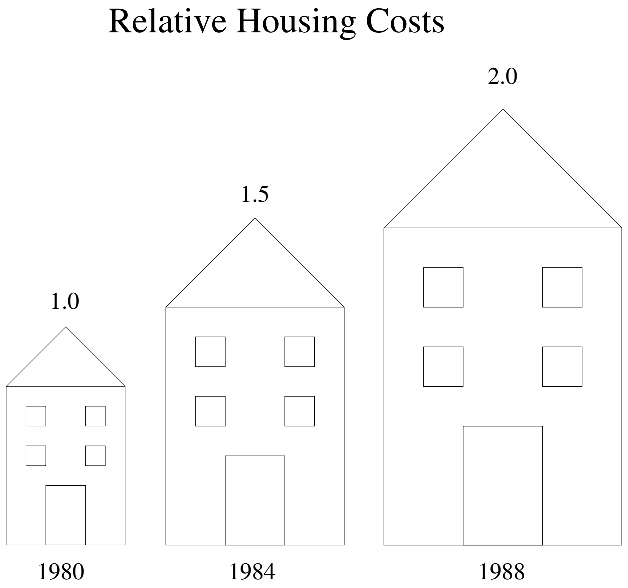


Figure 5.9: A graph subject to misinterpretation

5.3.5 Relevance

The relevance principle suggests that when identifying the encoding conventions of a presentation, viewers are likely to assume that the most salient dimensions along which the objects in the presentation vary are the ones which encode meaning. If this is ignored, misinterpretations may result when viewers try to interpret a non-semantic dimension. For example, in figure 5.9, the areas of the house icons are more salient than their heights, and viewers are likely to mis-estimate the relative numbers of houses built in different years. This point was made by Tufte (1983).

This principle has other applications, too. It suggests that non-semantic perceptual variation in a presentation should be minimized. For example, the graph in figure 5.10 is likely to be confusing because the varying vertical positions of the circles suggest incorrectly that vertical position carries meaning. The relevance principle also provides an explanation for why perceptual properties that seem to encode amounts (e.g., size) should not be used to represent data values that are coordinates (e.g., dates) (as noted in Roth & Mattis (1990)); perceptually salient proportional differences in sizes that have no semantics are likely to be confusing. Finally, relevance provides a rationale for evenly spacing perceptual values when representing nominal or ordered data domains. Extra variation would be meaningless, but the salience of the variation might lead a viewer to assume otherwise, as in the graph of figure figure 5.11.

design principle.

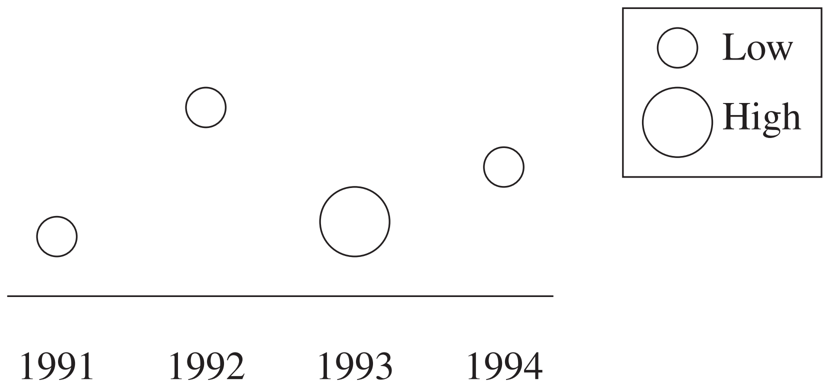


Figure 5.10: A graph with confusing non-semantic variation

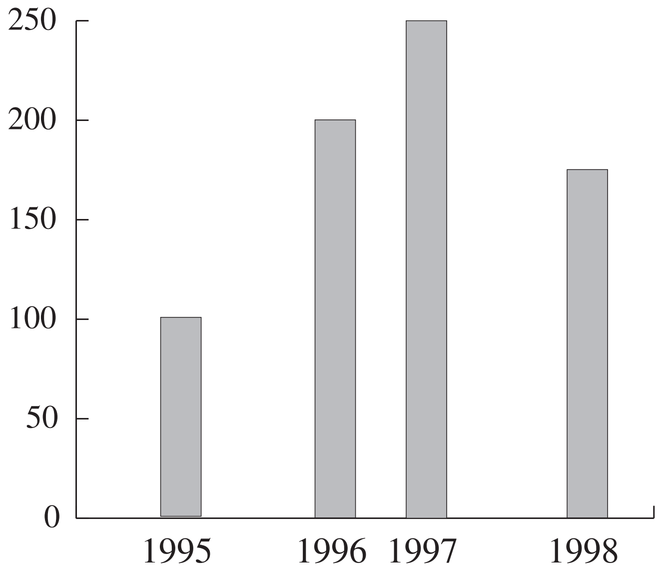


Figure 5.11: Another graph with confusing non-semantic variation

5.3.6 Conventionality

Certain conventions of graphic presentations cannot be accounted for with any of the more general principles, even metaphoric compatibility. For example, in western cultures, axes are generally displayed with values increasing upward and to the right. This in itself is a convention, but certain domains are treated even more specifically. Time is typically displayed along a horizontal axis. If it is displayed along a vertical axis, it is generally done so with greater values below lesser values, contrary to the standard convention.

There are many other conventions for the use of color, space, icons (shape), line-dashing, and other properties. From the point of view of an automatic presentation design system, specific conventions are easy to use if the data characterization language allows sufficient specification of the semantics of the domains and domain values for which the convention is appropriate.

5.4 Summary

This chapter addressed issues relevant to the interpretation of graphic presentations. Section 5.2 discussed some of the sources of interpretive tendencies and difficulties related to graphic presentations. These were cognitive limitations (including processing and memory limitations), constraints due to the nature of graphic communication, and graphic conventions. Section 5.3 outlined a set of design principles for ensuring interpretability, based on the interpretive tendencies and difficulties, and discussed how they apply to different aspects of graphical languages. These principles, repeated for convenience here, are:

- **Simplicity** Presentations should minimize the complexity of their encoding conventions.
- **Consistency** Presentations should use, when possible, consistent mappings of perceptual features to information.
- **Compatibility** Presentations should, if possible, use mappings compatible with perceptual judgments and semantic associations of perceptual dimensions and values.
- **Congruence** Presentations should not confuse any cognitive structure implicit in the information being presented.
- **Relevance** Salient perceptual features in a presentation should be semantically meaningful.
- **Conventionality** Conventions for presenting data should be used, where possible, all other things being equal.

While these principles overlap in some ways and are not intended to be complete, they provide a good starting point for determining how to create easily-interpreted presentations.

Chapter 6

Perceptual Principles

In the last two chapters, I discussed formal issues and algorithms for evaluating the logical expressiveness of graphic presentations and design principles intended to guarantee that graphic presentations will be easily interpreted. In this chapter, I turn to the question of how to design *perceptually* effective presentations, addressing the final set of issues set forth in chapter 3 as a requirement for creating an automatic presentation system. I will discuss in this chapter perceptual issues relevant to graphic presentation and how these principles can be employed to help design effective presentations.

When a viewer is first exposed to a presentation, it is largely visual perception that dictates how he or she determines its encoding conventions—i.e., what classes of objects are being used in the presentation, how compound objects are formed from component objects, and what properties of the objects are being used to encode information. Similarly, it is through visual perception that viewers accomplish given tasks using the presentation (e.g., determining net profits for 1987), by locating the objects of the presentation encoding the relevant information and determining precisely what information they encode. Clearly, knowledge about human perceptual capabilities will be of paramount importance in determining the effectiveness of a method of graphic presentation.

There exists a large body of psychological literature concerning visual perception. In order to make sense of the perceptual research relevant to graphic presentation, I will discuss it from two angles. In the next section of this chapter, I will give a brief overview of a number of issues in visual perception that are relevant to the effectiveness of graphic presentations. In section 6.2, I will address the question of how these issues fit into my framework, and how they can help guide the design of presentations.

6.1 Visual perception

I will not attempt to provide a detailed survey here of the vast amount of research that has been done on visual perception. Rather, I will attempt to highlight general issues in visual perception which are particularly relevant to the effectiveness of graphic presentations. The next section of this chapter will discuss

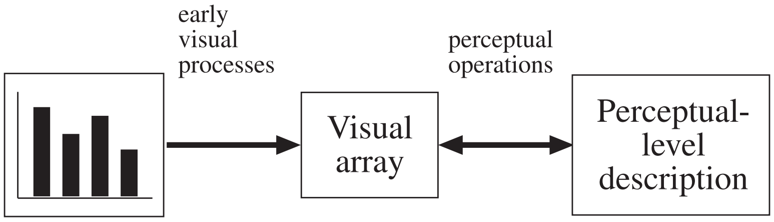


Figure 6.1: A rough model of visual perception (adapted from Pinker (1990))

precisely how these issues are relevant; here I will only explain concepts and describe phenomena.

Before discussing the details of various aspects of visual perception, it will be useful to clarify some basic assumptions. For purposes of analysis, I will consider perception to consist of two types of processes: bottom-up processes, employed autonomously when a viewer first sees a graphic presentation, and top-down processes, employed to accomplish specific tasks. The bottom-up processes will be assumed to produce some primitive visual representation in the mind of a viewer; the top-down processes operate on this representation to produce the perceptual-level description. This basic model, shown in figure 6.1, is roughly analogous to part of Pinker's model of graphic perception (Pinker 1990), discussed in chapter 2.

In chapter 3, I discussed the perceptual-level description of a presentation as an abstract entity, a subjective description corresponding to the objective description of a presentation at the graphic level. When trying to analyze perceptual effectiveness, however, it is important to realize that a viewer's mental representation of a presentation is not constructed all at once, but rather over a period of time. A viewer actively seeks to extract information from a graph, in accordance with his or her goals. This is accomplished through a series of top-down processes I will refer to as *perceptual operations*, including visual searches for objects with particular properties and determinations of other properties of those objects. These operations take time to carry out, and some are quicker or more accurate than others. Perceptual effectiveness is largely a function of the time and accuracy with which these operations can be executed.

Given this basic model, there are number of specific issues in visual perception, addressed by psychological research, that are relevant to the effectiveness of graphic presentations. These include perceptual organization, the dimensional structure of visual stimuli, and perceptual operations. I will discuss each of these in turn.

6.1.1 Perceptual organization

One early perceptual process which operates autonomously when a viewer first perceives a graphic presentation is the organization of the elements of the presentation into groups or patterns. Perceptual organization was studied first (in a general context) by the Gestalt psychologists, who came up with a set of principles describing which elements in an image will group together (see Palmer (in press)). The principles they originally described include proximity (nearby objects tend to group together), similarity (objects that

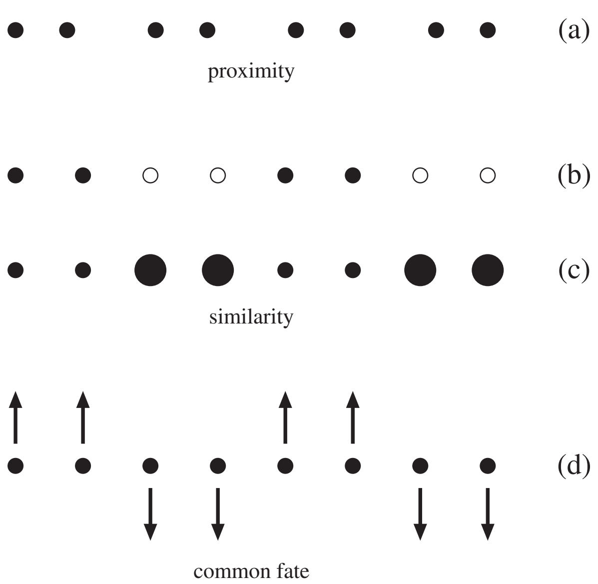


Figure 6.2: Principles of grouping (adapted from Palmer (in press))

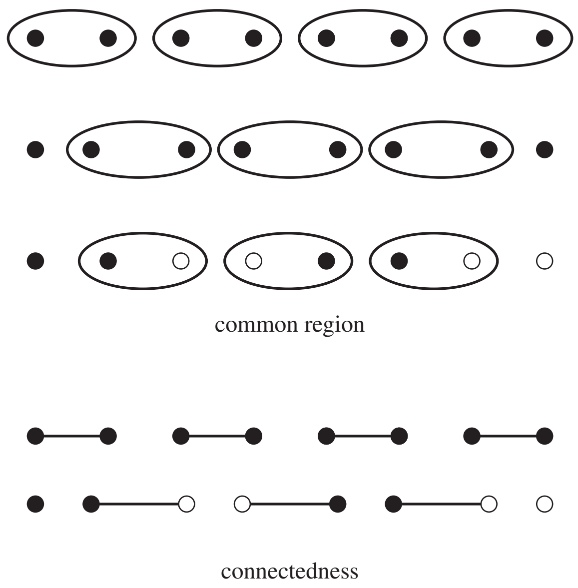


Figure 6.3: Recent principles of grouping (adapted from Palmer (in press))

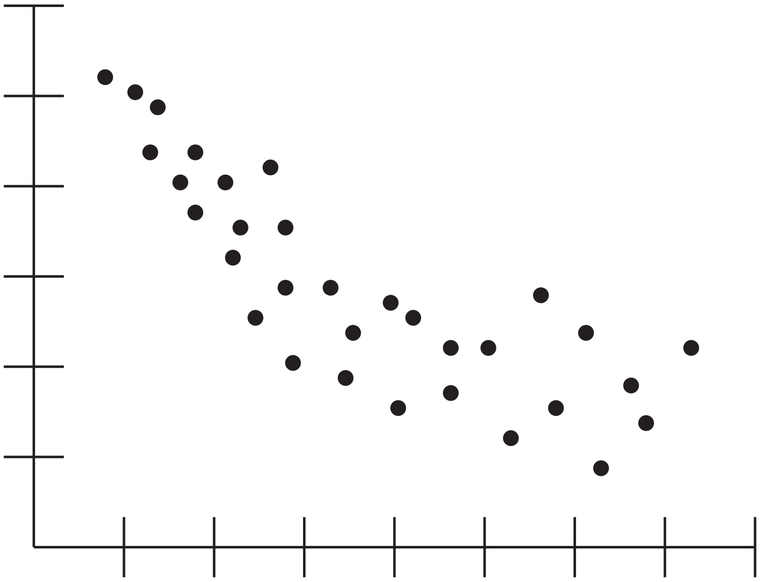


Figure 6.4: A scatterplot manifesting configural properties

are similar tend to group together), and common fate (objects moving in the same direction tend to group together). These principles are illustrated in figure 6.2 (adapted from Palmer (in press)): in (a) pairs of circles that are close to each other will group together; in (b) circles of the same shade group together; in (c) circles of the same size group together; and in (d), circles moving the same direction (indicated by the arrows) group together. More recently, some researchers have argued that connectedness (Palmer & Rock 1994) and common region (i.e., that two elements are enclosed in the same way) (Palmer 1992) should also be considered grouping principles. The effects of these principles are demonstrated in figure 6.3 (adapted from Palmer (in press)), which shows that common region can take precedence over proximity and similarity, as can connectedness. Note that in general, however, all of these principles are *ceteris paribus* rules, applying when all other aspects of the stimuli are equal.

It is also relevant for purposes of this thesis that because of perceptual organization, large parts of a presentation consisting of many objects may group together to form larger units, with properties (which I will refer to, following Pinker (1990), as *configural*) perceptible to viewers. For example, in a scatter plot with enough points and some degree of correlation (as in figure 6.4), the entire "cloud" of points may have a shape and density, both of which can be meaningful.[1]

6.1.2 Dimensional structure

In addition to perceptual organization, another early process that takes place when a viewer sees a presentation is the determination of the dimensional structure of the elements of the presentation. This is

[1] I will discuss the issue of configural properties further in section 6.2.2.

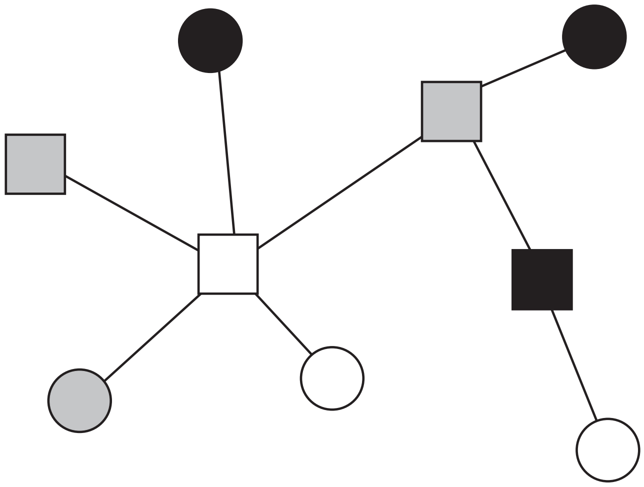


Figure 6.5: A graph in which multiple levels of classification are possible

a necessary step in beginning to understand the encoding conventions of the presentation, and is related to the psychological task of “free classification”—viewers must determine a basis for classifying elements of the presentation into an organized set of classes (i.e., character classes) with different meanings. There are generally many potential perceptual properties associated with any stimulus, and it is not always obvious which dimensions should be used for classification.

In my framework, I distinguish between classifying graphic objects into object classes and making further distinctions with regard to perceptual property equivalence classes (i.e., character classes defined by perceptual property values), but perceptually these tasks have much in common. Stimuli in an image are classified into groups (i.e., object classes) by different perceptual features, then the groups are broken down into subgroups (i.e., character classes) by other perceptual features. For example, the viewer of figure 6.5 might consider circles and squares separate object classes, then subclassify the circles and squares by their grayscale values.

Research on inferring dimensional structure provides us with two types of insights into how people classify stimuli. First, some perceptual dimensions are preferred to others when making classifications. For example, there is a general preference to use value over size (Handel & Imai 1972). Discriminability—i.e., the salience of variation within different dimensions—has also been found to have some effect on classification. This makes sense; as Garner points out, in the extreme case, variation in some dimension may be unnoticeable, and thus unuseable for classification (Garner 1974). However, this seems to be secondary to pre-existing dimensional preferences in free classification tasks.

Preference issues are complicated by a distinction made by some researchers (e.g., Garner (1974)) between *integral* and *separable* dimensions. Integral dimensions are dimensions which exist objectively in a stimulus but which are generally not perceived as separate dimensions by a viewer, such as value and saturation. Integral dimensions tend to act together in classification tasks—i.e., subjects have been found to make classification based on overall differences between two stimuli varying in integral dimensions, rather than on differences in one particular dimension.[2] Separable dimensions correspond more closely to an intuitive notion of perceptual properties, making dimensional structure much more important. Size and value are an example of separable dimensions. Note that dimensional preferences as described in the previous paragraph have been found to be much more reliable for separable properties (Garner 1974). This should be expected, given that integral properties tend to act as single dimensions.

A number of issues relating to performance have been explored with respect to the integral/separable distinction. These will be discussed shortly, in section 6.2.2.

6.1.3 Perceptual operations

After the bottom-up processes of perceptual organization and the inferring of dimensional structure have been accomplished, viewers must make more specific determinations in a top-down manner. They must locate individual objects in the presentation, determine perceptual relations, and determine the specific character classes into which individual objects fall.

Ullman (1985) has proposed a theory of *visual routines* as a general way of explaining how people determine perceptual relationships and object properties. He hypothesizes that the human visual system employs a small number of basic operations which can be combined in various ways to make many different determinations about an image. He further hypothesizes that these basic operations include shifting the processing focus, indexing (shifting the focus of attention to a particular point based on "odd-man-out" properties at that point, e.g., a red object in a field of green objects), bounded activation (determining the area enclosed by a curve), arc tracing (following a curve), and marking (noting an observed point for future reference). According to this theory, determining a perceptual relation involving connectedness (e.g., that a certain circle touches a certain line) would be composed primarily of arc tracing. Determining containment would involve bounded activation. Although I will not make use of the specifics of Ullman's theory, the general assumptions about perception made here are compatible with it.

Visual search and preattentive processing

Many results and theories concerning more specific perceptual operations exist independently of the theory of visual routines. Visual search—the perceptual operation of locating an object or objects with particular properties in a display—is one of the most important of these operations for purposes of determining perceptual effectiveness, since locating a relevant object in a presentation is essential for extracting information from that presentation. Visual search has been studied extensively by psychologists and is still an area of active research.

The speed with which visual search can be accomplished depends on the specific properties that are the subject of the search and on other properties that vary in the presentation. One of the largest determiners of

[2] That is, difference between stimuli seems to be based on a Euclidean distance metric rather than a city-block metric.

the speed of a search for an object with a particular property value is whether that property is *preattentively* processed. Certain perceptual properties are processed by the visual system in a bottom-up manner, very rapidly and seemingly in parallel, without requiring a shift in focus of attention— i.e., preattentively. Such efficiently-processed properties include hue (Nagy & Sanchez 1990; D'Zmura 1991), value (Treisman & Gormican 1988), size (Treisman & Gelade 1980), orientation (of a line or blob) (Julész & Bergen 1983; Wolfe *et al.* 1992), and length (Treisman & Gormican 1988). Psychologists have explored the effects of preattentive processing on a range of visual tasks, documenting how preattentively-processed properties improve subject performance on tasks such as grouping similar elements, detecting elements with a particular characteristic (i.e., property value), estimating the number of elements with a given property value, and visual search.

Early research seemed to indicate that features which are not processed preattentively are processed serially, with selective attention (Treisman & Gelade 1980). Although more recent research has undermined this simple dichotomy, and different theories explain current results in different ways, it is unquestionable that some features are definitely processed less efficiently than others and require more focused attention. These less efficiently processed features include certain aspects of shape and the orientation of complex objects. Generally speaking, a viewer will be able to search for an object or for all objects with a given preattentively-processed property value much more quickly than he or she will be able to search for an object with some other property value. For example, searching for a blue circle among red circles will be very rapid— i.e., essentially independent of the number blue circles in the image— because it makes use of information that has been "pre-computed"—i.e., computed preattentively. Conversely, searching for a 'T' shape among a set of 'L' shapes (where both shapes can be at a variety of orientations) will be much slower, since it appears to be done serially and is dependent on the number of objects in the display (Kwak *et al.* 1991).

This general picture is complicated by a number of factors. First, early research demonstrated that in many cases, conjunctive preattentive search is impossible (Treisman & Gelade 1980). It is possible to search quickly for a blue circle among red circles, or for a large circle among small circles, but not for a large blue circle among small blue and large red circles. To perform this task, a viewer must serially search all the blue circles for the large one, or all the large circles for the blue one. More recent research has shown that in some cases, however conjunctive search *is* possible. Depth (using binocular vision) and some aspects of motion have both been shown to be detected preattentively, but research has shown that subjects can search preattentively by color on a given depth plane without distraction from other depth planes (Nakayama & Silverman 1986), and can detect an X moving horizontally among a group of X's moving vertically (in sync) and a group of O's moving horizontally (in sync) (Driver 1992).

Another general issue affecting the efficiency of visual search with preattively-processed properties is the discriminability of values within the properties of the search. Search is more difficult when the searched-for property values are similar to the property values of other objects in an image, and also more difficult when there is a lot of variation in the property values of those other objects (Duncan & Humphreys 1989). There are also a number of issues related to the processing of particular perceptual dimensions. For color, it has been argued that in addition to general discriminability and the number of different colors used, search seems to be less efficient if the color being searched for is not linearly separable (in color space) from the other

Quantitative	Ordinal	Nominal
Position	Position	Position
Length	Grayscale Value	Color Hue
Angle	Color Saturation	Texture
Slope	Color Hue	Connection
Area	Texture	Containment
Volume	Connection	Grayscale Value
Grayscale Value	Containment	Color Saturation
Color Saturation	Length	Shape
Color Hue	Angle	Length
	Slope	Angle
	Area	Slope
	Volume	Area
		Volume

Table 6.1: Mackinlay's ranking for elementary perceptual tasks

colors in the image (D'Zmura 1991). For example, searching for an orange object among red and yellow objects is difficult. It has also been argued that certain colors are prototypical and easier to search for than other colors (Treisman & Gormican 1988). For orientation, it has been argued that the perceptual system is attuned to certain classes of orientations— steep, shallow, left-, and right-tilted—and searching for an object from one class among others from different classes is more efficient than searching under other conditions (Wolfe *et al.* 1992).

Finally, there is the general issue of interference. Research testing boundary segregation, another task involving preattentive processing, has shown that varying brightness interferes with segregation by hue, though the reverse is not true (Callaghan 1984). Similarly, varying hue interferes with segregation by form (shape), though varying form does not interfere with segregation by hue (Callaghan 1989). Other research has shown interference on classification tasks when integral dimensions are varied (Garner 1974). For example, a subject attempting to make a discrimination based on value will be slower if the saturation values of a set of stimuli are varied orthogonally to the grayscale values. This will presumably influence the speed of a serial search. Taken together, these results suggest that searching—preattentive and otherwise—will be adversely affected by variation in properties other than the ones being searched on, at least in some cases.

Discrimination and comparison

In addition to searching for objects with particular perceptual property values, other crucial perceptual operations for using graphic presentations include determining the values of an object's other properties, and comparing the values of some property for two different objects.

One obvious factor in the perceptual effectiveness of such operations is that some perceptual properties can be determined much more accurately and/or much more quickly than others. There have been a number of efforts to explore this phenomenon as it relates to graphic presentations. Based on a variety

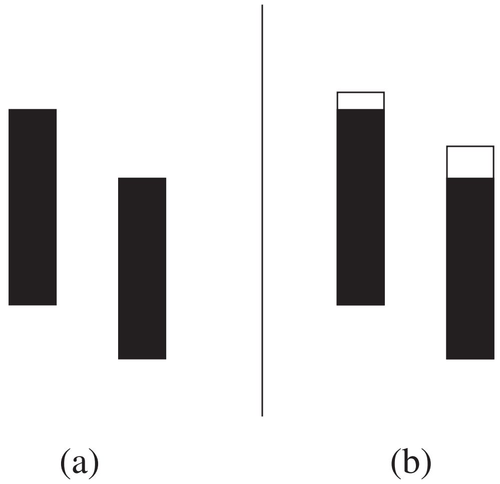


Figure 6.6: An illustration of Weber's law

of factors, Cleveland & McGill (1984) hypothesized a ranking for accuracy in the discrimination of quantitative values for several perceptual tasks, and performed experiments to confirm this ranking. Mackinlay (1986a) extended this ranking to cover additional perceptual tasks, and discrimination for quantitative, ordinal, and nominal information. His rankings (with the quantitative rankings incorporating those of Cleveland and McGill) are shown in table 6.1. Note that Mackinlay's notion of perceptual tasks includes both the discrimination of perceptual properties and the determination of perceptual relations (e.g., connection, equivalent to determining if two objects are touched by a line). His rankings also incorporate certain notions of interpretability in addition to effectiveness; this can be seen in the different ordering of perceptual tasks in different columns. Casner (1990) also constructed a ranking, separating search and discrimination. His rankings combine time and accuracy into a more general measure of "effectiveness".

Another well-documented factor in the effectiveness of perceptual discrimination is that, as with search, it becomes more difficult (i.e., slower and more error-prone) to discriminate perceptual property values when pairs of distinct values are perceptually close. Research in the field of psychophysics addressed this issue. Weber, studying line segment length judgments in the 19th century, concluded that $w_p(x)$, the length that must be added to a line of length x so that the new line can be discriminated to be longer with probability p is, for a fixed p, directly related to x—i.e.,

$$w_p(x) = k_p x$$

According to Cleveland & McGill (1984), Weber's law is applicable to a variety of perceptual judgments.

A compelling demonstration of Weber's law is given in figure 6.6 (adapted from Cleveland & McGill (1984)). Figure 6.6.a shows two rectangles of unequal vertical extent, though it is difficult to make this discrimination. In figure 6.6.b, the same rectangles are surrounded by rectangular frames of equal length, which makes the difference in lengths of the black rectangles readily apparent, because the relative difference in lengths of the white rectangles is much larger than that of the black rectangles.

Another perceptual law, Stevens' law (Stevens 1975), describes the relationship between perceived and actual perceptual quantities:

$$p(x) = c \times x^{\beta}$$

where $p(x)$ is the perceived quantity for an actual quantity x and where c is a constant. β has been estimated to be between 0.9 and 1.1 for length, 0.6 to 0.9 for area, and 0.5 to 0.8 for volume. In addition to describing the phenomenon of consistent underestimation of differences in area, these constants support the rankings given in table 6.1, since they show that length is determined more accurately than area and area more accurately than volume.[3]

The discrimination of color values is a bit more complicated, since color consists of multiple perceptual dimensions—e.g., hue, value, and saturation. However, some color spaces have been designed so that Euclidean distance in the space approximately corresponds to perceived differences in color. The common Munsell color model (Birren 1969) is an example of a perceptually-based color space.

As with search, the accuracy and speed with which property values can be discriminated can be affected by interference. Garner & Felfody (1970) found that with integral dimensions, when subjects were asked to discriminate based on one dimension (e.g., value) but the integral dimension was varied orthogonally (i.e., without any correlation with the value the subject was attempting to discriminate), discrimination times were longer. Conversely, when they were varied in a correlated way, discrimination times were shortened. These effects were not observed with separable dimensions.

Most of the results on discrimination are also relevant to comparison, which is effectively a discrimination of whether two values are different, and possibly how different they are. However, additional factors may affect the the accuracy of comparison. One particularly important factor noted by Cleveland (1985) is *distance*—i.e., when two stimuli are in close proximity, they can be compared more accurately than when they are further apart.

6.2 Applying perceptual knowledge

As discussed in chapter 3, there are two basic phases in a viewer's use of a graphic presentation. In the first phase, identification, a viewer determines (typically without much conscious effort) the encoding conventions of the presentation. This consists of determining the different classes of primitive objects used in the presentation, how they combine to form compound objects, and how certain properties of the primitive and compound objects are used to encode information. In the second phase, extraction, a viewer performs various perceptual tasks in order to extract information from the presentation, dependent on his or her goals. The identification phase is discussed in section 6.2.1; the extraction phase is discussed in section 6.2.2. Before discussing the details of these phases, though, it will be helpful to clarify the goals of this analysis.

[3] This is discussed by Cleveland & McGill (1984).

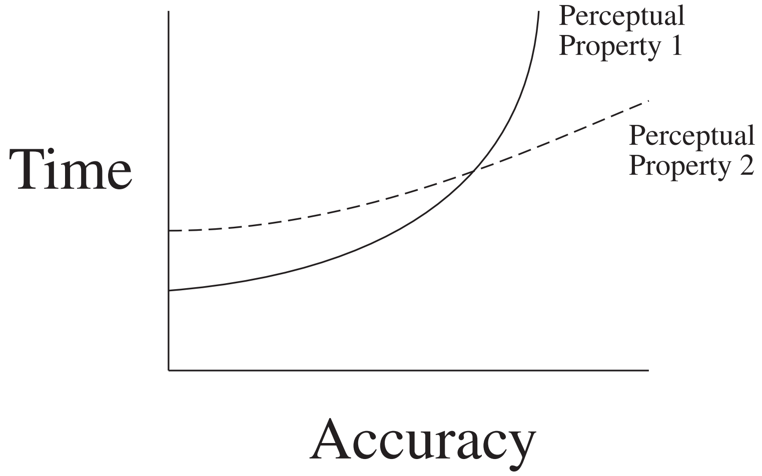


Figure 6.7: The time/accuracy tradeoff

The primary goal of this analysis is to determine some design principles that will help create perceptually effective graphic presentations. Perceptual effectiveness is not a unitary concept, however. As it relates to the identification phase, a perceptually effective presentation will be one which makes use of perceptual abilities to enable a viewer to quickly and accurately determine its encoding conventions. This has much in common with interpretive effectiveness; my design principles for perceptual effectiveness can thus be considered more specific versions of some of the interpretive effectiveness principles discussed in the previous chapter.

