AIDE: A Case-Based Approach for Designing Graphics From Locative and Temporal Data

by Craig Michael Kanarick

Bachelor of Arts, University of Pennsylvania, 1989.
Bachelor of Applied Science/Computer Science, University of Pennsylvania, 1989.

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning
in partial fulfillment of the requirements for the degree of
Master of Science in Visual Studies
at the Massachusetts Institute of Technology, September 1993.

signature of author [illegible]
Craig M. Kanarick
Program in Media Arts and Sciences
July 1, 1993

certified by [illegible]
Muriel R. Cooper, B. F. A.
Professor of Visual Studies
Thesis advisor

accepted by [illegible]
Stephen A. Benton
Chairman, Departmental Committee on Graduate Students

AIDE: A Case-Based Approach for Designing Graphics From Locative and Temporal Data

by Craig Michael Kanarick

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning
on July 1, 1993 in partial fulfillment of the requirements for the degree of
Master of Science in Visual Studies
at the Massachusetts Institute of Technology.

abstract

This thesis describes a research system called **aide: automated intelligent design expert** that creates graphics from data. It assists users of data who wish to design graphics by combining artificial intelligence techniques with current automated design software paradigms. The work extends previous research in four ways. First, it combines the artificial intelligence technique known as case-based reasoning with traditional rule-based automated layout techniques. Case-based reasoning is used to choose a graphical template from a library of examples and is also used to instantiate the graphical aspects of the image. This approach is investigated because it is well matched to the process that people take when designing -- adapting old designs to new situations.

Because of their cognitive importance, the rule-based components of **aide** have special design knowledge for *temporal* and *locative* information, allowing the system to present this type of information in unique ways.

Like other automatic layout systems, **aide** is capable of producing traditional static graphics. In addition, however, **aide** can also design and present dynamic graphics.

Finally, **aide** runs in a one-of-a-kind large high-resolution display environment. This, combined with a unique set of design skills and techniques, allows the system to produce graphical images of a higher quality than found previously.

The research presented here contributes to the fields of artificial intelligence and automated graphic design by effectively combining them in one system. **aide**, the resulting prototype application, can be used to generate a number of different high-quality graphical representations from application independent data and offers potential benefits for anyone who needs to analyze or present data.

Thesis advisor: Muriel Cooper, Professor of Visual Studies.

This work was sponsored in part by the Defense Agency Research Projects Administration DARPA), International Business Machines Incorporated (IBM), the Department of Transportation (DOT), and Nynex Incorporated.

AIDE: A Case-Based Approach for Designing Graphics From Locative and Temporal Data

by Craig Michael Kanarick

The following people served as readers for this thesis.

thesis advisor
Muriel Ruth Cooper, B. F. A.
Professor of Visual Studies, MIT Media Laboratory
Director, Visible Language Workshop

reader
Kenneth Haase, Ph. D.
Assistant Professor, MIT Media Laboratory

reader
Joe Marks, Ph. D.
Member of Research Staff, Digital Equipment Corporation
Lecturer in Computer Science, Harvard University

acknowledgments

Many people have helped with this thesis, and I would like to thank all of them.

table of contents

1 introduction

Number 6: What do you want?
Number 2: Information!
Number 6: Whose side are you on?
Number 2: That would be telling.

Number 2: We want information.
Number 2: Information.
Number 2: Information!
Number 6: You won't get it!
Number 2: By hook or by crook, we will.

-*The Prisoner*

No statistical package exists that can take raw data and analyze them a priori. The statistician must have ideas about the kinds of structures present in the data before applying even the most general analytical procedure. Possibly the most powerful tool for this initial exploratory data analysis is the graph.

- Colin Ware and John Beatty

overview

The current era has often been dubbed "the information age." As computers become a more integral part of society, they produce more data with each passing minute. Databases can describe a collection of objects, customers, employees, sales figures, the results of scientific experiments, medical histories, events, places, or just about any other phenomena. Not only is the number of such databases increasing, but their size and complexity are growing as well. Databases, as such, are not inherently useful, for they are just collections of data. However, buried within each database is *information.* We can consider *information* to be, among other things, relevant or important aspects of the data. That is to say that each set of data contains trends, patterns, and relationships among the elements and their characteristics (the lack of an easy-to-describe trend or pattern being a pattern in and of itself). These trends and patterns often tell important stories about the data in question and are needed to solve real problems. The ability to quickly and efficiently discover such information buried in a database is valuable and grows more difficult with its size and complexity

Data can be analyzed in many different ways. However, as Ware and Beatty point out, the most powerful way to analyze a database as a whole and to get a good overall view of the information within a database, especially if the user has no a priori knowledge of the database, is through the use of a graphical presentation (e.g., a chart, diagram, or map).

For example, consider the data in Figure 1.1:

	I		II		III		IV	
N	x	y	x	y	x	y	x	y
1	10.00	8.04	10.00	9.14	10.00	7.46	8.00	6.58
2	8.00	6.95	8.00	8.14	8.00	6.77	8.00	5.76
3	13.00	7.58	13.00	8.74	13.00	12.74	8.00	7.71
4	9.00	8.81	9.00	8.77	9.00	7.11	8.00	8.84
5	11.00	8.33	11.00	9.26	11.00	7.81	8.00	8.47
6	14.00	9.96	14.00	8.10	14.00	8.84	8.00	7.04
7	6.00	7.24	6.00	6.13	6.00	6.08	8.00	5.25
8	4.00	4.26	4.00	3.10	4.00	5.39	19.00	12.50
9	12.00	10.84	12.00	9.13	12.00	8.15	8.00	5.56
10	7.00	4.82	7.00	7.26	7.00	6.42	8.00	7.91
11	5.00	5.68	5.00	4.74	5.00	5.73	8.00	6.89

Figure 1.1. A tabular arrangement of Anscombe's quartet. For each of the four sets, N = 11, mean of x's = 9.0, mean of y's = 7.5, equation of regression line: y = 3 + .5x, standard error of estimate of slope = 0.118, t = 4.24, sum of squares of x = 111.0, regression sum of squares = 27.5, residual sum of squares of y = 13.75, correlation coefficient = .82, and t^2 = .67.

This data, called Anscombe's quartet (Anscombe, 1973; Tufte 1983) contains four sets of data each described by the same linear model. It is difficult to tell by looking at the table above, but even though the four sets are described by the same model, as the visual representations of the data illustrate, each of the four sets is quite different.

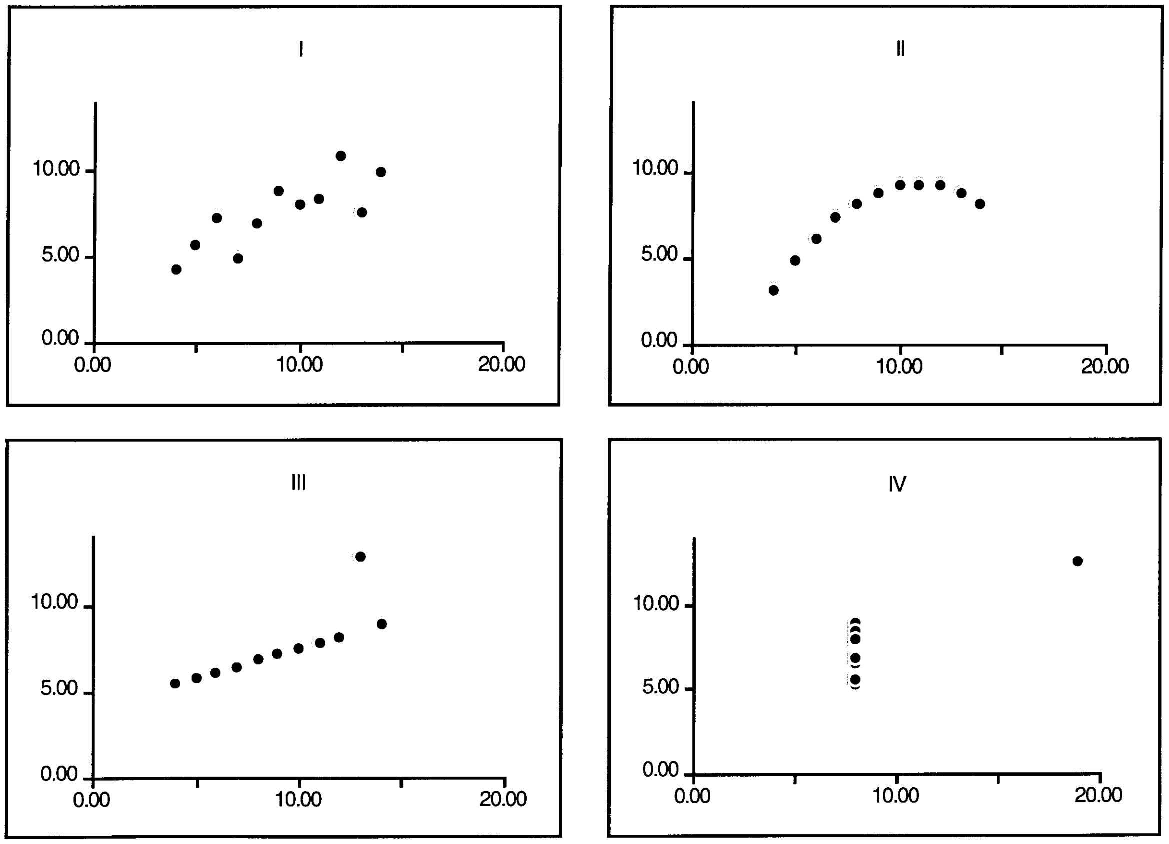


Figure 1.2. A graphical presentation of Anscombe's quartet. Each of the graphs is a scatterplot of x vs. y.

This examples illustrates how much easier it can be for a user to understand a set of data by looking at a visual representation than a tabular arrangement. It is easier to recognize patterns, determine trends, or discover other information by using our perceptual skills for recognizing shapes, colors, and positional arrangement of objects than it is to parse a table of textual information. Graphical images are often more inviting than tables of data and allow the user an opportunity to investigate and examine data in a way just not available with tables. In addition, graphics can often condense data, allowing a view to analyze thousands or millions of data in one visual experience. For hundreds of years people have been making such charts and diagrams from data.

However, the variations on graphical forms are limitless. Choosing the appropriate graphical form for a particular set of data is no easy task. Each aspect of the design, the color of the marks, the width of the lines, the size of the labels, all of these communicate information and play a role in the effectiveness of the graphical presentation.

For example, Figure 1.3 shows the same data in two different formats. The image on the left shows an obvious pattern, while the one on the right does not. One could argue that the left image is more useful (or more *effective*) than the one on the right.

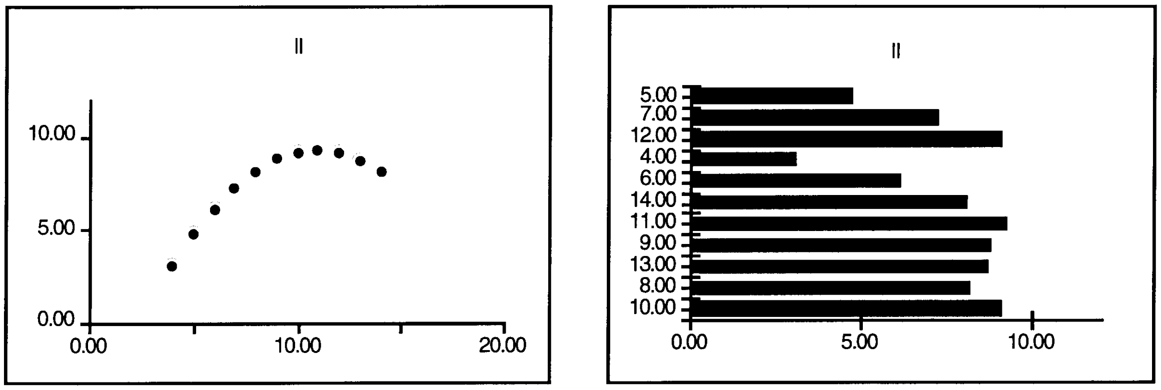


Figure 1.3. Two different graphical presentations of the same data from Anscombe's quartet. Both graphs are visualizations of Group II, however, the graph on the left is a scatterplot of x vs. y, while the graph on the right is a barchart of N vs. y.

Trained graphic designers are specialists in this area, and are often given the responsibility of turning tables of data into powerful images that illustrate the underlying information in the database. This process of turning data into images involves choosing the format of the graphic, choosing the colors and symbols that comprise the graph, choosing the fonts, and hundreds of other decisions that on the surface might seem simple, but are in reality extremely complex -- for each decision has a direct impact on the legibility, expressivity, and effectiveness of the final graphical image.

Since the need to go from data to image is so common, it is hardly cost-effective or time-effective to hire a trained designer every time a user needs a graphic. As a result, the ability to visualize data has been given to the general user through production software, spreadsheets, and statistical tools.

There are two major drawbacks to the current ways of putting this power in the users' hands. First, the graphical *intelligence* of these systems is severely limited. The user must often make many design choices, including which form to use (scatterplot, pie chart, bar chart, etc.), which characteristics should go on the x axis, which on the y axis, what the tick marks should be like, and so on. The user is often not a trained designer, and hence, is not equipped with the skills needed to produce effective graphics.

The second problem is the graphical *power* of such programs is limited. This means that even if the user makes reasonable design choices, the graphical quality of the images that can be created is not very high. In most cases, the system just doesn't provide enough control over the individual design aspects of the image, such as the size of the marks, the colors, and so on.

What are possible solutions to these problems? Providing more control over the various design aspects would solve the second problem, but would make the first even more acute. However, if we could abstract the process of turning data into graphs and generate a set of rules for this process, these rules could be codified in software and a virtual designer could perform the job of a trained graphic designer. Current research in this area has met with limited, although significant, success. One of the biggest problems is that the design process is too complex and large to be codified in a set of simple rules. It does appear that designers follow general guidelines and styles when designing. However, these styles and guidelines are so complex, it seems unlikely that they could be delineated in a comprehensive list of strict conditional rules.

An alternative theory about the process of design is that designers borrow ideas from other designs -- that is to say they create new designs based on old ones. Consider a particular diagram that communicates information about some specific set of data effectively and is easy to understand. If the data changes slightly (i.e., if the data is about horses, maybe the height of a horse changes), it would not be useful to completely redesign the graphic. Similarly, if a *few* of the values in the dataset change, it would not make sense to completely redesign the graphic. The alternative is merely to *modify* the original visualization to reflect the new state of the data. As such, these new images could be considered to be "based" on the original. Design often happens this way -- adapting other graphics to the state of the "new" data. The more different the new data from the old, the more redesign is needed. At some point, when the new data is significantly different from the original, a completely new design is required.

This idea is further illustrated in Figure 1.4. In this figure there are four graphs, all graphed in the same form. The values of the data being represented varies greatly from graph to graph, however. In the two left graphs the images represent time (year) and dollars, whose values range from -640,000,000 to 2,540,000,000. In the third graph, the image also represents time (year) and dollars, yet the dollar values range from -.91 to 3.6. The graph on the furthest right represents time (year) and percentage (values ranging from -3.5 to 10.5). Each of the four graphs displays time against some numerical value, yet the magnitude of the numbers varies drastically from image to image. However, the same graphical style is useful for each of the four datasets.

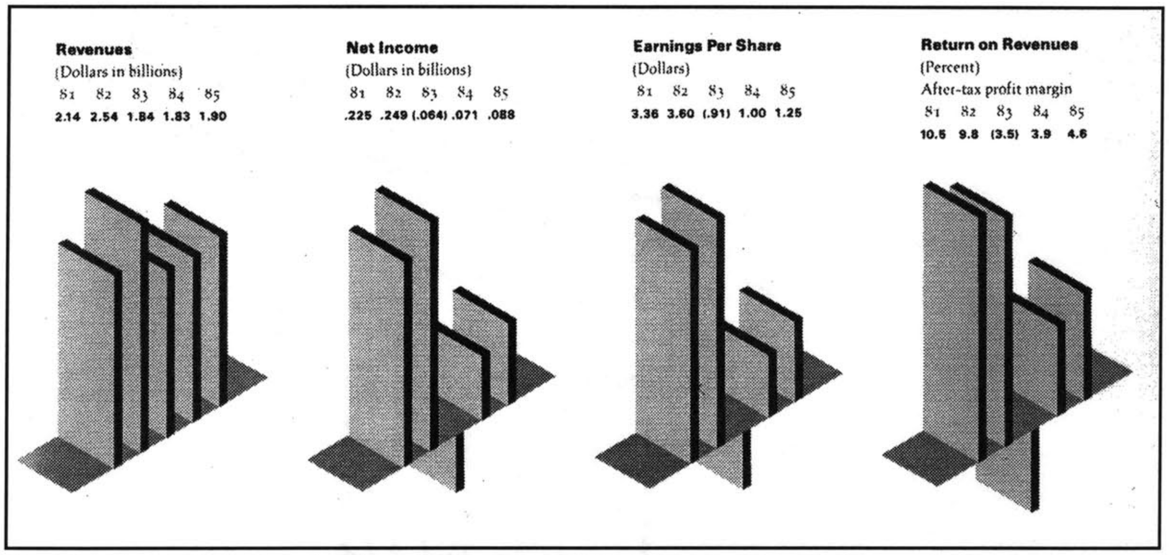
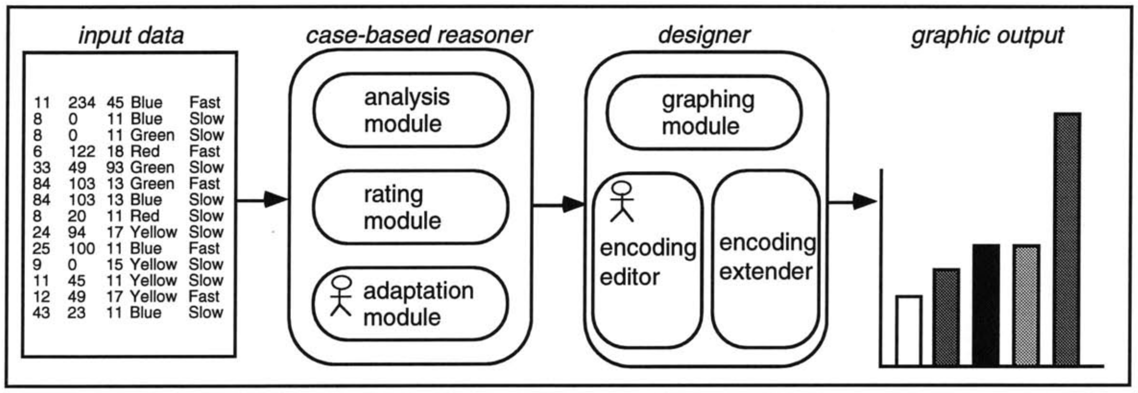


Figure 1.4. Four similar barcharts from the 1985 annual report of Baker International. (Graphis Diagram, 1988).