Perceptual effectiveness for the extraction phase is much more quantifiable, and has been the focus of previous work on the effectiveness of graphic presentations. This work has taken the effectiveness of a presentation to be a function of either the accuracy (Cleveland & McGill 1984; Mackinlay 1986b) or the speed (Lohse 1993) with which the presentation allows necessary perceptual operations to be performed, or a function of some metric merging both speed and accuracy (Casner 1991). Clearly, both of speed and accuracy are important. To a large degree, they are also covariant; some of the most accurate perceptual operations can also be performed most quickly. But this is only generally the case. As an example of an exception, a printed number with ten digits after the decimal point can be decoded much more accurately than any perceptual property, but not more quickly.

Generally, it is reasonable to assume that there will be a trade-off between speed and accuracy. In some cases, it may be possible to perform a given perceptual task more accurately by taking more time. There is thus an inherent potential difficulty in comparing the perceptual effectiveness of two presentations: the relative speed may be different at different levels of accuracy (see figure 6.7). Since there is not enough empirical psychological evidence to determine speed vs. accuracy curves for all the perceptual operations relevant to graphic presentation, it is necessary to make some simplifying assumptions. These can be summarized as follows:

- The effectiveness of a presentation with respect to the extraction phase is a measure of the time it will take to complete different goals at some fixed level of accuracy, using the presentation.

- Presentations that do not support this fixed level of accuracy can be considered inexpressive.

Note that the context in which a graphic presentation is used will determine the relative importance of the effectiveness of the identification phase and the extraction phase. If a method of presentation is being designed for a single presentation to be used only once, understanding the encoding conventions quickly will be very important. If a method of presentation is to be used over and over again by viewers who will become accustomed to the encoding conventions, the effectiveness of the extraction phase will be much more important.

For purposes of this discussion, it will be useful to divide the design principles for both the identification and the extraction phases into three types:

- **Expressiveness principles**—Principles that are so compelling, they allow an automatic design system to exclude certain types of graphical languages from consideration.

- **Choice (Effectiveness) principles**—Principles that provide a system with a way of determining that one graphical language is better than another, at least for satisfying particular goals.

- **Optimization (Effectiveness) principles**—Principles that provide guidance for augmenting a graphical language with additional constraints that make it more perceptually effective, or that help make fine-grained choices within a chosen but not fully specified graphical language.

When I describe design principles throughout the next two sections, I will note the classes into which they fall. It is worthwhile to note that the boundaries between these classes of principles are largely pragmatic and a function of my framework. It will be useful to distinguish expressiveness principles from choice principles because expressiveness principles allow a search tree of graphical languages to be pruned, while choice principles merely help to rank branches in the tree or to rank the resulting presentations. It will be useful to distinguish optimization principles from choice principles because, as described in chapter 3, the space of graphical languages in my framework does not fully constrain the languages, and optimization principles can be used to make a chosen graphical language more effective.

6.2.1 The identification phase: understanding the encoding conventions

When a viewer encounters a graphic presentation and tries to interpret it, he or she must first determine the encoding conventions of the presentation. One part of this task is to determine how individual component objects group into compound objects. Another part of the task is determining the dimensional structure of the elements of the presentation. This dimensional structure includes both the separation of objects into classes and the division of objects within a given object class into different character classes—i.e., different equivalence classes corresponding to different meanings—based on the properties of those objects.

As noted earlier, the task of understanding the encoding conventions is highly related to interpretation. Thus, the design principles I will set forth in this section will effectively be more specific or "operationalized" versions of principles discussed in the previous chapter.

Although compound objects have generally been discussed as abstract entities throughout this dissertation, from the standpoint of perceptual effectiveness, certain types of compound objects are more likely to be recognized as such. A viewer must be able to determine the set of components that together encode a tuple. Clearly, this is largely a question of perceptual organization. Thus, factors affecting how individual elements will be perceptually organized into groups will influence the ease or difficulty of determining the compound objects in a presentation. While it is clear that compound objects will be less easy to determine if they overlap, as in a standard network graph, it seems reasonable to assume that the same principles of perceptual organization are still valid. Therefore, we can state the following design principle:

Design Principle (choice): *Choose graphical languages without compound objects, or with perceptual relations causing the components of compound objects to group together perceptually.*

As an example, consider the network graph in figure 6.8.a and a possible alternative in figure 6.8.b, which uses a square *between* a pair of circles instead of a line *touching* a pair of circles to form compound objects. From a purely theoretical stance, this graphical language is valid, but perceptually it is terrible, because the rectangles group with each other rather than with the circles they are between.[4]

It is also possible to state a corollary to this principle:

Design Principle (optimization): *During the layout of a presentation, avoid accidental perceptual organization features such as grouping that have no meaning.*

As an example of a violation of this principle, consider the network graph of figure 6.9, in which certain pairs of circles seem to group together due to proximity. If this grouping is purely arbitrary, the encoding conventions of the graph are likely to be somewhat confusing, since there is a temptation to view the pairs as some sort of compound object, with some additional, unknown, semantics.

In considering dimensional structure, from a perceptual point of view, both class and property value distinctions can probably be considered equivalent. There is no *a priori* reason why the marks in figure 6.5, for example, should be divided into object classes according to shape and then further broken down into character classes by grayscale value, rather than being divided into object classes by value and then subclassified by shape. I am unaware of any research studying differences in classification processes of visual stimuli—i.e., which classifications are seen as more fundamental (object class distinctions) and which as secondary (property value distinctions). For graphic presentations, I expect that these differences are largely a function of higher-level cognitive processes based on expectation and experience. Most of the

[4] The perceptual relation *between* violates the assumptions of chapter 3 because it is not binary (i.e., a relation between a pair of objects), and this graphical language is likely to violate the strict expressiveness condition for complex datasets, but otherwise this example is reasonable.

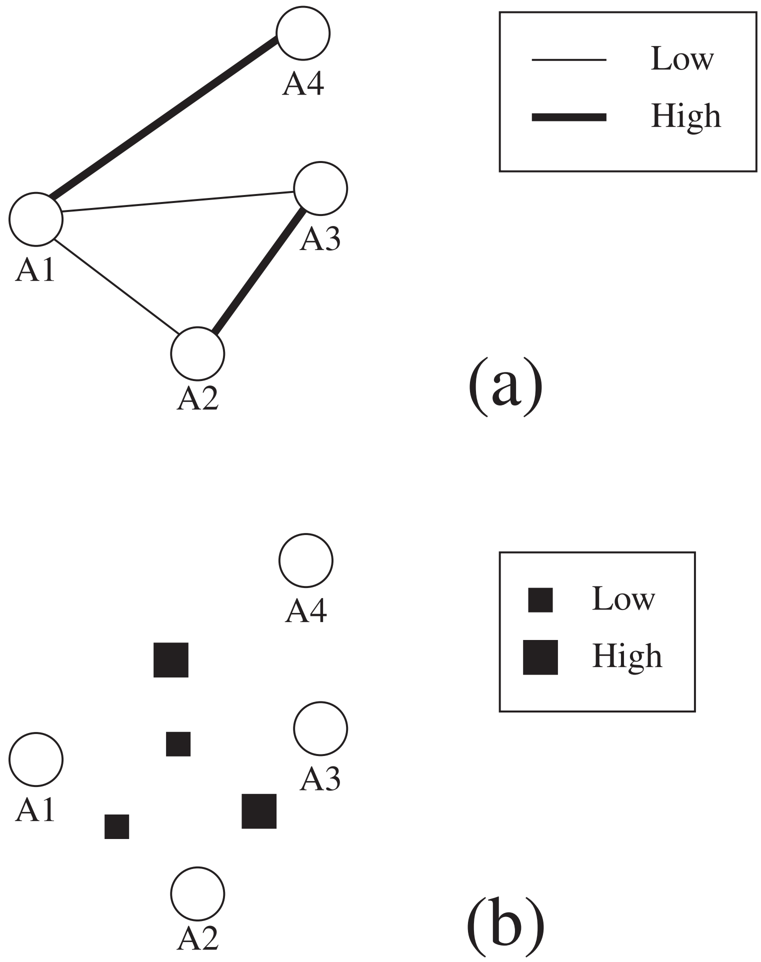


Figure 6.8: (a) A standard network graph (b) A perceptually-ineffective alternative

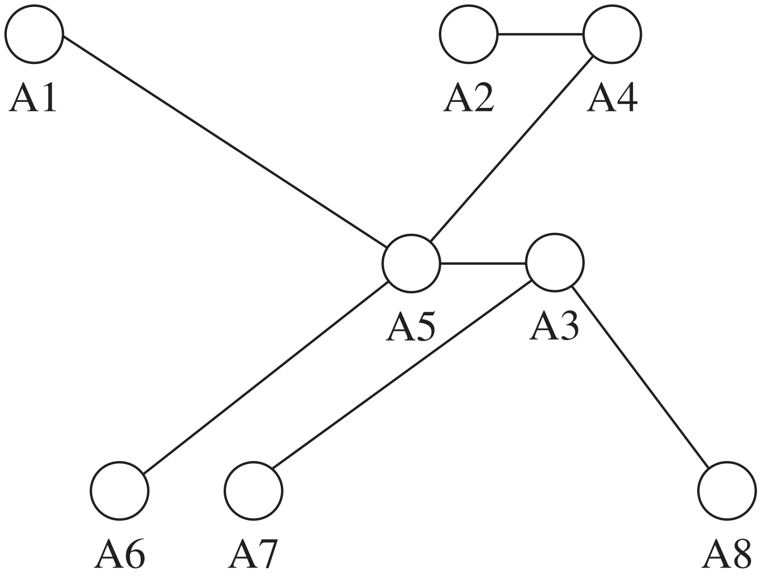


Figure 6.9: Accidental grouping in a network graph

examples discussed in this chapter, and also the AUTOGRAPH implementation to be discussed in chapter 7, assume that shape is more fundamental than other properties of a mark in determining object class, though further research may challenge this assumption.[5]

Given this uncertainty, it is still possible to postulate a design principle based on the research into dimensional structure discussed previously:

Design Principle (expressiveness): *Use separable dimensions for dividing character classes.*

This principle was explicit in the design of Casner's BOZ system (Casner 1991), which used a set of graphical languages based only on separable dimensions. As an example of a violation of this principle, consider using hue and value, known to be integral, to make an object class division and then a division into classes with different meanings. For example, red circles might be used to encode airports, green circles used to encode train stations, and blue circles used to encode bus terminals, with the values (i.e., shades) of the circles encoding the number of flights/trains/buses into and out of the airport/station/terminal. Since the dimensional structure is unlikely to be perceived—i.e., a viewer is likely to see a set of circles of different colors, rather than a set of blue circles with different shades, red circles with different shades, etc.—this presentation is likely to be misunderstood, at least at first.[6] A better alternative would be using shape and value, i.e., using the shapes circle, square, and triangle instead of the colors red, green, and blue.

Since salience of variation in perceptual dimensions can also influence categorization, it is possible

[5] It is also possible that classification will be related to the dimensional preferences discussed in section 6.1.2. That is, it is possible that the most strongly preferred dimensions for classification will be used to distinguish object classes, with less-preferred dimensions used for subclassification. This is speculative, however.

[6] As Garner (1974) points out, it is not impossible to perceive dimensional structure within integral dimensions, just relatively difficult and presumably involving cognitive processes in addition to perceptual ones.

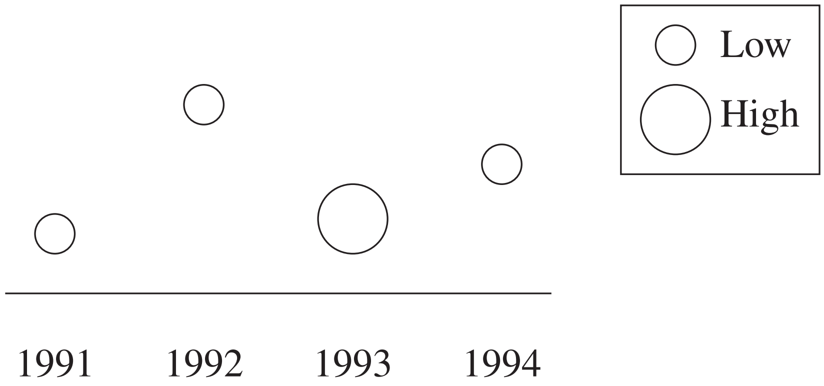


Figure 6.10: A graph with confusing non-semantic variation

to state another design principle, to help insure that the correct properties will be seen as semantic dimensions:

Design Principle (optimization): *Avoid variation in non-semantic properties where possible, particularly in the most perceptually salient non-semantic properties.*

As an example, consider figure 6.10, repeated from the previous chapter. The varying vertical positions of the circles may be confusing here, since they suggest that vertical position encodes something.

6.2.2 Extracting information

As described in the previous section, the perceptual effectiveness of the identification phase is based primarily on the perceptual issues of organization and dimensional structure. The perceptual effectiveness of the extraction phase, conversely, is determined mostly by the speed and accuracy of the perceptual operations a viewer will undertake when extracting information. These perceptual operations will be dependent on the information to be extracted, but this information is itself dependent on the viewer's goals.

Many systems for automatically designing graphic presentations (e.g., Mackinlay's (1986b) APT system) follow Bertin's (1983) classification of goals into three types: *elementary* readings, *comparative* readings, and *summary* readings. An elementary reading is one where the information a viewer desires can be determined from a single tuple. For example, determining the net profit for 1998 in a presentation of a set of tuples describing net profits for a series of years is an elementary reading. A comparative reading involves a comparison between the values of two or more tuples—e.g., determining whether profits increased or decreased in 1998 (i.e., whether they were larger in 1998 or 1997). Summary readings involve determinations based on a large number of tuples, possibly even entire datasets. They include goals such as determining the general trend in profits over the last twenty years, estimating the percent of years which were profitable, etc.

Ultimately, the goals a viewer has determine the sequence of perceptual operations he or she must perform to accomplish those goals using a graphic presentation. Once the encoding conventions of a presentation have been understood, the effectiveness of the presentation is largely determined by how quickly

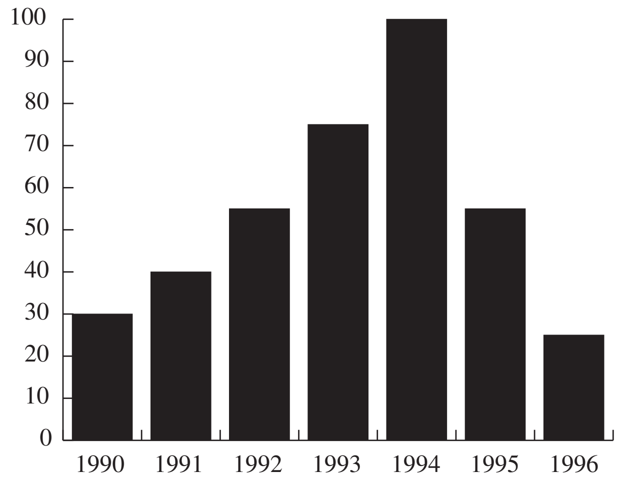


Figure 6.11: A simple bar graph

and how accurately these perceptual operations can be performed. Some previous research has attempted a fine-grained analysis of perceptual operations for very specific goals. Casner's (1991) BOZ system, for example, uses complex specifications of user tasks as input for designing presentations, analyzing the effectiveness with which corresponding sequences of perceptual operations can be performed using different presentations. Lohse's (1993) UCIE system, also, makes predictions about the effectiveness of different types of presentations using a very fine-grained decomposition of the use of a presentation into perceptual and cognitive operations.

Since perceptual effectiveness is not the sole focus of this dissertation, however, I will restrict my discussion to a somewhat coarser analysis and a much more limited set of potential goals. In particular, I will discuss perceptual issues involved in analyzing elementary and comparative goals and a limited set of summary goals. I believe these goals to be most important, and summary goals in particular have not been addressed adequately by previous work.

Elementary goals and perceptual operations

In my framework, each tuple is encoded by a simple or compound object. Thus, only three types of perceptual operations are necessary to accomplish any elementary goal: *search by perceptual property value*, *determination of perceptual relationships*, and *discrimination of perceptual property and perceptual function values*. That is, to retrieve information from individual tuples, based on some part of those tuples, viewers search for component objects relevant to their goal (i.e., the tuple they are querying), check spatial relationships to find compound objects encoding the relevant tuples if necessary, and determine the property values of any components (or perceptual function values between pairs of components) of the compound object needed to satisfy their goal. For example, in a simple bar graph of net profits for a given company such as the one in figure 6.11, if a viewer wants to know how much profit the company made in 1995, he or she does a visual search for the bar at the horizontal position encoding 1995, then determines the height of the bar and what that means.[7] As another example, in a network graph of flights between cities, if a viewer wants to determine if there is a flight between New York and San Francisco, they could locate a circle labeled "New York", locate a city labeled "San Francisco", and then determine if there is a line touching each circle.

I am making two significant assumptions here :

- The exact sequence of perceptual operations a viewer will undertake to satisfy some goal with the graphic presentation is predictable.
- No additional perceptual operations are needed to deal with looking up values on legends and axes.

The first assumption is perhaps simplistic, particularly when compound objects are employed in the graphic presentation. For example, with the network graph goal just described, it may be that a viewer will search for the circle labeled "New York", determine all other circles it is connected to (i.e., all compound objects it is made part of through perceptual relations), and check to see if any of them are labeled "San Francisco". It is quite conceivable that different viewers may employ different strategies, or that the same viewer may employ different strategies at different times. However, since I believe there is not enough empirical research to always predict the precise strategies a viewer will employ, and since many of the same perceptual operations will be involved no matter which strategy is used, I expect that this assumption will lead to reasonable conclusions in choosing among different graphical languages. Similarly, although the second assumption is also rather substantial, looking up values on axes and legends is not always necessary because mappings between perceptual and data values may be stored in short- or long-term memory.[8] Also, looking up values in a legend or on an axis is likely to involve similar perceptual operations for any perceptual properties (i.e., searching for particular labels in the axis or legend). I thus believe that this assumption will also lead to reasonable conclusions as to the relative merits of different presentations.[9]

Given these assumptions, since the performance of an elementary task will be composed of the performance of the individual operations, a relatively obvious design principle can be stated:

Design principle (choice): *Choose presentations with objects and properties expected to minimize the total time required for the search, discrimination, and perceptual relation determination operations a viewer will execute in satisfying his or her goals* .

For example, based on Mackinlay's rankings of perceptual operations, one would expect that determining the vertical position of an object in figure 6.12.a will be quicker than determining its value in figure 6.12.b, and thus figure 6.12.a will be preferrable. Note that many of the issues discussed in section 6.1 relating to when searches can be efficient should come into play in a full analysis of the time required to complete a sequence of perceptual operations.

[7] No determination of perceptual relationships is necessary here, because information is encoded using atomic rather than compound objects.

[8] Note that the interpretive design principles of the last chapter also guide a system towards choosing presentations which require less consulting of axes and legends.

[9] As noted earlier, both Casner (1991) and Lohse (1993) assume that a more fine-grained analysis of perceptual operations is possible. This type of fine-grained analysis is in no way incompatible with my overall framework.

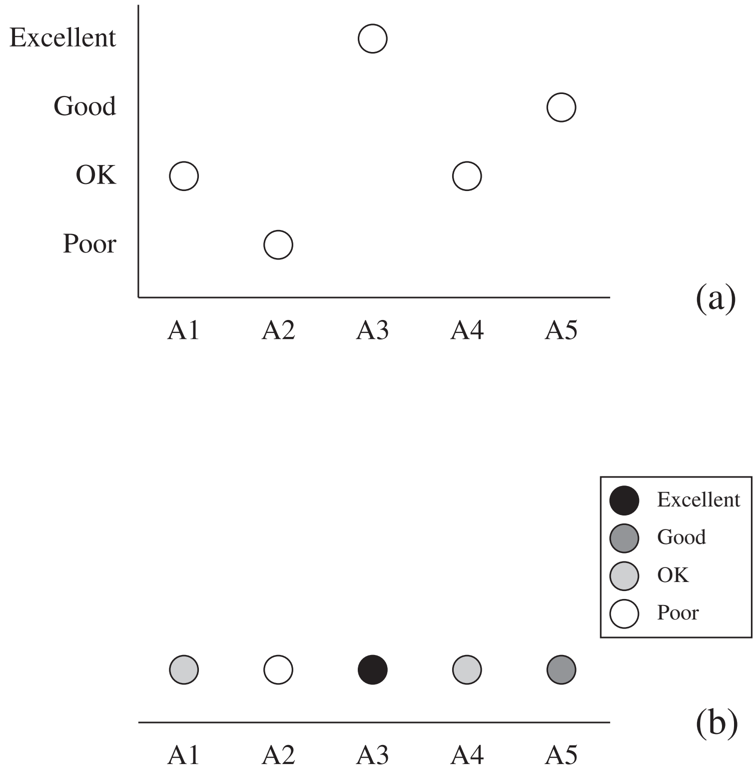


Figure 6.12: (a) A presentation designed to minimize perceptual operation times (b) A less effective presentation of the same data

As discussed in section 6.1.3, accuracy and speed of discrimination (i.e., classification) is based not only on the perceptual property being used to encode information, but also on the discriminability of individual pairs of values for the property. This suggests two other design principles:

Design principle (expressiveness): *Choose properties supporting acceptable degrees of accuracy for the number of distinct values in the domains they are representing.*

Design principle (optimization): *Maximize discrimination of adjacent values when representing ordinal or nominal domains.*

As an example of a violation of the first of these principles, consider figure 6.13, which employs more distinct grayscale values than can be discriminated accurately. Some estimates on the number of values different properties are capable of representing are given in table 6.2, adapted from Casner (1990). With regard to the optimization principle, Weber's law makes this task relatively simple for ordered properties such as length and area. For example, a graph using sizes to display some ordered set of values should use sizes chosen according to Weber's law, i.e., in a geometrically-increasing sequence rather than a linearly-increasing sequence. For color, in addition to maximizing discriminability, the choice of values should take into account other issues affecting the efficiency of visual search (e.g., linear separability). Healey (1996) describes a method for choosing colors for presenting data.

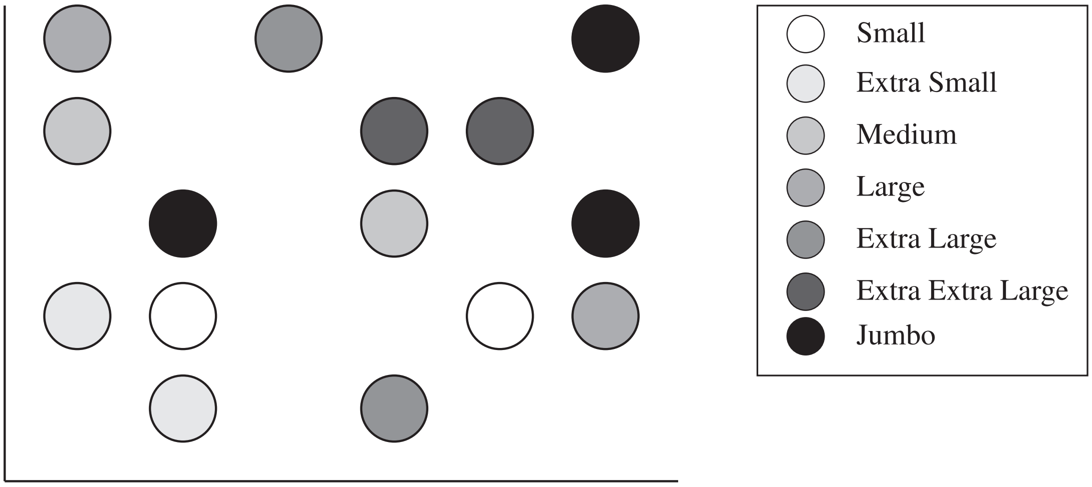


Figure 6.13: Too many grayscale values used for effective discrimination

Label	∞
Horizontal position	100
Vertical position	100
Height	50
Width	50
Line length	50
Color	12
Area	10
Shape	5
Value	4
Line thickness	3
Line dashing	2

Table 6.2: Estimated number of discrete values encodable by different perceptual properties

Finally, since search and classification times for specific perceptual properties are known to be increased by interference from other perceptual dimensions in some cases, and decreased by some sorts of redundancy, it is possible to suggest two more design principles to optimize the effectiveness of a presentation.

Design principle (optimization): *Minimize interference by using uniform values in interfering non-semantic properties.*

Design principle (optimization): *Use integral properties to increase the speed and accuracy with which discriminations of other properties can be performed.*

A principle similar to the first of these was described in section 6.2.1 which discusses the identification phase. The example given there, illustrated by figure 6.10 in which circles of different sizes are placed at arbitrary vertical positions, serves equally well here, assuming varying vertical position interferes to some small degree with the perception of size. As an example of the second principle, it is possible to vary saturation and value together, instead of just one or the other, to increase the discriminability of the values being represented. Because integral dimensions are perceived as one, this type of encoding should not seem redundant. It is worthwhile to note that these two design principles potentially contradict each other. Since the second principle suggests an optimization that may actually improve speed and accuracy, it should be preferred over the first when the two conflict.

Comparative goals

To a large degree, presentations that support fast and accurate satisfaction of elementary goals are likely to support fast and accurate satisfaction of comparative goals as well. Where elementary readings are accomplished through a combination of search, determination of perceptual relations, and discrimination of perceptual function and property values, comparative readings will be accomplished through a combination of search, determination of perceptual relations, and comparisons of perceptual function and property values. In a manner similar to the way times can be minimized for elementary readings for a desired degree of accuracy, times can be minimized for comparative readings as well.

One notable difference exists, however, from the perspective of expressiveness. Certain perceptual dimensions support more complex comparisons than others. For example, hue supports only equality/inequality comparisons. There is no innate perceptual ordering of red, green, blue, etc. Value does support ordering comparisons; if one object is darker than another, a comparison can indicate the larger of the two values. Finally, length supports ratio judgments; one rectangle can be perceived as 50% longer than another. Based on these qualities of perceptual properties, different types of data can be effectively represented. This suggests:

Design Principle (expressiveness): *Use perceptual dimensions supporting the types of comparisons that are part of the viewer's goals.*

All perceptual properties are suitable for nominal data, from a perceptual point of view.[10] Ordered and quantitative properties such as size and value are suitable for ordinal data. Only quantitative properties

[10] Note that from an interpretive point of view, the use of properties supporting ordered comparisons for presenting nominal data may be confusing, as discussed in chapter 5.

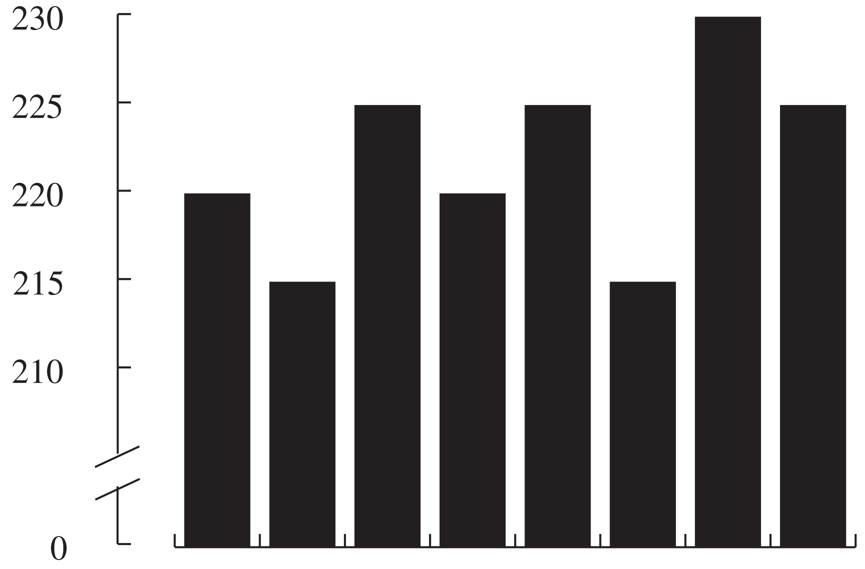


Figure 6.14: A truncated bar graph

(e.g., size and position) are suitable for quantitative data.

I am assuming here that if quantitative data is being presented and a viewer wants to compare two values, they will necessarily want to determine the relative magnitudes of the values. This is not necessarily the case, however; a viewer may be primarily interested only in which value is greater, in which case ordered properties might be suitable for representating quantitative data. As Mackinlay (1986a) points out, a bar graph with a truncated axis (as in figure 6.14) supports ordered but not ratio comparison, making it suitable for some purposes but not for others. Note that truncating an axis is likely to make individual values more discriminable, since the actual difference in length increases, so for some purposes, a bar graph like the one in figure 6.14 is likely to be more effective than a normal bar graph.

Summary goals and configural properties

Summary goals for graphic presentations are goals that require extracting information from large sets of tuples. These goals can include finding trends or correlations between two variables, analyzing the distribution of a single variable, determining if there are clusters in the data, etc. Satisfying summary goals is often one of the primary reasons for producing graphic presentations; in a perceptually effective presentation, these goals can often be satisfied in a single glance.

Consider the bar graph of figure 6.11. In addition to being able to determine the profits for any single year, a viewer can easily determine the overall trend in profits: rising steadily until 1994 and then falling off rapidly. This is possible because the perceptual task of determining the correlation between length and horizontal position for the set of rectangles can be accomplished effectively. Determining such a correlation

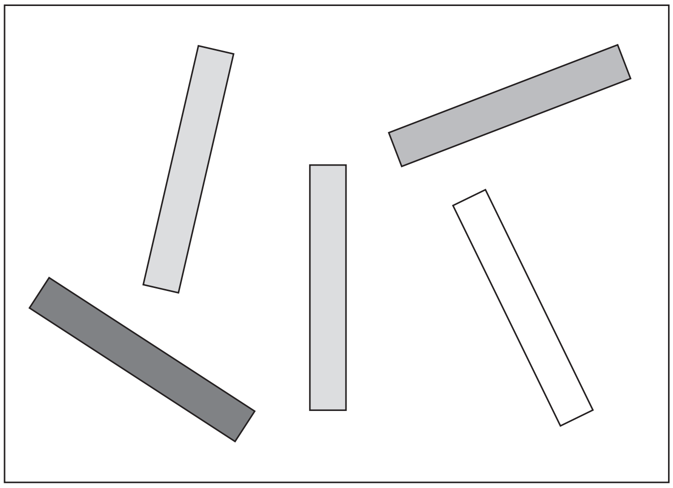


Figure 6.15: A difficult perceptual task (adapted from Pinker (1990))

is not easy for all pairs of object properties, however. As Pinker (1990) has pointed out, a task such as trying to determine the correlation between orientation and grayscale value in figure 6.15 is quite difficult, even though the values and orientations of the bars are easily discriminable. Such perceptually apparent correlations between two or more perceptual properties are examples of configural properties of a graphic presentation. Configural properties are what allow summary goals to be satisfied through simple perceptual operations.[11]

Graphic presentations can support even more sophisticated summary goals through configural properties. For example, the bar graph in figure 6.16.a supports determining the trend in profits by quarter for each year, while the bar graph in figure 6.16.b supports determining the trend in profits over a period of years for any given quarter. Also consider figure 6.17, which allows a viewer to determine the correlation between some domain X and another domain Y for either value of a third domain Z. A graph could also use similar perceptual properties in a different way, as in figure 6.18. Note that in this figure, the way a dependent variable (represented by a combination of value and saturation) varies with two independent variables (represented by horizontal and vertical position) can be perceptually detected.[12]

Several factors seem to be at work here. In all the examples just described, a correlation is perceived between some preattentively perceived property and one or more spatial properties (e.g., horizontal or vertical position, or in the case of figure 6.16, horizontal offset within a constrained horizontal extent).[13] In the examples of figure 6.16 and figure 6.17, indexing (either spatial or based on a preattentive property such as color) is also used.

[11] As far as I know, this observation was first made by Pinker (1990).

[12] Note that this figure is not an ideal illustration, since values and saturations were chosen manually, and probably do not maximize discriminability. In addition, there seem to be effects from the white background which interfere with the perception of values of the circles.

[13] The fact that configural properties are typically defined over space is noted by Pinker (1990).

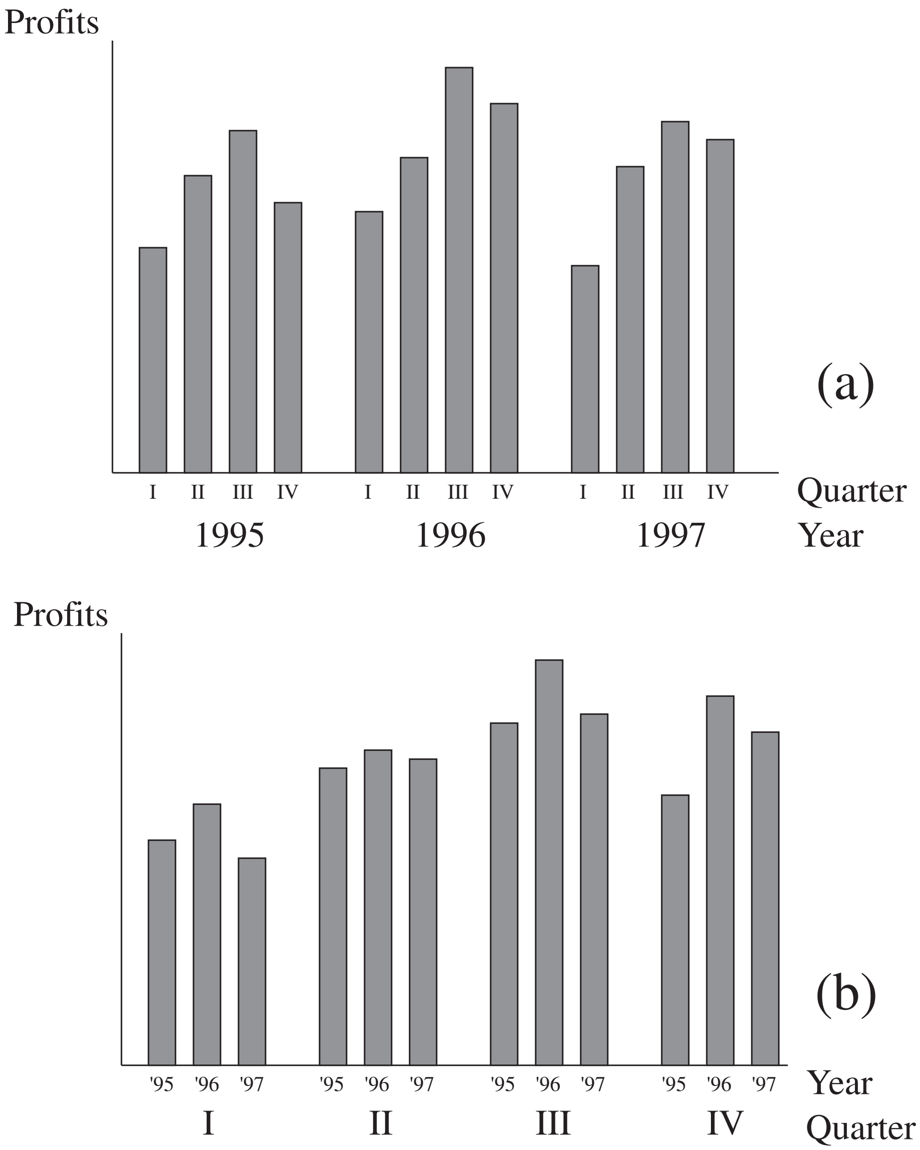


Figure 6.16: Presentations allowing quick determination of (a) quarterly trends for given years (b) trends over years for given quarters

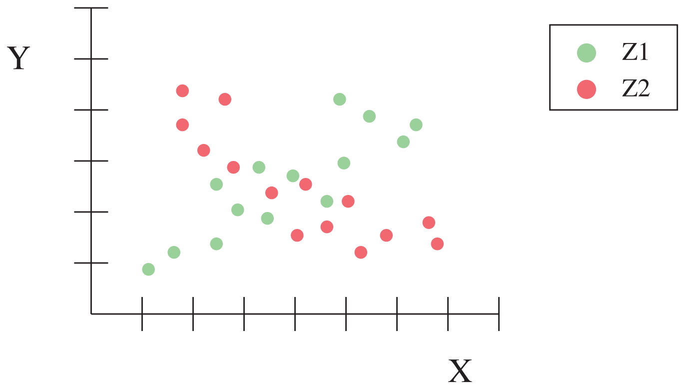


Figure 6.17: A presentation allowing selective trend determination

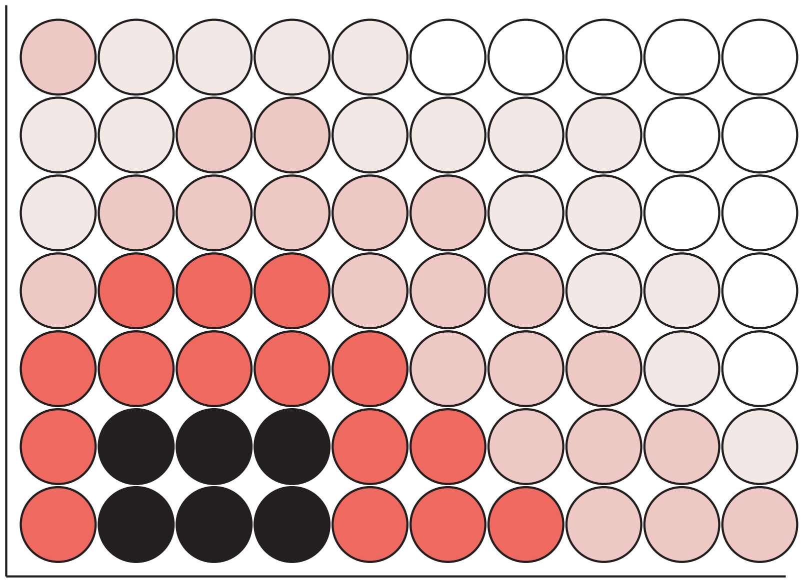


Figure 6.18: Value (and simultaneous saturation) variation over two dimensions

Configural properties of this sort seem to be a function of perceptual organization. Each configural property just described can, in my framework, be considered a property of a compound object encoding an entire dataset— i.e., a number of small elements perceptually organized into a larger "object". In figure 6.18, this larger object is composed of all the circles. In the examples of figures 6.16 and 6.17, the compound object with the configural property is a subset of the objects of the presentation that can be extracted from the rest of the presentation through either spatial or preattentive indexing. That is, the viewer can isolate the compound object by restricting their focus of attention to a certain spatially-contiguous region or "window" of the presentation, or to a set of objects with some particular property value (e.g., a particular color). A number of issues affecting preattentive indexing are thus likely to apply here as well. Interference is possible. Also, since detecting a preattentive property after preattentive indexing seems to be equivalent to conjunctive search, it is only likely to be possible under certain conditions, as described in section 6.1.3.

This analysis can be summarized with the following principle:

Design Principle (expressiveness): *If a user has a summary goal of determining the correlation of* Dependent *vs.* Independent *given* Index*, where* Dependent*,* Independent *and* Index *refer to sets of one or more domains, represent* Dependent *by some spatial or otherwise preattentively perceived property or properties,* Independent *by some spatial property or properties, and* Index *by some property or properties supporting preattentive or spatial indexing.* The spatial property representing *Independent* can only include offset within a constrained extent (e.g., as used in figures 6.16.a and 6.16.b) if the constrained extent encodes a value from *Index*. Also, the properties representing the domains of *Dependent* must be "orthogonal" to the properties representing the domains of *Independent* (i.e., not variations of the same property, such as horizontal position and horizontal position offset or width).

This principle should be considered a working hypothesis or heuristic. At best, it is an approximation of certain conditions under which configural properties will arise; with the "wrong" data values, only a subset of the desired component objects may group together, or the property values of the larger compound object (e.g., the shape of a point cloud in a scatter plot) may be difficult or impossible to interpret. As a heuristic, however, it seems to be a fairly effective, at least for the properties used in the current version of the AUTOGRAPH implementation. Future research may provide more details about precisely when such configural properties arise.

There are undoubtedly many other types of configural properties. In network graphs alone, different layouts can hilight or obscure different structural properties of the data. The graph in figure 6.19.a makes it obvious to a viewer that the links form a cycle, while the graph in figure 6.19.b of the same data makes it clear that the nodes can be divided up into two groups of equal size and that no nodes in these groups link to any others in the same group.[14]

[14] The work on ANDD (Dengler *et al.* 1993; Kosak *et al.* 1994), discussed briefly in chapter 2, is somewhat relevant to the configural properties of network graphs.

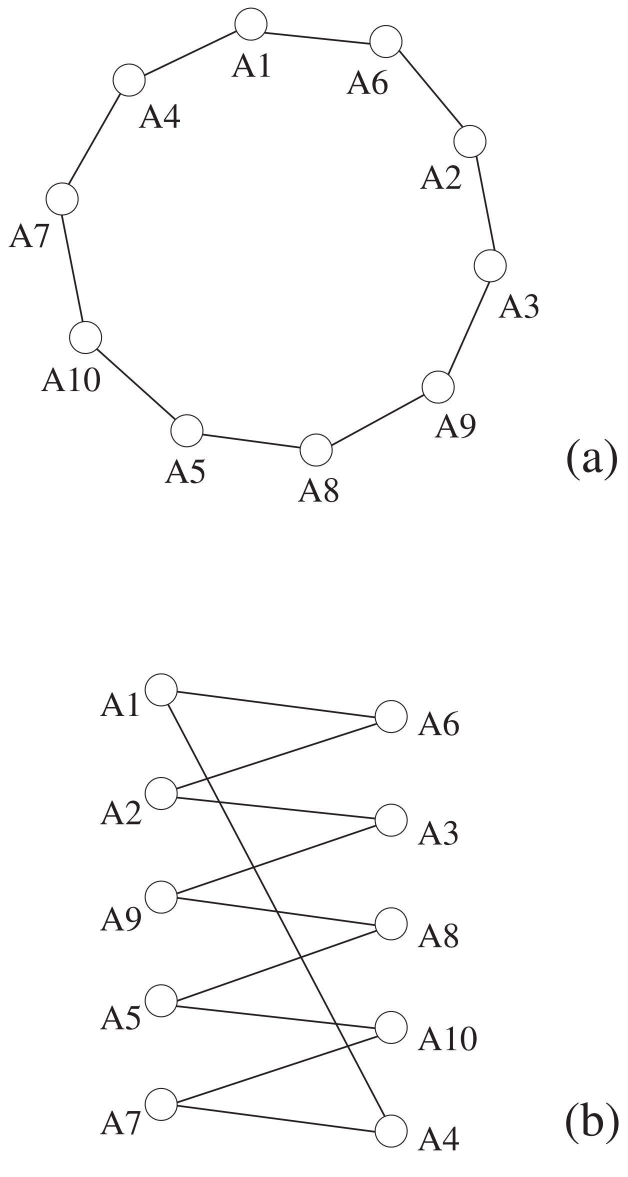


Figure 6.19: Equivalent network graphs with different configural properties

6.3 Summary

This chapter discussed perceptual issues relevant to graphic presentation and set forth number of design principles for ensuring perceptual effectiveness. The general issues discussed included perceptual organization, dimensional structure of visual stimuli, and perceptual operations (including visual search and discrimination). Particular emphasis was given to the last of these topics; some of the individual issues discussed included how preattentive processing affects visual search and some psychophysical laws describing discriminability.

Section 6.2 analyzed how these various perceptual issues could be used to guide the design of graphic presentations. It set forth some design principles for ensuring the perceptual effectiveness of both the identification and the extraction phases of the use of a graphic presentation. Some of the most important design principles included the principle that a presentation should minimize the time needed for viewers to carry out the perceptual operations needed to satisfy their elementary goals, and the specification of conditions under which viewers will be likely to be able to effectively satisfy their summary goals.

Chapter 7

Operationalizing the framework

Previous chapters of this thesis have provided a theoretical framework for describing the space of possible graphical languages for a given dataset, algorithms for determining whether a graphical language is suitable for a dataset and for whether multiple graphical languages can be composed, and design principles for graphic presentations based on issues of perception and interpretation. This chapter describes a prototype implementation that brings all of these pieces together, a program called AUTOGRAPH.

Given the discussions and analyses of previous chapters, many aspects of an implementation are straightforward. A large part of what AUTOGRAPH does is to search over a dynamically constructed space of graphical languages (the space described in chapter 3), then test candidate graphical languages to ensure logical expressiveness and composability (using the algorithms described in chapter 4). These candidates are then ranked using heuristics (based on the design principles described in chapters 5 and 6) and presented to the user. The key issues in the implementation that have not arisen in the discussions thus far are how to operationalize the design principles described in chapters 5 and 6—i.e., how to turn them into heuristics for pruning the search tree and for ranking candidate presentations—and how to use these heuristics and other techniques to manage the size of the search space, while ensuring that good presentations are found. Other issues include the user interface to the system and the construction of a layout engine capable of rendering datasets using any graphical language the system can construct.

Although the discussions of this chapter are couched in terms of the AUTOGRAPH implementation, much of what is discussed pertains more generally to operationalizing the framework presented in this dissertation. That is, some architectural issues and issues of turning perceptual and interpretive design principles into working heuristics are independent of the specific details of the current version of AUTOGRAPH.

In the next section of this chapter, I describe the overall system architecture of AUTOGRAPH and the details of particular modules of the system. I also discuss the heuristics used to prune the search tree, and their relationships with the design principles discussed in chapters 5 and 6. In section 7.2, I discuss a number of examples of presentations designed by AUTOGRAPH and analyze the effectiveness of the program.

7.1 System architecture

The basic architecture of AUTOGRAPH is shown in figure 7.1. It consists of:

- A *graphical user interface* which allows a user to conveniently characterize his or her datasets and goals.
- A *top-level controller* which coordinates the search for graphical languages for multiple datasets.
- A *search module* which finds a list of candidate graphical languages for a single dataset.
- An *expressiveness checking module* which filters the list of candidate languages, passing through only those which are logically expressive for the dataset being presented.
- A *ranking module* which rates and then ranks candidate graphical languages according to their predicted effectiveness.
- A *finalization module* which completes the specification of underdefined languages in a way which maximizes their effectiveness.
- A *layout manager* which uses the graphical languages and datasets to produce onscreen or PostScript renderings of presentations.

These modules are employed as follows:

1. A user specifies the dataset(s) he or she wishes to display, and characterizes them along with his or her goals for the use of the presentation, using the graphical user interface.
2. The dataset description of one of the datasets produced by the user interface is passed to the search module, which heuristically searches for candidate graphical languages for the dataset.
3. The list of candidates is pruned by the expressiveness checking module.
4. For each of the remaining (i.e., non-pruned) graphical languages, steps 2 and 3 are repeated to find compatible graphical languages for the remaining datasets.[1] That is, a new search for languages for the second dataset is performed for each choice of languages for the first dataset, and a search for languages for a third dataset is performed for each choice of languages for the first two datasets, etc.
5. The resulting set of candidate languages are ranked by the ranking module, using heuristics to predict their effectiveness, then more fully specified by the finalization module.
6. The layout manager produces graphic presentations using the different candidate languages and the user's datasets, presenting them to the user in order of their predicted effectiveness.

[1] Recall that composite languages in my framework are merely sets of simple graphical languages determined to be composable.

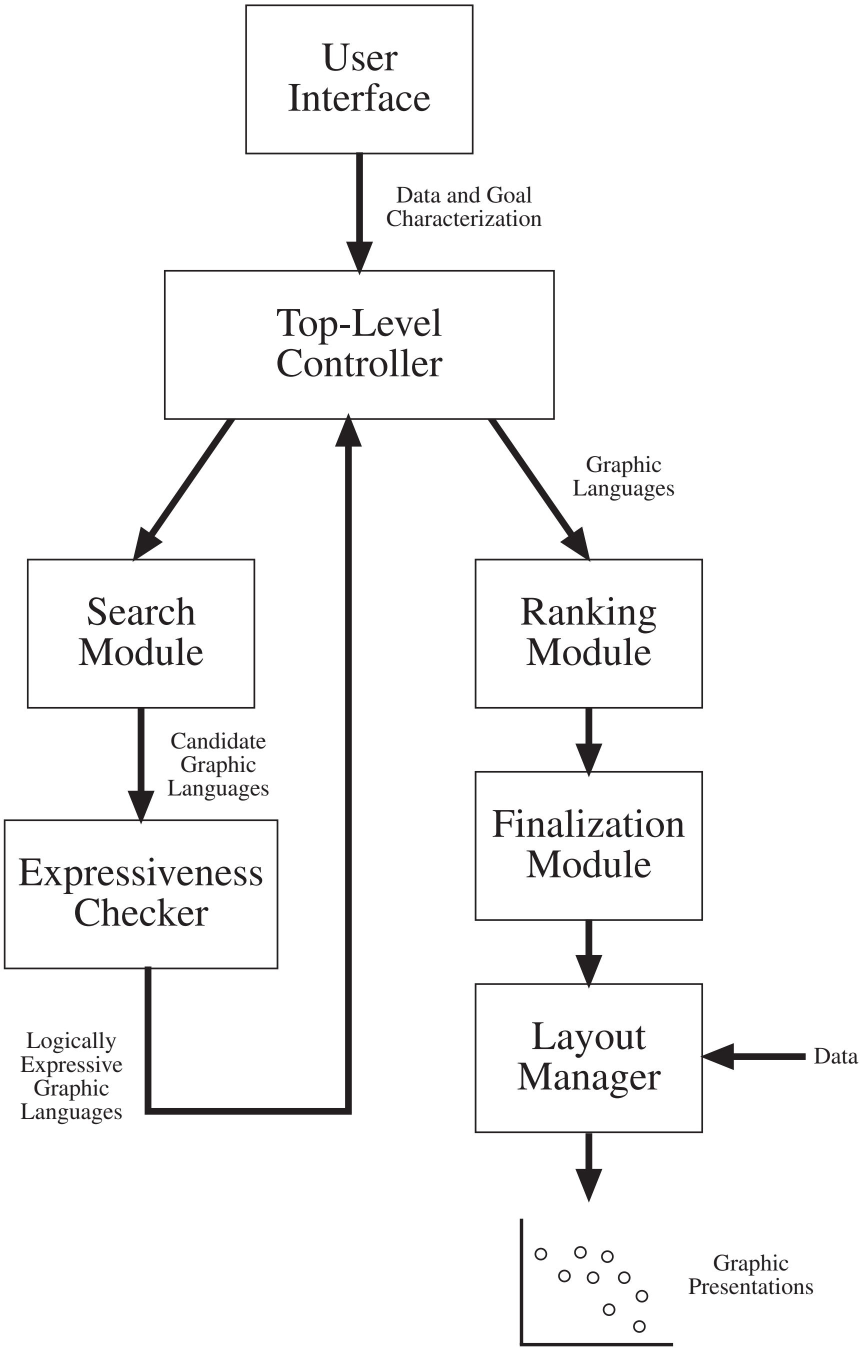


Figure 7.1: AUTOGRAPH architecture

This architecture is implemented in approximately 10,000 lines of code of Allegro Common LISP.

The remainder of this section will discuss the graphical user interface, search module, ranking module, completion module, and layout manager in detail. The top-level controller is straightforward, and the expressiveness-checking module is a fairly direct implementation of the algorithms discussed in chapter 4.

7.1.1 Graphical user interface

The user interface to AUTOGRAPH is somewhat *ad hoc*; its sole purpose is to allow a user to conveniently specify the files containing the datasets they wish to display, characterize the data contained in the files, and characterize the goals he or she has for the presentation. Before discussing the details of the GUI, then, it is important to be specific about the data and goal characterization system that AUTOGRAPH uses. This system is not *ad hoc*, having been designed to capture precisely the distinctions that AUTOGRAPH needs for designing presentations.

A data characterization for AUTOGRAPH consists of two types of specifications: (1) characterizations of individual datasets in terms of the domains data come from, the way values can co-occur, and semantic groupings of the domains of the dataset; (2) detailed characterizations of the domains used in all of the datasets. These types of specifications are generally as described in chapters 3 and 4, and in particular are compatible with the algorithms presented in chapter 4. I will describe both of these aspects of data characterization in AUTOGRAPH, then describe goal characterization.

Dataset characterization

A dataset characterization in AUTOGRAPH specifies, for each dataset:

- *Which domain each value of each tuple will be drawn from.* Characterizing datasets in AUTOGRAPH requires the user to assign different labels to individual domains of the same type, to conveniently distinguish between multiple instances. The types of these domains must also be captured by the characterization. For example, for a dataset describing prices of flights between pairs of cities on a certain airline in the form $< City_1, City_2, Price>$, the first two values of each tuple are drawn from the domain *City* and the third value is drawn from the domain *Price*.
- *Functional dependencies among domains*—i.e., statements which indicate that for each value in certain domains, there will be at most one value of some other domain represented in the set of tuples. For example, assuming there is only one price for each pair of cities in the example just cited, the dataset characterization will specify that *Price* is dependent on $City_1$ and $City_2$.
- *Completeness information about the domains as used in the dataset* —i.e., statements indicating that a tuple exists in the dataset for all possible values of a certain domain, or all possible combinations of values of a set of domains. For example, a dataset may contains prices for flights between every pair of cities from the domain *City*. If so, this must be captured in the characterization, since this information is necessary for determining composability.

- *Other logical relationships among the tuples in the dataset.* Because of the nature of the data they represent, certain datasets may be characterized by different logical properties. For example, some data is symmetric, meaning that a tuple $< x_1, x_2 >$ is equivalent to a tuple $< x_2, x_1 >$. In AUTOGRAPH, relationships like this can be described with implication rules and restriction rules, similar to those discussed in chapter 4. Implication rules specify that certain tuples in a dataset imply that other tuples must be in the dataset as well. Restriction rules specify that certain tuples in a dataset imply that other tuples must *not* be in the dataset.

- *Information about semantic groupings of values in the tuples.* For some datasets that describe relationships over multiple instances of the same types of domains, different instances of the domains are more conceptually related than others. For example, if a dataset contains tuples describing the longitudes and latitudes of start and end positions of troop movements, the longitude and latitude of the start position are more conceptually related than are the longitude of the start position and the latitude of the end position. In AUTOGRAPH, such relationships are described by grouping the related domain instances.

As a complete example, the dataset characterization for the "flights between cities" dataset referred to above might be:

```
(SETQ *DATA-DESCRIPTION* '((CITY1 CITY NIL)
                           (CITY2 CITY NIL)
                           (PRICE1 PRICE (CITY1 CITY2)))
(SETQ *DATA-COMPLETENESS-DESCRIPTION* '((CITY1 CITY2)))
(SETQ *LOGICAL-RULES* '(((((FLIGHT (VAR CITYI)(VAR CITYJ)(VAR PRICE)))
                          ((FLIGHT (VAR CITYJ)(VAR CITYI)(VAR PRICE))))
                        NIL))))
```

The first item, `*DATA-DESCRIPTION*`, labels the domains of the tuple, describes the types of each domain, and specifies the domains, if any, on which each domain value is functionally dependent. The second item, `*DATA-COMPLETENESS-DESCRIPTION*`, specifies any sets of the domains for which the dataset is complete, in this case indicating that the dataset contains a tuple for every possible pair of cities. Finally, `*LOGICAL-RULES*` specifies additional logical relationships in the form of a list of implication rules followed by a list of restriction rules. In this case, there is a single implication rule specifying that a tuple of the form $< city_i, city_j, price >$ is equivalent to a tuple of the form $< city_j, city_i, price >$.[2] No semantic groupings are needed to characterize this dataset.

It will be helpful to clarify the format of logical rules. Implication rules are described as a list of tuple descriptions with variables followed by a list of implied tuple descriptions with the same variables. That is, they specify how certain tuples in a legal dataset imply that other tuples must also be in the dataset. Restriction rules are specified as lists of tuple descriptions with variables followed by a list of disequality conditions. None of the disequality conditions can hold for any sets of distinct tuples in the dataset matching the tuple description. That is, restriction rules specify conditions under which a set of tuples co-occuring

[2]The rule given assumes that the dataset has been labeled `FLIGHT`.

in a dataset will not be legal. For example, in a dataset of parental relationships with tuples of the form $< Father, Mother, Child >$, the restriction rule

```
(((TUPLE40 (VAR FATHER) (VAR MOTHER) (VAR CHILD))
  (TUPLE40 (VAR FATHER) (VAR MOTHER2) (VAR CHILD2)))
 ((!= (VAR MOTHER) (VAR MOTHER2))))
```

could be used to specify that no father in the dataset has two (or more) children with different mothers.

As another simple example of data characterization, if a dataset contained information about the population and dominant industry for a set of countries, and there was one tuple for each country, it could be characterized:

```
(SETQ *DATA-DESCRIPTION* '((COUNTRY1 COUNTRY NIL)
                           (POP1 POPULATION (COUNTRY1))
                           (DOMIND1 INDUSTRY (COUNTRY1))))
(SETQ *DATA-COMPLETENESS-DESCRIPTION* '((COUNTRY1)))
(SETQ *LOGICAL-RULES* NIL)
```

This characterization specifies both the functional dependency of *population* and *dominant industry* on *country*, and that the dataset is complete with respect to the domain *country*.

Domain characterization

The dataset characterizations described above refer to domains, which must also be characterized. The characterization for each data domain consists of:

- A specification of whether the data domain is nominal, ordinal, or quantitative.
- A specification of the complete set of values for a discrete domain (i.e., a nominal or ordinal domain) or the range of values for a continuous (i.e., quantitative) domain.
- A specification of additional features of the domain, consisting of a flag for whether a quantitative or ordered domain is a coordinate (e.g., time of day) or an amount (e.g., profits), flags for whether the domain has inherent horizontal or vertical semantics (e.g., for latitude or longitude), and a flag indicating whether a domain represents temperature or not.

  The set of domain features used by AUTOGRAPH is somewhat arbitrary. Other features can easily be added and used to help make decisions about presentations. This is in keeping with the discussion of chapter 5.

As an example of how domain characterizations work, for the dataset of the prices of flights between cities on a certain airline, the domain characterization specifying the complete set of values for a nominal domain *City* and the range for a quantitative domain *Price* is the following:

```
(SETQ *DATA-DOMAINS*
      '((CITY NOMINAL (PARIS LONDON MILAN AMSTERDAM MADRID))
        (PRICE QUANTITATIVE (200 800))))
```

Goal characterization

In addition to characterizing the data, it is also necessary to characterize a viewer's goals. Goals in AUTOGRAPH may be of two types: elementary (or basic) goals and summary goals, as described in chapter 6. Elementary goals are specified as a set of *search* domains and a set of *lookup* domains; they indicate that a viewer is interested in finding one or more tuples with particular values in the search domains and seeing what the values of the lookup domains are for those tuples. For example, an elementary goal might indicate that a viewer wants to be able to determine a corporation's profits for a given year. Summary goals are specified as three sets of domains: *Independent*, *Dependent*, and *Index*. They indicate that a viewer is interested in seeing how the values of the domains in *Dependent* vary with the domains in *Independent*, for given values of the domains in *Index*. For example, a summary goal could specify that a viewer is interested in seeing how net profits (*Dependent*) vary over a period of years (*Independent*) for a given quarter (*Index*). Coupled with an elementary goal of seeing the profits for a given quarter and year, this would be represented in AUTOGRAPH as:

```
(SETQ *USER-GOALS* '((BASIC (YEAR1 QUART1) (PROF1))
                     (SUMMARY (PROF1) (YEAR1) (QUART1))))
```

The basic goal is expressed with the list of search domains followed by the list of lookup domains. The summary goal is expressed with a list of *Dependent* domains followed by a list of *Independent* domains and then the list of *Index* domains.

Other examples of dataset, domain, and goal characterizations will be given in section 7.2 of this chapter, which presents them along with the presentations AUTOGRAPH creates for them.

Given this framework for data and goal characterization, the user interface is rather straightforward. It consists of a series of dialog boxes allowing the user to specify the relevant information. Some screen shots are shown in figure 7.2 (repeated from chapter 1).

In the current version of AUTOGRAPH, no convenient method exists for entering general logical characterizations of datasets, though they may be entered manually (i.e., as LISP data structures). Future versions might have dialog box options for common types of logical characterizations (e.g., statements about symmetries such as `(city1 city2 price) -> (city2 city1 price)`). They might also employ heuristics to infer logical and other characterizations.[3]

7.1.2 Search module

As described in chapter 3, creating a graphical language for a single dataset can be described as a set of five decisions:

1. **Partitioning**—How to partition the domains of the dataset into groups of domains (i.e., subtuples).

2. **Mapping**—How to represent the groups of domains with objects and properties of those objects.

[3] Currently, the only characterization inferred is the distinction between quantitative and nominal/ordinal domains.