In the example above, each of the four datasets was illustrated using the same basic design. This was possible because the datasets were relatively similar. However, if one of the datasets were extremely different, a new design would have been needed. For example, if the revenues were categorized into "great", "good", "fair", and "bad", instead of being given in billions of dollars, the barchart shown above would not have been able to convey this information appropriately. As stated earlier, the more different the new data from the old, the more redesign is needed.

It is possible to use an artificial intelligence technique called case-based reasoning to model this behavior. This thesis describes a prototype of such a system, called **aide: automated intelligent design assistant. aide** is a system that uses case-based reasoning to automate part of the process of graphic design. **aide** takes a set of application-independent data as input and has a library of different data sets and their graphical representations. It compares the new data to the examples in the library, and based on input from the user, attempts to create a new design based on one of the examples. As an assistant, it does not merely take in data and produce graphics, but *assists* the user in the process of generating statistical graphics from data. This means that the user is an integral part of the design process. Figure 1.5 shows a very general block diagram of the system.[1] The stick-figure () indicates areas where user input is required.



11	234	45	Blue	Fast
8	0	11	Blue	Slow
8	0	11	Green	Slow
6	122	18	Red	Fast
33	49	93	Green	Slow
84	103	13	Green	Fast
84	103	13	Blue	Slow
8	20	11	Red	Slow
24	94	17	Yellow	Slow
25	100	11	Blue	Fast
9	0	15	Yellow	Slow
11	45	11	Yellow	Slow
12	49	17	Yellow	Fast
43	23	11	Blue	Slow

Figure 1.5. A general block diagram for ***aide.***

How does the system work? The first half of the process utilizes the artificial intelligence technique called case-based reasoning. The *analysis module* takes the input database and produces an abstract representation of the data, called the *shape*, by performing some traditional statistical analyses of the data and their characteristics. This analysis includes sorting the characteristics into *nominal*, *ordinal*, and *quantitative* categories. In addition, **aide** extends this paradigm by treating two *aspects* of characteristics as special. These aspects are *locative* and *temporal*, and are singled out because of their special cognitive value. People have a natural tendency to organize things with respect to places and time. Hence, any system that tries to design like a human being must have knowledge of these aspects, and appropriate design rules to visually represent this data.

Once the system creates a simple description of the input database, it then compares the input database's shape to a library of example graphs. The *rating module* compares the *shape* of the input data to each of the examples' *shapes* to assess each as a possible basis for the visual representation of the input database. **aide** visually displays a subset of the examples along with their ratings and in doing so, provides the user with guidance about how to proceed. At this

[1] A more complete schematic of the system can be found in Chapter 5.

point, the user chooses an example and the system passes the input data and the chosen example to the *adaptation module* that modifies the chosen example to fit the input database.

Once the example is adapted, **aide** then passes the adapted example and the input data to the *designer component*. This component lays out a large, full-color visual representation of the data based on the adapted encoding and provides the user with an easy and effective way to discover the information buried within the data. The user can then choose another example from the library, modify the graph manually, or in relevant situations, ask the system to try to "extend" the graphic to convey more information[2], through the use of a rule-based design component similar to those found in other work. Once a user is satisfied with a graphic, he can add the new image to the example library for future use.

aide extends previous research in a number of important ways: 1) It combines the artificial intelligence technique known as case-based reasoning with traditional rule-based automated layout techniques; 2) Because of their special cognitive value, the rule-based designer component of **aide** treats *temporal* and *locative* information as special and contains special design rules to represent these characteristics; 3) Just as other automated layout systems, **aide** can design and present static images, but the system is also capable of producing dynamic graphics; and 4) **aide** contains a number of sophisticated new design skills that combined with its unique display environment allow the system to create graphics of a higher quality than in previous work.

This thesis is composed of different sections. Chapter 2 contains a description of the user interface and how the system looks and feels to a user. Chapter 3 contains an overview of related work. Chapters 4 and 5 delve into the "behind-the-scenes" activities of **aide** -- Chapter 4 explains the case-based reasoning aspects of **aide** and Chapter 5 describes the graphing aspects of the system. Chapter 6 concludes the document with a discussion of performance criteria, a summary of the contribution of **aide** to various fields of research and a discussion of areas for future work.

aide was developed on the Large High Resolution Display (LHRD) Prototype, a 6044x2048, 100 pixel-per-inch display prototype developed in the Visible Language Workshop (VLW) at the MIT Media Lab (Mashiushi, et. al, 1991). The LHRD is composed of hardware including: Sony 2000-line monitors, frame-buffers made by Metheus, and an IBM RS/6000 workstation running AIX/3.2. The VLW window system, BadWindows version 2.0, and a number of VLW libraries were used to create the user interface. All other code was written by the author in C.

[2] **aide** has a "goal" to try and represent as much information about the data as possible. To this end, it tries to represent as many of the database's characteristics graphically. If the user chooses an example that visually presents fewer than all of the characteristics, it is possible in some cases to "extend" the image to provide more information.

2 the user interaction

One of the features of **aide** is a complex feedback loop that makes the user an integral part of the design process. **aide** is not a completely automated designer. Instead, it is meant to be used as an *assistant* in the design process. This section describes the user interface and interaction with the system.

the screen

The screen is organized into five primary sections. There is an area where the input data is presented to the user along with some information about its *shape*, an abstract description of the characteristics of the database. There is an area where the user can view the library of examples and a measure of how "good" the system thinks that example would serve as a basis for a visualization of the input data. There is a large section of the screen used by the designer component to draw the graphic, and a *key* to the graph. In addition, there is a visual history area of the screen. Figure 2.1 shows an image of the entire display.

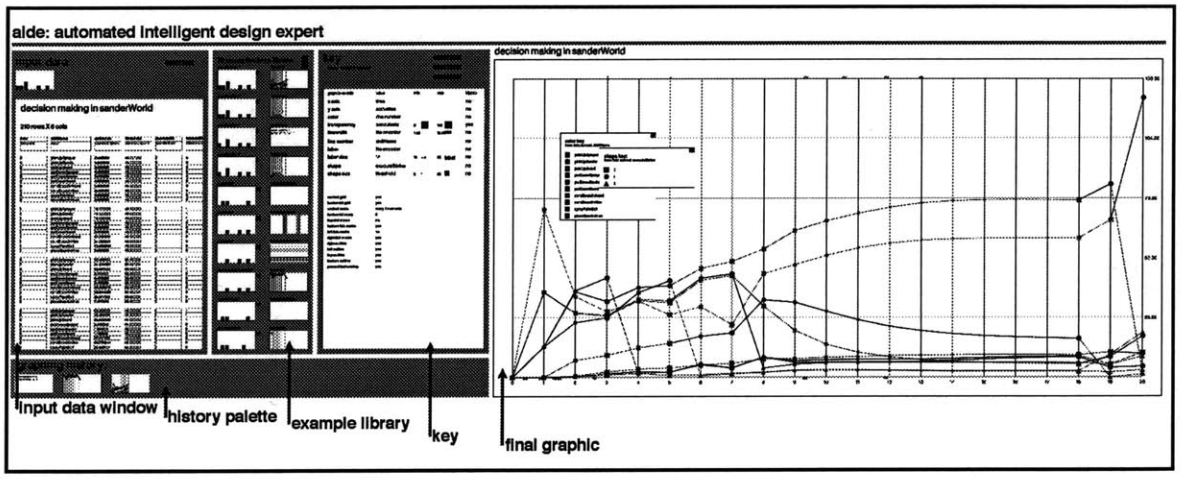
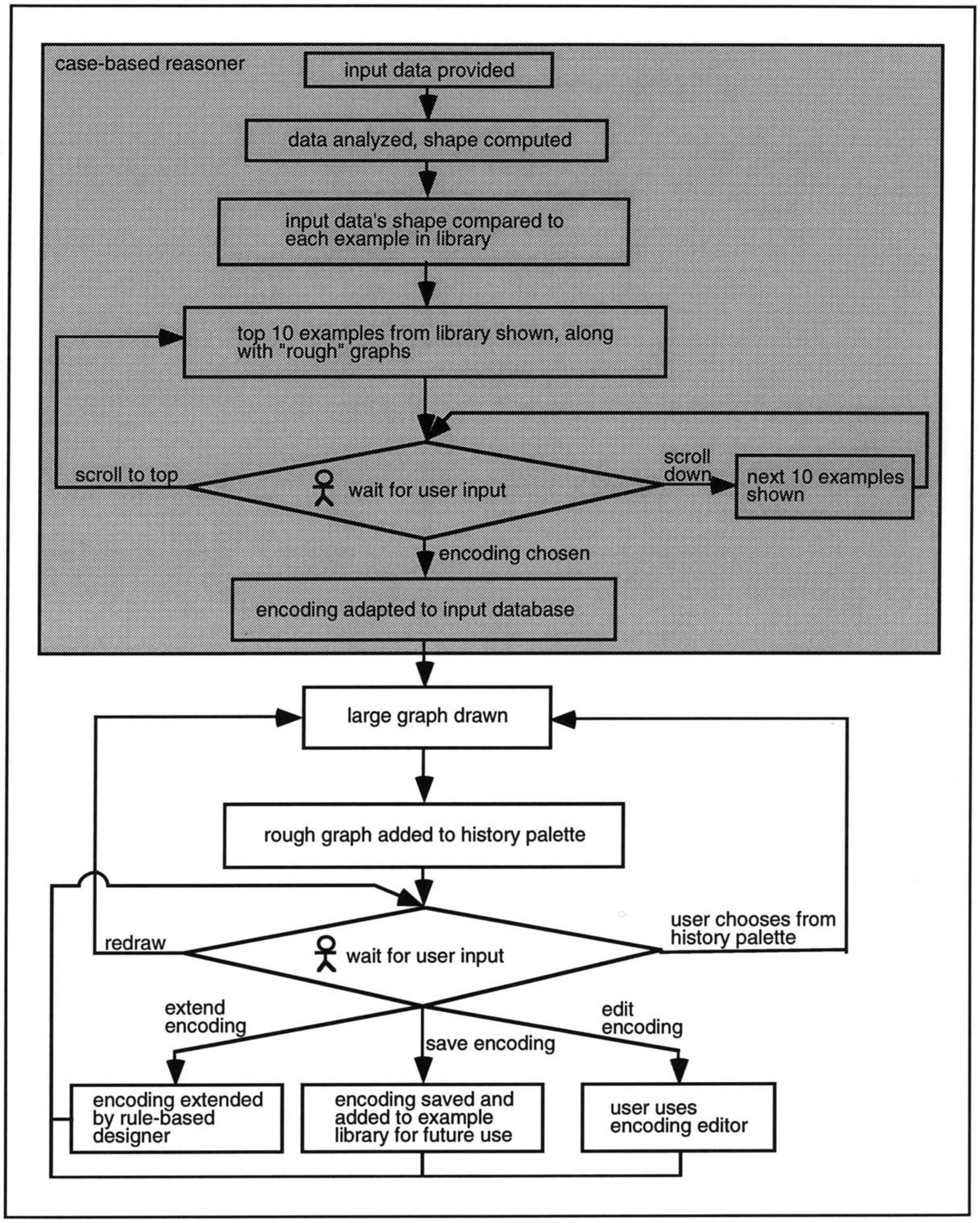

aide: automated intelligent design expert
decision making in sanderWorld
decision making in sanderWorld
input data window
history palette
example library
key
final graphic

*Figure 2.1. **aide**'s user interface.*

The section on the far left is an area for displaying the input data in tabular format. Directly to the right of this is a visualization of a subset of the example library. To the right of this is a the *key*. The large area to the right of this is the area for the fully rendered *final* graphic. In the lower left corner of the display is the *history palette*, where rough copies of the final graphics produced are archived. Each of these areas of the screen is described in more detail in this chapter.

aide is a highly interactive system. Once the user provides the system with some input data, he has a lot of control over how the system graphs the data. Figure 2.2 contains a schematic describing the interaction between the user and the system.



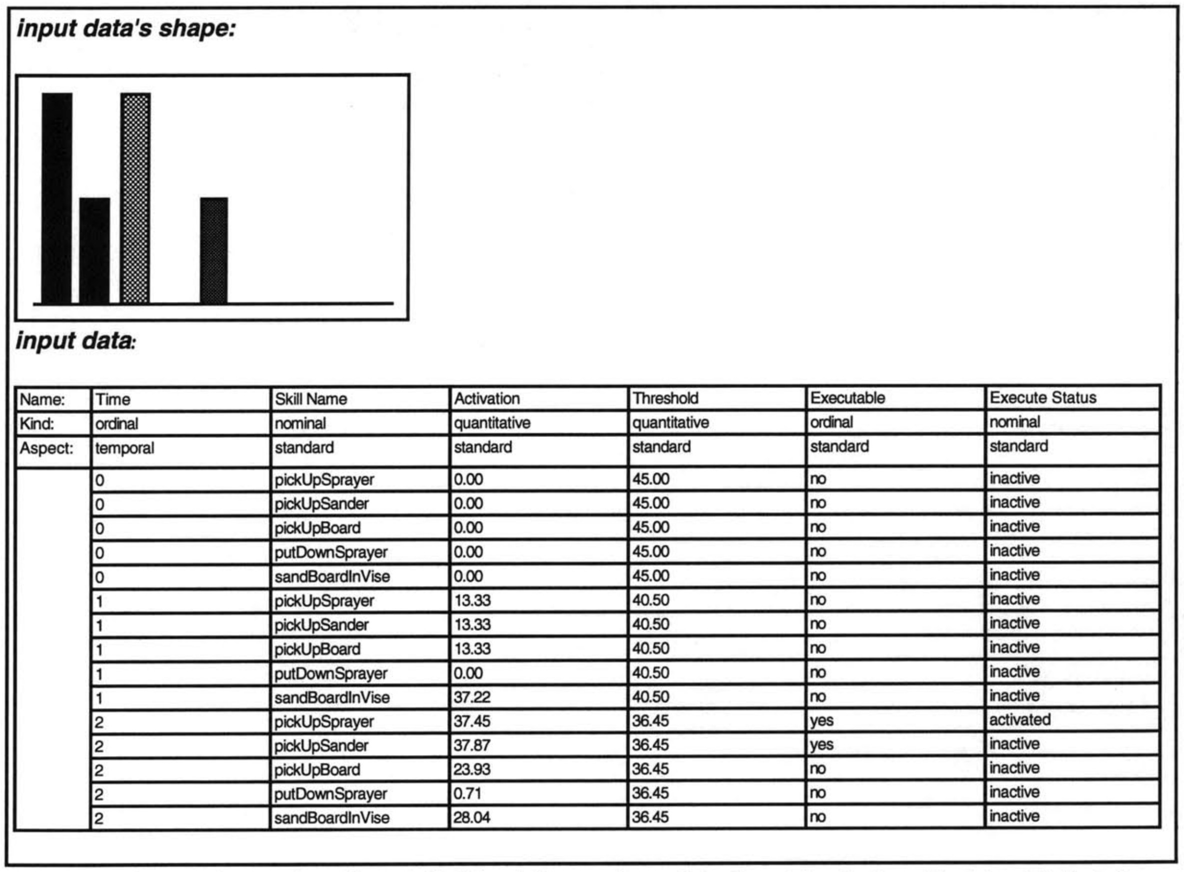
case-based reasoner
input data provided
data analyzed, shape computed
input data's shape compared to each example in library
top 10 examples from library shown, along with "rough" graphs
scroll to top
wait for user input
scroll down
next 10 examples shown
encoding chosen
encoding adapted to input database
large graph drawn
rough graph added to history palette
redraw
wait for user input
user chooses from history palette
extend encoding
save encoding
edit encoding
encoding extended by rule-based designer
encoding saved and added to example library for future use
user uses encoding editor

*Figure 2.2. A schematic for **aide**.*

the input data

When **aide** first starts up, it displays the input data to the user in the *input data window*. The data is displayed in a spreadsheet-like format in order to give the user an opportunity to see the data as a table. In the current implementation, the user can only look at the data in the tabular format. Future work includes adding the ability for the user to edit the input data, select subsets of the data to be graphed, and other similar tasks.

While the tabular format can be useful, **aide**'s main purpose is to assist in designing more powerful and informative visualizations. **aide** rests on the theory that the content of the data is not as important to creating a good design as the structure of the data itself. After the system displays the input data as a table, it computes an abstract description of the input data, called its *shape*. The *shape* of the data includes the number of different *kinds* (*nominal, ordinal,* or *quantitative*) of characteristics and the number of the different *aspects* (*temporal, locative,* or *standard*) of characteristics are in the database as well as some other statistical values. Once the *shape* of the input data is computed, a barchart visualization of it is presented to the user. Figure 2.3 shows the *input data window*, including part of an example input database and a visualization of its shape.



input data's shape:

input data:

Name:	Time	Skill Name	Activation	Threshold	Executable	Execute Status
Kind:	ordinal	nominal	quantitative	quantitative	ordinal	nominal
Aspect:	temporal	standard	standard	standard	standard	standard
	0	pickUpSprayer	0.00	45.00	no	inactive
	0	pickUpSander	0.00	45.00	no	inactive
	0	pickUpBoard	0.00	45.00	no	inactive
	0	putDownSprayer	0.00	45.00	no	inactive
	0	sandBoardInVise	0.00	45.00	no	inactive
	1	pickUpSprayer	13.33	40.50	no	inactive
	1	pickUpSander	13.33	40.50	no	inactive
	1	pickUpBoard	13.33	40.50	no	inactive
	1	putDownSprayer	0.00	40.50	no	inactive
	1	sandBoardInVise	37.22	40.50	no	inactive
	2	pickUpSprayer	37.45	36.45	yes	activated
	2	pickUpSander	37.87	36.45	yes	inactive
	2	pickUpBoard	23.93	36.45	no	inactive
	2	putDownSprayer	0.71	36.45	no	inactive
	2	sandBoardInVise	28.04	36.45	no	inactive

Figure 2.3. The input data window. The top half of the window contains a tabular format visualization of the data, while the bottom half contains a barchart representation of the data's shape.