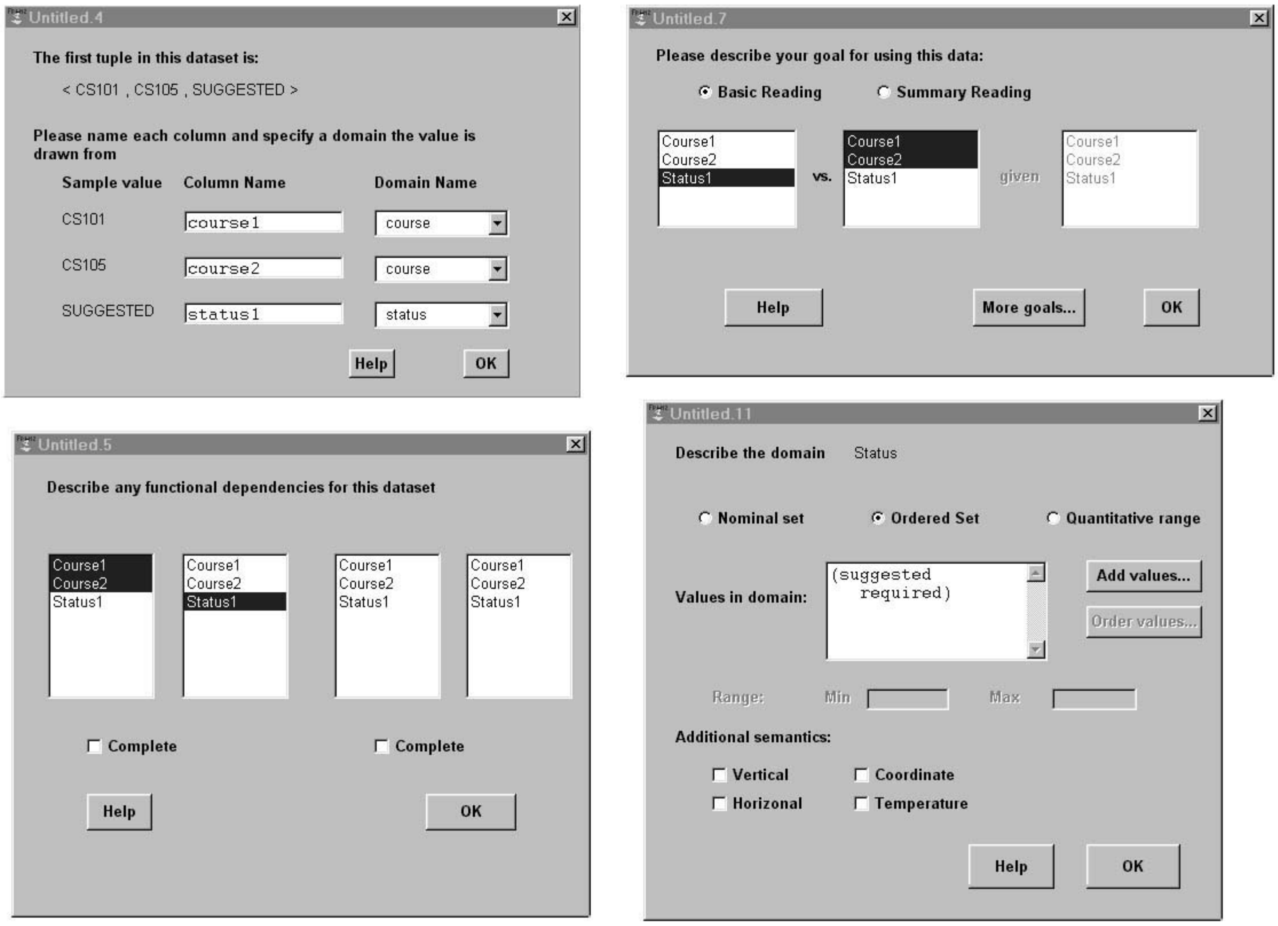


Figure 7.2: Screen shots of the AUTOGRAPH user interface

3. **Topology—**How to arrange the objects for a given tuple to form a compound object (i.e., the topology of the components).

4. **Connection (topology instantiation)—**How to make the topology concrete, using perceptual relations and possibly intermediate (non-semantic) objects.

5. **Sharing—**Which objects are shared and which are non-shared.

AUTOGRAPH considers the first four of these decisions as forming a search tree of four levels. Since there are more limited options for object sharing and since different decisions about object sharing lead to relatively similar presentations, object-sharing possibilities are considered secondary to the other decisions. The search module determines a set of candidate graphical languages for a dataset by exploring a subset of the branches of the search tree at each level and keeping a list of the leaves it reaches. Possible options for object-sharing are stored with the nodes of the search tree.[4]

As analyzed in chapter 3, the number of leaves of this search tree is potentially very large. This is true even with the limited set of objects, perceptual properties, and perceptual relations used in the current version of AUTOGRAPH, shown in table 7.1.[5] The candidate graphical languages are eventually (i.e., after being filtered by the expressiveness checking module) ranked, but it is not reasonable to consider all possible leaves in most cases. Thus, some amount of heuristic filtering (i.e., pruning the search tree) is necessary.

There is a clear trade-off between the expected quality of the best candidate and the space/time used by the search module—a standard problem in heuristic search. If fewer candidate graphic presentations are ranked (i.e., if more branches of the search tree are pruned), it is possible that the most effective one possible will be overlooked, but if fewer branches are pruned, the search will take longer and require more space. It is thus important to choose the amount to prune wisely, and to prune as intelligently as possible. Furthermore, in order to reasonably cover the space (i.e., to consider a wide variety of different presentations), pruning must be distributed evenly among the different levels. Considering only one possible partitioning branch but all the branches at the other levels will lead to a set of graphical languages that is generally more similar than the set that would be found by considering half of all possible partitionings and fewer possibilities for later decisions.

Overall branching in AUTOGRAPH is limited by an easily-changed constant specifying the desired maximum number of candidate graphical languages the search should retrieve. During the search, each branching level calculates a target number branches based on the maximum-number-of-candidates constant, and limits branching to this amount.[6] If multiple datasets are being presented, this number is scaled for datasets after the first one by a factor of $1/k$, where $k = 2$ in the current implementation. That is, AUTOGRAPH will search for n candidate graphical languages for the first dataset, then for n/k candidate graphical

[4] These options are filtered by logical expressiveness checking and by composability checking, so each branch at a certain point in the tree is stored with object-sharing options compatible with decisions about the graphical language made up to that point.

[5] The meanings of most of these properties should be obvious, with the exception of horizontal and vertical position offset. These properties refer to position within a limited spatial extent or group, as in a bar graph showing profits for different quarters, with years segregated into different groups. Note that no perceptual functions are used in the current version of AUTOGRAPH.

[6] This is done differently at different levels. If branches can be ordered, the best n are chosen. Otherwise, choices are made arbitrarily.

```
Objects: Rectangle, Circle, Line, Arrow

Properties:
  Rectangle: horizontal position (hpos), vertical position (vpos),
             width, height, color,
             horizontal position offset (hpos-offset),
             vertical position offset (vpos-offset)
  Circle:    horizontal position (hpos), vertical position (vpos),
             size, color,  horizontal position offset (hpos-offset),
             vertical position offset (vpos-offset)
  Line:      width, color
  Arrow:     width, value, color,
             endpoint1 horizontal position (ep1hpos)
             endpoint1 vertical position (ep1vpos),
             endpoint2 horizontal position (ep2hpos),
             endpoint2 vertical position (ep2vpos)

Perceptual Relations:  touches (line/circle, line/rectangle)
                       head-touches (arrow/circle)
                       tail-touches (arrow/circle)
```

Table 7.1: AUTOGRAPH's objects, properties, and perceptual relations

languages for the second dataset (for each language found for the first dataset), then n/k^2 candiates for the third dataset, etc. This is necessary to reduce combinatorial explosion.

Pruning heuristics

In addition to the general pruning control, specific rules are used at each level to prune the tree. In this section, I discuss these rules at length. I will analyze the effectiveness of these heuristics, and general issues in pruning and branching, in section 7.2 of this chapter.

There are two types of pruning rules used by AUTOGRAPH: rules based on logical issues (e.g., implementing the systematicity assumptions of chapter 3, such as the restriction that each instance of a given type of object must represent the same set of domains in the same way) and heuristics based on the design principles for interpretive and perceptual effectiveness described in chapters 5 and 6.

Of the heuristics, some are obvious implementations of the design principles while others are more approximate. In general, the earlier that heuristics can be applied (i.e., the higher the level in the search tree), the more time and space they will save, because they will eliminate larger numbers of potential graphical languages at a time. However, applying pruning heuristics earlier means that they must make decisions based on incompletely specified graphical languages, which generally makes them more approximate (i.e., more error-prone) and less obviously connected with the design principles they implement. In designing AUTOGRAPH, it was necessary to make some compromises along these lines.

I will summarize the heuristics used in AUTOGRAPH on a level-by-level basis and describe the perceptual and interpretive design principles they implement. I will note the logical rules only in passing, as a convenience, since they are fairly obvious given the framework and algorithms discussed in chapters 3 and 4.

Partitioning level First, when possible partitionings are considered, they are pruned using the following heuristics (expressed as criteria for acceptability):

- Partitionings into more than four groups are eliminated.
- Two subtuples sharing one domain of the same type must be identical— i.e, have only domains of the same types, and the same number of each type of domains. For example, a tuple of the form $< city_i, city_j, \mathit{traffic} >$ cannot be partitioned into $< city_i >$ and $< city_j, \mathit{traffic} >$.
- Each domain in each subtuple in the partitioning must be functionally dependent on the same domains, or each must be dependent on domains in the subtuple or not dependent on any domains. For example, for a dataset of traffic between two computers on a network, a partitioning could not place the traffic value with only one of the computers, since it is functionally dependent on the values of both.
- Grouping of domains into subtuples must not conflict with the semantic groupings described in the dataset characterization. Specifically, a conflict occurs when three conditions are met: (1) some domain in a subtuple is in a semantic group and (2) some domain in that semantic group is not in the subtuple and (3) some domain in the subtuple is not in the semantic group

The first of these heuristics implements the interpretive principle of simplicity—too many groups will lead to too many types of components or overly complex compound objects. The second heuristic implements the interpretive principle of consistency. The main function of the other two heuristics is to ensure that the physical structure of the presentation does not conflict with the conceptual structure of the data—i.e., they implement the interpretive principle of congruence.[7]

Mapping level At the mapping level, for each candidate partitioning, object types are chosen to represent subtuples, and perceptual properties of those object types are chosen to represent individual domains of the subtuples. A number of logical rules come into play here. First, each object type is allowed to represent only one set of domains, and each property of each object is allowed to represent only a single domain. These restrictions are based on the systematicity assumptions of chapter 3. Second, some perceptual properties can only be defined relative to other properties (e.g., horizontal or vertical offset within a horizontal/vertical extent cannot be defined unless horizontal/vertical position is being used to represent a discrete domain), and mappings not respecting these restrictions are removed.

In the current version of AUTOGRAPH, a very strict check is applied to the mappings to operationalize the interpretive principle of consistency: two subtuples in a partitioning representing the exact same

[7]The third heuristic listed also tends to lead to more economical presentations, because the number of objects of a given type that will be created is limited by the number of independent domains represented by the properties of that object type, as discussed in chapter 4, section 4.3.1. Thus grouping functionally dependent domains with domains they are not dependent on tends to lead to presentations with more objects than does grouping them with the domains on which they are dependent.

sets of domains must represent the domains in precisely the same ways. This eliminates some graphical languages that might be considered reasonable. For example, it might be possible to represent flights between two cities by lines between circles (cities of origin) and squares (destination cities), thus representing a directed link without using arrows. However, building this heuristic into the generation process reduces the combinatorics of finding mappings for certain partitionings.

Several heuristics are used to reduce these candidate mappings even further :

- A property must be appropriate to the domain it is representing.

  This heuristic incorporates several checks under the rubric of ‘appropriate’, all making use of information stored about the individual perceptual properties used by the system:

  - A property must support the judgments required by the domain type (nominal/ordinal/quantitative), as described in chapter 6. For example, hue cannot generally represent quantitative values. Also, a property must permit adequate discrimination given the number of values of the domain. These checks implement the perceptual design principle that properties must support adequate discrimination of values. Note that this heuristic assumes that the user’s goals are likely to involve as accurate comparisons of values as possible.[8]
  - A property must not suggest inappropriate inferences about domain type. Currently, this restricts the properties used to represent nominal domains, since certain perceptually-ordered properties (e.g., size) imply an order to the domain values they represent.
  - A property must have appropriate semantic associations. Currently, this imposes several restrictions. A domain specified as a coordinate (e.g., time of day) cannot be encoded by a property encoding an ‘amount’ (e.g., the height of a bar). A property with the semantics ‘horizontal’ (‘vertical’) must be encoded by horizontal (vertical) position, or some variant such as horizontal (vertical) offset. This check and the previous one implement the interpretive design principle of compatibility. Clearly, more sophisticated data characterization might allow for extensions to this heuristic.

- The mapping for each subtuple must be ‘consistent’. More specifically, domains of the same type must be represented by ‘similiar’ properties— i.e., properties mapping to similar perceptual dimensions such as the horizontal and vertical positions of an object, the height and width of a rectangle, horizontal position and horizontal position offset, etc. Lists of such properties are stored in a table, having been determined in advance. Also, equivalent properties of an object (such as the two horizontal endpoint positions of a line) must be used to represent the same domain type.[9]

[8] An exception to this heuristic is made for mapping temperature to colors ranging from saturated blues through gray to saturated reds, as in the presentation of Napoleon’s Russian campaign discussed in chapter 1 and also later in this chapter. Although this mapping is not very perceptually effective, it is permitted as a special case.

[9] An additional consistency check is performed for arrows as well— if the horizontal or vertical position of one endpoint is used to encode information, both the horizontal and vertical positions of both endpoints must encode information. This prevents certain reasonable uses of arrows—e.g., arrows that represent information with only the horizontal positions of each endpoint, and which are constrained to be horizontal. However, it prunes a large number of languages that are difficult to interpret.

Furthermore, each pair of subtuples must be consistent—i.e., instances of the same domain type must be represented with similar properties and equivalent properties (e.g., the horizontal position of a circle and the horizontal position of a square) must represent the same domain type. This latter restriction is suspended for labels; for example, the label of a rectangle and that of a circle are allowed to encode completely different domains.

These checks implement the interpretive design principle of consistency.

- Semantic grouping of domains must not conflict with grouping of object properties. This is analogous to the semantic grouping heuristic for partitionings, and implements the interpretive principle of congruence. At the moment, only the horizontal and vertical positions of the endpoints of lines and arrows are considered grouped. For example, if a dataset contains information about starting and ending longitudes and latitudes of troop movements, the starting latitude and ending longitude must not be represented by horizontal and vertical position of one endpoint of an arrow with the ending latitude and starting longitude represented by the other endpoint. Starting longitude and latitude are semantically grouped, as are ending longitude and latitude, and so each group should be represented by a single endpoint position, if endpoint positions encode information.

If the mapping is being chosen for a second or later graphical language that must be composed with a previously chosen graphical language, AUTOGRAPH also checks that the languages are composable and consistent with each other. The composability check is as described in chapter 4. Since this analysis is performed differently for shared and non-shared objects, this check actually prunes the set of object-sharing possibilities for different languages, and prunes languages with no valid object-sharing possibilities. To check consistency, AUTOGRAPH uses heuristics similar to the ones just described, only comparing the newly composed subtuples and sets of object properties.

In addition to these checks, potential mappings are pruned based on their expected perceptual effectiveness, in order to limit branching to the desired amount. This is done in the following manner:

1. Each possible mapping for each subtuple is rated by summing the 'expected quality' values for each object property, shown in table 7.2.[10]

2. The mappings for each subtuple are sorted in order of expected effectiveness.

3. Possible mappings for the set of subtuples are ranked based on the sum of the ranks of the individual subtuple mappings in their respective lists. That is, the mapping consisting of the best mappings for each subtuple is ranked first, followed by the mappings with any single subtuple's second-best mapping, followed by the mappings with one subtuple's third-best mapping and those with two subtuples' second-best mappings, etc. If there are multiple subtuples with the same types of domains, and thus the same representations, only one of the subtuples of each type is considered in this ranking.

[10] These are similar to, but not identical with, the values eventually used to rank candidate presentations. Ranking is discussed in section 7.1.4.

circle	hpos	1
	vpos	1
	hpos-offset	3
	vpos-offset	3
	color	3
	label	4
	size	4
rect	hpos	2
	vpos	2
	hpos-offset	3
	vpos-offset	3
	width	3
	height	3
	color	3
	label	4
line	width	3
	color	6
arrow	width	3
	color	6
	value	7
	ep1hpos	25
	ep2hpos	25
	ep1vpos	25
	ep2vpos	25

Table 7.2: “Expected effectiveness” of individual object properties

This heuristic is not as precise as later ranking heuristics—it doesn't take goals into account at all, for example, and thus does not analyze the different perceptual operations that will be used to extract information from a presentation—but it is quick and seems to be reasonably effective at pruning the worst graphical languages (according to later ranking heuristics) while keeping the best. It also preserves diversity in the mappings being considered, by ensuring that multiple mappings are considered for each subtuple, rather than considering many mappings for one subtuple and only a few for other subtuples. This is important because the most effective mappings at this stage may not be composable with effective mappings found later (i.e., for other datasets).

Topology level Topologies—i.e., schematic arrangements of component objects for forming compound objects—are an important determiner of when two languages will be composable. Pruning at the topology stages eliminates different object-sharing variations incompatible with each potential topology, and eliminates topologies not compatible with any object-sharing variations, based on checking composability as described in chapter 4.

Connection level Given a choice of partitioning, correspondence, and topology, the search module next looks for possible instantiations of the topology, using perceptual relations and possibly intermediate (i.e., non-semantic) objects. For example, given a topology of one object connected to another, AUTOGRAPH will make a line touch a circle or rectangle. If two circles are to be connected to each other, AUTOGRAPH will introduce a line or arrow touching each of them.[11] Of course, intermediate objects are not allowed to be of the same types as semantic objects—this would violate one of the systematicity assumptions given in chapter 3. No heuristics are currently used to prune the instantiations of the topology.

All of the pruning heuristics discussed thus far operate on incompletely specified graphical languages. One additional heuristic is used to filter out completely specified, composite graphical languages as well as incomplete ones, because it cannot be fully applied to incomplete languages. This is a heuristic used to filter out a set of particularly confusing presentations— presentations that are inconsistent in their use of spatial semantics. Earlier checks ensure that objects and properties are not used inconsistently, but they do not prevent cases where certain objects have spatial semantics and others do not, or where the spatial semantics of different objects are quite different. For example, in a scatter plot where the horizontal and vertical positions of circles encode information, rectangles whose horizontal positions and colors encode information would likely be confusing, if their vertical positions were meaningless. Spatial inconsistency checking eliminates languages like these. While there are some situations in which they might be reasonable, in the absence of more sophisticated layout heuristics they are more often likely to lead to difficult-to-interpret presentations.

The heuristic used to filter spatially incompatible graphical languages requires that all objects in the presentation must either have no spatial semantics (as in a standard network graph) or have identical spatial

[11] Making the two circles touch is not an option in the current version of AUTOGRAPH—i.e., the perceptual relation *touches* for two circles is not part of the system's vocabulary. In addition, AUTOGRAPH only considers putting single intermediate objects between pairs of semantic objects. This is in keeping with the assumption underlying the quantitative analysis of the space of graphical languages in chapter 3. As a result, putting a line or arrow between a pair of circles is the only way to instantiate the topology.

semantics—i.e., if horizontal or vertical position (or position within a constrained extent) is used to represent some domain for one object, it must be used similarly for other objects. Lines and arrows are exempt from this requirement unless the positions of their endpoints encode information. Note that this requirement is more strict than previous consistency checking, because it doesn't allow spatial properties to encode information for some objects and not for others, which isn't considered inconsistent by the normal consistency-checking heuristic.

Note that this heuristic, which still generally implements the interpretive principle of consistency, cannot be completely applied to an incomplete graphical language, because composing objects may alter their spatial semantics by adding to the set of properties which encode information. However, it can be (and is) used to prune incomplete languages which already have inconsistent spatial semantics (i.e., in which different objects use the same spatial properties to encode different types of information).

7.1.3 Expressiveness checking module

The expressiveness checking module implements the strict expressiveness checking algorithm as it is described in chapter 4. The intuitive expressiveness checking algorithm described in that chapter, however, is implemented only in a limited way. AUTOGRAPH checks for graphical language descriptions in which an object will be overconstrained because too many of its properties are being used to represent information, and for graphical languages in which an object will be overconstrained by its perceptual relations with other objects. No check is made for objects overconstrained due to a combination of perceptual relations and properties. Such a check is not necessary given the limited set of objects, properties, and perceptual relations used in the current version of the system.

In addition, no check for potential compound objects with too few distinct components (the first part of the intuitive expressiveness checking algorithm described in chapter 4) is made to insure intuitive expressiveness. This limits the ability of the current implementation to make correct decisions about potentially reflexive datasets —i.e., datasets which may have tuples containing multiple identical domain values. This is not a theoretical or architectural limitation, however; the check could be added in the future.

7.1.4 Ranking module

Having found a set of logically expressive graphical languages for the dataset or datasets, AUTOGRAPH ranks them. The rankings consist of a crude implementation of the design principles from chapters 5 and 6. Each (possibly composite) graphical language is assigned a rating and the languages are sorted based on this rating, to determine the order in which they will be presented to the user.

The ratings are based on the calculation:

$$R(P, G_E, G_S) = k_1 \sum_{G_E} POP(P, G_i) + k_2 \sum_{G_S} CP(P, G_j) + C(P) + \mathit{EXTRA}(P)$$

where P refers to a presentation, G_E to a user's set of elementary goals, G_S to a user's set of summary goals, $POP(P, G_i)$ to a function estimating the time required for a viewer to complete each elementary goal using a

Object	Property	Search time
circle	hpos	20
circle	vpos	20
rect	hpos	20
rect	vpos	20
circle	label	30
circle	color	30
rect	label	30
rect	color	30
arrow	ep1hpos	32
arrow	ep1vpos	32
arrow	ep2hpos	32
arrow	ep2vpos	32
line	color	35
arrow	color	35
arrow	value	40
circle	hpos-offset	50
circle	vpos-offset	50
rect	hpos-offset	50
rect	vpos-offset	50
rect	height	200
rect	width	200
circle	size	200
line	width	200
arrow	width	200

Table 7.3: Perceptual property relative search times

given presentation, $CP(P, G_i)$ to a function estimating whether a summary goal can be completed effectively using a given presentation, $C(P)$ to a function rating the visual "clutter" in a presentation, *EXTRA*(P) to a function adjusting the ranking of the presentation slightly for any additional factors, and where k_1 and k_2 are constants (with $k_1 = 1$ and $k_2 = 1000$ in the current implementation).

The nature of each of the terms is as follows:

$POP(P, G_i)$ is a rating of the effectiveness of the **p**erceptual **op**erations needed to fulfill some elementary goal G_i. It is intended to be a rough estimate of the relative times needed to carry out the operations needed to complete this goal.

In its current form, AUTOGRAPH divides perceptual operations into *search* operations and *lookup* operations, which correspond to the search and lookup domains of an elementary goal. That is, an elementary goal is satisfied by searching for an object with property values corresponding to domain values of the search domains and determining the property values corresponding to the domain values of the lookup domains. For example, to find a corporation's profits for a given year in a bar graph of such information, a viewer must search for the rectangle located at the horizontal position representing the appropriate year, then determine the height of the rectangle, which encodes the profits. Each perceptual operation is assigned an estimated relative time based on the tables shown in tables 7.3, 7.4 and 7.5. The absolute numbers in these tables are

Object	Property	Quantitative lookup time
rect	height (aligned)	20
rect	width (aligned)	20
circle	hpos	20
circle	vpos	20
rect	hpos	20
rect	vpos	20
circle	label	30
rect	label	30
circle	color	30
rect	color	30
arrow	ep1hpos	32
arrow	ep1vpos	32
arrow	ep2hpos	32
arrow	ep2vpos	32
line	color	35
arrow	color	35
line	width	40
arrow	value	40
arrow	width	40
rect	height	50
rect	width	50
circle	size	70
circle	hpos-offset	100
circle	vpos-offset	100
rect	hpos-offset	100
rect	vpos-offset	100

Table 7.4: Perceptual property relative lookup times for quantitative encodings

Object	Property	Discrete lookup time
rect	height (aligned)	20
rect	width (aligned)	20
rect	hpos	20
rect	vpos	20
circle	color	20
rect	color	20
circle	hpos	20
circle	vpos	20
circle	label	30
rect	label	30
rect	height	30
rect	width	30
circle	size	30
line	color	30
arrow	color	30
line	width	40
arrow	width	40
arrow	value	40
circle	hpos-offset	50
circle	vpos-offset	50
rect	hpos-offset	50
rect	vpos-offset	50
arrow	ep1hpos	60
arrow	ep1vpos	60
arrow	ep2hpos	60
arrow	ep2vpos	60

Table 7.5: Perceptual property relative lookup times for discrete encodings

not meaningful; only the relative values are important.[12] Note that AUTOGRAPH takes into account whether certain layout optimizations (e.g., alignment of rectangles) might affect the time required to carry out certain operations.[13] Also note that continuous and discrete domains receive different ratings, because they require different levels of accuracy in discriminating perceptual values.[14] These values are summed to yield a total estimate of the time required to complete a basic task, and the values for each goal are added together to determine a final rating

In the current version of AUTOGRAPH, the time needed to determine perceptual relations is ignored. One justification for this is that the perceptual relations used in the current version of AUTOGRAPH (lines and arrows touching circles and rectangles) are probably determined in a bottom-up manner, prior to a viewer extracting information from a presentation. As a result, determining particular perceptual relations during the extraction phase should be extremely quick. Additional psychological research might suggest ways of computing time values for these operations, however.[15]

$CP(P, G_S)$ is a determination of whether a summary goal can be carried out effectively with a given presentation—i.e., whether there exists a **c**onfigural **p**roperty in the presentation supporting the effective satisfaction of the goal. Since not enough empirical research has been done on determining the relative perceptual salience of different configural properties, this value is boolean—i.e., it is 0 if the summary goal is supported by a configural property and 1 otherwise. Because k_2 is large, this has the effect of downgrading presentations that do not allow the effective satisfaction of summary goals.

AUTOGRAPH considers a summary goal of estimating the correlation of some set of domains *Dependent* with some other set of domains *Independent* given particular values of some third set of domains *Index* to be supported by a configural property if:

- The domains of *Independent* are represented by spatial properties (e.g., horizontal or vertical position or offset, with horizontal (vertical) offset only considered spatial if horizontal (vertical) position represents a domain in *Index*).

- The domains of *Dependent* are represented by preattentive properties.

- The properties representing *Index* can be spatially or preattentively indexed (e.g., horizontal position, vertical position, color).

- The properties representing *Independent* are "orthogonal" to the properties representing *Dependent* (i.e., not variations of the same property, such as horizontal position and horizontal position offset or width).

---

[12] Some anomalies are present in these tables. Notably, color is ranked fairly high for quantitative lookup. However, color can only be used for quantitative lookup in special cases—i.e., where it is being used to encode temperature with a special mapping.

[13] These layout optimizations are normally determined by the finalization module, but are also determined separately for use by the ranking module.

[14] A still more accurate rating scheme might assign times based on the number of domain values of the discrete domains, since these also affect discriminability. Such as scheme would require much more knowledge about perception to make accurate estimates, however.

[15] Overlooking the time needed to determine perceptual relations is partly remedied by the clutter term, $C(P)$, to be described shortly. This term penalizes languages that use more objects (and thus more perceptual relations).

This function is a fairly direct implementation of the heuristic described in the last chapter. Some examples of how it is used will be described shortly.

$C(P)$ is an estimate of the relevant amount of visual "clutter" in a presentation. This takes into account both the simplicity principle for interpretability and the perceptual principle that operations can be carried out more quickly and more accurately if there are fewer distracting stimuli in a presentation. In part, it adjusts for simplifications made in the computations of time needed to execute perceptual operations. It is computed with the following formula, which takes into account the objects that will be created for the presentation and whether the presentation seems to be a constrained network graph for which it is likely to be difficult to produce a clean layout:

$$\begin{aligned} C(P) \quad = \quad & k_3 \mathit{SHARED}(P) + k_4 \mathit{NON\text{-}SHARED}(P) + k_5 \mathit{INTERMEDIATE}(P) + \\ & k_6 \mathit{NET\text{-}CONSTRAIN}(P) + k_7 \mathit{LINETHICK}(P) \end{aligned}$$

In this formula, *SHARED*(P) refers to the number of shared components in the compound objects used in the presentation, *NON-SHARED*(P) refers to the number of non-shared but semantic components, and *INTERMEDIATE*(P) refers to the number of intermediate (i.e., non-semantic) components.[16]. In the current version of AUTOGRAPH, $k_3 = 10$, $k_4 = 20$, and $k_5 = 8$. The rationale for this scheme is that there are likely to be more non-shared objects created than shared objects, for any given class of objects, since they are created on a per tuple rather than a per domain value basis. Intermediate objects, however, are less likely not to be distracting because they tend to be part of perceptually organized (and thus easy to perceive) compound objects.

NET-CONSTRAIN(P) in the above formula refers to a determination of whether the presentation seems to involve a "constrained network graph"—i.e., compound objects involving lines or arrows between two objects that use position to encode information. Such graphs often present layout problems—i.e., they lead to layouts with poor perceptual organization. The value of *NET-CONSTRAIN*(P) is 1 if there are lines or arrows in the presentation between objects with either horizontal or vertical position used to encode information, 2 if both of these properties encode information, and 0 otherwise. In the current version of AUTOGRAPH, $k_6 = 70$.

LINETHICK(P) in the above equation takes on the value of 1 when the thickness of lines or arrows is used to encode quantitative information and these lines or arrows touch circles or rectangles. When using thickness to encode quantitative information, AUTOGRAPH permits the use of very thick lines and arrows, and these tend to obscure the circles or rectangles they touch. In the current version of AUTOGRAPH, $k_7 = 150$.

The *EXTRA*(P) term in the original ranking equation is intended to make small adjustments to the ranking based on other heuristics. Currently, its value is the count of the number of arrows in the compound

[16]The number is computed counting each type of compound object only once, and disregarding merged components. For example, if a normal network graph language using lines between circles is composed with a plot language using the locations of circles, AUTOGRAPH will count two shared circles (from the compound object used in the network graph) and one intermediate line

objects for a presentation that represents information through their endpoint positions, and are guessed to point in the "wrong" direction. The heuristic for determining this is whether the head of an arrow encodes domain values earlier in a tuple than the tail of the arrow. Obviously, this is a very crude heuristic, but it is useful in the absence of a more sophisticated semantic characterization of datasets.

There are number of ways in which this ranking function is unsatisfying from a theoretical point of view. Most obviously, it is heterogeneous, combining multiple types of ratings that are not commensurate. While this is rather unappealing, it also seems to be necessary; all of these issues influence the effectiveness of a presentation. Examining the qualitative behavior of the ranking function provides another justification for this heterogeneous approach. Different factors have different magnitudes. The effects of the configural property term are very large, much larger than any other term. The clutter term $C(P)$ is also rather large in magnitude compared to the perceptual operations term, and the *EXTRA* term is very small compared to all the others. The effects of these magnitude differences are that the larger terms effectively segregate presentations into different classes—e.g., languages that will not satisfy a viewer's summary goals, languages that will satisfy the summary goals but which are likely to be very cluttered and thus problematic, and languages that seem reasonable. Within each of these classes, the smaller magnitude terms ensure that the predicted best language will be chosen. Viewed in another way, the individual terms of the ranking function act like *ceteris paribus* principles; when all other terms are equal, they will rank graphical languages appropriately.

There are other problems with the ranking equation, including the fact that it treats all goals as equally important, which may not be the case, and the fact that it only crudely estimates the perceptual operations that will be used in extracting information from a presentation.

It would be inappropriate to claim that this ranking function is the best one possible, even given the various assumptions that have been made, and it has not been subject to empirical testing, though it could be. It could undoubtedly be improved. Nevertheless, even in its current form, it seems to provide reasonable results for a large range of problems, demonstrating the viability of a first-principles approach. Some of the results of running AUTOGRAPH on different problems are described in section 7.2, which discusses experiences with the implementation.

7.1.5 Finalization module

The graphical languages rated and ranked by AUTOGRAPH are incomplete. They specify mappings of data domains to perceptual properties but fail to specify the precise mappings between particular domain values and particular property values. In addition, they do not specify how the values of the non-semantic properties of the objects in a presentation can be determined. Although the layout manager can set these values using default assumptions, the finalization module ensures that, where possible, perceptual effectiveness is maximized.

The output of the finalization module is a set of directives understood by the layout manager, describing adjustments to be made to a layout produced for a given graphical language. In terms of my framework, these should be seen as a combination of more complete specifications of the mappings of a graphical language (i.e., from domains to perceptual properties) and the extra constraints that may be placed on the objects of graphical language.

The directives take the form of *set constraints* and *hints*. Set constraints are constraints over groups of values. Currently, the set constraints used by AUTOGRAPH consist of the following:

- **Uniform**(*ObjectType*, *Property*, *Value*)—The property values for *Property* for all objects of type *ObjectType* are set to *Value*.

- **Ordered**(*ObjectType*, *Property*)—The property values for *Property* for all objects of type *ObjectType* are ordered similarly to the domain values they represent.

- **EquallySpaced**(*ObjectType*, *Property*)—The property values for *Property* for all objects of type *ObjectType* are perceptually equally spaced.

- **LinearProportional**(*ObjectType*, *Property*)—The property values for *Property* for all objects of type *ObjectType* should be given values linearly proportional to the domain values they represent.

- **TempToColorMapping**(*ObjectType*, *Property*)—The property values for *Property* (which must be the color of an object) for all objects of type *ObjectType* are computed by a special-purpose function treating the domains values being represented as temperatures and using colors ranging from saturated blues through gray to saturated reds.

Hints are more abstract than set constraints, and more *ad hoc*. Although they do not play a large role in the current version of AUTOGRAPH, they include directives like "make all lines wider than usual".

The details of how hints and set constraints are used will be explained shortly; for now I will just describe how the finalization module makes decisions. The finalization module determines set constraints and hints in three steps :

- It places set constraints on property values encoding individual domains, establishing how the mappings of domain values to property values will be constructed. For example, for nominal values assigned to spatial properties (e.g., horizontal position), the values should be equally spaced, and for ordinal domains represented by spatial properties, the values should be equally spaced and ordered in the same way as the domain values they represent. These set constraints are in keeping with the perceptual and interpretive principles discussed earlier in this thesis.

- It attempts to increase the perceptibility of semantic perceptual properties with set constraints and hints. At the moment, this is rather *ad hoc*, consisting of the following rules:

  - For rectangles, if the height is semantic, align the bottom edges if possible with a set constraint. If the width is semantic, align the left edges if possible. Alignment is an important feature of standard bar graphs.

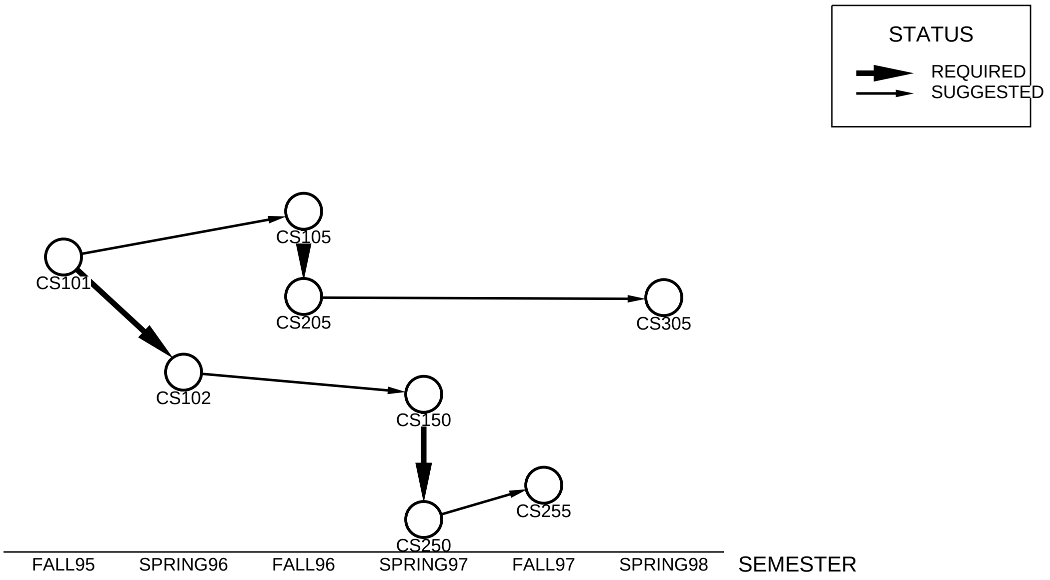


Figure 7.3: A constrained network graph

- For lines and arrows, if the color or value is semantic, make the lines a bit thicker than normal, using a hint.[17]

- When possible, it places set constraints on the non-semantic properties of each object, starting with the most salient properties, making all of their values uniform. This is done only when the objects will not be overconstrained. For example, if each of a set of lines is constrained to touch particular circles, the orientations and lengths of the lines can be constrained to be uniform, but it will not be possible to also constrain the endpoint positions of the lines. To make the determination needed for this step, the finalization module makes use of a crude ranking of the properties in terms of their salience, and a hand-coded table of sets of properties and perceptual relations that can be used together without overconstraining different types of objects.

7.1.6 Layout manager

Once a graphical language has been fully specified, the layout manager uses it to display the user's data. The layout manager must be capable of handling a wide variety of presentations, from relatively determined layouts such as bar graphs to those requiring more sophisticated layout techniques such as network graphs, and also hybrids such as the constrained network of figure 7.3 (repeated from chapter 1). Typically, very different methods are used for laying out such different types of graphs. Since the "different types of

[17] These attempts to increase perceptibility reveal the blurriness of the line between graphical languages and layouts. While my framework allows these types of constraints to be part of a graphical language, it is also possible to think of them as a convenience that helps the layout manager find the most perceptually effective layout for the less-constrained graphical language.

graphs" are not known in advance in my framework, however, AUTOGRAPH uses a relatively flexible system. It treats each layout problem as a set of constraints that must be satisfied, either deterministically or through iterative optimization.

AUTOGRAPH's layout manager sets up a satisfaction/optimization problem as follows:

- Based on the graphical language, it creates a set of data structures for the perceptual objects of the presentation—i.e., arrays of perceptual property values. It also creates a set of data structures for the corresponding graphic objects—arrays of graphic parameter values.

- It sets up a correspondence between the perceptual property values and graphic object parameter values, in the form of functions that compute parameter values from property values and vice-versa. There are a variety of these functions, since different graphical languages may specify different combinations of properties. For example, if the lengths and orientations of a set of lines are semantic, this requires a different method of setting the graphic parameters of the lines than if the horizontal and vertical positions of the endpoints of the lines are semantic. Furthermore, most of the functions are bidirectional, so that perceptual property values can be determined from graphic property values as well.

- Based on the set constraint directives issued by the finalization module, it constructs a set of hard constraints (i.e., constraints that can be satisfied by propagation) determining the values of the perceptual properties from the data domain values, from each other, and from constants specified by the set constraints. The set constraints used by the finalization module lead to simple, deterministic solutions, at least in the current version of AUTOGRAPH.

- Based on individual sets of perceptual relations for individual objects, it sets up hard constraints and optimization forces on the values of the perceptual object properties. Optimization forces are a way of creating "soft constraints", pushing values one way or another based on functions, rather than setting them directly. The layout manager uses them to create the a version of the spring-mass system common to many graph drawing algorithms, first described by Eades (1984).

The layout manager also determines the initialization routines that will set the initial values of the object parameters prior to layout, and also sets up "opportunistic" optimization forces that it determines will help with the layout. For example, it makes all rectangles and/or circles in a presentation slightly repel each other.[18]

Once the constraints and optimization forces have been set up, the actual layout phase begins. This consists of an initialization phase which either initializes property values randomly or sets them to default values, depending on the initialization routines that have been set, followed by an iterative loop of a fixed number of steps during which optimization terms and hard constraints act on the perceptual property values in sequence, eventually producing a layout. At each step in the iteration or only on the final step, depending on a variable setting, the layout manager calls a redefinable set of drawing functions—current sets exist for

[18]These determinations are heuristic, and somewhat *ad hoc*. They rely in part on certain expectations about the types of presentations the layout manager will be used to generate, and are based in part on standard graph-drawing techniques.

both onscreen and PostScript display—to produce a graphic presentation.[19] Axes and legends are drawn appropriately, based on a hand-coded system of rules.

A specific example will help illustrate how the layout manager operates.[20] Consider the constrained network graph shown in figure 7.3, which was discussed in chapter 1. This network graph represents two datasets: a dataset of course prerequisites (either suggested or required) and a dataset describing which semester each course is offered.

Prior to the layout manager being called, the finalization module will have asserted some set constraints on the properties of the objects of the graph:

- The horizontal positions of the circles will be ordered similarly to the semesters they represent, and they will be equally spaced.
- The widths of the arrows will also be ordered, corresponding to the order of the data domains they represent (i.e., *suggested* followed by *required*).

When the layout manager is first called, it will set up the data structures for the graphic objects of the presentation, with arrays for all of the parameters of these objects. For circles, these parameters are: *x*, *y*, *radius*, *red*, *green*, and *blue*—i.e., parameters for the position, size, and color of each circle. For arrows, the parameters are *x1*, *y1*, *x2*, *y2*, *width*, *red*, *green*, and *blue*—i.e., parameters for the endpoint positions, thickness, and color of each arrow. The layout manager will also set up default initialization procedures for each of these parameter values, and mappings between the parameters of the graphic objects and the semantic properties of the corresponding perceptual objects:

- The horizontal position of the perceptual-level circles will be mapped to the *x* parameter of the graphic-level circles.
- The thickness of the perceptual-level arrows will be mapped to the *width* parameter of the graphic-level arrows.

The mappings are one-to-one in this case—i.e., there is a corresponding single perceptual property for each of these parameters—though this is not always true.

Next, the layout manager processes the set constraints, determining functions that will set the values of individual perceptual properties from individual data values:

- The horizontal position of each perceptual-level circle will be set by taking the index of the semester represented by the circle in the ordered list of semesters and multiplying this index by some constant and adding another constant to map the data value into the appropriate range for horizontal position. The constants are chosen based on the range of property values for horizontal position and the number of data values in the domain *Semester*.
- The width of each perceptual-level arrow will be set in a similar fashion.

[19] In addition, the user is permitted to save a PostScript version of any presentation rendered onscreen.

[20] This description will be slightly simplified, for clarity.

After processing the set constraints, the layout manager processes the set of individual perceptual relations—i.e., each arrow *head-touches* some circle and *tail-touches* some other circle. Processing the *sets* of relations, the layout manager determines an initialization routine that will be used to set the positions of the circles—one that positions them randomly. Then, for each individual pair of relations (i.e., the pair of relations for each arrow):

- The layout manager places a set of hard constraints on each graphic-level arrow—functions that set the position of the arrow based on positions of the circles it touches.
- The layout manager sets up optimization forces on the pair of circles that the arrow touches. These will compute changes to the positions of each circle to try to keep them at a specific constant distance from each other. When they are processed, the effects of the distance-equalization forces for each pair of circles joined by an arrow will be computed:

$$\Delta x = \Delta x + k_1 \frac{d_x(d_{xy} - D_{xy})}{d_{xy}}$$

$$\Delta y = \Delta y + k_1 \frac{d_y(d_{xy} - D_{xy})}{d_{xy}}$$

  where Δx and Δy are how much the horizontal position and vertical positions of the circle will be changed at the next update, where d_x and d_y and d_{xy} are the horizontal, vertical, and Euclidean distances between the pair of circles, and where D_{xy} is the optimal distance between the pair of circles.[21]

Finally, by looking at the objects in the presentation, the layout manager sets up some "opportunistic" constraints on the parameters of the objects:

- The positions of all the circles are constrained to stay within the normal rectangle allocated to the presentation—i.e., if they stray from this rectangle they are forced back into it.
- A repulsion force is set up between each pair of circles in the presentation. When they are processed, the effects of the repulsion forces will be computed:

$$\Delta x = \Delta x - k_2 \frac{d_x}{d_{xy}}$$

$$\Delta y = \Delta y - k_2 \frac{d_y}{d_{xy}}$$

After this pre-layout phase has been accomplished, the parameters of all the objects are initialized using the initialization routines that have been determined. All parameters are initially set to constants, in this case, and then the circles are positioned randomly. The layout manager then runs iteratively, processing the optimization forces followed by the hard constraints. More specifically, during each iteration:

[21] The signs of d_x and d_y are opposite for each of a pair of circles. That is, to compute the change for Δx for circle 1 as it is affected by circle 2, $d_x = x_2 - x_1$, while to compute the change to Δx for circle 2, $d_x = x_1 - x_2$.

1. Each circle is moved (horizontally and vertically), based on the repulsion forces between each pair of circles and the distance-equalization forces between pairs of circles connected by arrows.

2. The horizontal position of each circle is set to the proper value by multiplying the index of the semester the course is offered (in the ordered list of semesters) by a constant and adding another constant. The width of each arrow is set to the appropriate value, depending on whether the prerequisite it represents is *suggested* or *required*, in a similar manner. The positions of the circles are clipped to within the presentation rectangle, if necessary. Finally, each arrow is positioned to touch the appropriate pair of circles.

In this way, the layout manager implements a fairly specific (and standard) graph-drawing algorithm in a fairly general way. It is worth noting that standard graph-drawing techniques are typically thought of as heuristics for establishing a number of "aesthetic principles" for graph drawing—e.g., that lines or arrows should tend to be of equal lengths and that connected nodes should be placed near each other. These might best be thought of as principles of perceptual effectiveness; standard graph drawing techniques seem to heuristically guarantee good perceptual organization.

The layout manager in the current version of AUTOGRAPH has a number of limitations, including its fixed number of iterations (which may be many more than necessary for a deterministic layout like a bar graph), and its lack of an explicit energy function for evaluating layouts.[22] Furthermore, although they are fairly general, there is no guarantee that the techniques just summarized can be extended to arbitrary types of constraints. Finally, like most graph-drawing programs, the layout manager is not guaranteed to produce optimal results. Layout is not the focus of this dissertation, however, and given its limitations the layout manager is fairly effective. Some of its strengths and weaknesses will become apparent in the next section of this thesis, which includes a number of presentations created by AUTOGRAPH.

7.2 Examples and experiences

The previous section described the architecture and operation of AUTOGRAPH. This section presents some examples of presentations generated by AUTOGRAPH and describes some experiences with the implementation. In section 7.2.1, I describe the operation of AUTOGRAPH in detail for a specific simple dataset and then for the same dataset paired with another dataset to be presented simultaneously. In section 7.2.2, I present several other examples, demonstrating AUTOGRAPH's sensitivity to various aspects of data and goal characterization. Finally, in section 7.2.3, I analyze more empirically branching and pruning issues in the context of a pair of fairly complex examples.

[22] Graph layout can be seen as minimizing some explicit energy function, allowing layouts to be evaluated during the layout process (Davidson & Harel 1996).

7.2.1 A simple example

As a simple example of how AUTOGRAPH works, consider how it will generate presentations for a simple dataset describing annual profits for some corporation. The dataset might be:

```
1992 200
1993 300
1994 450
1995 700
1996 400
1997 700
```

and the characterization of the data and goals might might be:

```
(SETQ *DATA-DESCRIPTION* '((YR1 YEAR NIL) (PR1 PROFITS (YR1))))
(SETQ *DATA-COMPLETENESS-DESCRIPTION* '((YR1)))
(SETQ *USER-GOALS* '((SUMMARY (PR1) (YR1) NIL) (BASIC (YR1) (PR1))))
(SETQ *DATA-DOMAINS* '((PROFITS QUANTITATIVE (200 700) NIL)
                       (YEAR ORDERED (1992 1993 1994 1995 1996 1997)
                             (COORDINATE))))
```

This characterization represents the dataset as a set of tuples of the form `(YR1 PR1)`—i.e., tuples with values from the domains *year* and *profits*, where `PR1` is functionally dependent on `YR1`, and the set contains a tuple for each year (i.e., `YR1` is complete). It further characterizes the domain *year* as an ordered set of coordinate values (i.e., a year is not an amount) and *profits* as a quantitative range, and characterizes the user's goals as determining how profits vary with year and determining the profits for any individual year.

When AUTOGRAPH is run, it first considers the possible partitionings for the arity-2 tuple `(YR1 PR1)`:

```
All partitionings:
[2](((YR1 P1)) ((P1) (YR1)))
```

Neither of these two partitionings are pruned. The first will lead to the consideration of graphical languages using one object to represent each tuple. This second will lead to the consideration of graphical languages using compound objects with two semantic components to represent each tuple.

For the partitioning grouping both domains together, considering only properties appropriate for ordinal, coordinate domains and for quantitative domains, AUTOGRAPH comes up with the following mappings:

```
(((OBJ-PROP CIRCLE VPOS HPOS))
 ((OBJ-PROP CIRCLE HPOS VPOS))
 ((OBJ-PROP RECT VPOS HPOS))
 ((OBJ-PROP RECT HPOS VPOS))
 ((OBJ-PROP CIRCLE VPOS VPOS-OFFSET))
 ((OBJ-PROP CIRCLE HPOS HPOS-OFFSET))
 ((OBJ-PROP RECT VPOS VPOS-OFFSET))
 ((OBJ-PROP RECT VPOS HEIGHT))
```

```
((OBJ-PROP RECT VPOS WIDTH))
((OBJ-PROP RECT HPOS HPOS-OFFSET))
((OBJ-PROP RECT HPOS HEIGHT))
((OBJ-PROP RECT HPOS WIDTH))
((OBJ-PROP CIRCLE VPOS SIZE))
((OBJ-PROP CIRCLE HPOS SIZE)))
```

The second value in each list is the perceptual object class used to represent the tuples, and the next two values are the perceptual properties of that object class used to represent the *year* and *profits* values of each tuple, respectively.

Since topologies and connections between objects are only applicable to presentations involving compound objects, each of these mappings leads to a single leaf of the search tree. That is, each can be made into a complete graphical language in a deterministic manner (i.e., by describing the mappings in more detail and possibly adding additional constraints on the non-semantic properties of the objects). Many of these mappings will lead to fairly similar presentations. For example, using the horizontal and vertical positions of a circle to represent one tuple is quite similar to using the horizontal and vertical position of a rectangle to represent the tuple, and also similar to the presentation created by switching the horizontal and vertical axes of either of these languages.[23]

For the other partitioning, which represents the two domains of each tuple with separate components of a compound object, AUTOGRAPH generates 36 mappings, then determines that half of them are inconsistent, thus narrowing the set of mappings to 18, as follows:

```
[36](((OBJ-PROP CIRCLE VPOS) (OBJ-PROP CIRCLE VPOS))
     ((OBJ-PROP CIRCLE VPOS) (OBJ-PROP CIRCLE HPOS))
     ((OBJ-PROP CIRCLE HPOS) (OBJ-PROP CIRCLE VPOS))
     ((OBJ-PROP CIRCLE HPOS) (OBJ-PROP CIRCLE HPOS))
     ((OBJ-PROP RECT VPOS) (OBJ-PROP CIRCLE VPOS))
     ((OBJ-PROP CIRCLE VPOS) (OBJ-PROP RECT VPOS))
     ((OBJ-PROP CIRCLE HPOS) (OBJ-PROP RECT VPOS))
     ((OBJ-PROP RECT VPOS) (OBJ-PROP CIRCLE HPOS))
     ((OBJ-PROP CIRCLE VPOS) (OBJ-PROP RECT HPOS))
     ((OBJ-PROP RECT HPOS) (OBJ-PROP CIRCLE VPOS))
     ((OBJ-PROP RECT VPOS) (OBJ-PROP RECT VPOS))
     ((OBJ-PROP CIRCLE HPOS) (OBJ-PROP RECT HPOS))
     ((OBJ-PROP RECT HPOS) (OBJ-PROP CIRCLE HPOS))
     ((OBJ-PROP ARROW WIDTH) (OBJ-PROP CIRCLE VPOS))
     ((OBJ-PROP ARROW WIDTH) (OBJ-PROP CIRCLE HPOS))
     ((OBJ-PROP RECT HPOS) (OBJ-PROP RECT VPOS))
     ((OBJ-PROP RECT VPOS) (OBJ-PROP RECT HPOS))
     ((OBJ-PROP LINE WIDTH) (OBJ-PROP CIRCLE VPOS))
     ((OBJ-PROP ARROW WIDTH) (OBJ-PROP RECT VPOS))
     ((OBJ-PROP LINE WIDTH) (OBJ-PROP CIRCLE HPOS))
     ((OBJ-PROP RECT HPOS) (OBJ-PROP RECT HPOS))
     ((OBJ-PROP RECT HEIGHT) (OBJ-PROP CIRCLE VPOS))
```

[23]This suggests that there should be some more efficient way of describing sets of possible mappings. I will discuss the drawbacks of AUTOGRAPH's "brute force" search and possible alternatives in chapter 8.

```
      ((OBJ-PROP ARROW WIDTH) (OBJ-PROP RECT HPOS))
      ((OBJ-PROP RECT HEIGHT) (OBJ-PROP CIRCLE HPOS))
      ((OBJ-PROP LINE WIDTH) (OBJ-PROP RECT VPOS))
      ((OBJ-PROP RECT WIDTH) (OBJ-PROP CIRCLE VPOS))
      ((OBJ-PROP RECT WIDTH) (OBJ-PROP CIRCLE HPOS))
      ((OBJ-PROP CIRCLE SIZE) (OBJ-PROP CIRCLE VPOS))
      ((OBJ-PROP LINE WIDTH) (OBJ-PROP RECT HPOS))
      ((OBJ-PROP RECT HEIGHT) (OBJ-PROP RECT VPOS))
      ((OBJ-PROP RECT HEIGHT) (OBJ-PROP RECT HPOS))
      ((OBJ-PROP CIRCLE SIZE) (OBJ-PROP CIRCLE HPOS))
      ((OBJ-PROP RECT WIDTH) (OBJ-PROP RECT VPOS))
      ((OBJ-PROP CIRCLE SIZE) (OBJ-PROP RECT VPOS))
      ((OBJ-PROP RECT WIDTH) (OBJ-PROP RECT HPOS))
      ((OBJ-PROP CIRCLE SIZE) (OBJ-PROP RECT HPOS)))
Filtering by inconsistent-corr?:
[18](((OBJ-PROP CIRCLE HPOS) (OBJ-PROP RECT VPOS))
      ((OBJ-PROP RECT VPOS) (OBJ-PROP CIRCLE HPOS))
      ((OBJ-PROP CIRCLE VPOS) (OBJ-PROP RECT HPOS))
      ((OBJ-PROP RECT HPOS) (OBJ-PROP CIRCLE VPOS))
      ((OBJ-PROP ARROW WIDTH) (OBJ-PROP CIRCLE VPOS))
      ((OBJ-PROP ARROW WIDTH) (OBJ-PROP CIRCLE HPOS))
      ((OBJ-PROP LINE WIDTH) (OBJ-PROP CIRCLE VPOS))
      ((OBJ-PROP ARROW WIDTH) (OBJ-PROP RECT VPOS))
      ((OBJ-PROP LINE WIDTH) (OBJ-PROP CIRCLE HPOS))
      ((OBJ-PROP RECT HEIGHT) (OBJ-PROP CIRCLE VPOS))
      ((OBJ-PROP ARROW WIDTH) (OBJ-PROP RECT HPOS))
      ((OBJ-PROP RECT HEIGHT) (OBJ-PROP CIRCLE HPOS))
      ((OBJ-PROP LINE WIDTH) (OBJ-PROP RECT VPOS))
      ((OBJ-PROP RECT WIDTH) (OBJ-PROP CIRCLE VPOS))
      ((OBJ-PROP RECT WIDTH) (OBJ-PROP CIRCLE HPOS))
      ((OBJ-PROP LINE WIDTH) (OBJ-PROP RECT HPOS))
      ((OBJ-PROP CIRCLE SIZE) (OBJ-PROP RECT VPOS))
      ((OBJ-PROP CIRCLE SIZE) (OBJ-PROP RECT HPOS)))
```

Note that AUTOGRAPH filters out all the mappings using the same object to represent each of the two different domains (which violates a systematicity assumption) and also the mappings using comparable properties (e.g., the horizontal position of a circle and horizontal position of a rectangle) to represent different domains. The former are pruned for logical reasons, the latter for heuristic reasons (i.e., the interpretive principle of consistency). Many of the remaining mappings will later be rejected for spatial inconsistency reasons (e.g., `((OBJ-PROP CIRCLE HPOS) (OBJ-PROP RECT VPOS)))`.

For each of these mappings, since only two objects are involved, there can be only one possible topology—i.e., the objects must be connected, one to the other. Of course, AUTOGRAPH makes this determination separately for each mapping.[24]

Considering the first mapping, using this topology, AUTOGRAPH finds the single way of connecting a circle and a rectangle available to the system—creating an intermediate object, a line, that touches

[24] Note that this is somewhat inefficient, but only in that possible topologies must be re-computed many times. This does not account for a significant amount of processing time.

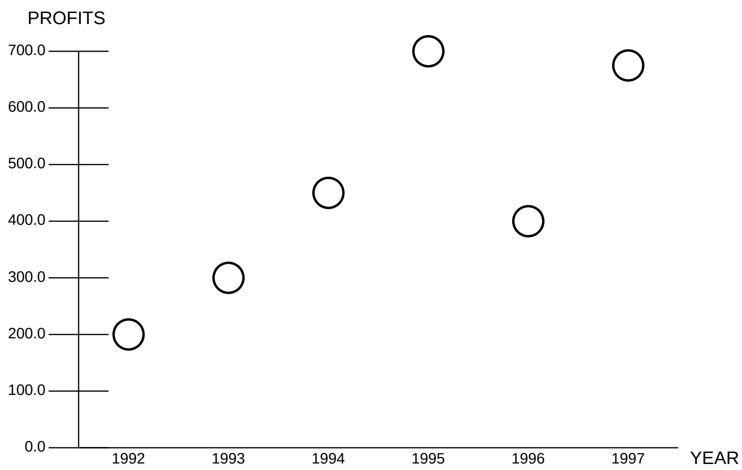


Figure 7.4: A top-ranked presentation

each of them—and determines that the resulting graphical language is logically expressive (i.e., it doesn't get filtered):

```
All connections:
[1]((((INTER-OBJ LINE 202 (1 2)) (TOUCHES 1 202) (TOUCHES 202 2))))
Filtered complete reps from
[1](((TUPLE27 YR1 P1)
      ((P1) (YR1))
      ((OBJ-PROP CIRCLE HPOS) (OBJ-PROP RECT VPOS)
       (INTER-OBJ LINE 202 (1 2)) (TOUCHES 1 202) (TOUCHES 202 2))))
to
[1]((((TUPLE27 YR1 P1)
       ((P1) (YR1))
       ((OBJ-PROP CIRCLE HPOS) (OBJ-PROP RECT VPOS)
        (INTER-OBJ LINE 202 (1 2)) (TOUCHES 1 202) (TOUCHES 202 2))))))
```

In describing the connections, intermediate objects are referred to by unique integers above 100 (e.g., 202), and semantic objects are referred to by their indices in the mappings, with the first object in the mapping numbered 1.

Similar determination of possible connections are made for the other mappings. For some of the mappings, AUTOGRAPH is unable to find any method of connecting the two component objects encoding the domains to create a compound object. For example, AUTOGRAPH has no means of connecting an arrow and a rectangle either directly or using an intermediate object.[25] For some other mappings, there are two ways to connect the components. For example, to connect a circle and an arrow, the head of the arrow can touch the circle, or the tail of the arrow can touch the circle.

[25] Arrows cannot directly touch rectangles in the current version of AUTOGRAPH because the relevant graphic constraint has not been implemented in the layout engine.

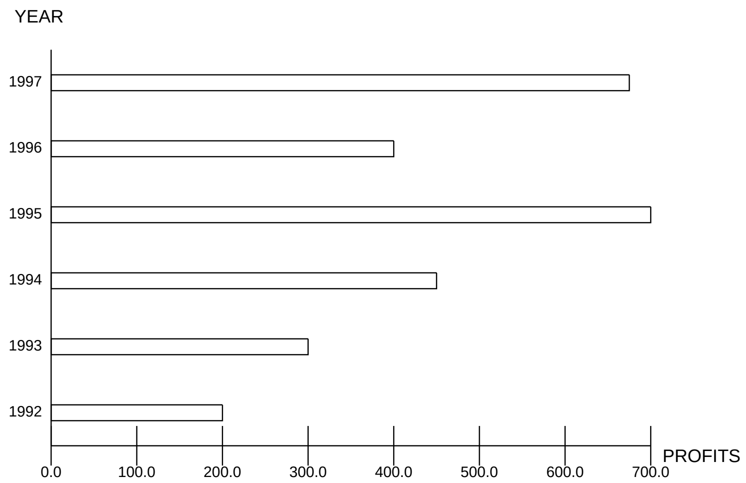


Figure 7.5: Another top-ranked presentation

After collecting all of the graphical languages from both partitionings, AUTOGRAPH ends up with a set of 29 different ones. After rejecting eight of these as spatially inconsistent (e.g., all the languages that use the horizontal position of a circle to represent one thing and the vertical position or width or height of a rectangle to represent another domain), it is left with 21 graphical languages. Many of these are similar, however, as noted above. After ranking the possible presentations, AUTOGRAPH eventually presents the dot plot shown in figure 7.4 as its first choice and the equally ranked bar graph of figure 7.5 as its second choice. After discarding presentations determined to be similar to others that are at least as effective, AUTOGRAPH presents only three other presentations—all of them ranked much lower than the first two because they are determined not to effectively satisfy the summary goal, and to satisfy the basic goal less effectively as well. Figure 7.6 presents these three presentations top to bottom in the order they are ranked by AUTOGRAPH.[26]

Now consider the case where the user wants to present the same dataset of annual profits, but simultaneously present another dataset containing notes about particular years in the corporation's history:

```
1993 "Introduction of XZ300"
1996 "Acquisition of YY Inc."
```

The updated data characterization, including a new basic goal of seeing the note for any given year, would look like:

```
(SETQ *DATA-DESCRIPTION* '((YR2 YEAR NIL) (NOTE1 NOTE (YR2))
                           (YR1 YEAR NIL) (PR1 PROFITS (YR1))))
(SETQ *DATA-COMPLETENESS-DESCRIPTION* '((YR1)))
(SETQ *USER-GOALS* '((BASIC (YR2) (NOTE1))
                     (SUMMARY (PR1) (YR1) NIL)
```

[26] Some minor problems in drawing the axes and legends can be seen in these presentations, particularly the overlapping of the numbers in the axes of the bottom presentation.