This data is a set of values from *sanderWorld*, (Johnson, unpublished) a simple virtual environment in which a robot consisting of various autonomous skill agents is given two potentially conflicting goals: to spray paint itself and to sand a board (Maes, 1990). The difficulty arises because the robot has limited resources and constraints on its actions. The data represents the various parameters of each of the individual agents during a run of the system over a span of time, including the name of the agent, its "activation" level (how badly it "wants" to execute), its threshold for activation, and its execute state.

looking at the library of examples

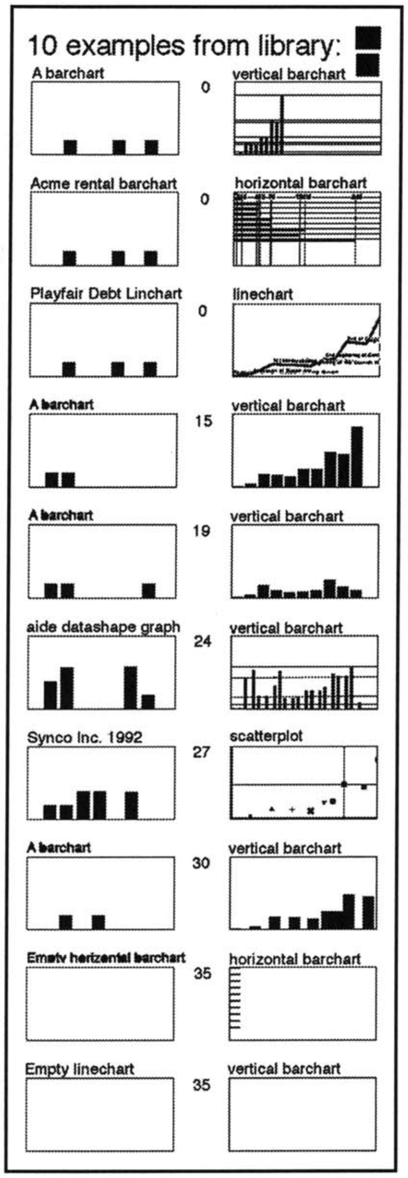


Figure 2.4. A visual representation of the ten best-rated examples from the library of examples.

Once **aide** computes the *shape* of the database, it uses this as a basis for suggesting an example on which to base the graphical visualization of the input data. **aide** contains a library of examples, each of which includes a graphical image, the *shape* of the database visualized in that example, and an abstract description of the design of the image, called the *encoding*. After computing the input data's *shape*, it is compared to each of the example's *shapes*, and each example is rated as a possible basis for the new visualization. This process is described in detail in Chapter 4. Once each example is rated, the ten most similar examples are presented to the user along with their rating, providing guidance about how to continue. Figure 2.4 illustrates how examples are presented.

The column on the left shows a graphical representation of the *shape* of the example. It is a rather simple graphic: since there are three *kinds*[3] and three *aspects* [4] of characteristics, there are nine *kind-aspect* combination. The graphical representation of the *shape* is a simple barchart showing the number of each type of characteristic in the database. To the right of each of these graphs is a small rough draft of what the graph would look like if the user chose that example. The number between the two images is the *rating*, or difference, between the *shape* of the input data and the *shape* of the data from the example. If the user is unsatisfied with any of the examples shown, he can scroll through the library by using the two scroller buttons in the upper right corner.

By providing the user with rough drafts of the potential graph, the user can quickly compare the different potential graphics and choose one to his liking. This pre-visualization step can save the user valuable time, as the rendering of the final graphic can often take a significant amount of time, especially in the case of dynamic graphics. To create a *final graphic* from one of the examples shown, the user need only click on the rough draft. After the user chooses an example, the system proceeds to draw the larger, fully rendered version of the graphic.

---

[3] *Nominal, ordinal,* or *quantitative.* A complete description of these kinds is given in Chapter 3.

[4] *Temporal, locative* or *standard* (neither *temporal* nor *locative*).

"final" graphics

After the user chooses an example from the database, the system adapts that example to the input data. A detailed description of this adaptation process is given in later chapters.

One of the problems with traditional data analysis packages that allow a user to "graph" data is that the graphical options are fairly limited. Assume that you are a user of a typical spreadsheet package and you have a large set of data (many rows and columns) that you wish to visualize. If you choose to make a scatterplot of the data, for example, you are severely limited in terms of how much control you have over the image. With most programs, you could only choose which characteristics to place on the x and y axes. However, there are many other aspects of the graph that are important parts of the design -- the tick marks, the grid lines, the size, color, transparency, and texture of the marks, and so on. Design aspects such as the size and color of the marks, however, can carry additional information about the data. Most visualization packages, however, don't allow the user to assign a characteristic of the data to the color or size of the marks. The most common scenario is that the machine picks a value and assigns all the marks the same size and color. In some advanced systems, the user can set the color and size of the marks individually, but must do so for each mark on the graph. In the first situation, the graph is not as expressive as it could be. In the latter case, the user just has to work to hard to make the graph expressive. For a system to be able to make expressive statistical chart graphic, it must be able to have characteristics of the data assigned to as many graphical aspects of the image as possible.

In **aide**, there are a large number of graphical aspects of the images that can convey information about the input data. These graphical aspects are called the set of *graphic variables* and differ for each graphical form (scatterplot, linechart, barchart, etc.) Each of the graphic variables can be instantiated in a visualization in different ways -- they can either be a constant value or they can be a function of one of the characteristics of the input data. For example, in a scatterplot of the *sanderWorld* data, the shape of the marks could be determined by the characteristic "executable".

As described earlier, each of the examples in the library contains a graphic image, the *shape* of the data for that graphic, and an abstract description of the design of the image. This description is called the *encoding* of the graph. An *encoding* contains a list of the *graphic variables* for the specific graphical form, and a list of *encoders* that describe how each of the *graphic variables* is instantiated. An *encoder* can be assigned a constant value, a description of a characteristic in a database, or be empty (in which case the system chooses a reasonable default value). In addition, the system can assign the same characteristic to more than one *graphic variable.* For example, both the color *and* shape of marks in a scatterplot could be determined by the same characteristic of a database.

Once the user chooses an example from the library, **aide** adapts its *encoding* to the input data (this process is described in the section about the *adaptation module*). Once the *encoding* has been adapted to the input data, the system renders a large graphical visualization of the input data in a 4000x2000 pixel window. This large image is referred to as a *final graphic.*

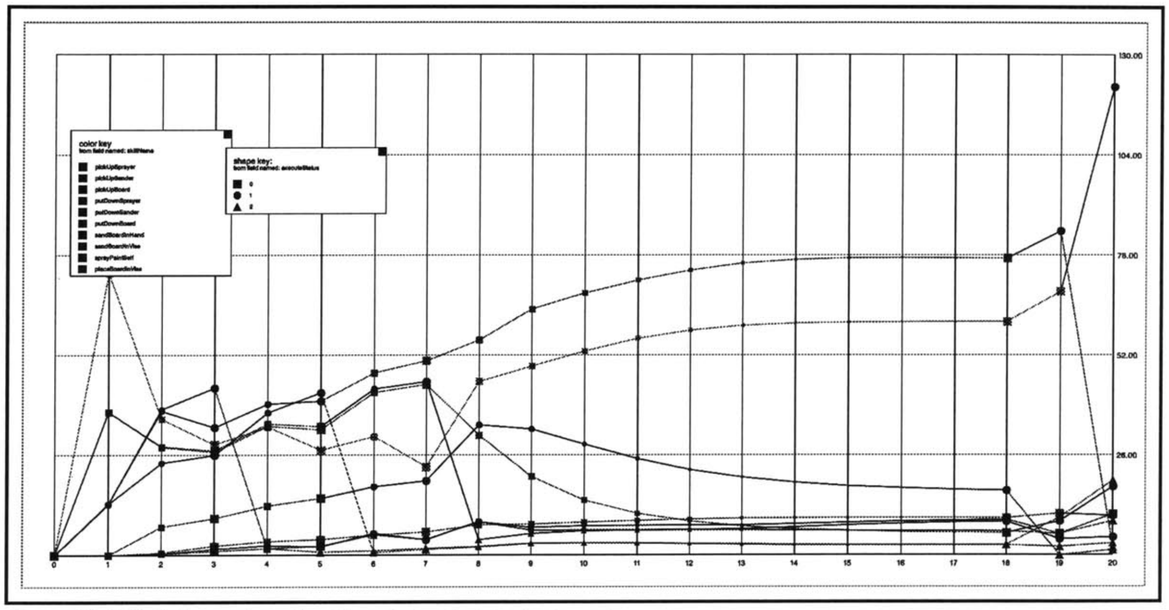


Figure 2.5. This is the graph produced by aide for the data about agents.

Figure 2.5 shows a visualization of the data from the *sanderWorld* dataset. In the image, the characteristic named "Time" is assigned to the x-axis and "activation" is assigned to the y-axis. Both graphic variables "line number" and "color" are instantiated from the characteristic named "Skill Name". The shapes of the plot marks are determined by the characteristic "execute status", and the transparency of the line and the marks is determined by the characteristic named "executable". As a result, all of the characteristics of the database are represented visually in different visual aspects of the image.

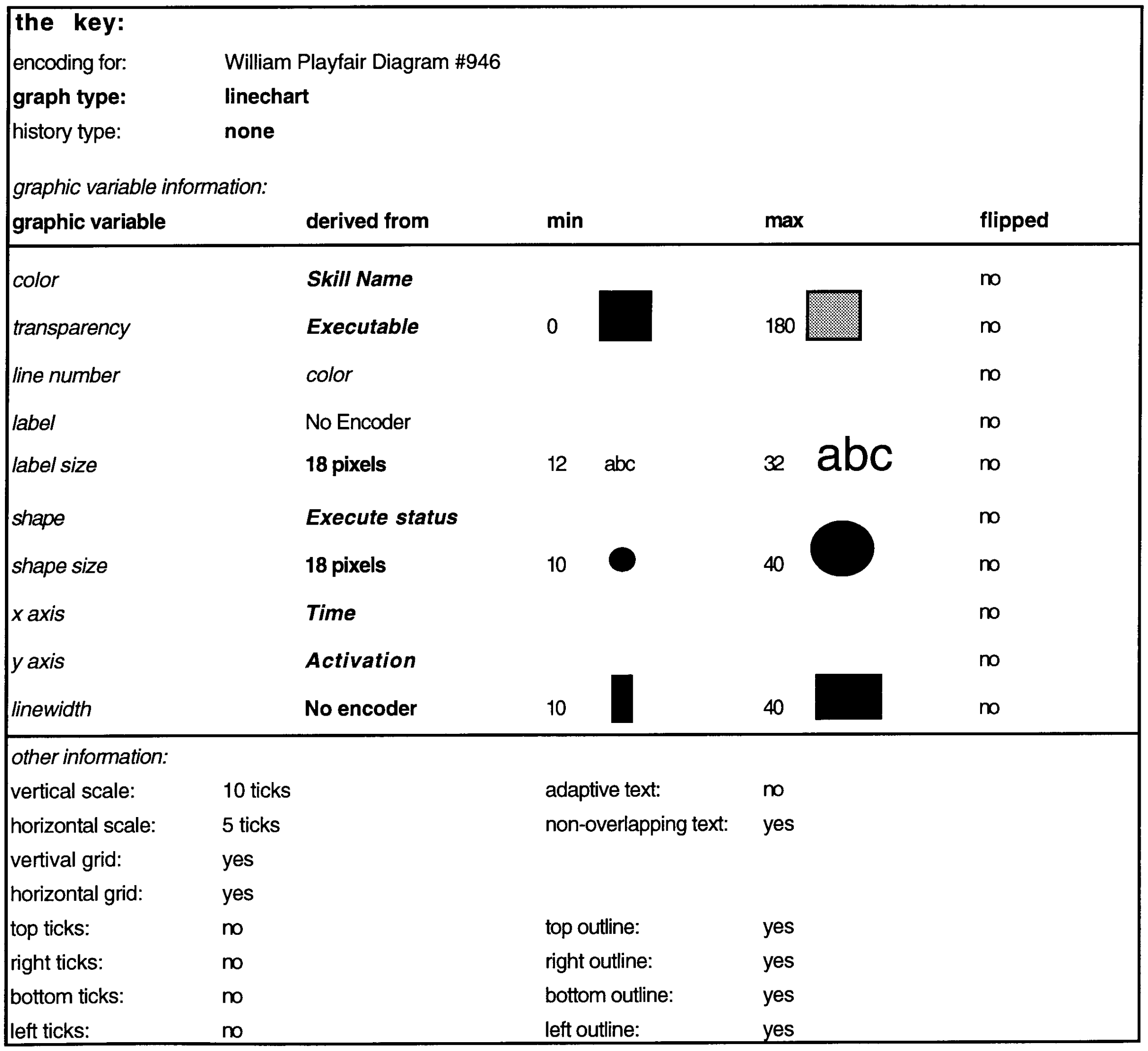
This is an effective way of displaying information because of the natural ability of humans to perceive the different visual components of the image separately. For example, if two objects have the same color but different shape, a typical person can still tell them apart. Humans have the ability to filter, as it were, based upon one or more visual property. By doing so, the user can quickly analyze the data and discover the information within the image.

aide was used to visualize the data from a few different runs of the *sanderWorld* system in order to find a problem with the robot. A situation arose in which the robot refused to activate a particular agent, even though all of its conditions for activation appeared to have been met. After spending a number of hours perusing the code and looking at the data in the tabular format, the bug could not be discovered. However, only a few seconds after **aide** presented the information visually, the bug was spotted. By being able to see the data from all of the agents from the entire run at the same time, finding the problem took only a few seconds, and provided a solution to a problem that would have been much more difficult to discover with other methods.

the key

As the system renders the *final graphic* version of the data, a *key* is displayed to the user, showing how the system creates the graphic.

aide's key is a simple one: it lists all of the *graphic variables* and a simple explanation of how they were instantiated. At the top part of the key, the information for each of the *graphic variables* is listed. On the color display, each row is color coded to reflect whether it is assigned a constant value, derived from a characteristic of the data, or derived from another *graphic variable*. At the bottom half of this window, other information included in the *encoding* is displayed, including the information about the grid, axis scales, and tick marks. Figure 2.6 shows a key from the linechart visualization of the *sanderWorld* database shown earlier.



the key:

encoding for:	William Playfair Diagram #946
graph type:	**linechart**
history type:	**none**

graphic variable information:

graphic variable	**derived from**	**min**		**max**		**flipped**
color	***Skill Name***					no
transparency	***Executable***	0	■	180	▨	no
line number	*color*					no
label	No Encoder					no
label size	**18 pixels**	12	abc	32	abc	no
shape	***Execute status***					no
shape size	**18 pixels**	10	●	40	●	no
x axis	***Time***					no
y axis	***Activation***					no
linewidth	**No encoder**	10	▮	40	■	no

other information:

vertical scale:	10 ticks	adaptive text:	no
horizontal scale:	5 ticks	non-overlapping text:	yes
vertival grid:	yes		
horizontal grid:	yes		
top ticks:	no	top outline:	yes
right ticks:	no	right outline:	yes
bottom ticks:	no	bottom outline:	yes
left ticks:	no	left outline:	yes

Figure 2.6. The key provides the user with information about the adapted encoding.

In this example, the characteristic named "Time" is assigned to the x-axis and "activation" is assigned to the y-axis. Both graphic variables "line number" and "color" are instantiated from the characteristic named "Skill Name". The shapes of the plot marks are determined by the characteristic "execute status", and the transparency of the line and the marks is determined by the characteristic named "executable".

Even though it appears as a simple list of the graphical aspects of the image, the key is an *active key*, as clicking on different areas of the key provide the user with more detailed information. For example, if the user clicks on any of the names of the graphic variables, a visual presentation of the values used for that graphic variable is shown to the user. In the key shown in Figure 2.6, if a user clicked on the word "color", the following color key could pop up:

Color key:
(derived from characteristic named "Skill Name")

□	pickUpSprayer
▒	pickUpSander
▓	putDownSander
■	sandBoardInVise
▨	pickUpBoard

Figure 2.7. A color sub-key.

keeping track

For each *final graphic* drawn, a rough copy of the graphic image is copied to the "history palette" displayed at the bottom left corner of the screen. This palette provides the user with a visual history of the steps taken to produce the current graphic. The user can click on any of the rough graphs in the history palette and the system will redraw the "final" version of that graphic. Figure 2.8 shows an example history palette from a session with **aide**.

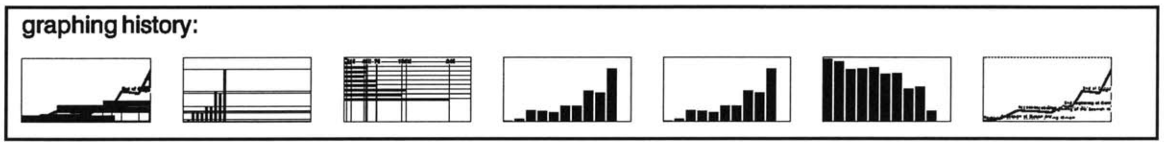


Figure 2.8. The history palette.

In the example above, we can see (reading from left to right) that the user first created a linechart, then a vertical barchart, then a horizontal barchart. He then created three similar vertical barcharts. The graph on the right end is another linechart. In the current implementation, there is space for ten rough drafts (even though the example above only shows seven). The program will soon be extended to allow for more rough drafts to be saved and for the user to be able to scroll through the history palette.

3 previous related work

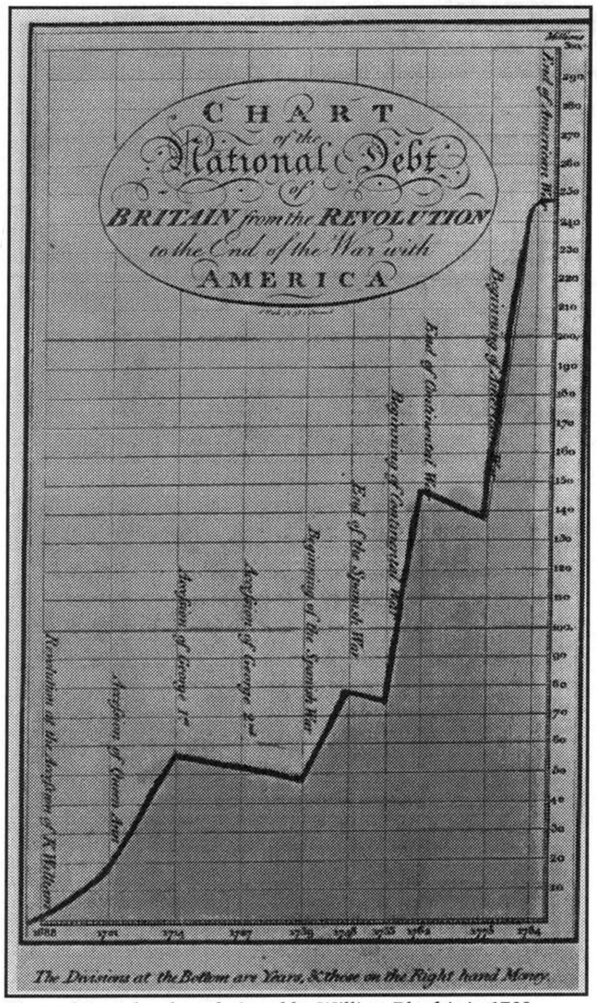


Figure 3.1. A barchart designed by William Playfair in 1780.

The use of graphs, bar charts, scatterplots, and other types of diagrams can be traced back to at least the late 1700's and the work of Playfair and De Fourcroy (Playfair, 1786; de Dainville, 1958). An example of one of Playfair's earliest works is shown in Figure 3.1, to the left. At least three areas of research in the field of statistical graphics design need to be examined: systematic algorithms for creating static visualizations of information, perceptual psychology studies of statistical graphics methods, and the implementation of computer generated visualizations of information.

the graphical presentation of static information

Over the years there have been many efforts to formalize the process designing statistical charts and maps. One of the most comprehensive and acclaimed efforts in this area comes from Jacques Bertin and is outlined in his *Semiology of Graphics* (Bertin, 1983). In this work, Bertin describes a series of general guidelines for creating graphs and charts. Much of his work rests on the theory that while the content of the data is important, characteristics can be grouped into general kinds, and that the distribution of these characteristics within a database is just as important, if not more so.

The first step of Bertin's methodology involves an analysis of the characteristics of the data as a whole. This analysis is a multi-step process beginning with computing the number of "components" for the elements in the database.[5] The next step is sorting each of these characteristics into three categories, *nominal*, *ordinal*, and *quantitative*. This categorization is mostly a description of the granularity of the values for the characteristic in question.

[5] Bertin uses the word "component" to mean some sort of characteristic of an element of the database which can vary from element to element (e.g., sex, width, length, speed, age, name, etc.). I use the word "characteristic" for this same notion.

These three categories are defined as follows:

- *nominal*: the values of *nominal* characteristics fall into non-ordered categories. These characteristics are sometimes referred to as *qualitative* .
- *ordinal*: ordered characteristics where the distance between the different possible values is the same for every possible value
- *quantitative*: the values of *quantitative* characteristics are actual values, usually numerical.

For example, let's consider a database consisting of Olympic medalists, their physical characteristics, and their favorite brands of soft drinks. Characteristics like "height "and "weight" would be quantitative. The kind of medal awarded (i.e., gold, silver, bronze) would be ordinal, as these values aren't real-world quantities, but are sortable categories. Favorite brand of soft drink, however, would be a nominal characteristic, as it is not a quantity, nor are the values ordered (e.g., is Coke more or less than Pepsi?).

For every characteristic, the *length* of each characteristic (the number of instantiated values for that characteristic) is computed. For example, if one of the characteristics of an element was "sex," the *length* of this characteristic would be either 1 or 2 (if all of the data elements were only male or female, the *length* would be 1. If there were some of each, it would be 2). For *quantitative* characteristics, the *range* of the values is also computed, which is the ratio between the largest and smallest value for that characteristic.

Bertin then defines a visual system, composed of a small set of fundamental retinal variables[6]: x-dimension, y-dimension, size, value, texture, color, orientation, and shape. The rest of his work is devoted to guidelines for diagram construction based solely on the previously described analysis of the data. The main task of this construction is choosing the correct graphic variables for the particular characteristics of the database. As Bertin explains (Bertin, 1983),

> ...any retinal variable can be used in the representation of any component. But it is obvious that each variable is not suited to every component. It is the notion of level of organization which provides the key to solving this problem.

Bertin's guidelines outline how to instantiate graphic variables from characteristics of the data to produce effective presentations of the data. One of the strengths of his system is noticing that certain graphic variables are not useful for particular types of characteristics. In order to explain and justify this assertion, he defines "efficiency" as a metric for judging visualizations, which is the application of Zipf's notion of "mental cost" applied to the visual domain (Zipf, 1935). Bertin states that the most efficient visualization is one with which "any question, whatever its type and level, can be answered in a single instant of perception, that is, in a single image."

---

[6] Referred to herein as *graphic variables.*

The basis for Bertin's guidelines is summarized in a simple set of relationships between graphic variables and the notions they can convey. For example, it is easy for people to recognize when two hues are different or similar, but impossible to order hues, as it is meaningless to try to asses whether one hue is "more" or "less" than another (e.g., is red more or less than blue?). He outlined relationships between the graphic variables and the perceptual properties "similar", "different", "ordered", and "proportional". These relationships provide structure for his Semiology, and is the basis for his design paradigm. For example, since color hue is not ordered, one should not use color to represent ordered values.

Edward Tufte, a statistician, has studied graphic design principles in depth, paying particular attention to the effectiveness of graphical presentations of data. He has produced a number of guidelines, which he calls his "Theory of Data Graphics," regarding the elimination of extraneous information in chart graphics to make diagrams more "effective." Tufte outlines a number of general principles aimed at increasing "the number of dimensions that can be represented on plane surfaces" and increasing the amount of information per unit area of statistical graphics. (Tufte, 1983, 1990). He summarized his guidelines as follows:

- Above all else, show the data.
- Maximize the data-ink ratio.
- Erase non-data-ink.
- Erase redundant data-ink.
- Revise and edit.

perceptual psychology and statistical graphics

Most of Bertin's work was based on his own feelings about color, texture, sizes and the types of information that they could convey. Cleveland and McGill, however, actually made the empirical observation that people do perform differently in tasks concerning each of the different graphical variables (Cleveland, 1984). They performed a number of studies concentrating on the perception of quantitative information display in different ways. The result was a set of rankings for different "tasks" of perception of quantitative information. Their ranking, from most easy to most difficult was: position, length, angle or shape, area, volume, and color.

Ware and Beatty (Ware, 1988) have done extensive empirical studies on the perceptual abilities of people to pick out clusters of marks based on color. They have shown that color is an effective way to convey information about data characteristics. They discuss the effectiveness of color and people's ability to discern different colors under different conditions.

automated graphical presentations

One of the earliest approaches to automating the design of diagrams was the BHARAT system (Gnanamgari, 1981). BHARAT is a simple procedural system capable of producing a pie chart, bar chart, or line chart of a single unary function. The system is limited in that design choices, like font and color, are hard-wired into the system. In addition, the user is required to enter large amounts of important information about the data.

Jock Mackinlay (Mackinlay, 1986) developed a system called *APT* (A Presentation Tool) which analyzes a database and creates a graphical representation of this data using an automated design algorithm similar based in part on Bertin's semiology. *APT* focuses on generating a variety of two-dimensional static presentations. *APT*'s approach is to encoding graphic design knowledge in a precise graphical language that formalizes the arrangement and properties of graphical elements. The algorithm for creating graphical sentences was based mainly on the work by Cleveland and McGill. Mackinlay used the data from this study to create his own unique rankings of perceptual tasks for quantitative, ordinal, and nominal categories. In doing so, he models the language of chart graphics as a basis set of primitive graphic languages and composition operators. *APT* then creates visualizations by constructing legal sentences in this language that encode the data. The system uses a depth-first backward chaining version of a deductive algorithm in LISP and has a rule set of around 200 rules.

Graphic design, however, is not strictly procedural. Although Bertin outlines a methodology for creating such visualizations, and Mackinlay's system shows that a procedural approach to visualization can work in certain situations, others have tried using a case-based approach in the domain of design. The problem with rule-based systems is they require a programmer to try to define the design process as a set of rules, which is often impossible. Designers often draw on other work as a basis for new designs. As a result, a case-based approach is better matched to the actual activities of a trained designer.

One of the first case-based systems in the domain of design is TYRO (MacNeil, 1989, 1990). TYRO is a visual programming environment built as a set of design constraint networks and designed to function as a designer's apprentice. The system has mainly been used in the domain of the design of technical diagrams and maps. TYRO's cases contain sequences of design actions for an isolated design decision and an abstract description of the conditions required for that rule to be instantiated.

LIGA (Colby, 1991) is another example of using a case-based approach to design. LIGA, written in LISP, uses case-based reasoning alternative layouts of pictorial and textual information. Figure 3.2 shows two layouts of the same information generated by LIGA.

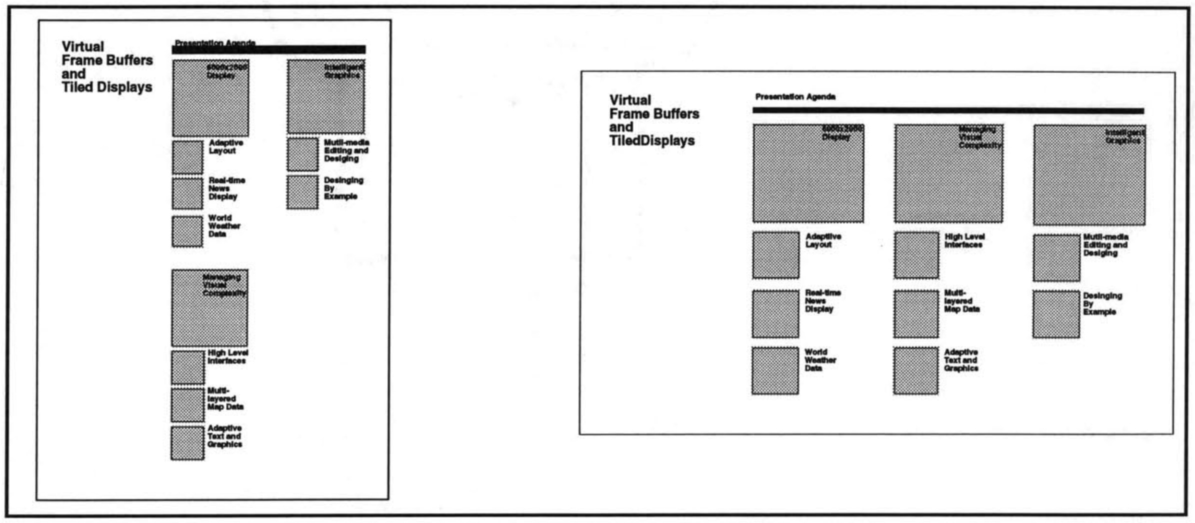
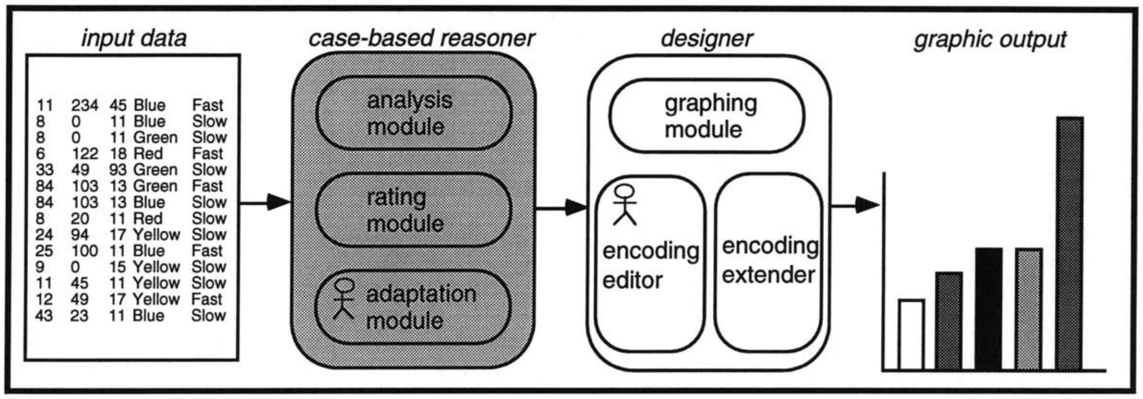


Figure 3.2. Sample output from LIGA. The sizes and shapes of the two displays are different, so the layouts are different. Both, however, contain the same amount of information and the same general "feel".

In LIGA, a case is a description of a grid and a set of constraints between abstract objects in the display, along with a description of how flexible each constraint is. New "problems" are new display characteristics (e.g., different sized display), and LIGA adapts a set of text and images from one situation to another by adapting both the constraints and the grid, making sure that the final image retains the relevant semantic feeling and content.

4 case-based reasoning in aide



11	234	45	Blue	Fast
8	0	11	Blue	Slow
8	0	11	Green	Slow
6	122	18	Red	Fast
33	49	93	Green	Slow
84	103	13	Green	Fast
84	103	13	Blue	Slow
8	20	11	Red	Slow
24	94	17	Yellow	Slow
25	100	11	Blue	Fast
9	0	15	Yellow	Slow
11	45	11	Yellow	Slow
12	49	17	Yellow	Fast
43	23	11	Blue	Slow

This section describes the motivation for case-based reasoning and the particular approach used in **aide**. The case-based reasoning algorithm can be broken down into three main modules: the *data analysis module*, the *rating module*, and the *adaptation module*. Each of these modules is now described in more detail.

case based reasoning

Case-based reasoning originated at Yale University in the late 1970's as an approach to natural-language comprehension (Riesbeck, 1989). As described in Riesbeck, a case-based reasoner solves new problems by adapting solutions that were used to solve old problems. This is opposed to rule-based systems that contain a large set of conditional rules, also called "if-then" rules, that when combined together describe solutions to problems. Rule-based computer programs are effective in situations where the process being automated is a simple one or can be reduced to a simple set of conditional rules. However, it seems unlikely that design happens this way. It is reasonable to assume that designers borrow ideas from other designs and base new visual images on old ones. It seems impractical, if not impossible, to try and delineate a set of rules for design. Instead, it seems that case-based reasoning is better matched to the process that graphic designers often take when creating graphics. As a result, this is the approach used by **aide**.

aide has a library of "solutions to old problems," called the *example library*. Each example consists of an image, abstract description of the database being visualized in the example image (the *shape*), and an abstract description of how the data was turned into a graphical image (the *encoding*). The original data itself is not part of the example because it isn't needed and would increase the storage requirements unnecessarily. This implies that while the content of a database is important, its characteristics as a database are equally, if not more, important to the design process. **aide** is capable of adapting each of the examples in the library to the current problem (i.e., the input database) through a series of steps described in this chapter.

Riesbeck explains that the first step a case-based reasoner takes is finding the cases that solved problems similar to the current problem. The *analysis module* computes the *shape* of the input data and passes this information to the *rating module.* This module rates each of the examples in the library to find the similar solutions. According to Riesbeck, the next step is to "adapt the previous solution or solutions to fit the current problem, taking into account any difference between the current and previous situations." In **aide,** this is performed by the *adaptation module.* This section contains a detailed description of the of case-based components of **aide**. For the purposes of explanation, the database shown in Figure 4.1 will be discussed.