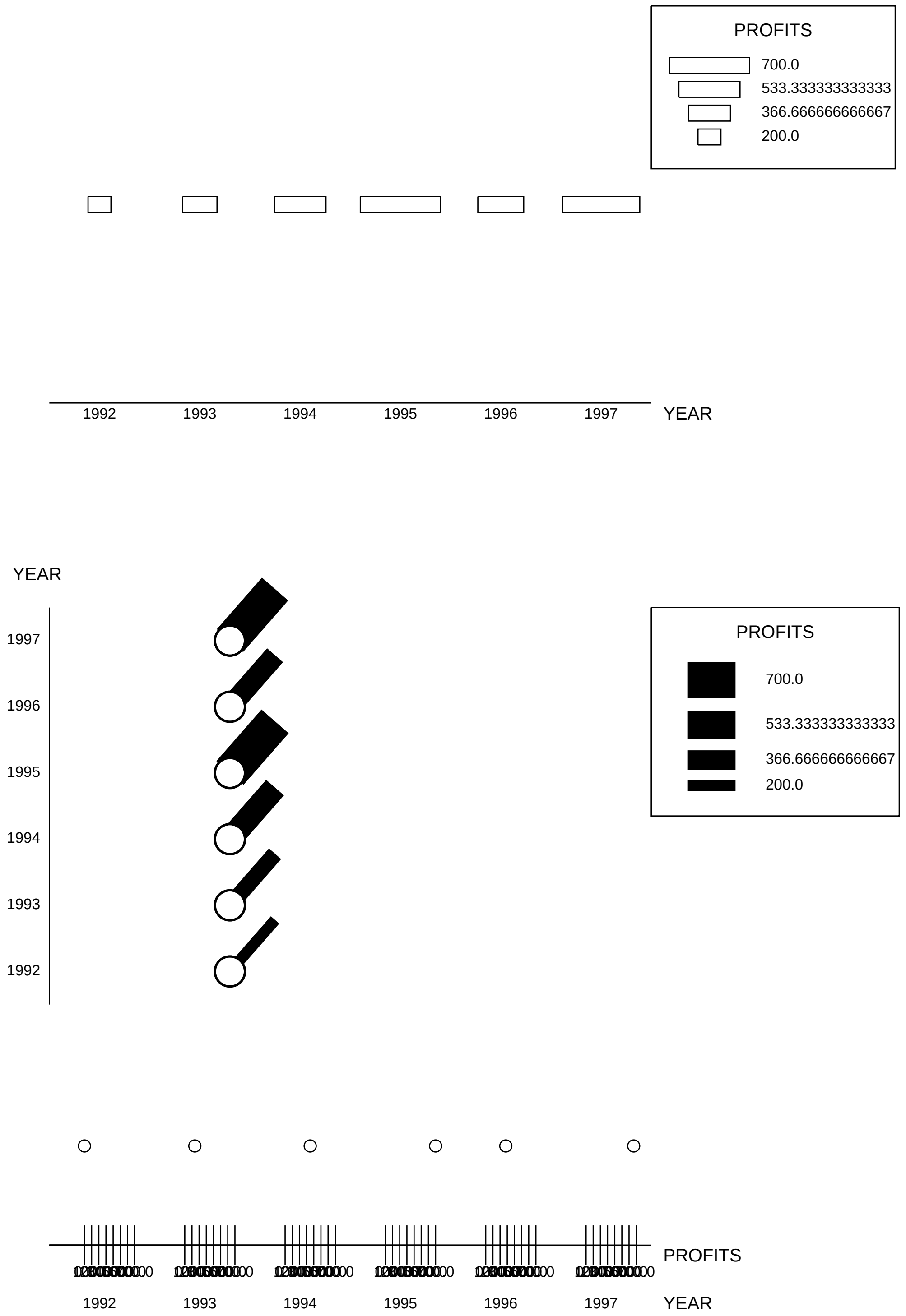


Figure 7.6: Other less effective presentations

```
                     (BASIC (YR1) (PR1))))
(SETQ *DATA-DOMAINS* '((PROFITS QUANTITATIVE (200 700) NIL)
                       (YEAR ORDERED (1992 1993 1994 1995 1996 1997)
                             (COORDINATE))
                       (NOTE NOMINAL ("Introduction of XZ300"
                                      "Acquisition of YY Inc.") NIL)))
```

with additional statements that the datasets to be presented contain tuples of the forms `(YR1 PR1)` and `(YR2 NOTE1)`.

In order to present this pair of datasets, AUTOGRAPH will first find a set of appropriate graphical languages for one of the datasets. Assuming this is the original annual profits dataset, AUTOGRAPH will find the same set of graphical languages it found before.[27] For each of these graphical languages, AUTOGRAPH will then try to find compatible graphical languages for presenting the notes dataset.

Consider this search when the graphical language for the first dataset represents years with the vertical positions and profits with the horizontal positions of a set of circles (i.e., a plot similar to that of figure 7.4, only with the axes flipped). As with the profits dataset, the notes dataset leads to only two possible partitionings: `((YR2 NOTE1))` and `((NOTE1) (YR2))`. For the first partitioning, AUTOGRAPH initially constructs 25 mappings that might potentially be compatible with the choice of graphical languages for the first dataset:

```
(((OBJ-PROP CIRCLE VPOS HPOS))
 ((OBJ-PROP CIRCLE HPOS VPOS))
 ((OBJ-PROP RECT VPOS HPOS))
 ((OBJ-PROP RECT HPOS VPOS))
 ((OBJ-PROP CIRCLE VPOS VPOS-OFFSET))
 ((OBJ-PROP CIRCLE VPOS HPOS-OFFSET))
 ((OBJ-PROP CIRCLE VPOS COLOR))
 ((OBJ-PROP CIRCLE HPOS-OFFSET VPOS))
 ((OBJ-PROP CIRCLE HPOS-OFFSET HPOS))
 ((OBJ-PROP CIRCLE HPOS VPOS-OFFSET))
 ((OBJ-PROP CIRCLE HPOS HPOS-OFFSET))
 ((OBJ-PROP CIRCLE HPOS COLOR))
 ((OBJ-PROP RECT VPOS-OFFSET VPOS))
 ((OBJ-PROP RECT VPOS VPOS-OFFSET))
 ((OBJ-PROP RECT VPOS COLOR))
 ((OBJ-PROP RECT HPOS-OFFSET HPOS))
 ((OBJ-PROP RECT HPOS HPOS-OFFSET))
 ((OBJ-PROP RECT HPOS COLOR))
 ((OBJ-PROP CIRCLE VPOS LABEL))
 ((OBJ-PROP CIRCLE HPOS LABEL))
 ((OBJ-PROP RECT VPOS LABEL))
 ((OBJ-PROP RECT HPOS LABEL))
 ((OBJ-PROP CIRCLE HPOS-OFFSET VPOS-OFFSET))
```

[27] The order of the datasets in searching for graphical languages is determined in the current version of AUTOGRAPH by the order in which the user characterizes the datasets. Future versions may alter this, since ordering may affect the efficiency of the search. Note also that in saying that AUTOGRAPH will find the same graphical languages for the same dataset, there is an assumption that the search space for this particular dataset is small enough that random pruning will not come into play. This is of course dependent on the maximum branching factor, as discussed in section 7.1.2.

```
 ((OBJ-PROP CIRCLE HPOS-OFFSET COLOR))
 ((OBJ-PROP CIRCLE HPOS-OFFSET LABEL)))
```

After checking these mappings more carefully for compatibility with the graphical language chosen for the first dataset (i.e., the language leading to the dot plot of figure 7.4), AUTOGRAPH narrows possible mappings for this second dataset down to only three:

```
(((OBJ-PROP RECT VPOS VPOS-OFFSET))
 ((OBJ-PROP RECT VPOS COLOR))
 ((OBJ-PROP RECT VPOS LABEL)))
```

Some of the pruned mappings violate the consistency heuristic, using horizontal or vertical position to represent a different domain than the previous graphical language. Others are not composable with the previous language. Note that if the second dataset had been characterized as complete for `YR2`—i.e., if there were a note for every year— some mappings using circles would have been considered composable. Making the color of each circle in the original dot plot correspond to a particular note, for instance, would be possible. This is not an option with the current characterization, however, since there would not be a color value for every circle.

When considering compatible mappings for the second dataset given the other partitioning (i.e., `((NOTE1) (YR2))`), AUTOGRAPH initially considers 60 possible mappings, rejects half of these as internally inconsistent, and rejects 24 of the remaining 30 mappings as incompatible with the graphical language for the first dataset. The remaining six mappings (presented with the first object property encoding *note* and the second object property encoding *year*) are:

```
(((OBJ-PROP RECT COLOR) (OBJ-PROP CIRCLE VPOS))
 ((OBJ-PROP RECT LABEL) (OBJ-PROP CIRCLE VPOS))
 ((OBJ-PROP ARROW COLOR) (OBJ-PROP CIRCLE VPOS))
 ((OBJ-PROP LINE COLOR) (OBJ-PROP CIRCLE VPOS))
 ((OBJ-PROP ARROW COLOR) (OBJ-PROP RECT VPOS))
 ((OBJ-PROP LINE COLOR) (OBJ-PROP RECT VPOS)))
```

Note that the mappings that use the vertical position of a circle to represent *year* are compatible with the graphical language for the first dataset (thus far) because they represent *year* in a similar way. The other mappings are compatible (thus far) because they do not use circles at all, and thus cannot be inconsistent.

These mappings lead to the same simple topologies as in the example just discussed, and similar connections (i.e., lines or arrows touch circles, lines touch rectangles, rectangles and circles are connected by non-semantic lines).

When the searches have been completed for each of the graphical languages for presenting the first dataset, AUTOGRAPH ends up with a total of 263 composite languages, 69 of which are spatially consistent and thus reasonable.

The first two composite presentations AUTOGRAPH presents, both equally ranked, are shown in figure 7.7. The next three presentations (filtering out similar languages in each case) are shown in figure 7.8, positioned from top to bottom with the most effective presentation on top and the least effective on the

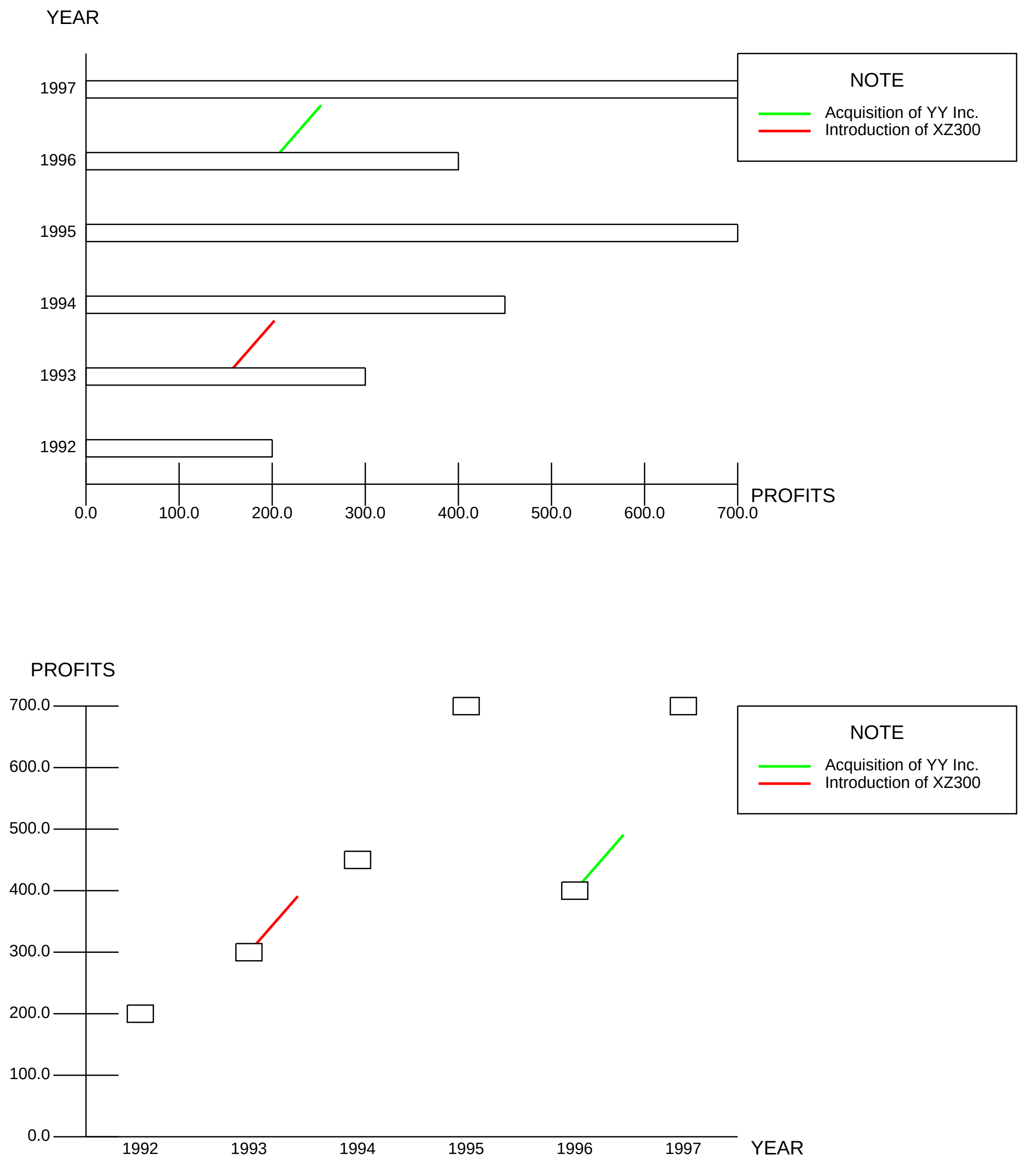


Figure 7.7: A top-ranked presentations for the profits and notes datasets

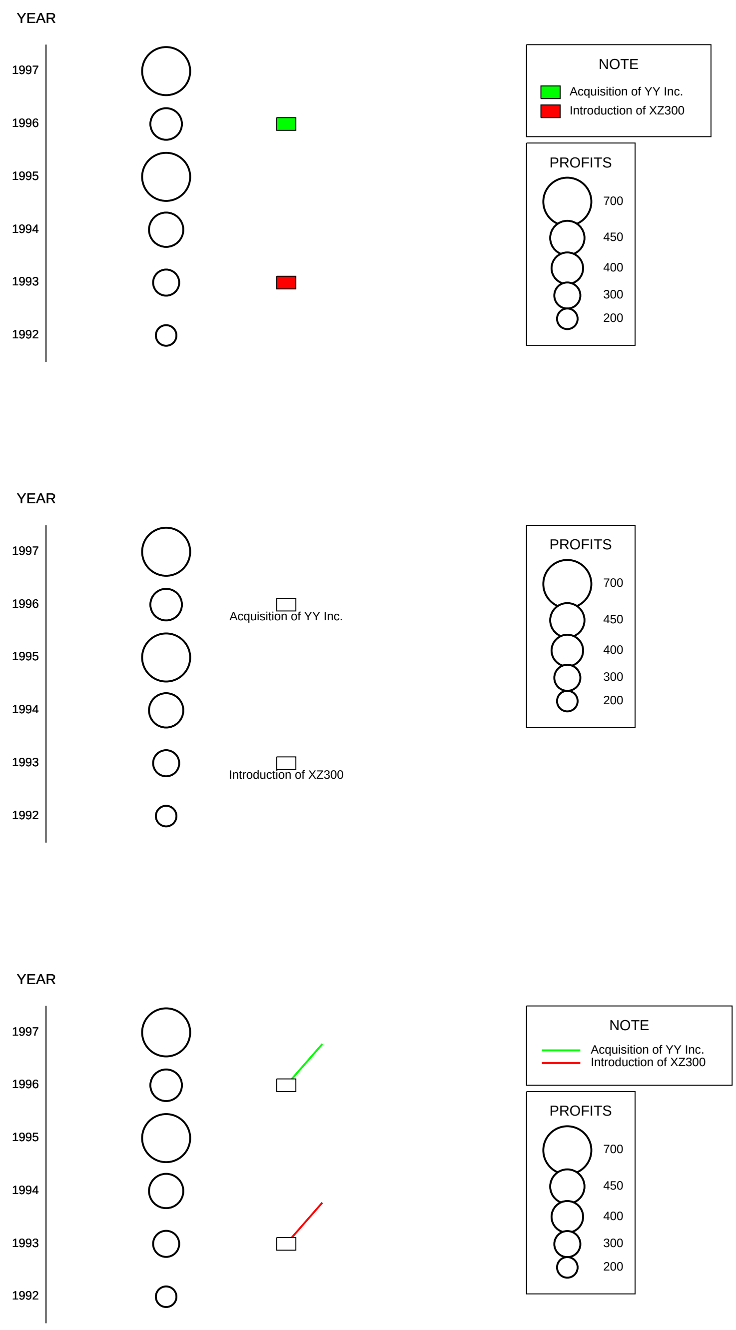


Figure 7.8: Less effective presentations for the profits and notes datasets

bottom. The five other presentations AUTOGRAPH will reveal to the user are shown in figure 7.9. All of these presentations it considers considerably poorer than the others.

It is worth noting that perhaps the most intuitive way of adding notes to particular years, labeling the objects directly, is not available to the current version of AUTOGRAPH, because labels are considered perceptual properties of circles and rectangles, rather than independent objects, and if any objects of a given type use some property to represent information, all objects of that type must use the property in the same way. That is, since the notes dataset is not complete with respect to *year*, exact composition of the labeling graphical language with a language like a bar graph or dot plot is impossible. One way of modifying the system to allow such a composite graphical language would be to treat labels as independent objects, with possible perceptual relations with circles and rectangles. Another way would be to allow for the method of inexact composition described in chapter 4 where “null values” are introduced for certain properties to allow composition with missing information. A null value for a label, of course, would simply be a missing label.

7.2.2 More examples

The preceding simple example demonstrates the basic operation of AUTOGRAPH, and gives some insight into the way it creates languages out of combinations of objects and properties. Some further examples will illustrate its sensitivity to different goals and data characterizations, and the range of presentations it is capable of creating.

Consider again some data about a corporation’s annual profits, now broken down by quarter, looking something like the following:

```
1994 I 200
1994 II 220
1994 III 140
1994 IV 200
1995 I 210
1995 II 220
...
```

One possible goal for this data would be to see how the profits correlate with quarter, and to be able to see this correlation for any given year. Another possible goal would be to see how the profits vary with year for any given quarter. Presumably, different presentations would be most effective for each of these goals.

In the first case, AUTOGRAPH was given the following data and goal characterization:

```
(SETQ *DATA-DESCRIPTION* '((YEAR1 YEAR NIL)
                           (QUART1 QUARTER NIL)
                           (PROF1 PROFITS (YEAR1 QUART1))))
(SETQ *DATA-COMPLETENESS-DESCRIPTION* '((YEAR1 QUART1)))
(SETQ *USER-GOALS* '((BASIC (YEAR1 QUART1) (PROF1))
                     (SUMMARY (PROF1) (QUART1) (YEAR1))))
(SETQ *DATA-DOMAINS* '((YEAR ORDERED (1994 1995 1996 1997) (COORDINATE))
```

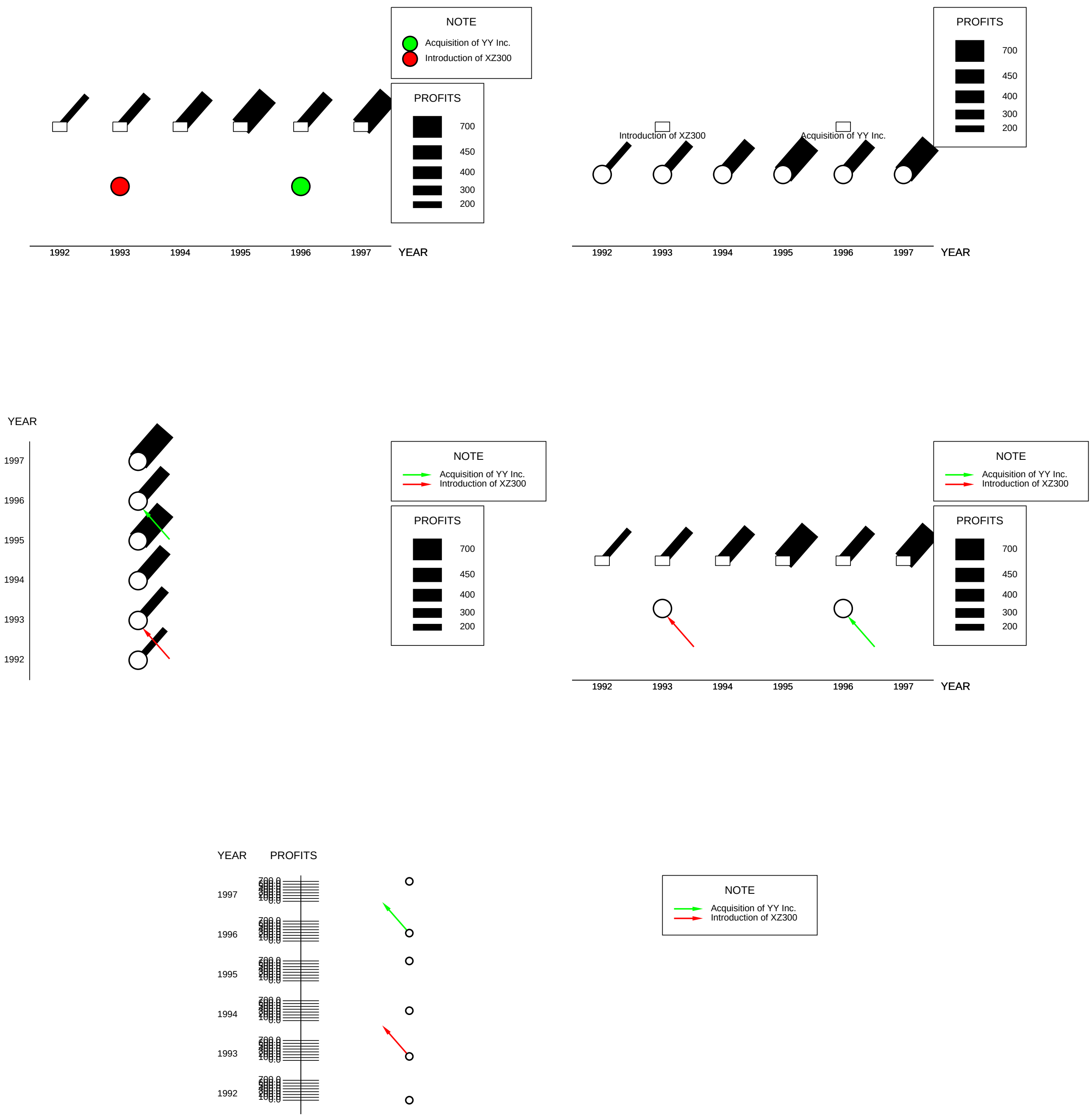


Figure 7.9: Even less effective presentations for the profits and notes datasets

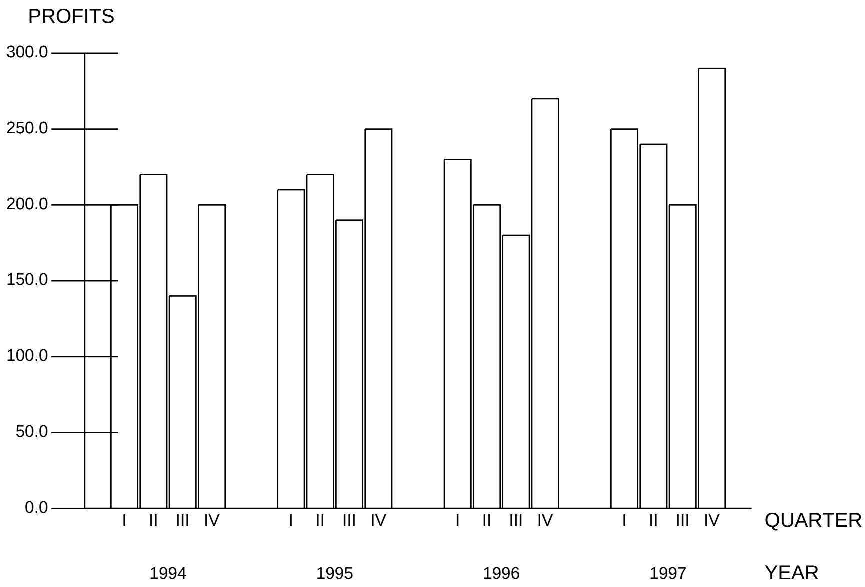


Figure 7.10: A top-ranked presentation for viewing corporate profits by year

```
(QUARTER ORDERED (I II III IV) (COORDINATE))
(PROFITS QUANTITATIVE (140 290) NIL)))
```

The goals in this case were the basic goal of seeing profits for any given year and quarter, and the summary goal of seeing how profits vary with quarter for any given year.

The top three equally-ranked presentations AUTOGRAPH produced for this characterization are shown in figures 7.10 (the most conventional presentation for this data) and 7.11. When AUTOGRAPH was given the second summary goal instead of the first, it produced a similar set of presentations, only with certain property to data domain mappings switched around. Contrasting figure 7.12, produced for the second goal, with figure 7.10, produced for the first, demonstrates how AUTOGRAPH's evaluation heuristic for summary goals enables it to choose an appropriate presentation. In each case, a viewer is able to narrow the focus of their attention to one group of bars—corresponding to one year in figure 7.10 or one quarter in figure 7.12—and determine how the lengths of the bars vary with the position offset of the bar within the group.

When presented with both goals simultaneously, the best AUTOGRAPH can do is the presentation shown in figure 7.13. Although this presentation is the only one in AUTOGRAPH's search space that can reasonably satisfy both summary goals, it is clearly less effective for the basic goal of determining profits than the previous graphs, and is deemed so by AUTOGRAPH.

AUTOGRAPH's evaluation function for summary goals enables it to choose effective presentations for a large variety of presentation problems. Consider a dataset of information about medical tests on a number of subjects, where data is given in the form of arity-four tuples $< \textit{Salt-intake}, \textit{Blood-pressure}, X, Y >$, where the first two domains can take on the values 'low' or 'high', X and Y are quantitative domains, and

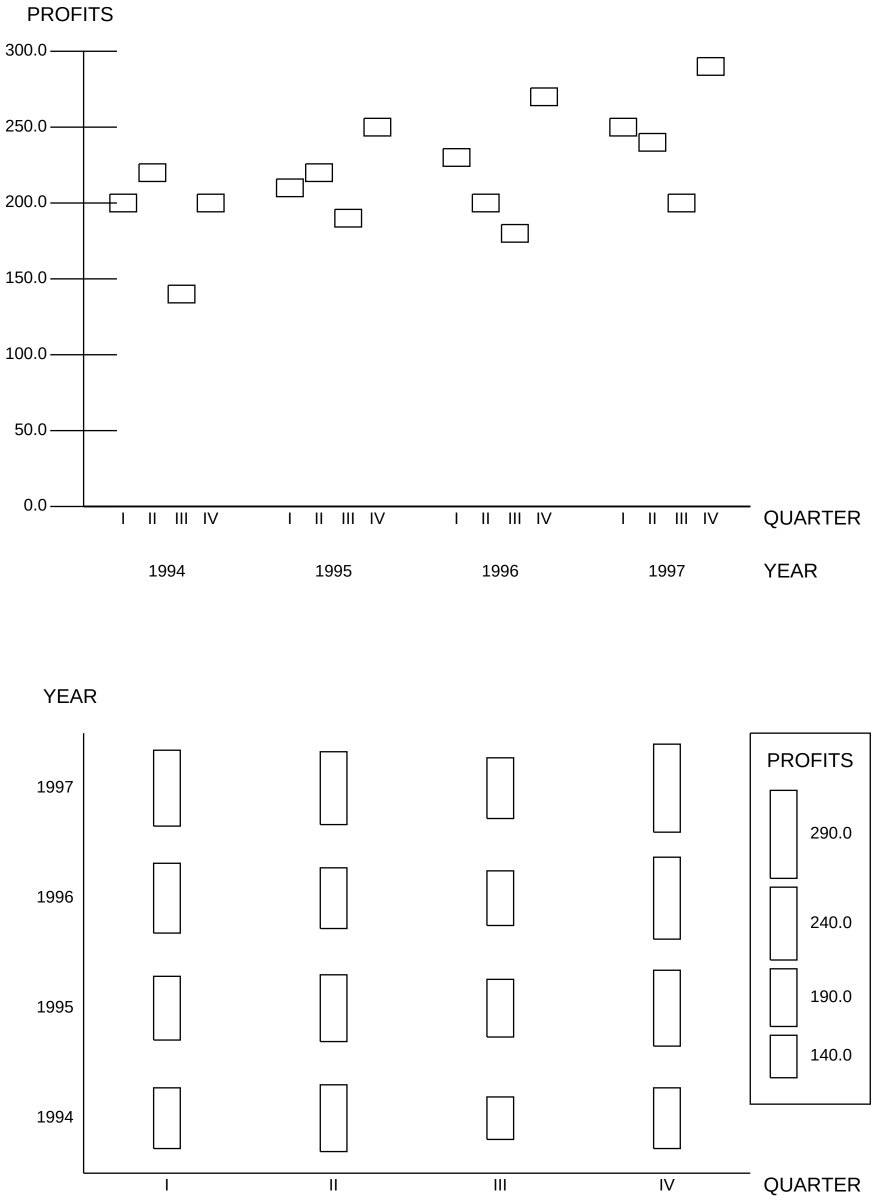


Figure 7.11: Other top-ranked presentations for viewing corporate profits by year

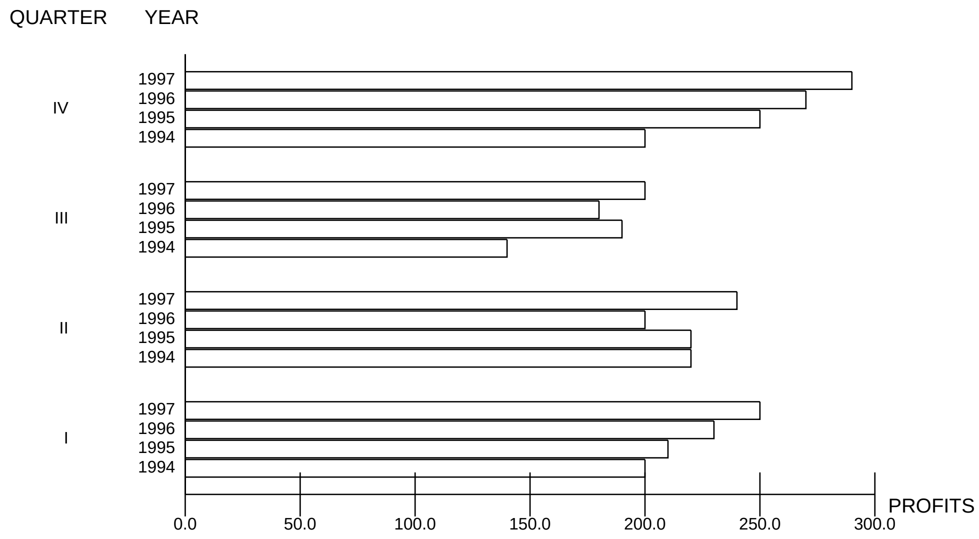


Figure 7.12: A top-ranked presentation for viewing corporate profits by quarter

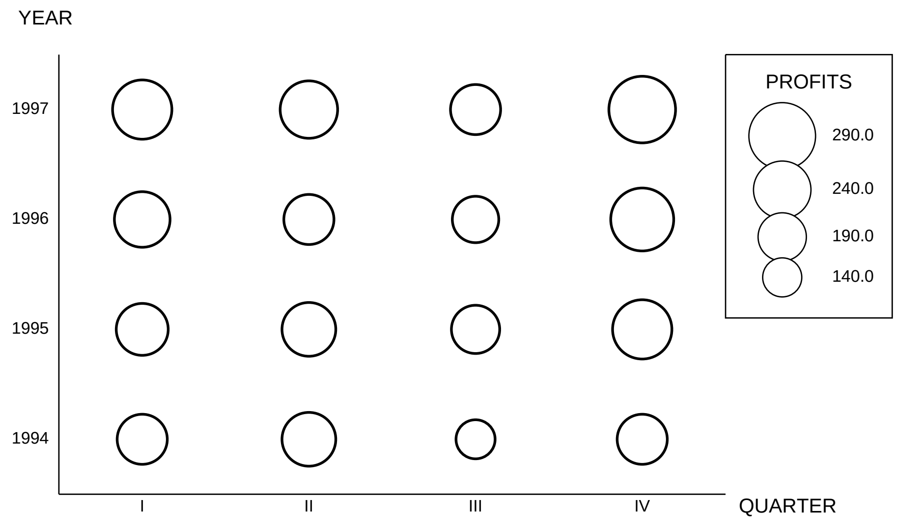


Figure 7.13: A top-ranked presentation for viewing corporate profits by year or quarter

the user's goal is to determine how Y varies with X for any values of *Blood-pressure* and *Salt-intake*. Given a dataset:

```
Low Low 30 50
Low Low 40 55
Low Low 50 40
Low Low 60 80
Low Low 70 100
Low Low 80 90
High Low 30 30
High Low 40 40
High Low 50 40
High Low 60 50
High Low 70 55
High Low 80 60
Low High 30 90
Low High 40 100
Low High 50 80
Low High 60 60
Low High 70 40
Low High 80 30
High High 30 90
High High 40 75
High High 50 55
High High 60 60
High High 70 80
High High 80 100
```

and the characterization:

```
(SETQ *DATA-DESCRIPTION* '((LH1 SALT-INTAKE NIL)
                           (LH2 BLOOD-PRESSURE NIL)
                           (X1 X NIL)
                           (Y1 Y NIL)))
(SETQ *USER-GOALS* '((BASIC (LH1 LH2 X1) (Y1))
                     (SUMMARY (Y1) (X1) (LH1 LH2))))
(SETQ *DATA-DOMAINS* '((SALT-INTAKE ORDERED (LOW HIGH) (COORDINATE))
                       (BLOOD-PRESSURE ORDERED (LOW HIGH) (COORDINATE))
                       (X QUANTITATIVE (30 80) (COORDINATE))
                       (Y QUANTITATIVE (30 100) (COORDINATE))))
```

AUTOGRAPH produces as its top-ranked selection the graph shown in figure 7.14, which is quite effective for satisfying the goal, because the desired *Blood-pressure* and *Salt-intake* values can be spatially indexed (i.e., one quadrant of the presentation can be isolated) and the correlation between Y and X is represented by a configural property of the circles in the appropriate region of the presentation.[28]

In addition to being sensitive to a user's goals, AUTOGRAPH is sensitive to the data characterization. Some simple aspects of this have already been demonstrated; functional dependencies and completeness

[28] Unfortunately, the effectiveness of this presentation is limited by bugs in the *ad hoc* axis-drawing code.