Name:	Title	Location	Format	Age	Length	Category	Priority
Kind:	standard	locative	standard	temporal	standard	standard	standard
Aspect:	nominal	quantitative	nominal	quantitative	quantitative	nominal	ordinal
	Bombs damage homes in Northern Ireland	Belfast, Northern Ireland	text	446	53	international	major
	Authorities to protect Taj Mahal from further pollution damage	Bombay, India	text	912	20	international	major
	Paid killers track down three police informers	Rome, Italy	text	1000	17	international	urgent
	Cambodians turn out in force to vote	Phnom Penh, Cambodia	text	1034	71	international	major
	Bosnian deputy prime minister wounded in Sarajevo shelling	Sarajevo, Bosnia	text	2028	150	international	urgent
	Cairo car bomb death toll rises to seven	Cairo, Egypt	text	2380	51	international	regular
	Venezuelan president faces congressional curtain call	Caracas, Venezuela	text	3612	82	international	urgent
	Parliament rejects prime minister's resignation	Sarajevo, Bosnia	text	3906	23	international	major
	Official vote tally shows Bosnian Serbs rejected peace plan	Sarajevo, Bosnia	text	6475	178	international	regular
	No. 2 boss in Sicilian Mafia arrested	Rome, Italy	text	7913	56	international	major
	Authorities dig for remains at 1980 massacre site	Buenos Aires, Argentina	text	7892	45	international	regular
	Yeltsin: Victory incomplete without new constitution	Moscow, Russia	text	9357	67	international	major
	New poll says Danes will grudgingly approve Maastricht treaty	Maastrict, Netherlands	text	9392	81	international	regular
	Fighting in Angola as peace talks remain deadlocked	Luanda, Angloa	text	9489	37	international	major
	Eastern German metal workers vote on whether to end strike	Berlin, Germany	text	9496	45	international	regular
	Street fighting flares in shell-ravaged town of Mostar	Mostar, Bosnia	text	9510	166	international	urgent
	Back to court for August 1991 coup plotters	Moscow, Russia	text	9593	73	international	regular
	Peace envoy expresses dismay at referendum	Jerusalem, Israel	text	9779	24	international	major
	India and Pakistan quibble over suspects	Islamabad, Pakistan	text	11147	86	international	regular
	Rainforest Damage Growing Rapidly	Brasilia, Brazil	series	8300	5	international	regular
	Racism Still Common	Johannesburg, S.A.	series	8100	4	international	regular
	Ukraine Votes on New Government	Kiev, Ukraine	series	6888	5	international	urgent
	Saddam Hussein Speaks	Bahgdad, Iraq	image	7333	8	international	regular
	Yugoslavian unrest continues	Zagreb, Croatia	image	11079	9	international	major
	Bill Clinton in Washington	Washington, D.C.	image	9500	8	washington	daily
	Air Force Marks Planes With Giant Flags	Lima, Peru	video	8500	18	national	major
	9 Killed in Stadium Collapse	Bastia, Corsica	video	7500	17	international	regular
	German Workers Strike Ends	Bonn, Germany	video	5079	15	international	major
	Russia Unveils Plan for Ruble	Moscow, Russia	video	6079	17	international	regular
	McDade Indicted for Racketeering	New York, New York	text	7080	15	usa	major
	Congressman Larry Smith Apologizes for Illegal Spending	Miami, Florida	text	9080	15	usa	major
	Drugs confiscated in South America	Santiago, Chile	image	8079	13	international	regular

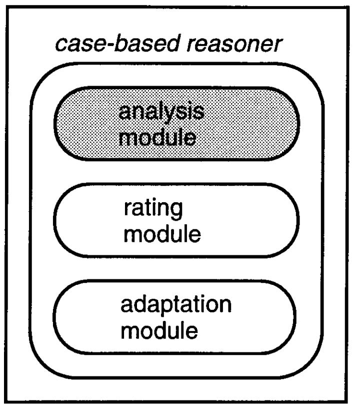
Figure 4.1. A tabular arrangement of a database containing information about news articles.

This data is a collection of news articles generated by different sources over a two week period. The "Age" characteristic is the age of the story in minutes. The stories are from all over the world and have different priorities and formats.

the analysis module

Statistics may be presented graphically in many different ways, but there should always be a sound reason for choosing the particular form of presentation. By and large it is the material itself that will determine which kind is to be used, for it will naturally be visually clearer in that form than in any of the others. The purpose of a chart is to clarify or make visible the facts that otherwise would lie buried in a mass of written material, lists, balance sheets, or reports.

- N. Holmes



The input data is first displayed as a table, allowing the user to scan the data. In future versions of the program, the user might be able to edit the data in the tabular format. Once the data has been presented in a tabular format, **aide** analyzes this data in order to create an abstract description of the data. This abstract description, called the *shape* of the database, is used to compare the input data with the *example library*. As Riesbeck explains, to find a relevant example, old solutions have to be labeled and organized so that features of input problems can be used in the search. Once the input problem is characterized, it can be compared to the cases in the example library to choose the most appropriate case. The *shape* is this characterization upon which the comparison is based. In the best case, the *shape* would be the minimum description necessary to compare the input data with the examples. For the input data described earlier, its shape would include:

overall information:

number of elements:	32		
number of fields:	7	number of standard quantitative:	1
number of quantitative:	3	number of standard ordinal:	1
number of ordinal:	1	number of standard nominal:	3
number of nominal:	3	number of locative quantitative:	1
number of standard:	5	number of temporal quantitative:	1
number of locative:	1		
number of temporal:	1		

characteristic information:

name:	**Title**	**Location**	**Format**	**Age**	**Length**	**Category**	**Priority**
kind:	nominal	quantitative	nominal	quantitative	quantitative	nominal	ordinal
aspect:	standard	locative	standard	temporal	standard	standard	standard
length:	32	24	4	32	25	4	4
average:	n/a	1653, 1025	n/a	6811	46	n/a	n/a
minimum:	n/a	816, 430	n/a	446	4	n/a	regular
maximum:	n/a	2564, 1242	n/a	11147	178	n/a	urgent
range:	n/a	3, 2	n/a	24	44	n/a	n/a
median:	n/a	1752, 1120	n/a	7902	23	n/a	n/a
standard deviation:	n/a	456, 238	n/a	3210	46	n/a	n/a
average deviation:	n/a	355, 186	n/a	2623	35	n/a	n/a
skew:	n/a	0, -1	n/a	0	1	n/a	n/a
kurtosis:	n/a	0, 0	n/a	0	1	n/a	n/a

Figure 4.2. The shape *of the news database shown earlier.*

aide computes the *shape* of a database by first sorting the characteristics of the data into three *kinds, nominal, ordinal,* and *quantitative. Quantitative* fields are quantities, numerical characteristics, such as height and weight. *Ordinal* characteristics are those whose values fall into sortable categories, such as "Olympic medal awarded" or "honors status". For a characteristic to be *ordinal,* each of the possible values must be considered to be equidistant from the previous (or latter) possible value. In the case of "Olympic medals", for example, this characteristic would be *ordinal* only if the difference between "gold" and "silver" was considered to be the same as the difference between "silver" and "bronze". *Nominal* characteristics are values that fall into categories that can not be sorted. For example, "favorite brand of soft drink" or "family name". This classification is based in part on Bertin's work.

However, sorting characteristics into only three *kinds* is not precise enough for **aide**'s graphing module. While it is true that systems like APT can create useful graphics resting only on this type of categorization, functionality could be improved by increasing the number of possible types of characteristics. The more detailed description of the data provided to the automated designer, the more sophisticated decisions the system can make. As a result, **aide** does more than just break characteristics down into three *kinds.*

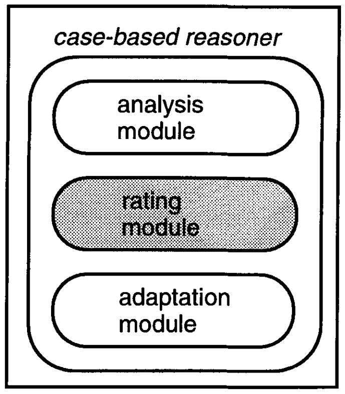
There are two important *aspects* of characteristics that **aide** treats as special. These two aspects are *temporal* and *locative. Temporal* characteristics (or rather, characteristics with a *temporal* aspect) are any characteristics dealing with time, *locative* ones are those which describe locations. Why these two? Because people have a natural tendency to think about things with respect to time and space. As Bertin explains (Bertin, 1983),

> Time, like geographic order, can be introduced into any analysis. As a naturally ordered component, it is a universally identifiable concept, on which innumerable comparisons can be based.

Because temporal and locative aspects have special cognitive value -- people are used to sorting things by location or time, any automated design system should be able to know when a characteristic is *locative* or *temporal.*

In addition to breaking the characteristic down into three *kinds* (*quantitative, ordinal,* and *nominal*) and three *aspects* (*temporal, locative,* and *standard*), various statistical parameters of the characteristics of the data are computed and included in the *shape.* These include the maximum, minimum and average values, the mode, median, standard deviation, average deviation, range (ratio of maximum value to minimum value), and the length (number of unique values).

the rating module

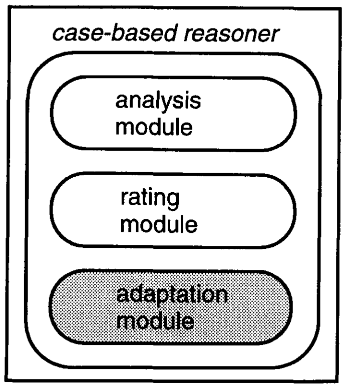


Once the *shape* is computed, the next step is to select appropriate cases from the example library on which to base the new solution. Before choosing an example, each example must be rated to asses its value as a possible basis for the final graphic. As mentioned, each of the examples in the library contains the *shape* of the data that produced that example. It is this abstract description that is used in the rating process. The *shape* of the input data is compared to the *shapes* of each the examples. This comparison is performed by the *rating function*. The *rating function* compares two *shapes* and returns a theoretical distance, or difference, between the two. In the current implementation, the *rating function* for *shapes* is defined as:

distance = [diff(number of standard ordinal fields) + diff(number of standard nominal) + diff(number of standard quantitative) + diff(number of locative ordinal) + diff(number of locative quantitative) + diff(number of temporal ordinal) + diff(number of temporal quantitative)] * weight_1 + [diff(number of ordinal) + diff(number of nominal) + diff(number of quantitative) + diff(number of standard) + diff(number of locative) + diff(number of temporal)] * weight_2 + diff(length) + diff(range) + diff(max) + diff(min) + diff(average deviation);

One of the difficult aspects of building a case-based reasoning tool is often called the *indexing problem* -- in the real world, similarity between problems is usually an abstract, nebulous relationship. Since **aide** is an automated system, the process of comparing the "new" problem to previous solutions is performed by a computer. As a result, the relevant aspects need to be listed in some relatively strict set of criteria, rules, or structures. In the current implementation, the relatively simple approach described above is taken. As **aide**'s example library grows, this *rating function* will need to be replaced with a more complex one, and the system is designed such that replacing this function is fairly easy.

the adaptation module



After each of the examples in the library is rated, a subset of them is visually presented to the user, along with their ratings and a rough draft of what a graph based on that example could look like. Once these are presented, the user is prompted to choose one of the suggested examples to be adapted for the new solution, called the *basis example.*

At this point, the user has only chosen an example on which to *base* the visualization of the input data. In almost all situations, a perfect match between the *shape* of the input data and one of the examples in the library will not be found. As a result, **aide** must *adapt* the chosen example to the input data. This process is carried out by **aide's** *adaptation module* .

Before detailing the adaptation process, it is first review the structure of the "solution" (the *basis example*) to be adapted. Not only does each example in the library contain the *shape* of the data that generated the example, it also contains an abstract description of the design characteristics of the graphical image, called the *encoding.* Figure 4.3 contains part of an example *encoding* for a graph, and is followed by a description of *encodings* and how they are adapted.

graph type:	**scatterplot**
history type:	shadow

graphic variable information:

graphic variable	**encoder kind**	**constant**	**other gv**	**kind**	**aspect**	**range**	**length**	**std dev**
color:	characteristic	-	-	nominal	standard	-	12	0.234
shape:	characteristic	-	-	quant	standard	18	5	0.944
shape size:	constant	14 pixels	-	-	-	-	-	-
frame number:	characteristic	-	-	ordinal	temporal	800	800	0.1934
location:	characteristic	-	-	quant	locative	213	24	0.73
x axis:	empty	-	-	-	-	-	-	-
y axis:	empty	-	-	-	-	-	-	-
transparency:	characteristic	-	-	quant	temporal	85	32	0.10033
label:	empty	-	-	-	-	-	-	-
label size:	other gv	-	shape size	-	-	-	-	-

other information:

vertical scale:	10 ticks	adaptive text:	no
horizontal scale:	5 ticks	non-overlapping text:	yes
vertival grid:	yes		
horizontal grid:	yes		
top ticks:	no	top outline:	yes
right ticks:	no	right outline:	yes
bottom ticks:	yes	bottom outline:	yes
left ticks:	yes	left outline:	yes

Figure 4.3. An encoding for a scatterplot.

Most of the *encoding* is a set of *encoders*, one for each of the *graphic variables* for that graphic. An *encoder* describes how to instantiate a specific *graphic variable* for each element in the database and can contain a constant value, an abstract description of a characteristic from a database, a pointer to another *encoder*, or be empty. The set of *graphic variables* differs for each of the different types of graphs. For example, in a scatterplot, the *graphic variables* include: the location of the marks, the shape of the marks, the size of the marks, the transparency of the marks, etc. In Figure 4.3, an *encoder* is illustrated as a row to the right of each graphic variable listed.

In this example, the color of the marks is assigned an *encoder* that describes a characteristic of a database that was *nominal*, not *temporal* or *locative*, had a length of 12 and a standard deviation of .234. Similarly, the shape of each mark was determined by a characteristic that was quantitative, had a range of 18, a length of 5 and a standard deviation of .944. It is important to note that the description of the characteristic contains information about its *kind*, *aspect*, and various statistical information, but does not contain any of the original values from the original database. Also found in the example above, the size of the shapes (the graphic variable "shape size") was a constant value of 14 pixels. Notice that the *encoder* for the size of the labels points to the *graphic variable* "shape size", so they also were a constant of 14 pixels.

In addition to *encoders* for *graphic variables*, an *encoding* includes a graph type (scatterplot, linechart, barchart, etc.) and some additional information about axis labels, axis scales, grid lines, etc. Each example in the library contains a *shape* and an *encoding*.

The *encoding* can be thought of as a template, for it describes the salient characteristics of a graphical image in terms of the data used to construct it. However, the example chosen by the user was derived from a different set of data than the input data, so **aide** must *adapt* the chosen *encoding*.

There are a few general types of adaptation schemes in case-based reasoning. One method is called *structural adaptation*, where the adaptation rules merely change the chosen solution. When a solution containing rules about object *x* are adapted for object *y*, all instances in the solution that refer to *x* are merely replaced by *y*. The solution is then checked for possible conflicts.

Another approach to adapting solutions is called *derivational adaptation* or *derivational analogy*. (Carbonell, 1993).This approach works best for solutions that are stored as a set of rules. This adaptation process creates new solutions by recreating the line of reasoning used to generate the original solution for the new conditions. In effect, a new solution is generated by, as Riesbeck puts it, "re-executing parts of the original solution process." **aide**'s adaptation process is a combination of the two and will now be described in more detail.

The first step of the adaptation process is to create a new *encoding* for the new graph. This is done so that the *basis encoding,* selected by the user from the example library, is not changed. This *encoding* will describe the new *final graphic* is called the *adapted encoding.* The *adapted encoding* is assigned the same graphical form (scatterplot, barchart, or linechart) as the *basis encoding.* Then, each of the graphic variables is assigned an empty *encoder,* referred to as *adapted encoders* (the *encoders* in the *basis encoding* are called *basis encoders*). Next, each of the *adapted encoders* is adapted from its corresponding *encoder* in the *basis encoding.*

The adaptation of *encoders* occurs in the following manner: If the *basis encoder* is instantiated with a constant value, the *adapted encoder* for that *graphic variable* is instantiated with the same constant value. If the *basis encoder* points to another *graphic variable,* the *adapted encoder* will point to the same *graphic variable.*

The interesting case occurs when the *basis encoder* contains a description of a characteristic from a database. In this case, a rating function is used to find the most similar characteristic in the input data base. If the most similar characteristic hasn't already been assigned to a *graphic variable* in the *adapted encoding,* it is assigned to the *encoder* being adapted. If it has been, then the most similar characteristic (within a certain range) that *isn't* assigned is used. In the current implementation, the *rating function* for data characteristics is a function of the *kind* of characteristic (*ordinal, nominal,* or *quantitative*), the *aspect* (*temporal, locative,* or *neither*), and the various statistical parameters of the field (i.e., maximum value, minimum value, standard deviation, range, etc.) This *rating function,* like that for the rating the *shapes,* is also easily expanded, and is listed below.

```
/* this function compares two characteristics, ec and dc. ec is the characteristic from the encoding and dc
is the characteristic from the input data. The higher the rating, the more similar the characteristics. */

rating = 0;
if (ec->aspect == dc->aspect)
      retval += c1;
if (ec->kind == dc->kind)
      retval += c2;
if ((ec->kind == ORDINAL) && (dc->kind == NOMINAL))
      retval += c3;
if ((ec->kind == QUANTITATIVE) && (dc->kind == ORDINAL))
      retval += c4;
retval -= diff(ec->length, dc->length) * c5;
retval -= (diff(ec->max, dc->max) + diff(ec->min, dc->min)) * c6;
retval -= (diff(ec->median, dc->median) + diff(ec->standard_deviation, dc->standard_deviation)) * c7;
retval -= diff(ec->range, dc->range) * c8;
retval -= diff(ec->average_deviation, dc->average_deviation) * c9;
retval -= (diff(ec->skew, dc->skew)) + diff(ec->variance, dc->variance)) * c10;
return(retval);
```

For example, assume that the user has chosen the *encoding* shown earlier and the news database. One of the *encoders* is for the color of the marks. The color *encoder* in the *basis encoding* is:

graphic variable	encoder kind	kind	aspect	range	length	std dev
color	characteristic	nominal	standard	-	12	0.234

Figure 4.4. A color encoder.

The most similar field in the input data is either "Format" or "Priority," so either of these would be assigned to the color *encoder* in the *adapted encoding.*

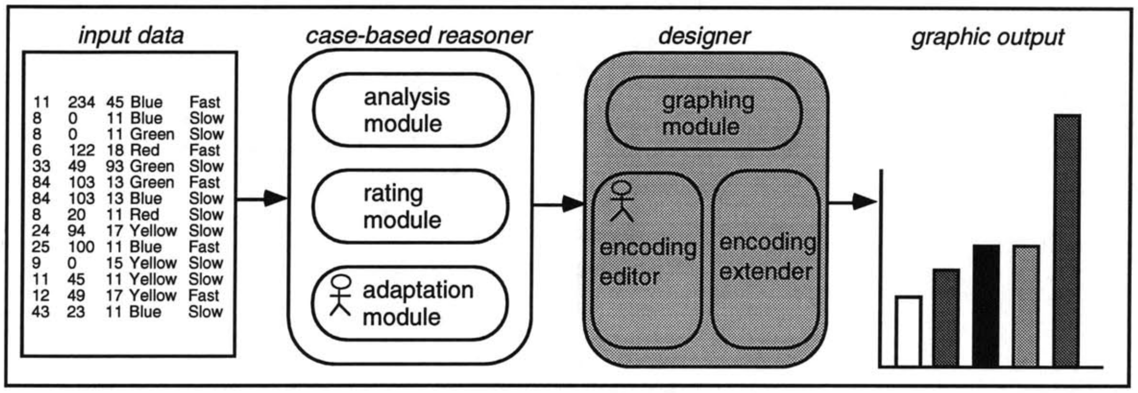
In summary, the adaptation of *encoders* is a relatively simple activity. For cases where encoders are assigned constant values or point to other graphic variables, this information is merely copied (a structural adaptation of sorts). When the *encoder* describes a characteristic, the most similar unused characteristic is assigned. Figure 4.5 summarizes the adaptation of *encoders.*

basis encoding	adapated encoding
empty	empty
constant value	same constant value
other graphic variable's encoder	same other graphic variable's encoder
characteristic of database	best-rated unassigned characteristic of input data

Figure 4.5. A summary of the adaptation rules for encoders.

Once the system has adapted the *encoding* from the example, it passes the newly *adapted encoding,* along with the input database, to the *designer component* that creates the visual presentation.

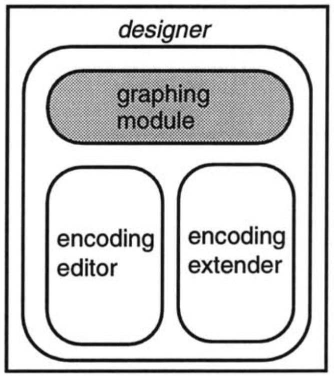
5 the designer component



11	234	45	Blue	Fast
8	0	11	Blue	Slow
8	0	11	Green	Slow
6	122	18	Red	Fast
33	49	93	Green	Slow
84	103	13	Green	Fast
84	103	13	Blue	Slow
8	20	11	Red	Slow
24	94	17	Yellow	Slow
25	100	11	Blue	Fast
9	0	15	Yellow	Slow
11	45	11	Yellow	Slow
12	49	17	Yellow	Fast
43	23	11	Blue	Slow

Once **aide** has completed adapting an example's *encoding*, it passes the new *encoding*, along with the input database to the *designer component* of the system. The *designer component* consists of a *graphing module* that contain a set of design skills that allow **aide** to draw graphs. In addition, **aide** contains an *encoding editor* that allows the user to provide feedback to the system about its design choices. The system also has an *encoding extender*. This rule-based module has a set of design rules that attempt to make a graph present as much information about a database as possible, given its unique characteristics. This chapter describes the process of designing and drawing, the *encoding editor* and *encoding extender*.

the graphing module



aide functions as a graphic designer's assistant. To that end, the system is not fully automated, it turns tabular data into visual images with the user being an integral part of the design process. The *graphing module* is responsible for creating the visual image. Each diagram is internally described by an *encoding*, consisting of a list of *graphic variables* corresponding to the various visual parameters of the diagram, and information about how to instantiate each for the data.

The *graphing module* contains a set of design skills and heuristics that allow it to create a number of design choices. The system has a few basic design priorities: to maintain legibility of text, to provide context for *locative* information, to make graphs as information-rich as possible. In turn, the system has a number of design skills and methods for achieving these goals. These skills or methods include the use of *basemaps*, adaptive-text, methods for preventing overlap of text, and a set of rules to match data characteristics to appropriate graphic variables.

graphic variables

As described in the previous chapter, each of the different graphic forms (scatterplot, linechart, barchart, etc.), is partially defined by its own unique list of *graphic variables*. These *graphic variables* are the set of variables to be instantiated in order to draw the final graphic. For example, for a scatterplot, the list of *graphic variables* is: location, shape, size, color, and transparency.

One of the ways **aide** is different from previous work is that its set of graphic variables is different than other systems. For example, *transparency* is one graphic variable that is used in **aide** that has not been used in previous work. Transparency is useful for a number of reasons, not the least of which being that if two transparent objects overlap, the one that is "in back" does not get occluded by the other object. Like brightness, people are able to discern different levels of transparency. As a result, information can be presented through the careful use of transparency. As another example, many previous systems use *texture* as a graphic variable, whereas **aide** does not.

The system contains a set of graphic design knowledge about how to instantiate each graphic variable for the different kinds of characteristics. For example, the system is capable of drawing up to 255 different levels of transparency. **aide**'s *designer component* contains routines to analyze a characteristic of a database and convert the values into the range 0-255 appropriately. **aide** has a different set of design heuristics for each graphic variable and each *kind* and *aspect* of characteristic. The graphic design knowledge associated with the graphic variables is explained during the discussion on the *encoding extender.*

displaying the graph

aide was developed on the Large High-Resolution Display prototype, a 6000x2000 pixel display. The display has 24-bit color and 100 pixels per inch and was developed at the VLW. The relatively large display area and high resolution allows for images of a quality close to that of printed material. In addition, the large amount of screen real estate allows for the simultaneous presentation of a large amount of information. Each of these was taken into account when designing the user interface described in Chapter 2.

traditional graphical forms

aide currently has knowledge about four forms of traditional static chart graphics: scatterplots, linecharts, horizontal barcharts, and vertical barcharts. Because these are fairly common, a detailed description of them will not be provided here.

chartmaps

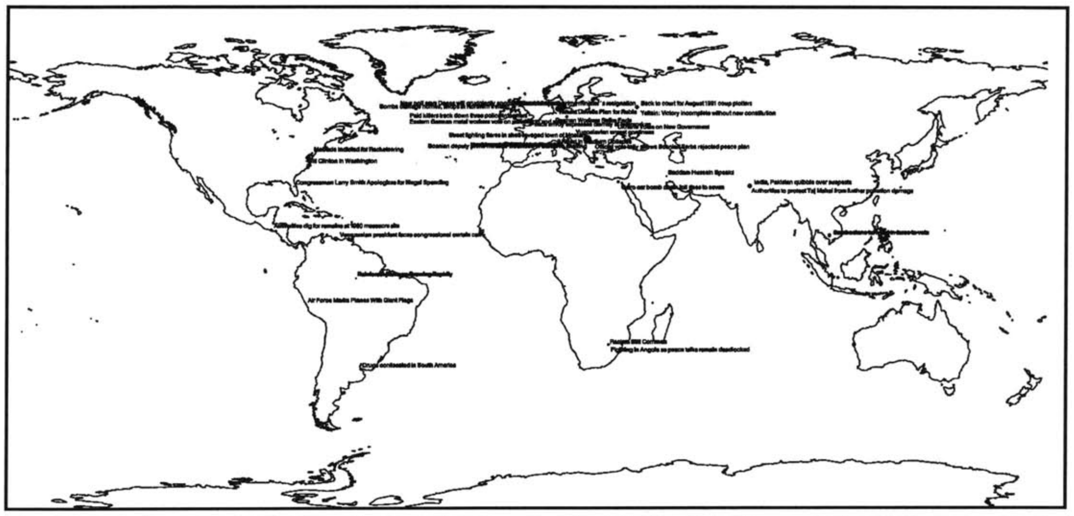
One of the reasons that **aide** treats *locative* information as special is because people have a tendency to think about things with respect to locations. On of the graphical forms that **aide** has special design knowledge about is a specialized form of scatterplot, called a *chartmap*. The main difference between a traditional scatterplot and a *chartmap* is that the marks of a *chartmap* are not drawn on a solid background. Instead, the graphing area is filled with an image, or *basemap*. Bertin describes the use of basemaps (Bertin, 1983):

> The base map consists of the set of known reference points which are necessary and sufficient for situating the as yet unknown elements of the new information being mapped.
>
> The drawing of a base map always runs up against the following contradictions:
>
> 1) The base map must include all the elements of identification necessary for the construction and reading of the map on all levels.
>
> 2) The base map must be dominated by the new elements being mapped; the reader must be able to select and group them for information on the intermediate and overall levels.

The other main difference between scatterplots and *chartmaps* is that in *chartmaps*, each of the locations of the marks are determined by a *locative* characteristic of the database. One of the powerful effects of *chartmaps* is that the location of the marks in the graphing area can convey an enormous amount of cognitive information. Not only can the user notice clustering or other patterns just as in traditional scatterplots, but the basemap provides information *about* the clusters. The user can get context for the information being displayed, and determine relationships not only among the elements themselves, but between the elements and the information on the *basemap*. In the example above, not only can the user discover the real-world geographic distribution of news articles, but can also get information about the geographic relationships between two or more stories, and between an individual story and the real-world geography of that location.

It is important to note that while in the example above, locations corresponded to real-world locations, a *locative* field can be any kind of location. One could use the same technique to display, say, a floor map of a building and place over it information about the employees in the building. Or, a basemap could be an image of a human body, and the data could be medical information about an individual. Another example of a *chartmap* would be a schematic diagram of an electrical circuit used as a basemap for a set of information about its components.

One of the design principles that **aide** uses is suggested by Bertin, "...the first task of the graphic designer is to separate clearly the lines belonging to the basemap from those constituting the new information" (Bertin, 1983). As a result, the *basemap* of a *chartmap* is often drawn more lightly than the marks derived from the database. Figure 5.1 shows an example of a *chartmap* derived from the news data. The locations of the marks are determined by the "location" characteristic and the size of each mark is determined by the "length" characteristic.



Back to court for August 1991 coup plotters
Yeltsin: Victory incomplete without new constitution
Paid killers track down three police
Eastern German metal workers vote on
on New Government
Street fighting flares in shell-ravaged town of
Bosnian deputy
Bosnian Serbs rejected peace plan
Saddam Hussein Speaks
India, Pakistan quibble over suspects
Authorities to protect Taj Mahal from further pollution damage
Indicted for Racketeering
Clinton in Washington
Congressman Larry Smith Apologizes for Illegal Spending
dig for remains at 1980 massacre site
president faces congressional censure
Air Force Marks Planes With Giant Flags
Drugs confiscated in South America
Fighting in Angola as peace talks remain deadlocked

Figure 5.1. A chartmap presentation of the input data shown earlier. The headlines of news articles are drawn over the locations from which the articles were written.

automated temporal diagrams

When such a rendering is achieved [a space-time diagram], the spectator himself will mentally recreate the process of growth; and in doing so, he is exercising vision in motion. The rendering itself is, in any space-time diagram, motion arrested for vision. Whether one studies statistics or looks at the space-time rendering of the hand, his contribution is to unfreeze the sign language of vision and recreate the motion. Later, after he liberates himself from traditional vision, he will be able to apprehend this *emotionally as well as intellectually*; see it, feel it, know it. Once he has achieved this power he will have broken [as the isolation of things to a deeper and richer unity of insight.

- L. Moholy-Nagy

Not only do people mentally organize things with respect to locations, they also have a tendency to think of things with respect to time. Even though time is a linear, single-dimensional characteristic, its special cognitive value requires unique design techniques.

As Bertin explains (Bertin, 1983), there are two effective ways of presenting temporal information:

> When the information involves both TIME and spatial or GEOGRAPHIC ORDER, the correspondences translate a MOVEMENT...but when the two planar dimensions are utilized to represent space, no planar dimension remains available to represent the 'time' component: This is the basic problem with the representation of movement in cartography. There are three solutions:
>
> a) Construct a series of images. As in the cinema, this solution can be applied to the most complex of movements. But here, the number of images is limited by the reading process: with a long series, it is difficult to suggest motion.
>
> b) Represent the path and direction of a moving body. This solution can suggest a continuous movement on the plane, i.e., MOTION.
>
> c) Utilize a *retinal variable*. The time component is divided into ordered categories represented by the different steps of an ordered retinal variable.

aide obviously has the ability to do the third of these suggestions, as any of the *graphic variables* could be assigned to a *temporal* characteristic. More interestingly, **aide** also has a set of design heuristics that provide it with ability to present information in the first two ways suggested by Bertin. **aide** does this by creating *temporal*, or *dynamic*, diagrams.

Temporal diagrams are presented as a series of separate images derived from the data. In most cases, the data is rendered once for each value of a *temporal* characteristic and images are presented sequentially, like a movie. This animated sequence allows the viewer's own ability to perceive change over time to discover how the various characteristics of the data change with respect to time.

aide creates sequences of images by including a *graphic variable* called *frame number* in some of its *encodings*. The result of this is that a characteristic of the data can be assigned to the frame of the sequence, and frames are presented to the user by sorting on that characteristic. As mentioned, the *frame number encoder* is most often assigned a *temporal* characteristic of the input data, but **aide** contains design knowledge about how to instantiate any characteristic for the *frame number*.

One of the main problems with animated sequences is that they rely on the viewer's memory to perceive change. In other words, the user compares the currently presented image with a memory version of the previous image. While this can be very effective for short sequences, it can become problematic as the length of the animated sequence increases. It is often necessary to provide the viewer with context for the current image, and this context is the previously viewed images. Just as the basemap in a *chartmap* provides the user with more context, **aide** employs special graphic techniques to provide viewers of dynamic graphics with a sense of history and context. There are many effective ways to provide such information (Sivasankaran), and **aide** implements a few of these. One of the most simple ways is to retain images on the screen as the dynamic presentation progresses. Current images are then drawn over previous ones. Bertin explains:

> The movement of a vehicle, for example, is best expressed by the "trace" of a moving body. This trace corresponds to a change in implantation: a point traces a line, a line or an area traces an area. But with complex movements, the area becomes cluttered and indicates direction poorly

At the end of the animated graphic, marks are left that represent the data at all time values. Figure 5.2 shows one of the most famous visualizations of this type, one that displays the number of troops under Napoleon over a continuum of time and space (Marey via Tufte, 1983).

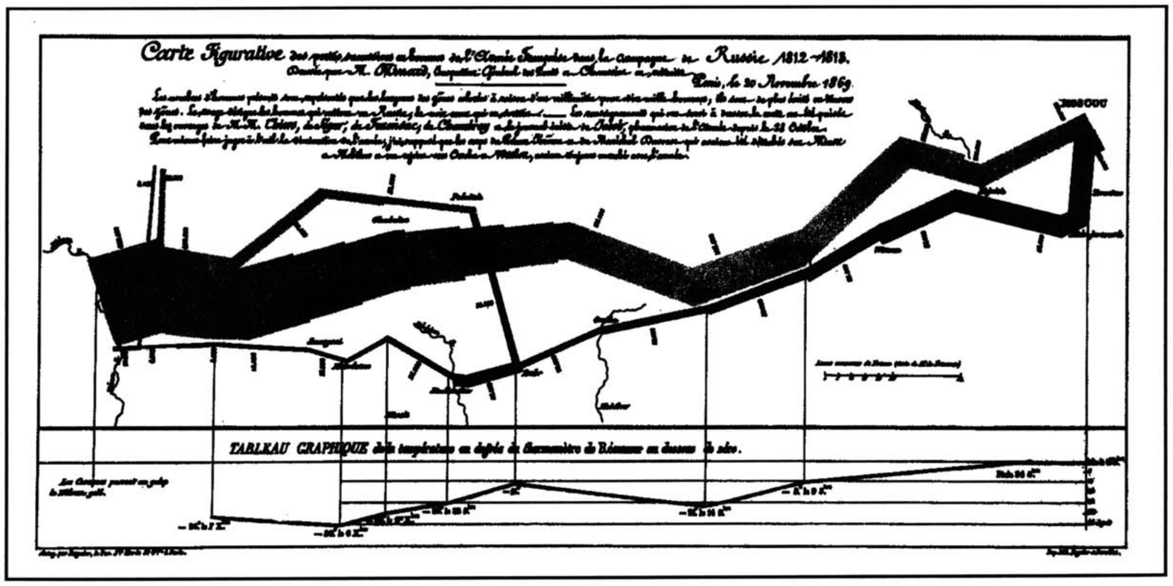

Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813.
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.

Figure 5.2. A graphical presentation of data about Napoleon's campaign against Russia.

Figures 5.3-5.5 show a few frames from a *trace* dynamic scatterplot produced by **aide** using this method on a set of data describing simulated air traffic patterns. The characteristics of the database include: flight number, time of day, location, airline, air speed and altitude. The encoding instantiates *frame number* from "time of day," *location* from "location," and *shape size* from "altitude". **aide** draws a fully rendered image for each value of "time of day," producing an animation of 250 frames. When the complete series of images is finished, there is a mark over every location visited by every plane.[7]

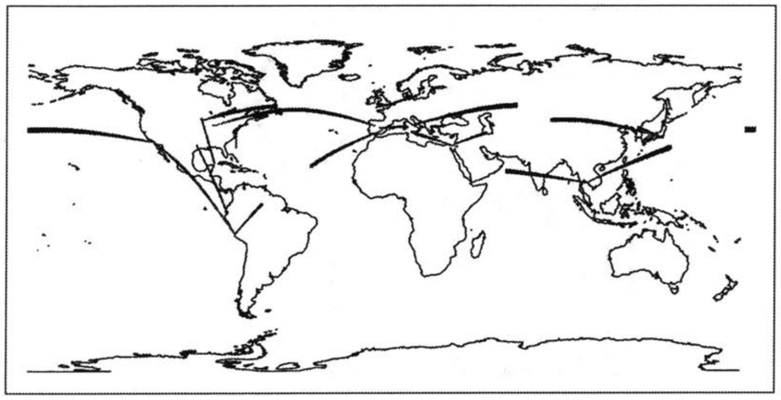


Figure 5.3. A frame from a dynamic chartmap drawn by ***aide****. Frame 100 out of 250.*

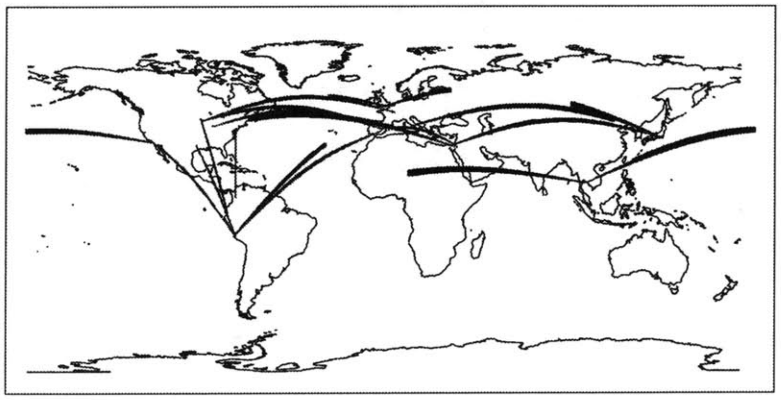


Figure 5.4. Frame 175 out of 250.