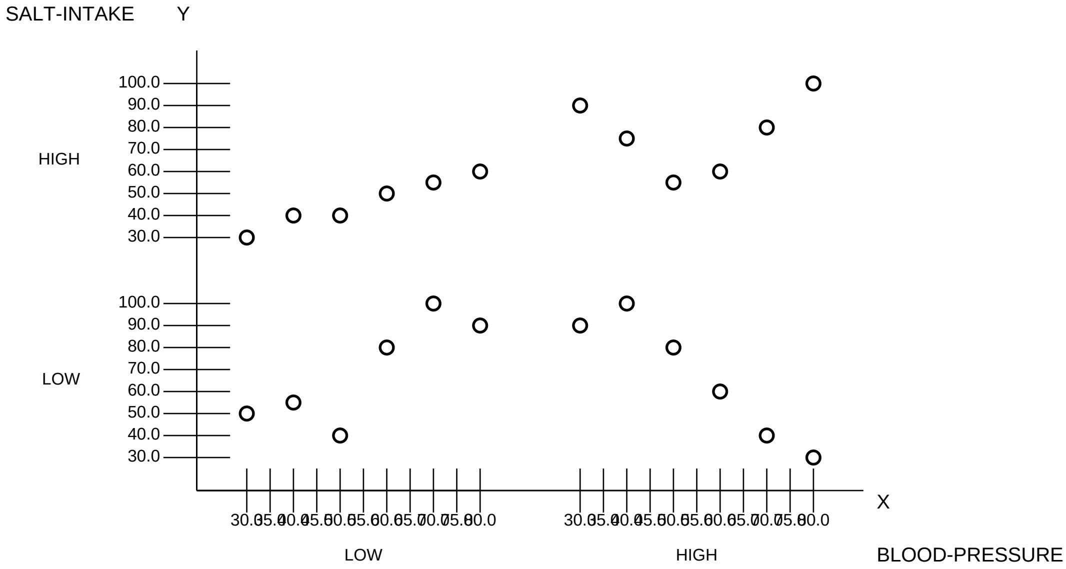


Figure 7.14: Presentation supporting complex summary goal

are crucial for determining composability, for example. AUTOGRAPH's expressiveness checking algorithms allow it to deal with more complicated logical characterizations as well.

Consider three small datasets, one showing parental relationships, and two showing the gender of the parents and children. The parental dataset might be:

```
Homer Marge Bart
Homer Marge Lisa
George Jane Judy
George Jane Elroy
```

and the most basic data characterization given AUTOGRAPH would be:

```
(SETQ *DATA-DESCRIPTION* '((P3 PARENT NIL)
                           (G1 GENDER (P3))
                           (C2 CHILD NIL)
                           (G2 GENDER (C2))
                           (P1 PARENT NIL)
                           (P2 PARENT NIL)
                           (C1 CHILD NIL)))
(SETQ *DATA-COMPLETENESS-DESCRIPTION* '((P3) (C2)))
(SETQ *DATA-DOMAINS* '((CHILD NOMINAL (ELROY LISA BART JUDY) NIL)
                       (PARENT NOMINAL (HOMER MARGE GEORGE JANE) NIL)
                       (GENDER NOMINAL (MALE FEMALE) NIL)))
```

with the datasets to be represented containing tuples of the forms `(P1 P2 C1)` (the parental relationship dataset), `(P3 G1)` (the parent gender dataset), and `(C2 G2)` (the child gender dataset).

A set of elementary goals for this dataset would be expressed:

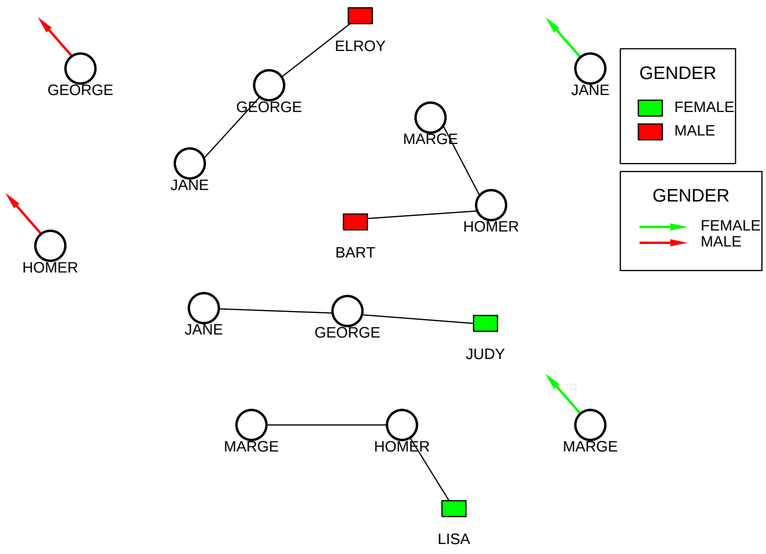


Figure 7.15: A top-ranked presentation for parental and gender datasets with no logical characterization

```
(SETQ *USER-GOALS* '((BASIC (P3) (G1))
                     (BASIC (C2) (G2))
                     (BASIC (P1 P2) (C1))))
```

Now, without any additional logical characterization, the best presentation that AUTOGRAPH can find for these three datasets is the one shown in figure 7.15. Without knowing that children only have two parents, or that the tuple `(HOMER MARGE BART)` is equivalent to the tuple `(MARGE HOMER BART)`, AUTOGRAPH is forced to use non-shared objects to represent both parents and children, in order to avoid violating the strict expressiveness condition. As an additional complication, since AUTOGRAPH cannot compose shared colored circles for representing gender with non-shared circles showing parental relationships without introducing redundancies, it must show the gender of the parents with additional circles.[29] Furthermore, the circles cannot be colored directly, since the other circles of the presentation are not colored.[30]

If AUTOGRAPH is given some logical rules describing the parental dataset, however, it can do better. The appropriate rules for this dataset (assuming it contains data about *biological* parents) are an implication rule stating that a tuple of the form `(PARENT1 PARENT2 CHILD)` is equivalent to `(PARENT2 PARENT1 CHILD)` and a restriction rule stating that a child will only have two parents, expressed as:

```
IMPLICATION RULE: (((TUPLE40 (VAR A1)(VAR A2) (VAR S1)))
                   ((TUPLE40 (VAR A2)(VAR A1) (VAR S1)))))
```

[29] This is a limitation of AUTOGRAPH's ability to compose presentations. Composing with redundancy is a form of inexact composition (as discussed in chapter 4) that AUTOGRAPH does not currently support.

[30] This presentation is also hindered by the limitation of the *ad hoc* legend-drawing routines in the current version of AUTOGRAPH; the presentation should somehow indicate that circle labels encode parents and rectangle labels encode children.

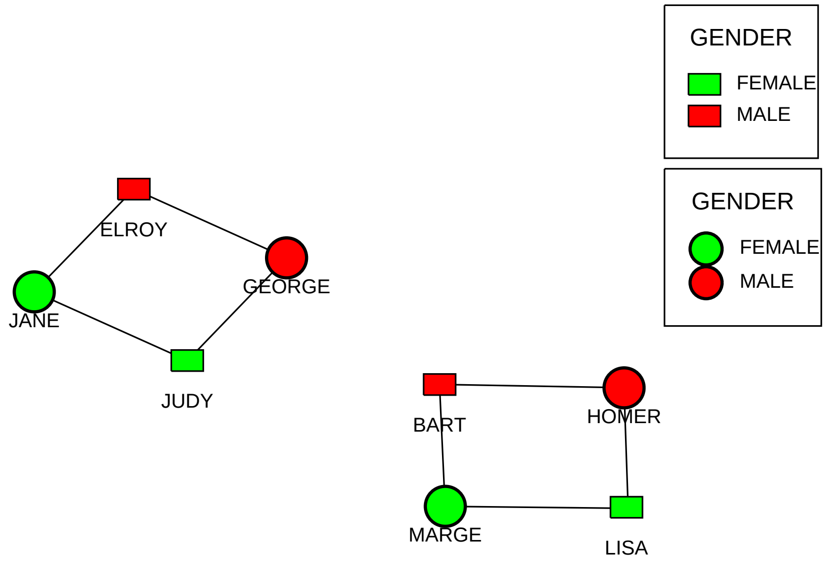


Figure 7.16: A top-ranked presentation for parental and gender datasets with some logical characterization

```
RESTRICTION RULE: (((TUPLE40 (VAR A1) (VAR A2) (VAR S1))
                    (TUPLE40 (VAR A3) (VAR A4) (VAR S1))
                   ((!= (VAR A2) (VAR A3)))))
```

The restriction rule can be translated as "If there are two different tuples in the dataset describing the parents of a single child, the dataset will be inconsistent if the second parent in one tuple is not equivalent to the first parent in the other tuple."[31] In conjunction with implication rule, this is equivalent to stating that if there are two tuples in a dataset describing the parents of a some child, they *must* be permutations of each other.

Given these rules, AUTOGRAPH generates as its top choice the presentation shown in figure 7.16. Note that if the dataset were not symmetric, or if a child could have more than two parents, this graphical language would lead to violations of the strict expressiveness condition, since representing `(MARGE HOMER BART)` would implicitly represent `(HOMER MARGE BART)` and representing `(MARGE HOMER BART)` and `(GEORGE JANE BART)` would represent several additional tuples.

Other logical data characterizations lead to other top choices for graphical languages. For example, the dataset of parental relationships might not be characterized as symmetric, but be characterized with a restriction rule specifying that no second parent (i.e., the parent described by second value in each tuple) will have children with more than one first parent, as follows:

```
RESTRICTION RULE: (((TUPLE40 (VAR A1) (VAR A2) (VAR S1))
                    (TUPLE40 (VAR A3) (VAR A2) (VAR S2)))
                   ((!= (VAR A1) (VAR A3))))
```

[31] Recall that restriction rules are given to AUTOGRAPH as conditions under which a dataset will be inconsistent, consisting of a list of tuples with variable values and then a list of constraints on the values of the variables.

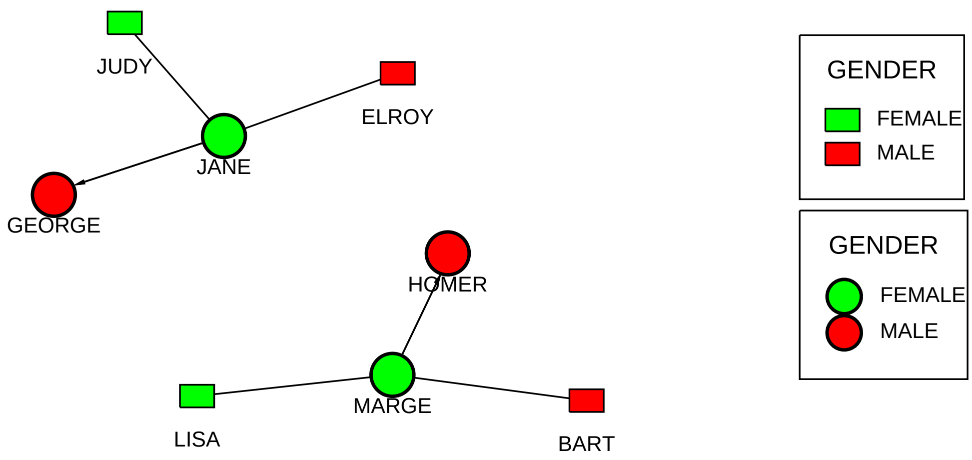


Figure 7.17: Top-ranked presentation for parental and gender datasets with alternate logical characterization

Given this characterization, AUTOGRAPH chooses the presentation shown in figure 7.17. The presentations of figures 7.16 and 7.17 are considered equal based on interpretive and perceptual design principles by AUTOGRAPH's rating function (both are ranked higher than the presentation of figure 7.15, of course)—but each is expressive only for data characterization for which it is generated.

7.2.3 Empirical analysis

To get a more quantitative feel for how AUTOGRAPH operates, it will be useful to consider in some detail how it handles two somewhat complex presentation problems. I will first discuss how AUTOGRAPH handles a particular set of related datasets it has trouble with—i.e., for which the default branching parameter controlling overall pruning must be increased and for which the system takes a large amount of time. I will then, for comparison, discuss how AUTOGRAPH effectively handles a different pair of complex datasets.

The first group of datasets describe:

1. Likes and dislikes among a set of students.

2. The department of each student.

3. The size of each department.

They are as follows:

```
GEORGE MARY LIKES
GEORGE PAUL DISLIKES
PAUL MARY LIKES
MARY SUSAN DISLIKES
MARY KATHERINE LIKES
```

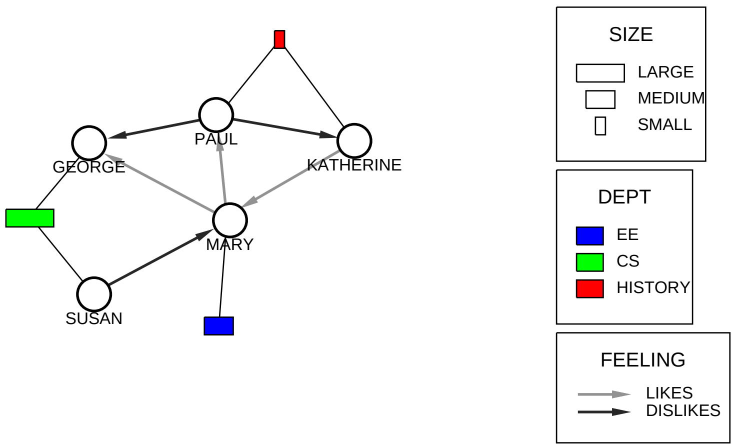


Figure 7.18: A complicated presentation

```
KATHERINE PAUL DISLIKES

GEORGE CS
MARY EE
PAUL HISTORY
SUSAN CS
KATHERINE HISTORY

CS LARGE
EE MEDIUM
HISTORY SMALL
```

The data and goal characterization (assuming the domains of the first dataset are described as `(ST2 ST3 LD1)`, those of the second as `(ST1 DEPT2)` and the those of third as `(DEPT1 SIZE1)`) are:

```
(SETQ *DATA-DESCRIPTION* '((ST2 STUDENT NIL)
                           (ST3 STUDENT NIL)
                           (LD1 FEELING (ST2 ST3))
                           (ST1 STUDENT NIL)
                           (DEPT2 DEPARTMENT (ST1))
                           (DEPT1 DEPARTMENT NIL)
                           (SIZE1 SIZE (DEPT1))))
(SETQ *DATA-COMPLETENESS-DESCRIPTION* '((ST1) (DEPT1)))
(SETQ *DATA-DOMAINS* '((SIZE ORDERED (SMALL MEDIUM LARGE) NIL)
                       (DEPARTMENT NOMINAL (HISTORY CS EE) NIL)
                       (STUDENT NOMINAL (GEORGE MARY SUSAN KATHERINE
                                         PAUL)
                                NIL)
                      (FEELING ORDERED (DISLIKES LIKES) (COORDINATE))))
(SETQ *USER-GOALS* '((BASIC (ST2 ST3) (LD1))
                     (BASIC (ST1) (DEPT2))
                     (BASIC (DEPT1) (SIZE1))))
```

This characterization specifies that the second and third datasets have functional dependencies and are complete for the independent domain (*student*). It also specifies that the user's goals are simply to read off information stored in the graph.

AUTOGRAPH produced presentations for most of the examples presented throughout this chapter in under a minute (sometimes well under) on a 150 MHz Pentium PC, with the maximum branching parameter (a limit on the number of leaves of the search tree for a single graphical language) set to 150. For the data characterization described above, it required ten minutes with the branching factor set to 600 in order to eventually produce its top choice, shown in figure 7.18.[32]

AUTOGRAPH handles the student department dataset, `(ST1 DEPT2)`, first. It finds two possible partitionings. Representing tuples in this dataset with one object (the first partitioning), it finds 32 possible mappings (i.e., 32 languages). For the partitioning into two groups, it finds 100 mappings, then reduces them using the consistency heuristic to 50 mappings. These 50 mappings lead to 50 graphical languages—not precisely one-to-one—for a total of 82 graphical languages for this dataset.

AUTOGRAPH handles the department size dataset, `(DEPT1 SIZE1)`, next. This dataset also can also be partitioned in two ways. Paritioning it into two groups leads to an average of 119 mappings (per each of the 82 languages chosen for the previous dataset), reduced to an average of 76 mappings by consistency checking, and to an average of 5.2 after checking for composability and consistency with the language chosen for the previous dataset. For the partitioning into one group, an average of 40 mappings are reduced not at all by self-consistency filtering, but reduced to an average of 4.7 mappings consistent with the language chosen for the previous dataset. A total of 805 mappings are eventually considered, and lead to 484 composite graphical languages after searching for topologies and ways of instantiating the topologies, and expressiveness checking.

Qualitatively, it is clear that there are many more ways of representing these two simple datasets than there were of representing just one of them, but making the representations consistent and composable keeps the number from exploding.

AUTOGRAPH finally considers methods of representing the dataset of student likes and dislikes, `(ST2 ST3 LD1)`. It does this, of course, 484 separate times—once for each of the composite languages found for the previous two datasets. There are five ways of partitioning an arity-3 tuple; AUTOGRAPH prunes two of these, and examines branches of the search tree for the remaining three partitionings—into two objects `((LD1) (ST2 ST3))`, into one object `((ST2 ST3 LD1))`, and into three separate objects `((LD1) (ST2) (ST3))`. The first of these partitionings leads to 233 mappings, on average, only 7.7 of which are internally consistent and none of which are consistent with and composable with the languages chosen for the first two datasets. The second partitioning leads to an average of 95.3 mappings, reduced to 10.7 by self-consistency checking and to 0 by checking composability and consistency with the previously chosen languages. Finally, the third partitioning leads to an average of 76.2 mappings, 43.6 of which are self-consistent, and 0.8 of which, on average, work with the previously chosen languages. The 387 total choices

[32] The time increase is largely due to the increased branching factor, of course.

of languages for the first two datasets plus mappings for the third dataset lead to 280 complete graphical languages after considering different topologies and ways of connecting component objects, 35 of which are spatially consistent.

Qualitatively, it is worth noting that while it is again true that composability and inter-dataset consistency pruning is most effective, for this (slightly) more complex third dataset, pruning at the level of partitioning and pruning mappings by self-consistency are also effective at limiting the size of the search space.

More generally, this example suggests that while the space of graphical languages grows exponentially with the number of objects and properties available to a system, pruning the search by logical and psychological principles can sharply limit the number of reasonable graphical languages as datasets get larger and/or more numerous.

For comparison, consider how AUTOGRAPH creates the presentation of figure 7.19 for seemingly more complex datasets describing Napoleon's Russian campaign. AUTOGRAPH finds this presentation in under 15 seconds, with a branching factor of only 150.

The first dataset describes troop movements, expressed tuples with the latitude and longitude of the start and end of the movements, the number of troops, and whether the movement was an advance or retreat:

```
54.8 24.0 55.0 25.4 422000 advance
55.0 25.4 54.8 26.0 400000 advance
55.0 25.4 55.5 25.4 22000  advance
54.8 26.0 55.3 26.6 50000  advance
55.3 26.6 55.2 28.3 33000  advance
54.8 26.0 55.0 29.0 250000 advance
...
```

The second dataset expresses some recorded temperatures for specific dates and places (expressed as the latitude and longitude of the place, a string for the date, and the temperature in degrees celsius):

```
55.1 36.6 8-18-1844  0
54.7 32.0 9-9-1844  -9
54.4 31.0 9-14-1844 -21
...
```

The full data and goal characterization for these datasets is:

```
(SETQ *DATA-DESCRIPTION* '((LAT2 LATITUDE MANY NIL)
                           (LONG2 LONGITUDE MANY NIL)
                           (LAT3 LATITUDE MANY NIL)
                           (LONG3 LONGITUDE MANY NIL)
                           (TROOP1 TROOPS MANY NIL)
                           (AR1 ADV-RET MANY NIL)
                           (LAT1 LATITUDE MANY (DATE1))
                           (LONG1 LONGITUDE MANY (DATE1))
                           (DATE1 DATE MANY NIL)
                           (TMP1 TEMPERATURE MANY (DATE1))))
(SETQ *DATA-COMPLETENESS-DESCRIPTION* '((DATE1)))
```

```
(SETQ *DATA-SEMANTIC-GROUPINGS* '((LAT2 LONG2) (LAT3 LONG3)))
(SETQ *DATA-DOMAINS* '((TEMPERATURE QUANTITATIVE (-30 0)
                                    (COORDINATE TEMPERATURE))
                       (DATE NOMINAL (10-7-1844 10-6-1844 10-1-1844
                                      9-28-1844 9-20-1844 9-14-1844
                                      9-9-1844 8-18-1844) NIL)
                       (LATITUDE QUANTITATIVE (53.8 55.9)
                                 (VERTICAL COORDINATE))
                       (LONGITUDE QUANTITATIVE (23.0 38.0)
                                  (HORIZONTAL COORDINATE))
                       (TROOPS QUANTITATIVE (12000 422000) NIL)
                       (ADV-RET ORDERED (ADVANCE RETREAT) NIL)))
(SETQ *USER-GOALS* '((BASIC (LAT2 LONG2 LAT3 LONG3) (TROOP1 AR1))
                     (BASIC (LAT1 LONG1) (DATE1 TMP1))))
```

Some important features to note in this data characterization are that a number of the domains have associated semantics—i.e, latitude and longitude have "vertical" and "horizontal" semantics, respectively, and temperature has "temperature" semantics. This will allow AUTOGRAPH to take advantage of its knowledge of how domains with particular semantics can or should be represented. All of these domains are also classified as being coordinates (rather than amounts). Also important in the characterization is the fact that the starting longitudes and latitudes of the troop movements are characterized as being grouped together semantically, as are the ending longitudes and latitudes. AUTOGRAPH needs this knowledge to make use of the interpretive principle of congruence when constructing graphical languages.[33]

AUTOGRAPH handles the troop movement dataset first. It initially finds 203 possible partitionings, but prunes away all but 7 of these, using the heuristics described in section 7.1.2. It finds an average of 101 mappings per partitioning, but reduces these to 9.3 per partitioning using heuristics. After completing the search for complete graphical languages for the dataset, it finds 44, which are filtered to 37 after expressiveness checking and only 5 after spatial consistency checking.

There are 15 possible partitionings for the second dataset, none of which are pruned. AUTOGRAPH finds exactly 30 mappings for each possible partitioning for each of the five graphical languages found for the first dataset. 30 is a "clipped" value—i.e., it is limited by the overall branching factor. These 30 mappings are reduced to an average of only 0.7 mappings per choice of original language and partitioning for the second language by normal consistency checking, and reduced further by checking consistency and composability of the mappings with the choice of original language. After this stage, only a total of 4 choices of first languages plus mappings remain. These lead to four composite graphical languages, all minor variations on the graph shown in figure 7.19 (e.g., circles instead of rectangles, or arrows pointing the opposite direction).

Qualitatively, this example again shows the power of the heuristics used to prune the search tree.

[33] One final noteworthy feature of this data characterization is that *date* is characterized as *nominal* rather than *ordered*, which is actually inappropriate. The reason for this is that the current version of AUTOGRAPH does not have in its space of possible languages, as far as I know, a presentation like the one in figure 7.19 that also shows the order of the dates using some perceptual property. Since the current implementation cannot ignore the constraint that ordered domains must be represented by perceptually-ordered properties, it is unable to find a presentation with *date* characterized as ordered.

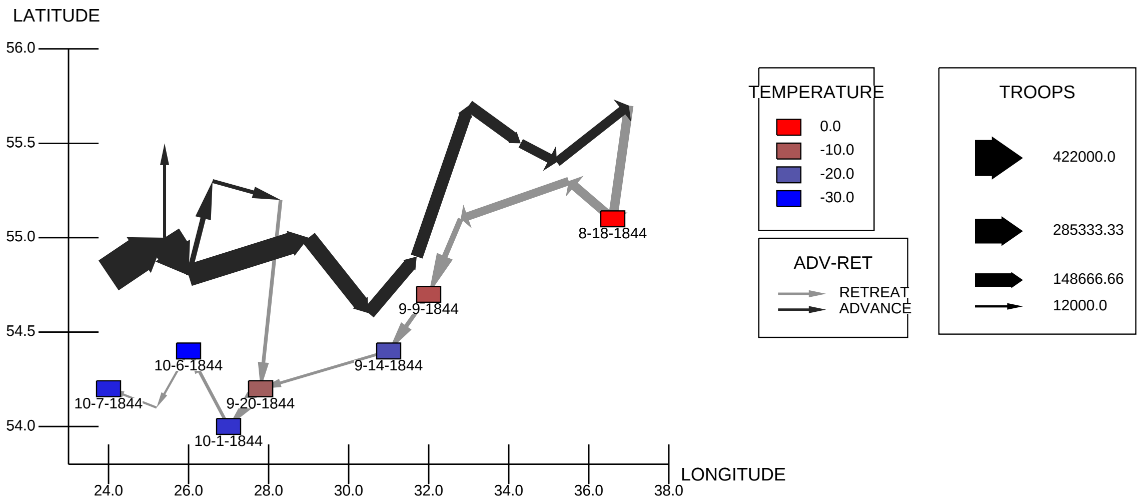


Figure 7.19: Another complicated presentation

Although the search tree for these more complex datasets is theoretically much larger than the tree for the previous example, the size and complexity of the datasets actually help to constrain the space of reasonable graphical languages, enabling a solution to be found quickly.

7.3 Summary

This chapter described a prototype implementation bringing together the general ideas and themes of this dissertation, a program called AUTOGRAPH. It also discussed more generally some issues important to making operational the framework and principles discussed by previous chapters. Such issues included how to turn the design principles of chapters 5 and 6 into heuristics for pruning the search tree of possible graphical languages and for ranking non-pruned languages, and how to use those heuristics and other techniques to manage the size of the search space.

Section 7.1 discussed the overall system architecture of AUTOGRAPH. AUTOGRAPH includes a graphical user interface, a search module, an expressiveness-checking module, a ranking module, a finalization module, and a layout manager. In describing the user interface, that section described data and goal characterization in AUTOGRAPH. In describing the search module, it discussed at length the heuristics AUTOGRAPH uses to prune its search tree. The subsection on the ranking module discussed AUTOGRAPH's complex system for ranking potential presentations, a crude implementation of many of the design principles of chapters 5 and 6.

Having described how AUTOGRAPH works, the chapter went on to present some examples of presentations generated by AUTOGRAPH for different input. It described a simple example in depth to illustrate how AUTOGRAPH operates, then described some other examples to illustrate AUTOGRAPH's sensitivity to various aspects of goal and data characterization. Finally, it attempted to give some quantitative feeling for branching factors and pruning in AUTOGRAPH, using some fairly complex groups of datasets as input.

Chapter 8

Conclusions

This dissertation has outlined the basis for a first-principles approach to automatic graphic presentation design, rooted in logical, interpretive, and perceptual principles. Although some of the issues covered here have been addressed by other research, I have tried to develop a more general framework with fewer implicit assumptions than previous approaches and to provide a broader and more systematic analysis of design principles than previous work. In this chapter, I will briefly review some of the contributions of this thesis, then discuss its many limitations and opportunities for future work.

8.1 Contributions

While my framework is not completely novel, it does go further than previous work in providing a general and fine-grained analysis of how graphic presentations can encode information. Chapter 3 started with an extremely general notion of how information can be encoded, adapted from Goodman, then developed this into a useful framework for graphic presentations of data by making a number of *explicit* assumptions about the systematicity of usable graphic presentations. The idea that the properties of objects encode information is not new to my framework, but the analysis of how compound objects encode information, how perceptual functions as well as properties can encode information, and how to systematically construct a space of possible graphical languages goes beyond previous approaches.