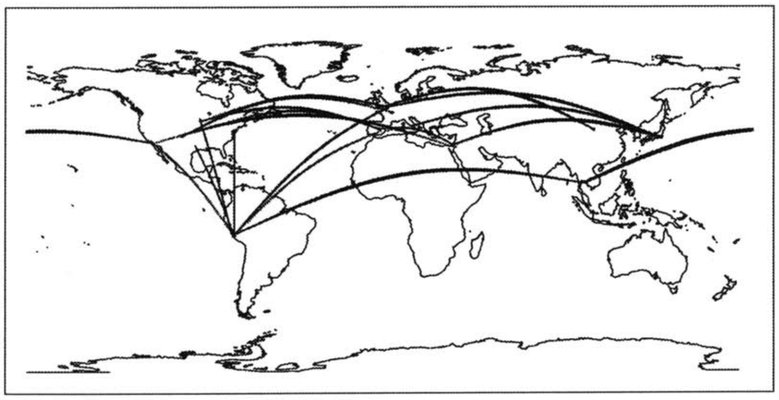


Figure 5.5. Frame 250 out of 250.

[7] This type of presentation of images -- a few from a sequence all shown at once, side-by side, is often called a *storyboard* presentation and is another kind of temporal graphic. While aide is not currently able to produce such graphics, it is an area for future work. See Chapter 6 for a more detailed discussion of storyboards.

While this type of graphic is effective for some data, sometimes the "old" images clutter the screen to the point where they interfere with the "current" information. In many situations, it is desirable to retain the "old" information, yet with a lower visual priority (much like the basemap in a chartmap). In **aide**, this technique is called generating a *shadowed* dynamic graphic. In this type of graph, when a new frame is drawn, the marks representing the data in the previous frame are made more transparent and smaller, leaving a shadow-like history of the data.

In situations where even the shadowed information interferes with the "current" data, it is desirable to completely erase the visualized information after each frame. In addition to the two types of dynamic graphics describe above, aide is capable of creating such *no-history* dynamic graphics.

The implementation of dynamic graphs is done by adding a graphic variable named *frame number* to the *encoding* of particular graph forms. As a result, the user can choose any characteristic to instantiate in the *encoder* for *frame number.* In the example above, if the user chose "airline" instead of "time of day," the user would see one frame with all of the airplanes of one airline, and the next frame would have all of the planes of another airline, and so on. The result is that users can use real-world time as a way to sort information and encode it in the graphical presentation.

labeling of data points while maintaining legibility

Automated layout systems have the potential for legibility of labels to be diminished in at least three ways: 1) labels can overlap, 2) the background image can interfere with legibility due to spatial frequencies, and 3) in colored systems, the color of the background can interfere with discerning the shapes of the characters. **aide** contains design heuristics that attempt to diminish the effects of each of these.

One activity an automatic layout system must perform is to place labels on a graph -- the system must decide exactly where to place these labels. If the system is not sophisticated in its label placement, it can easily place labels over each other, over other points on the graph, or over other important graphical information. Optimal label placement is a condition where labels do not overlap at all. Finding this state (if it exists at all) for a set of data has been shown to be an NP-hard problem for automated systems (Marks, 1991; Formann, 1991). It is a challenging task even for a trained human designer to effectively layout a graphic and be able to label many points. Current research into what is often called "the cartographic label-placement task" has shown that certain low-cost algorithms exist that can find near-optimal solutions. One such algorithm, called "gradient-descent" has been implemented in **aide** (Christensen, et. al, 1991). Whether labels are allowed to overlap or not is part of the *encoding* of a graph and as such, the user can modify this parameter.

The gradient-descent algorithm has been shown to provide considerable improvement over random placement of labels, and is relatively computationally inexpensive. In many cases, it has been shown to be the second-best method available in terms of performance (the first being simulated annealing). The gradient-descent algorithm is simple, and is described as follows:

1. For each feature, place its label randomly in any of the available potential positions.
2. Repeat until no further improvement is possible:
   (a) For each feature, consider moving the label to each of the alternative positions.
   (b) For each such repositioning, calculate the change in the objective function which would result if the label were moved.
   (c) Implement the single label repositioning that results in the most improvement.

While it is true that the gradient-descent algorithm has the disadvantage of only choosing local minima instead of global minima, it does provide a significant improvement with limited cost. This, combined with the simplicity of the code needed to implement the algorithm, make it a beneficial addition to **aide**. In the current implementation, there are four possible locations: to the upper-right, upper-left, lower-right, and lower-left of the point being labeled.

Figures 5.6-5.8 illustrate different methods of labeling points. Figure 5.6 has all of the labels to the upper right of the symbols, 5.6 has the labels in random positions, and 5.8 is a set of points labeled with the gradient-descent method.

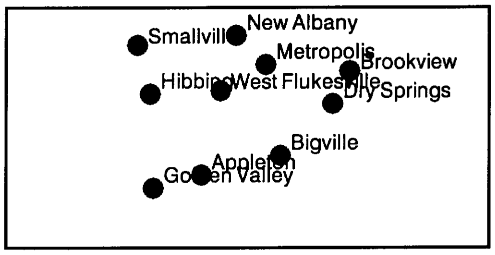


Figure 5.6. Labels to the upper right of symbols.

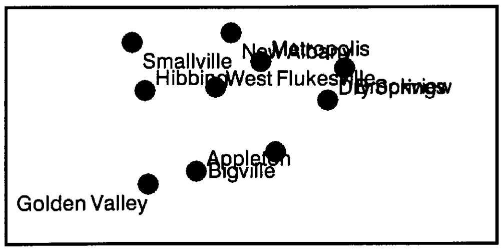


Figure 5.7. Labels with random positions.

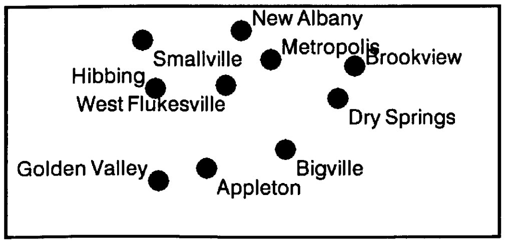


Figure 5.8. Labeling using the gradient-descent method.

An additional potential problem with labeling is often most profound in *chartmaps* or other graphical forms in which text is placed over a non-uniform background. When the background has high spatial frequencies, these frequencies interfere with the shapes of the characters in the text, making them difficult to discern, which in turn causes a decrease in legibility The solution to this problem is to remove the spatial high frequencies of the background image in the bounding box of the text. This can be achieved in many ways (television often uses a "drop-shadow," whereas subtitles in movies are often printed over solid opaque rectangles). **aide** removes the high spatial frequencies of the background by using a technique called *adaptive text* that draws a blurred rectangle of the average background color behind the text (Bardon, 1991). This technique has been shown to be more effective than drop-shadows, outlines, and solid background rectangles because it adds less extraneous information to the letter forms. Figure 5.9 illustrates the difference between adaptive and non-adaptive text.

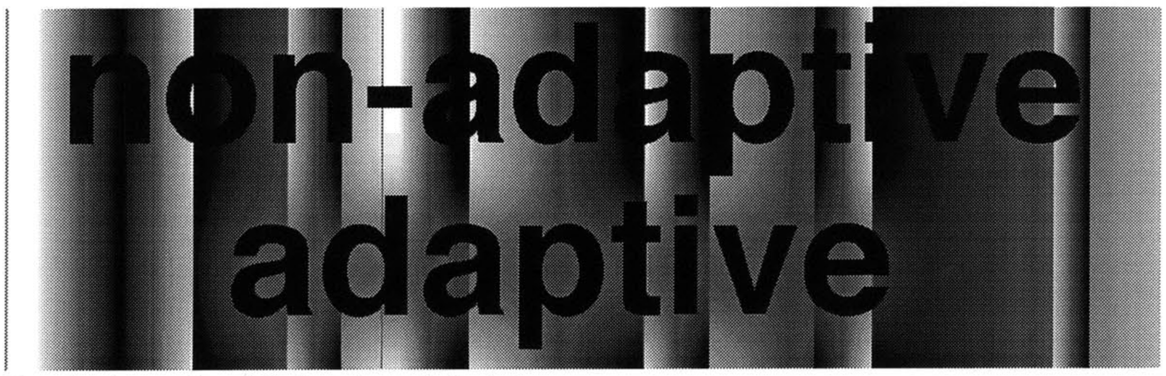
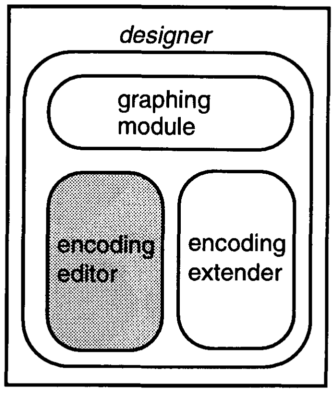


Figure 5.9 A comparison of adaptive and non-adaptive text.

If the color of a label is similar to the background color in either hue or brightness, recognizing where a text character ends and the background image begins can be difficult (i.e., blue text on a blue background is difficult to read if the blues are similar). In colored automatic layout systems, this situation must be accounted for. All the text in **aide** is *color adaptive.* Color adaptive text maintains legibility by adjusting the color of a piece of text before it is drawn so that it is always perceived to be a specific color, regardless of the background. This is done by adjusting the color of the text to compensate for the interactive effect of the background color (Ishizaki, 1991).

Finally, all of the text in **aide** is anti-aliased, also contributing to its legibility.

editing the encoding

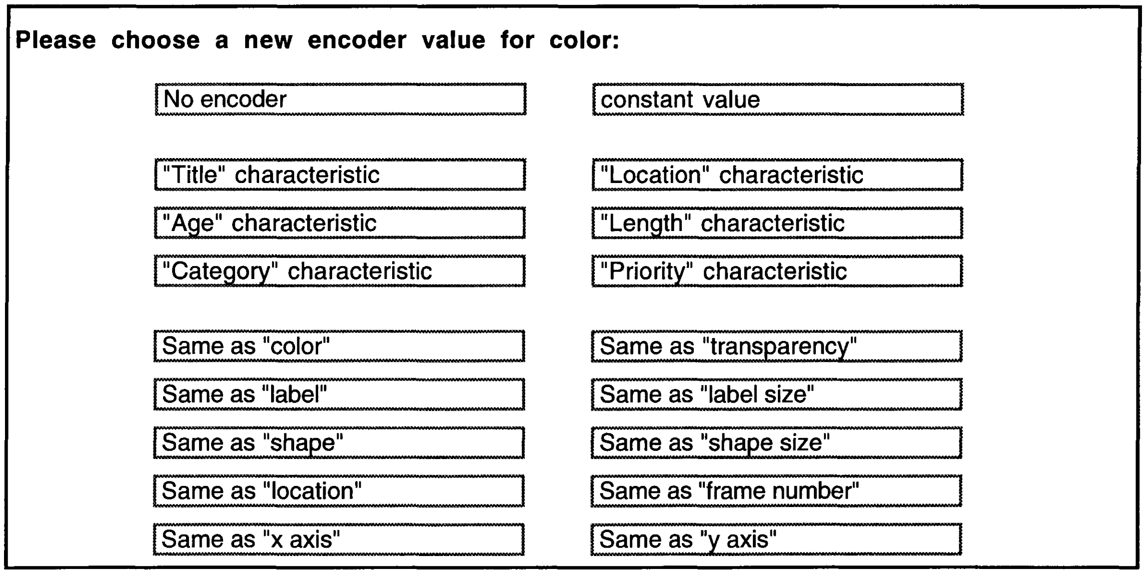


Once the *final graphic* version of the image is rendered, it is possible that the user is unsatisfied with some of the aspects of the graphic. The user can edit the image through the *key* window by clicking on any of the values in the window. If the user clicks on any of the values shown in the second column of the key window, the user can change the *encoder* assigned to that graphic variable. For example, in the news chartmap example shown earlier, the color *encoder* was assigned to the characteristic named "Format". The line of the *key* is shown in Figure 5.10.

graphic variable	derived from	min	max	flipped
color	***Format***	-	-	no

Figure 5.10. The color encoder line from the key for the news chartmap shown in Figure 5.1.

If the user clicked down on the word "Format" in order to change how the colors of the marks were determined, the *encoder editor* window would appear. This window is a palette containing one button for each of the characteristics of the database, along with buttons for constant values, an empty encoder, or the same encoder that is assigned to a different graphic variable. A sample *encoder editor* for this example is shown below.



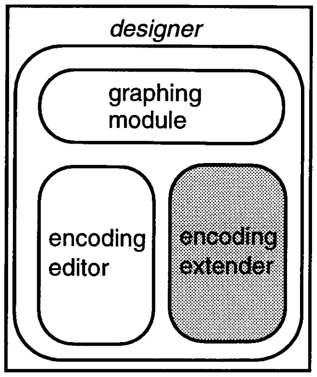
Please choose a new encoder value for color:

No encoder	constant value
"Title" characteristic	"Location" characteristic
"Age" characteristic	"Length" characteristic
"Category" characteristic	"Priority" characteristic
Same as "color"	Same as "transparency"
Same as "label"	Same as "label size"
Same as "shape"	Same as "shape size"
Same as "location"	Same as "frame number"
Same as "x axis"	Same as "y axis"

Figure 5.11. The encoder editor.

In addition, the user can change the maximum and minimum values for certain graphic variables, such as *transparency* and *shape size* by clicking on the values in the "max" or "min" columns. The user can change the orientation of a particular graphic variable by clicking in the "flipped" column.

extending the encoding



One of the assumptions made in designing **aide** was that often, a user has no a priori knowledge about the data and wants to get a good overall view of the data. One of the modules of **aide** rests on the notion that as much information as possible should be encoded in the final image, and tries to design (or re-design) graphics in such a way that as many characteristics of the data as possible are visualized. This module is called the *encoding extender* and uses a rule-based approach similar to that of Mackinlay and others.

To *extend* a graph, a user can click on the "extend" button drawn above the *key*. This button triggers a set of rules that check each of the graphic variables. If a graphic variable has no encoder assigned to it, and there are characteristics of the data that are not graphically represented, **aide** tries to find an appropriate match. The basis for this matching is summarized in Figure 5.12.

graphic variable	kind desired	aspect desired	other
color	nominal	any	length should be close to (but not more than) size of color palette
shape	nominal	any	length should be close to (but not more than) size of shape palette
label	nominal	any	range should be a factor of the total number of elements
line number	nominal	any	range should be a factor of the total number of elements
transparency	quantitative	temporal	
location	quantitative	locative	
x axis	quantitative	temporal/any	
y axis	quantitative	any	
shape size	quantitative	any	
label size	quantitative	any	
bar height	quantitative	any	
bar width	quantitative	any	
linewidth	quantitative	any	
frame number	ordinal	temporal	

Figure 5.12. A summary of the rules used to extend an encoding.

For example. assume that for the database of news articles described earlier, the user chose an encoding that created a very simple graph, one in which only two graphic variables were assigned encoders: location was assigned the "Location" characteristic, and "label" was assigned the "Title" characteristic. In this case, the other graphic variables (color, transparency, shape, shape size, etc.) were all assigned empty encoders. In addition, "Age", "Priority", and "Format" would not be visually represented. If the user directed **aide** to try to extend the encoding, the system would discover that the color encoder was not assigned to a characteristic, and would search through the input data for a non-represented characteristic that was nominal and had a length less then the number of available colors (12 in the default palette). As a result, it would assign the "Format" characteristic to the color encoder, extending the amount of information in the final graphic.

This prioritization is loosely based on the work done by Cleveland and McGill. However, it needed to be adapted to **aide's** unique set of graphic variables. For example, no previous automated design system uses transparency in its diagrams, so a measure of the visual effectiveness of transparency needed to be assessed. This new prioritization was researched and created by the author. While the table above describes the types of characteristics that are desirable for each graphic variable, it does not tell the entire story. In addition, there is a specific order in which **aide** tries to assign the graphic variables, and this list is different for each of the different graphical forms. Figure 5.13 contains these priorities.

scatterplots	**linecharts**	**vertical barcharts**	**horizontal barcharts**
location	x axis	x axis	y axis
x axis	y axis	bar height	bar width
y axis	line number	color	color
shape size	color	bar width	bar height
color	shape	transparency	transparency
shape	label	label	label
label	shape size		
frame number	label size		
transparency	transparency		
label size	line width		

Figure 5.13. The priorities for instantiating graphic variables when extending an encoding.

6 conclusions

This section contains an analysis of **aide**. This includes a summary of the system, an analysis of its functionality, a list of areas for future work, and some conclusions.

summary

Because of the large demand high-quality visualizations of data, and because this process is complicated and often expensive, automated layout systems have been created to try and perform the job of graphic designer. Most of these systems use rule-based design algorithms to convert data into images. While these systems function well in certain situations, their functionality is severely limited.

The prototype research system **aide** was implemented advance the field of automated layout in four distinct ways. These are: 1) combining a case-based approach with traditional rule-based automatic layout paradigms, 2) treating locative and temporal characteristics as special and creating specific design rules for these types of data characteristics, c3 creating dynamic graphics, and 4) building a system capable of creating more information-rich graphics through the use of advanced design and display technologies.