Using a general framework for describing graphical languages rather than a predefined set of graphical languages necessitates addressing several issues in more generality than they have been addressed in previous research. Chapter 4 presented general analyses of logical expressiveness and of the composition of two or more graphical languages. While the algorithms described in that chapter are limited in various ways, the foundations should be useful in the design of more general algorithms.

In addition to analyzing logical expressiveness issues from a general standpoint, making use of a framework without predefined graphical languages requires that the qualities which make a presentation easy or difficult to interpret be analyzed as well. Chapter 5 provided such an analysis, which I believe to be more general and systematic than any previous analysis. My framework also provides guidelines for the ways in which the various general principles can be applied, another useful contribution. My analysis of perceptual principles, presented in chapter 6, has more in common with previous approaches, from which it borrows. However, I believe it makes small contributions in its discussion of the roles of perceptual organization and dimensional structure in graphic presentations and also in the way it addresses the role of configural properties as they relate to summary goals. This is something that has barely been addressed by previous work (with the exception of Pinker (1990)) but which is an extremely important issue in the design of graphic presentations.

Finally, the AUTOGRAPH implementation, discussed in chapter 7, demonstrates how general principles of interpretation and perception can be operationalized, i.e., how relatively abstract design principles can actually be used in making decisions about how to present data.

Combining these different threads, this dissertation presents a new approach to the problem of automatically designing graphic presentations. Based on the evidence presented in chapter 7, I think this approach can be considered a qualified success. As the examples of that chapter demonstrate, a system operating from first principles can be capable of designing a wide range of graphic presentations, including standard ones such as bar and network graphs, variations on these standards, and even rather complicated and unusual ones such as the graph of Napoleon's Russian campaign. While a system making use of predefined languages can certainly be capable of designing standard presentations and some variations of these presentations, getting it to design unusual variations or new types of presentations will be difficult unless the techniques are known in advance and pre-programmed.

Having said this, I believe my framework has not been pushed nearly as far as it can go. In attempting to pull together many threads, this dissertation has focused on breadth (i.e., exploring different types of principles), occasionally to the exclusion of depth (e.g., creating the most general or efficient algorithms possible). The current AUTOGRAPH implementation, using only a small set of objects and properties, is not capable of designing very many presentations that previous systems could not design. However, it seems likely that a future version of this program could design truly novel presentations—at least for complex data, since the space the presentations for simple data has already been well explored by human designers. First, however, some of the limitations of the framework, principles, and implementation must be addressed.

8.2 Limitations and future work

The framework outlined in this dissertation, the principles that flesh out the framework, and the prototype implementation all have a number of limitations which are worth mentioning here. I will outline some of the most significant limitations, mentioning some of the ways they might be addressed, and finally discuss briefly some other ways this work might be extended.

Before discussing the limitations of the work described in this dissertation, however, it might be useful to briefly consider some of the limitations inherent in its definition of the graphic presentation design problem.

8.2.1 Limitation of the problem definition

As detailed in chapter 1, the task taken on by this dissertation was to design graphical languages based on data and goal characterizations, without using details of the actual data to be presented. Although this approach makes the presentations designed by a system more flexible (i.e., usable for different datasets of the same type), it has a substantial drawback. By being insensitive to the actual data to be presented, a system is significantly limited in its ability to judge the relative effectiveness of different choices for presentations. For example, network graphs are fairly effective for presenting sparse data (e.g., a few flights between a few different cities), but get very messy when presenting very dense datasets. Without special layout techniques, dense network graphs are likely to have a lot of visual clutter (in the terms of chapter 7) and poor perceptual organization. A system that does not look at the actual data to be presented can only roughly estimate the effectiveness of a network graph, and such an estimate may be poor. Similarly, to accurately determine the time required to perform an inefficient (i.e., "serial") visual search, it is essential to know the number of objects in a presentation, something which will often not be possible without knowing the specific data being presented.[1]

Fortunately, there is little in the technical framework and principles described in this dissertation to prevent the use of knowledge about the specific data to be presented. To some extent, it would be possible to address the problems described above by adding information about the "density" of a dataset to the characterization of that dataset. Even without modifying the characterization language, though, there is no reason why information about actual data to be presented could not be used by slightly modified versions of the ranking heuristics used in the current version of AUTOGRAPH. Such a change would be relatively easy to make, and might improve the effectiveness of the presentations that the system chooses.

8.2.2 Limitations of the framework

Chapter 3 began by outlining a very general logical and psychological framework, then made a number of assumptions—i.e., systematicity assumptions—to make this framework more useful. Both the general framework and the assumptions have certain limitations, however.

The limitations of the framework arise mostly from assumptions about the cognitive, perceptual, and graphic levels of description of a presentation, as well as from the framework's notions of perception and interpretation. The graphic level is assumed to be static and two-dimensional, which limits the potential output of an automatic presentation system. It would be relatively simple to expand the graphic level to include parameters for a third dimension and for time, then make corresponding changes to the perceptual level in order to produce three-dimensional and animated presentations. It is not clear how well the rest of the framework would accommodate these changes, however.

On the input side, the data handled by the framework is assumed to be described in terms of tuples and domains. This is somewhat restrictive, but the notion of tuples and domains could probably be extended

[1] Of course, the *relative* times required by different languages may not be as sensitive to the actual data.

fairly easily to any compositional description of information. More significantly, perhaps, since the cognitive-level description is taken to be simply a representation of the data, the framework does not address issues such as the memorability of a presentation, or how comfortable viewers will feel with a particular method of presentation.

Another limitation of the framework is its restricted notion of how information is extracted from a presentation. As discussed briefly in chapter 3, it is possible to envision a more comprehensive model of extracting information which includes cognitive operations (i.e., inferences). Such a model would allow the design of a wider range of presentations, although these new presentations would be likely to be less effective than those in which information is conveyed in a direct perceptual manner.

The limitations imposed by the systematicity assumptions are also significant, given that one of the goals of this dissertation was to provide as general a framework as possible for graphical languages. Most glaring among these limitations is that the systematicity assumptions limit the framework to describing "discrete" graphical languages in which there is a simple correspondence between individual tuples and the compound objects that encode them. Although component objects can be shared, there exist "continuous" visual representations in which objects encode many tuples or pieces of simple objects encode parts of different tuples. Such presentation methods are common in scientific visualization. For example, the surface temperature of a three-dimensional object might be represented by coloring the surface of that object. In the realm of more conventional, two-dimensional presentations, the very common line chart uses multi-segmented lines to represent entire datasets, with no single segment corresponding to a single tuple. I do not think it would be very difficult to extend the framework or an implementation of the framework to handle certain continuous languages (such as line charts). One approach might be to introduce a new class of compound object with multiple parts (e.g., a segmented line) which could encode an entire dataset. Addressing this issue would be quite valuable for the purpose of creating a more practical system.

Other limitations imposed by the systematicity assumptions include the restriction that all objects of a given class must encode the same type of information—i.e., that they cannot be distinguished by context. While the notion of class is somewhat malleable, as I will discuss shortly, it is clear that in some graphic presentations, similar types of objects encode different things. For example, it is possible to draw a network graph with labeled circles that encode different types of entities (e.g., cities and the airports that serve them) without visually distinguishing the different types. While such graphs may be confusing, it is probably not ideal to completely exclude them from the space of graphical languages.

Even the systematicity assumption that every tuple of a dataset must be encoded by the same type of objects is somewhat limiting. As described in chapter 4, some graphic languages such as network graphs violate the intuitive expressiveness condition for datasets that can include reflexive relations (e.g., $< a_1, a_1 >$), because the compound objects required to represent such relations are not well-formed. That is, they have too few distinct components. To extend such a language to handle reflexive data, it would be useful to allow different types of compound objects to represent reflexive tuples. For example, a network graph language which normally uses circles connected by straight lines to represent tuples could be extended to represent tuples of the form $< a_i, a_i >$ using compound objects consisting of circles with a looping arc attached to them. Such an extension would require relaxing the systematicity assumptions, however.

Most of the systematicity assumptions can probably be relaxed in a number of ways. They could be replaced with explicit interpretive principles—mostly applications of the consistency principle—which would be more flexible. Doing so would be likely to make the analysis of logical expressiveness and composition more complicated, however.

Some limitations of the framework of a different sort arise because it does not address the question of what defines an object class, a property, etc. It was assumed—explicitly in chapter 3, though only for convenience—that there are closed classes of perceptual objects, properties, perceptual relations, and perceptual functions. Perception and cognition are complex, though, and permit flexible and widely varied definitions of class and object properties. On the perceptual side, people are capable of discriminating a wide range of complicated perceptual relationships among objects, rather than a small set of simple ones. On the cognitive side, it is relatively easy for a person to see circles and squares as belonging to a single object class, but varying in the property of shape, and a viewer is certainly capable of treating lines of different colors as different classes of objects (e.g., with black lines encoding roads and blue lines encoding rivers on a map). Similarly, it is fairly common to put on a single page several presentations using similar types of objects to encode different types of information. One way of looking at this in my framework would be to treat objects in different presentations on one page as belonging to separate classes, with spatial region being an important determiner of object class. While a truly general formulation of what people are capable of perceiving is well beyond the scope of my framework, a more flexible classification scheme could certainly be integrated into it. This would greatly expand the variety of presentations that a system based on the framework could generate.

A more flexible classification scheme would also increase the independence of the framework from the way data is schematized. Currently, different types of tuples must be represented with different types of (possibly compound) objects, to avoid violations of the strict expressiveness condition. Similar types of tuples must be represented with the same types of object. Different schematizations of the same data, however, might split information into different datasets. For example, information about trains and buses between pairs of cities might be specified as a dataset of train times of the form $\{City_1, City_2, Time\}$ and a similar but separate datasets for bus times, or the datasets might be combined into a single set of tuples of the form $\{City_1, City_2, Time, Train\text{-}or\text{-}Bus\}$. Such schematizations would lead to different representations in the current framework. A more flexible definition of object class could remedy this problem.

8.2.3 Algorithm limitations

The algorithms for intuitive and strict expressiveness checking described in chapter 4 have a number of limitations. In some ways, the intuitive expressiveness checking algorithm is the more limited of the two, since it has limitations even given the assumptions under which it operates. As discussed in chapter 4, the algorithm is overconservative —i.e., it may reject some expressive graphical languages as inexpressive— because of the way it analyzes how perceptual relations affect the degrees of freedom of the objects they constrain. As is also discussed in that chapter, it seems likely that this limitation could be addressed.

Both the strict and intuitive expressiveness algorithms described in chapter 4 operate only under certain assumptions about data characterization. One major limitation is their inability to make use of data characterizations expressing mathematical relationships between data values. Such characterizations are potentially quite valuable—knowing that *duration* = *end-time* – *start time* could enable an expressiveness checking algorithm to use the width of a rectangle and the positions of its left and right edges to represent these three domain values. I believe it would be fairly easy to integrate simple types of mathematical data characterizations into the expressiveness checking algorithms—e.g., modifying the intuitive expressiveness algorithm to deal with interrelated properties of atomic objects—though it is not clear how easy it would be to use these characterizations in a fully general way.

Another constraint on the expressiveness checking algorithms is that the perceptual relations they deal with must be mutually independent (i.e, assertable without regard to other perceptual relations that have been asserted). This eliminates the possibility of using perceptual relations such as *above*, *contains*, and *same-color*. It seems clear that an algorithm could be developed for binary relations and limited data characterizations of these relations, determining that *anti-symmetric*, *transitive* relations could be represented by containment, for example. It is less certain that it could be extended to do this with tuples of greater arity and the current range data characterizations.

8.2.4 Design principle limitations

The principles discussed in chapters 5 and 6 are almost certainly incomplete. In addition, as mentioned in chapter 6, there are a number of simplifying assumptions underlying my analysis of the perceptual operations needed to extract information for elementary goals. For example, the specific ways that axes and legends affect information extraction are not considered. The analysis of summary goals is also hampered by currently limited knowledge of configural properties. Future research may expand understanding of when configural properties occur in presentations and allow analysis of a wider range of such properties. In general, more sophisticated principles and analyses should be easy to incorporate into a system based on my framework.

Another general limitation of both the interpretive and perceptual principles described in this dissertation is that they are for the most part not quantitative. Future empirical research may lead to more precise knowledge about both perception and interpretation, and it is likely that such knowledge could be incorporated into my framework. However, it would require some changes to the framework to take advantage of such quantitative metrics.

8.2.5 Implementation limitations

In addition to the limits of the framework, algorithms, and principles, there are a number of ways in which the implementation does not take full advantage of the potential of the framework. Some notable cases of this include:

- Many of the perceptual and interpretive principles described in chapter 5 and 6 are not actually used by the implementation. For example, most conventions are ignored (e.g., time is represented as often vertically as horizontally, and will be graphed increasing from top to bottom, contrary to convention). Furthermore, the current ranking functions and the methods for determining specific perceptual values for mappings take into account only crudely psychological knowledge about preattentive processing, interference, dimensional integrality, etc.

- Some methods of inexact composition which are described in chapter 4 are not supported, limiting the composite presentations that the implementation is capable of producing.

- The layout manager is rather *ad hoc*, and will not always produce good layouts for all graphical languages the system can design.

Most of these limitations can probably be easily remedied; they are generally the result of practical considerations (i.e., the amount of time required to write code) rather than theoretical ones.

In addition to these limitations, the implementation is restricted by its inefficiencies. While it handles many presentation problems in under a minute, many even in 10 seconds or so, when presenting multiple simple datasets it often slows down drastically. In addition to making the system more convenient to use, addressing these inefficiencies would in some cases allow the search module to consider a wider range of choices for graphical languages, possibly leading to more effective presentations.

Several ways of improving the efficiency of the implementation are evident:

- Parallelization—Since the implementation works with a well-defined search tree, it should be a relatively simple matter to parallelize it to run on several or many computers rather than a single one.

- Abstraction—The way the current implementation considers different choices for graphical languages, especially for mapping data domains to perceptual properties, seems highly inefficient. AUTOGRAPH treats every choice of object properties as completely separate, but many mappings have much in common and choices for mappings may be independent of other choices for a graphical language. For example, whether the thickness or the grayscale value of lines between two circles is used to represent information has little bearing on whether the graphical language will be expressive or spatially consistent.

  It seems likely that deferring committment to specific properties and using constraint satisfaction techniques might lead to a more efficient implementation.

- Case-based reasoning—The data characterization used in the search for graphical languages in the current version of AUTOGRAPH is sufficiently abstract that many presentation problems the system is given are likely to be similar to problems it has been given in the past. Thus, it should be possible to use a record of previous searches for graphical languages to avoid new searches in many cases, by retrieving (and possibly modifying) the results of old searches for similar problems.

Some or all of these techniques could very likely make AUTOGRAPH a much more useful system.

8.2.6 Other opportunities for future work

Many limitations of this thesis could be addressed in order to build more powerful systems. Of the approaches to remedying these limitations just described, I think the most promising would be incorporating a more flexible approach to defining object classes and properties, improving efficiency through abstraction, and designing new algorithms that permit the use of a wider range of perceptual relations and inter-related perceptual properties in graphical languages. Expanding the graphic-level and perceptual-level descriptions to include three-dimensional and animated presentations may also be promising.

In addition to these improvements, however, there are other interesting ways in which I believe the research described in this dissertation could be extended. I have already mentioned the possibility of generating animated presentations using this framework. More generally, I believe it might be possible to extend the framework to address the automatic generation of interactive presentations of information, and even more generally the design of user interfaces. Such an extension would require research into design principles for interaction as well as additional principles for interpretation and a broadening of the framework, but I believe this would ultimately be possible.

Another interesting direction to extend this research would be to place the operation of visualization in a broader context of data analysis or problem solving. The current framework presumes that a user knows what information he or she wishes to present and what goals he or she has for the presentation. There are many situations in which this will not be the case, however. If a user is casting about for new ways of looking at his or her data, or trying to design a visual representation to help solve some problem, it may not always be obvious what information should be presented visually. I believe it might be possible to build an environment for exploratory visualization or problem solving that would incorporate a first-principles approach to presentation design, as well as heuristic methods for determining when and what types of presentations would be useful.

8.3 Concluding remarks

The information accessible to people is continually increasing, but without means for understanding this information, it may be useless. Graphic presentations are an effective aid for understanding information, but finding effective ones is not always easy. This dissertation has demonstrated that a first-principles approach to automatic design is a promising ways of creating effective presentations, and has the potential to improve the ways that people interact with information.

Bibliography

BERGMAN, L. D., B. E. ROGOWITZ, & L. A. TREINISH. 1995. A rule-based tool for assisting colormap selection. In *Proceedings of IEEE Visualization '95*, 118–125. IEEE Computer Society Press.

BERTIN, J. 1981. *Graphics and graphic information processing*. New York: de Gruyter.

BERTIN, J. 1983. *Semiology of graphics*. Madison, WI: University of Madison Press. Translated by William J. Berg.

BIRREN, FABER (ed.) 1969. *A grammar of color; a basic treatise on the color system of Albert H. Munsell*. New York: Van Nostrand Reinhold Co.

CALLAGHAN, T. C. 1984. Dimensional interaction of hue and brightness in preattentive field segregation. *Perception & Psychophysics* 36(1):25–34.

CALLAGHAN, T. C. 1989. Interference and domination in texture segregation: Hue, geometric form, and line orientation. *Perception & Psychophysics* 46(4):299–311.

CASNER, STEPHEN M., 1990. *Task-analytic design of graphic presentations*. University of Pittsburgh dissertation.

CASNER, STEPHEN M. 1991. A task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics* 10(2):111–151.

CASNER, STEPHEN M., 1993. Graphical composition of relational information. Draft of unpublished article.

CHOMSKY, N. 1965. *Aspects of the theory of syntax*. Cambridge, MA: MIT Press.

CHUAH, MEI C., STEVEN F. ROTH, JOHN KOLOJEJCHICK, JOE MATTIS, & OCTAVIO JUAREZ. 1995a. SageBook: Searching data-graphics by content. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '95)*, 338–345.

CHUAH, MEI C., STEVEN F. ROTH, JOE MATTIS, & JOHN KOLOJEJCHICK. 1995c. SDM: Malleable information graphics. In *Proceedings of the IEEE Symposium on Information Visualization '95*, 36–42.

CHUAH, MEI C., STEVEN F. ROTH, JOE MATTIS, & JOHN KOLOJEJCHICK. 1995b. SDM: Selective dynamic manipulation of visualizations. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '95)*, 61–70.

CLEVELAND, WILLIAM S. 1985. *The elements of graphing data*. Monterey, California: Wadsworth.

CLEVELAND, WILLIAM S., & ROBERT MCGILL. 1984. Graphical perception: Theory, experimentation, and application to the development of graphcal methods. *Journal of the American Statistical Association* 79(387):531–554.

DAVIDSON, RON, & DAVID HAREL. 1996. Drawing graphics nicely using simulated annealing. *ACM Transactions on Graphics* 15(4):301–331.

DENGLER, E., M. FRIEDELL, & J. MARKS. 1993. Constraint-driven diagram layout. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*, 330–335. IEEE Computer Society Press.

DISESSA, A. A., D. HAMMER, B. SHERIN, & T. KOLPAKOWSKI. 1991. Inventing graphing: Meta-representational expertise in children. *Journal of Mathematical Behavior* 10(2):117–160.

DRIVER, J. 1992. Motion coherence and conjunction search: Implications for guided search theory. *Perception & Psychophysics* 51(1):79–85.

DUNCAN, J., & G. W. HUMPHREYS. 1989. Visual search and stimulus similarity. *Psychological Review* 96(3):433–458.

D'ZMURA, M. 1991. Color in visual search. *Vision Research* 31(6):951–966.

EADES, PETER. 1984. A heuristic for graph drawing. *Congressus Numerantium* 42:149–160.

FRICK, A., C. KESKIN, & V. VOGELMANN. 1996. Integration of declarative approaches. In *Proceedings of Graph Drawing '96*, volume 1190 of *Lecture Notes in Computer Science*. Springer-Verlag.

FUJISHIRO, ISSEI, YURIKO TAKESHIMA, YOSHIHIKO ICHIKAWA, & KYOKO NAKAMURA. 1997. GADGET: Goal-oriented application design guidance for modular visualization environments. In *Proceedings of IEEE Visualization '97*, 245–252.

GARNER, W. R. 1974. *The processing of information and structure*. Potomac, MD: Lawrence Erlbaum Associates.

GARNER, W. R., & G. L. FELFODY. 1970. Integrality of stimulus dimensions in various types of information processing. *Cognitive Psychology* 1:225–241.

GIBSON, J. J. 1979. *The ecological approach to visual perception*. Boston: Houghton Mifflin.

GOLDSTEIN, JADE, & STEVEN F. ROTH. 1994. Using aggregation and dynamic queries for exploring large data sets. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '94)*, 23–29.

GOLOVCHINSKY, GENE, THOMAS KAMPS, & KLAUS REICHENBERGER. 1995. Subverting structure: Data-driven dynamic diagram generation. In *Proceedings of IEEE Visualization '95*, 217–223. IEEE Computer Society Press.

GOODMAN, N. 1968. *Languages of art*. New York: Bobbs-Merrill.

GRICE, H. P. 1975. Logic and conversation. In *Syntax and semantics 3: Speech acts*, ed. by P. Cole & J. L. Morgan, 41–58. New York: Academic Press.

GRICE, H. P. 1978. Further notes on logic and conversation. In *Syntax and semantics 9: Pragmatics*, ed. by P. Cole, 113–128. New York: Academic Press.

HANDEL, S., & S. IMAI. 1972. The free classification of analyzable and unanalyzable stimuli. *Perception & Psychophysics* 12:108–116.

HE, W., & K. MARRIOTT. 1996. Constrained graph layout. In *Proceedings of Graph Drawing '96*, volume 1190 of *Lecture Notes in Computer Science*, 217–232. Springer-Verlag.

HEALEY, CHRISTOPHER G., 1996. *Effective visualizing of large multidimensional datasets*. The University of British Columbia dissertation.

HEALEY, CHRISTOPHER G., KELLOGG S. BOOTH, & JAMES T. ENNS. 1993. Harnessing preattentive processes for multivariate data visualization. In *Proceedings of Graphics Interface '93*, 107–117, Toronto, Ontario, Canada. Canadian Information Processing Society.

JULÉSZ, B., & J. R. BERGEN. 1983. Textons, the fundamental elements in preattentive vision and perceptions of textures. *The Bell System Technical Journal* 62(6):1619–1646.

KAMPS, T., J. KLEINZ, & J. READ. 1995. Constraint-based spring-model algorithm for graph layout. In *Proceedings of Graph Drawing '95*, volume 1027 of *Lecture Notes in Computer Science*, 349–360. Springer-Verlag.

KOLOJEJCHICK, JOHN A., STEVEN F. ROTH, & PETER LUCAS. 1997. Information appliances and tools in Visage. *Computer Graphics and Applications* 17(4):32–41.

KOSAK, COREY, JOE MARKS, & STUART SHIEBER. 1994. Automating the layout of network diagrams with specified visual organization. *IEEE Transactions on Systems, Man, and Cybernetics* 24(3):440–454.

KOSSLYN, STEPHEN M. 1989. Understanding charts and graphs. *Applied Cognitive Psychology* 3:185–226.

KOSSLYN, STEPHEN M. 1994. *Elements of graph design*. New York: W.H. Freeman.

KWAK, H., D. DAGENBACH, & H. EGETH. 1991. Further evidence for a time-independent shift of the focus of attention. *Perception & Psychophysics* 49(5):473–480.

LOHSE, GERALD LEE. 1991a. A cognitive model for the perception and understanding of graphs. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '91)*, 137–144.

LOHSE, GERALD LEE, 1991b. *A cognitive model for understanding graphical perception*. The University of Michigan dissertation.

LOHSE, GERALD LEE. 1993. A cognitive model for understanding graphical perception. *Human-Computer Interaction* 8(4):353–388.

LOHSE, GERALD LEE, NEFF WALKER, KEVIN BIOLSI, & HENRY RUETER. 1991. Classifying graphical information. *Behaviour and Information Technology* 10(5):419–436.

MACKINLAY, J., 1986a. *Automatic design of graphical presentations*. Stanford University dissertation.

MACKINLAY, J. 1991. Search architectures for the automatic design of graphical presentations. In *Intelligent user interfaces*, ed. by Joseph W. Sullivan & Sherman W. Tyler. New York: ACM Press.

MACKINLAY, JOCK. 1986b. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics* 5(2):110–141.

MACKINLAY, JOCK. 1988. Applying a theory of graphical presentation to the graphic design of user interfaces. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software (UIST '88)*, 179–189.

MILLER, G. A. 1956. The magical number seven, plus or minus two: some limits on our capacity to process information. *Psychological Review* 63:81–97.

MITTAL, VIBHU, STEVEN F. ROTH, JOHANNA D. MOORE, JOE MATTIS, & GIUSEPPE CARENINI. 1995. Generating explanatory captions for information graphics. In *Proceedings International Joint Conference on Artificial Intelligence*. IJCAI.

NAGY, A. L., & R. R. SANCHEZ. 1990. Critical color differences determined with a visual search task. *Journal of the Optical Society of America A* 7(7):1209–1217.

NAKAYAMA, K., & G. H. SILVERMAN. 1986. Serial and parallel processing of visual feature conjunctions. *Nature* 320:264–265.

PALMER, S. E. 1992. Common region: A new principle of perceptual grouping. *Cognitive Psychology* 24:436–447.

PALMER, S. E. in press. *Vision science: Photons to phenomenology*. Cambridge, MA: Bradford Books/MIT Press.

PALMER, S. E., & I. ROCK. 1994. Rethinking perceptual organization: The role of uniform connectedness. *Psychonomic Bulletin and Review* 1(1):29–55.

PCSSCA. 1986. Report of the Presidential Commission on the Space Shuttle Challenger Accident. Washington, DC.

PINKER, S. 1990. A theory of graph comprehension. In *Artificial intelligence and the future of testing*. Lawrence Erlbaum Associates.

REICHENBERGER, K., T. KAMPS, & G. GOLOVCHINSKY. 1995. Towards a generative theory of diagram design. In *Proceedings of IEEE Symposium on Information Visualization '95*, 217–223, Los Alamitos, USA. IEEE Computer Society Press.

ROGOWITZ, BERNICE E., & LLOYD A. TREINISH. 1993. An architecture for perceptual rule-based visualization. In *Proceedings of IEEE Visualization '93*, ed. by Gregory M. Nielson & Dan Bergeron, 236–243. IEEE Computer Society Press.

ROTH, STEVEN F., MEI C. CHUAH, STEPHAN KERPEDJIEV, JOHN A. KOLOJEJCHICK, & PETER LUCAS. 1997. Towards an information visualization workspace: Combining multiple means of expression. *Human-Computer Interaction* p. in press.

ROTH, STEVEN F., PETER LUCAS, JEFFREY A. SENN, CRISTINA C. GOMBERG, MICHAEL B. BURKS, PHILIP J. STROFFOLINO, JOHN A. KOLOJEJCHICK, & CAROLYN DUNMIRE. 1996. Visage: A user interface environment for exploring information. In *Proceedings of the IEEE Symposium on Information Visualization '96*, 3–12, San Francisco, CA.

ROTH, STEVEN F., & JOE MATTIS. 1990. Data characterization for intelligent graphics presentation. In *Proceedings of the ACM Conference on Human Factors in Computer Systems (CHI '90)*, 193–200.

ROTH, STEVEN F., & JOE MATTIS. 1991. Automating the presentation of information. In *Proceedings of the IEEE Conference on AI Applications*, 90–97.

ROTH, STEVEN F., JOE MATTIS, & XAVIER MESNARD. 1991. Graphics and natural language as components of automatic explanation. In *Intelligent user interfaces*, ed. by Joseph Sullivan & Sherman Tyler, chapter 10, 207–239. Reading. MA: Addison-Wesley.

RYALL, KATHY, JOE MARKS, & STUART SHIEBER. 1996. An interactive system for drawing graphs. In *Proceedings of Graph Drawing '96*, volume 1190 of *Lecture Notes in Computer Science*, 387–394. Springer-Verlag.

RYALL, KATHY, STUART SHIEBER, & JOE MARKS. 1997. An interactive constraint-based system for drawing graphics. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '97)*, 97–104, New York. ACM Press.

SENAY, HIKMET, & EVE IGNATIUS. 1991. Compositional analysis and synthesis of scientific data visualization techniques. In *Scientific Visualization of Physical Phenomena (Proceedings of CG International '91)*, ed. by N. M. Patrikalakis, 269–281. Springer-Verlag.

SENAY, HIKMET, & EVE IGNATIUS. 1994. A knowledge-based system for visualization design. *IEEE Computer Graphics and Applications* 14(6):36–47.

STEVENS, S. S. 1975. *Psychophysics*. New York: John Wiley.

TREISMAN, A., & G. GELADE. 1980. A feature-integration theory of attention. *Cognitive Psychology* 12:97–136.

TREISMAN, A., & S. GORMICAN. 1988. Feature analysis in early vision: Evidence from search asymmetries. *Psychological Review* 95(1):15–48.

TUFTE, EDWARD R. 1983. *The visual display of quantitative information*. Cheshire, CT: Graphics Press.

TUFTE, EDWARD R. 1990. *Envisioning information*. Cheshire, CT: Graphics Press.

TUFTE, EDWARD R. 1997. *Visual explanations: Images and quantities, evidence and narrative*. Cheshire, CT: Graphics Press.

TVERSKY, BARBARA. 1995. Cognitive origins of graphic productions. In *Understanding images: Finding meaning in digital imagery*. Telos.

ULLMAN, SHIMON. 1985. Visual routines. *Cognition* 18:97–159.

WANG, XIAOBO, & I. MIYAMOTO. 1995. Generating customized layouts. In *Proceedings of Graph Drawing '95*, volume 1027 of *Lecture Notes in Computer Science*, 504–515. Springer-Verlag.

WEHREND, STEPHEN C., 1990. A categorization of scientific visualization techniques. Master's thesis, University of Colorado, Boulder.

WEHREND, STEPHEN C., & CLAYTON H. LEWIS. 1990. A problem-oriented classification of visualization techniques. In *Proceedings of IEEE Visualization '90*, 139–143.

WOLFE, J. M., S. R. FRIEDMAN-HILL, M. I. STEWART, & K. M. O'CONNELL. 1992. The role of categorization in visual search for orientation. *Journal of Experimental Psychology: Human Perception and Performance* 18(1):34–49.

ZHOU, MICHELLE X., & STEVEN K. FEINER. 1996. Data characterizaton for automatically visualizing heterogeneous information. In *Proceedings of IEEE Symposium on Information Visualization '96*, 13–20.

ZHOU, MICHELLE X., & STEVEN K. FEINER. 1997. The representation and usage of a visual lexicon for automated graphics generation. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI '97)*, 1056–1062. Morgan Kaufmann.

ZHOU, MICHELLE X., & STEVEN K. FEINER. 1998. Visual task characterization for automated visual discourse synthesis. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '98)*, 392–399, Los Angeles.