The main motivation for **aide** was to build a system that designed statistical graphics in a manner similar to that of a trained graphic designer. Namely, by trying to create new graphics based upon previously successful designs. By using an artificial intelligence technique called case-based reasoning, **aide** creates visual representations of data by analyzing the data, comparing it to a library of examples of other graphical presentations, and adapting one or more of the examples to the input data provided. Using this methodology, **aide** has been successful in producing effective graphics from input data. By comparing the input data to a library of example and recommending examples to the user for use as a basis for new visualizations of the input data, **aide** puts the power of a graphic designer in the hands of an ordinary user, allowing him to create effective, high quality graphics from data without significant training.

The system is also interesting because it extends some of the theories of automated design. **aide** was built with the notion that *temporal* and *locative* characteristics are special and that unique design paradigms must be employed to present these types of information. As a result, **aide** has knowledge of these aspects of characteristics and can use this accordingly. The system's ability to display this information in unique ways demonstrates the need for any automated system to be able to sort data characteristics in as many ways as possible, and to have special design paradigms for characteristics that have special cognitive value, like *temporal* and *locative* ones.

Like other automated layout systems, **aide** capable of designing and displaying static visualizations of data in many different forms, including standard scatterplots, barcharts and linecharts. In addition, however, the system expands the field of automated layout by being able to design and present dynamic graphics as well as static ones.

aide demonstrates that case-based reasoning is an effective way to generate graphics from data. By combining this technique with traditional rule-based automated layout techniques, extending traditional rule-based algorithms for automated layout by having special knowledge of certain important types of characteristics, by being able to produce dynamic and static graphics combined with its advanced display environment and design skills, **aide** is a powerful system capable of producing a wide range of effective statistical graphics that contribute to both the fields of artificial intelligence and automated layout.

areas for future work

The success of a case-based system not only lies in its ability to adapt old solutions to problems, but in the ability to choose appropriate old solutions to adapt. The more solutions available to the system, the more options it has to choose from. In other words, a case-based system is only as good as its example library. As of the time of this printing, **aide**'s library contained around 30 examples, but these examples were not well distributed throughout the entire space of possible data *shapes*. Most of the examples in the library had less than ten characteristics, and most had no more than three characteristics of any one *kind* or *aspect*. The system could be improved by expanding the example library.

One of the ways in which **aide** expands the field of automated design is by treating *temporal* and *locative* information as special types. While it is true that singling out these two *aspects* allow a more robust categorization than without, merely adding these two does not cover the entire range of characteristics that could benefit from having special design rules. The current implementation requires a moderate amount of work to add new *aspects*. It is ineffective to rewrite the system for every new *aspect*, and **aide** could benefit from using a grammar to define aspects and their special design rules. This would allow the system to be expanded more easily.

In the current implementation, values for *locative* characteristics must be precomputed for the size of the screen. There should be a more system-independent format of *locative* information. For example, real-world locative information could be stored in latitude/longitude. However, with such a scheme, a conversion routine from latitude/longitude to the screen position must be provided. This is easy for real-world coordinates, but for arbitrary *basemaps*, there is no simple way to make this conversion routine separate from the system code, unless an interpreter of some sort was written.

Both Bertin and Tufte argue that for images such as certain kinds of maps or graphs that represent cyclical data (such as a 24-hour span), the image should be rendered in such as way as to reflect this aspect of the data. This is done by "wrapping" the image, printing part of the left portion of the image to the right of what would normally be the right edge. This technique can provide the user with the ability to study any subset of the continuum without an unnatural break and will soon be added to **aide.**

While **aide** is not strictly a graphic production system, it does provide the user with some ability to edit the parameters of the graph. These details, such as tick marks, graph size, and grid lines, along with the graphic variables, are part of the graph's *encoding*. The list of these production details is severely limited however and could benefit from some expansion. In addition, there are some graphic variable that could be added to the encodings. For example, texture could be added to each of the graphic forms, and line style (solid, dashed, etc.) could be added to linecharts. These additions would increase the expressivity of the graphics and would be relatively simple to add to the system.

The system is currently able to design only four basic forms of graphic diagrams (scatterplots, linecharts, vertical and horizontal barcharts). Even with the addition of dynamic graphics and other extensions, the number of different graphical forms in the real world is much larger. **aide** should be expanded to be able to draw more complicated forms of graphics, such as span graphs, box plots, and others.

Dynamic graphics are a powerful way to provide the user with information about *temporal* characteristics, but they are not they only way to present temporal information. One of the most common ways to present temporal information in a static display is by use of a *storyboard.* These presentations are made by rendering a different image for each of the time values and presenting them side by side. Figure 6.1 shows an example of a storyboard presentation of the population of Chinese poets during for major dynasties (Cheng-Siang via Tufte, 1990). Although it would be easy to extend **aide** to be able to create scatterplots, the current implementation does not provide this functionality.

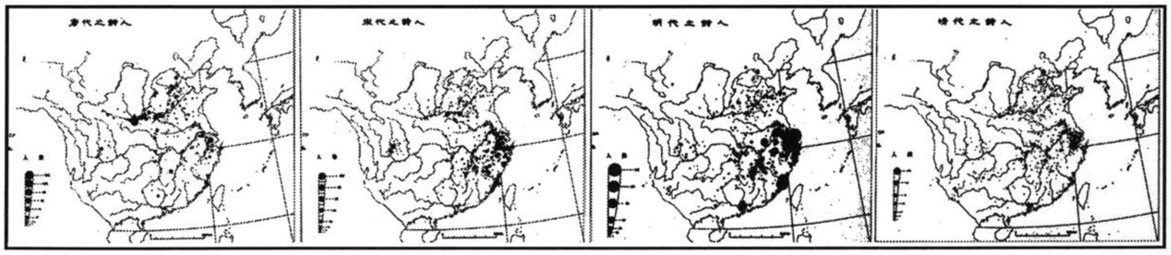


Figure 6.1. An example of a storyboard.

In addition, **aide** needs a more flexible structure for the input data. Currently, the data is organized into a two-dimensional structure. However, the system is built to display multidimensional data that is often difficult to reduce to a simple two-dimensional structure. As a result, the input data file format needs to be expanded.

One way to improve **aide**'s performance would be to improve its rating function. In the current implementation, the rating function was created and adapted by the author over the course of writing the software. It would be nice to add a "learning" module to the code that watched the choices that the user made or to conduct an ablution study to determine what measures were most important in enabling the "best" match.

aide was written to function as an assistant to a user. There are situations, however, where design needs to happen without human intervention. There should be an optional mode for **aide** in which it makes all of the design decisions, removing the user from the process entirely.

evaluation

There are many parts of **aide** that can be evaluated, but the most important issue is whether or not the system is capable of taking data and producing relevant graphs from that data. The answer to this question is yes.

aide's rule-based graphic design paradigm is similar to that of previous work, and as such produces encodings that can create graphics similar in style to that of previous work. The enhanced display capabilities and additional rules for locative and temporal information increase the quality of the graphical presentations. The case-based section of **aide** performs quite well when the input data's shape is very similar to that of one or more of the examples in the library. When the input shape is far outside the space spanned by the examples, the system still does a reasonable job, but there is still a lot of room for improvement. One of the easiest ways is to increase the example space by entering more encodings. In addition, the adaptation module routines need to be improved.

One interesting aspect of the system is that it makes its design choices based on input from a user. If the user chooses an example with a shape very different than that of the input data, the results are unpredictable. The system can notify the user that he has made a "poor" choice, but nonetheless continues to try to adapt the chosen example to the input data. It is often the case that this process does not produce good results and the resulting graphic is unintelligible.

Even with its limitations, **aide** is a powerful tool for anyone who needs to create graphics from data. It expands previous research in four important ways, and is capable of creating high-quality, informative, effective graphics from data. **aide** contributes to both the field of automatic layout and artificial intelligence by playing the role of a graphic designer's assistant.

references

Anscombe, F. J.," Graphs in Statistical Analysis", *American Statistician*, Volume 27, 1973.

Bardon, D.," Adaptive Typography for Dynamic Mapping Environments", Visible Language Workshop, Media Laboratory, MIT, Cambridge, MA, USA, 1991.

Bertin, J., Semiology of Graphics, W. J. Berg translation, University of Wisconsin Press, Madison, WI, USA, 1967, 1983.

Carbonell, J., "Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition", *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems*, B. G. Buchanan and D. C. Wilkins, ed., Morgan Kaufmann, San Mateo, CA, USA, 1993.

Cheng-Siang, C., *An Historical and Cultural Atlas of China*, Tokyo, 1981.

Christensen, J., Marks, J., and Shieber, S., "Algorithms for Cartographic Label Placement", *Proceedings of the American Congress on Surveying and Mapping*, Volume 1, New Orleans, LA, USA, 1983.

Cleveland, W. S. and R. McGill, "Graphical perception: Theory, experimentation and application to the development of graphical methods", *Journal of The American Statistics Association*, Volume 79, 1984.

Colby, G. "Intelligent Layout for Information Display: An Approach Using Constraints and Case-based reasoning". S.M. Thesis, Media Arts and Sciences, Massachusetts Institute of Technology. Cambridge MA, USA, 1991.

de Dainville, F., *Population 13*, Number 3, France, 1958.

Freeman, H. and J. Ahn, "A packing problem with applications to lettering of maps", *Proceedings of the Seventh Annual Symposium on Computational Geometry*, ACM, North Conway, NH, USA, 1991.

Gnanamgari, S., Information Presentation through Default Displays, Computer and Information Sciences Report 81-05-02, University of Pennsylvania, Philadelphia, PA, USA, 1981.

Ishizaki, S., et. al., "Adjusting simultaneous contrast effect for dynamic information display", *Proceedings of the 8th Joint Conference on Color Technology*, pp. 81-84, 1991. (in Japanese)

Johnson, M., "WavesWorld: A Distributed Parallel Testbed for Autonomous Animation", unpublished.

Mackinlay, J., "Automating the Design of Graphical Presentations of Relational Information", *ACM Transactions of Graphics*, Volume 5, Number 2, USA, 1986.

MacNeil, R., "Adaptive Perspectives: Case-Based Reasoning with TYRO, the Graphic Designers Apprentice", *Proceedings of the IEEE 1990 Workshop on Visual Languages*, USA, 1990.

MacNeil, R.," TYRO: A Constraint Based Graphic Designers Apprentice", *Proceedings of the IEEE 1989 Workshop on Visual Languages*, USA, 1989.

Maes, "How to do the right thing", *Connection Science Journal*, February, 1990. Also, MIT AI-LAB Memo 1180.

Marks, J. and S. Sheiber, "The computational complexity of cartographic label placement", Technical Report TR-05-01, Harvard University, March 1991.

Marey, E. J., *La Methode Graphique*, Paris, France, 1885.

Mashiushi, T., et al., "6000x2000 Display Prototype", Visible Language Workshop, Media Laboratory, MIT, Cambridge, MA, USA, 1991.

Moholy-Nagy, L., *vision in motion,* Paul Theobald and Company, Hillson & Etten Company, printers, Chicago, IL, USA, 1947, 1961.

Playfair, W., *Commercial and Political Atlas,* London, England, UK, 1786.

Riesbeck, C. and R. Schank, *Inside Case-Based Reasoning,* Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1989.

The Prisoner, ITC Entertainment, London, England 1968.

Sivasankaran, V. K. and C. L. Owen, "Data Exploration: Transposition Operations in Dynamic Diagrams", Design Processes Laboratory, Institute of Design, Illinois Institute of Technology, Chicago, IL, USA.

Tufte, E. R., *Envisioning Information*, Graphics Press, Cheshire, CT, USA, 1990.

Tufte, E. R., *The Visual Display of Quantitative Information,* Graphics Press, Cheshire, CT, USA, 1983.

Zipf, G., *The Psycho-biology of Language*, Houghton-Mifflin, Boston, MA, USA, 1935.

appendix a - related readings

Adams, W., "An Ergonomical Approach to Bertin's Method", *Information Design Journal,* Volume 3, 1989.

Albers, J., *Interaction of Color,* New Haven Press, New Haven, CT, USA, 1963.

Ayers, L. P., *The War with Germany,* Washington, D. C., USA, 1919.

Bertin, J., *Graphics and Graphic Information Processing,* W. J. Berg and P. Scott translation, Walter de Gruyter & Co., Berlin, Germany, 1970.

Casner, S. M., "A Task-Analytic Approach to the Automated Design of Graphic Presentations", *ACM Transactions on Graphics,* Volume 10, Number 2, Association for Computing Machinery, New York, NY, USA, 1991.

Chernoff, H. "The Use of Faces to Represent Points in k-Dimensional Space Graphically", *Journal of the American Statistical Association,* Volume 68, 1973.

Christ, R., Review and Analysis of Color Coding Research for Visual Displays, *Human Factors,* Volume 17, Number 6, 1975.

Colby, G. and L. Scholl," Transparency and Blur as Selective Cues for Complex Visual Information", Visible Language Workshop, Media Laboratory, MIT, Cambridge, MA, USA, 1991.

Cooper, M., *Design Quarterly,* Volume 142, MIT Press, Cambridge, MA, USA, 1989.

Crichton, M., *Electronic Life: How to Think About Computers,* Ballantine Books, New York, NY, USA, 1983.

Davidoff, J., "The Role of Colour in Visual Displays", *International Reviews of Ergonomics,* Volume 1, D. J. Oborne, ed., Taylor and Francis, 1987.

Davis, W. and A. McCormack, *The Information Age,* Addison-Wesley, Reading, MA, USA, 1979.

Dondis, D. A., *A Primer of Visual Literacy,* MIT Press, Cambridge, MA, USA, 1973.

Feiner, S., "A Grid-Based Approach to Automating Display Layout", *Proceedings of Graphics Interface '88,* Morgan Kaufmann, Palo Alto, CA, USA, 1988.

Friedell, M. "Automatic Synthesis of Graphical Object Descriptions", *Computer Graphics,* Volume 18, Number 3, 1984.

Friedell, M., J. Barnett, and D. Kramlich, "Context-Sensitive Graphic Presentation of Information", *Computer Graphics,* Volume 16, Number 3, 1982.

Graphis Diagram 1, The Graphic Visualization of Quantitative Information, Procedures, and Data, B. Martin Pedersen, ed., Graphis Press, Zurich, Switzerland, 1988.

Hollands, J. G., and I. Spence, "Judgments of Change and Proportion in Graphical Perception", *Human Factors,* Volume 34, Number 3, 1992.

Holmes, N., *A Designers Guide to Creating Charts and Diagrams,* Watson-Guptill Publishers, New York, NY, USA, 1984.

Huff, D., *How to Lie with Statistics,* Norton, New York, NY, USA, 1954.

Hulburt, A., *The design concept,* Watson-Guptill Publications, New York, NY, USA, 1981.

Hurlburt, A., *Layout: the design of the printed page*. Watson-Guptill Publishers, New York, NY, USA, 1977.

Johnson, M. B., "Build-a-Dude", S.M. Thesis, Media Arts and Sciences, Massachusetts Institute of Technology, Cambridge, MA, USA, 1991.

Lieberman, H., "Communication of Expert Knowledge in Graphic Design", Visible Language Workshop, Media Laboratory, MIT, Cambridge, MA, USA, 1988.

Meyer, J., and D. Shinar, "Estimating Correlations from Scatterplots", *Human Factors*, Volume 43, Number 3, 1992.

Muller-Brockmann, J., *The Graphic Designer and His Design Problems*, D. Q. Stephenson translation, Hastings House Publishers, New York, NY, USA, 1983.

Pinker, S., "Visual Cognition: an introduction", *Cognition*, Volume 18.

Pinker, S., "A Theory of Graph Comprehension", MIT Brain and Cognitive Sciences Department, Cambridge, MA, USA, 1981.

Pokorney, J. and V. C. Smith, "Colorimetry and Color Discrimination", Eye Research Laboratories, The University of Chicago, Chicago, IL, USA.

Robertson, P., Visualizing Color Gamuts: A User Interface for the Effective Use of Perceptual Color Spaces in Data Displays, *Computer Graphics and Applications*, 1988.

Tukey, J. and M. B. Wilk, *Data Analysis and Statistics: Techniques and Approaches, The Quantitative Analysis of Social Problems*, E. Tufte, ed., Reading, MA, USA, 1970.

Tukey, J., "Some Graphic and Semigraphic Displays," *Statistical Papers in Honor of George W. Snedecor,* T. A. Bancroft, ed., Ames, IA, USA, 1972.

Tukey, J., *Exploratory Data Analysis*, Reading, MA, USA, 1977.

Wainer, H. and D. Thissen, "Graphical Data Analysis," *Annual Review of Psychology,* Volume 32, 1981.

Ware, C. and J. C. Beatty, "Using Color as a Tool in Discrete Data Analysis", Computer Science Department, University of Waterloo, Report CS-85-21, Waterloo, CANADA, 1985.

Ware, C. and J. C. Beatty, "Using Color Dimensions to Display Data Dimensions", *Human Factors*, Volume 30, Number 2, 1988.

Weitzman, L., "Designer: A Knowledge-Based Graphic Design Assistant," MCC Technical Report ACA-HI-017-88, Austin, TX, USA, 1988.

Wurman, R. S., *Information Anxiety,* Doubleday Dell Publishing Group, Inc., New York, NY, USA, 1989.

Yee, M. M., "System Design and Cataloging Meet the User: User Interfaces to Online Public Access Catalogs", *Journal of the American Society for Information Science,* Vol. 42, John Wiley & Sons, 1991.

appendix b - biographies of readers

Ken Haase is a member of the faculty at the MIT Media Lab as part of the Common Sense and Learning Group. His research focus is currently Knowledge Representation, and his FRAMER system is being used in many ongoing research projects at the Media Lab.

Joe Marks is a member of the research staff at Digital Equipment Corporations Cambridge Research Lab. Dr. Marks received his Ph.D. in Computer Science from Harvard University in 1991. Prior to his graduate studies, he was employed at BBN Laboratories in Cambridge, MA, where he worked on simulation and training systems, and at Wang Laboratories, where he worked on office-information systems. His research interests include computer graphics, artificial intelligence, user interfaces, and stochastic optimizations. In addition to his research activities, he has taught several courses on computer graphics, graphical user interfaces, theory of computation, and introductory programming at Harvard University and Harvard Extension School.