A COGNITIVE APPROACH TO THE DESIGN OF INFORMATION GRAPHICS

Technical Report AIP - 143

Stephen Casner

University of Pittsburgh
Pittsburgh, PA 15260

May, 1989

This research was supported by the Computer Sciences Division, Office of Naval Research, under Contract Number N00014-86-K-0678. Reproduction in whole or in part is permitted for purposes of the United States Government. Approved for public release; distribution unlimited.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AIP-143	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Carnegie-Mellon University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Computer Sciences Division Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) Department of Psychology Pittsburgh, Pennsylvania 15213	7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, Virginia 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Same as Monitoring Organization 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0678
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS p4000ub201/7-4-86 PROGRAM ELEMENT NO. N/A; PROJECT NO. N/A; TASK NO. N/A; WORK UNIT ACCESSION NO N/A

11. TITLE (Include Security Classification)

A Cognitive Approach to the Design of Information Graphics

12. PERSONAL AUTHOR(S)
Casner, Stephen

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 86Sept15 TO 91Sept14	1989, May	62

16. SUPPLEMENTARY NOTATION

submitted manuscript

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

Design, Human Factors, Algorithms, Theory, Graphics Design, Task Analysis, Perception, Visual Languages, User Interface

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

SEE REVERSE SIDE

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Alan L. Meyrowitz	22b. TELEPHONE (Include Area Code) (202) 696-4302 22c. OFFICE SYMBOL N00014

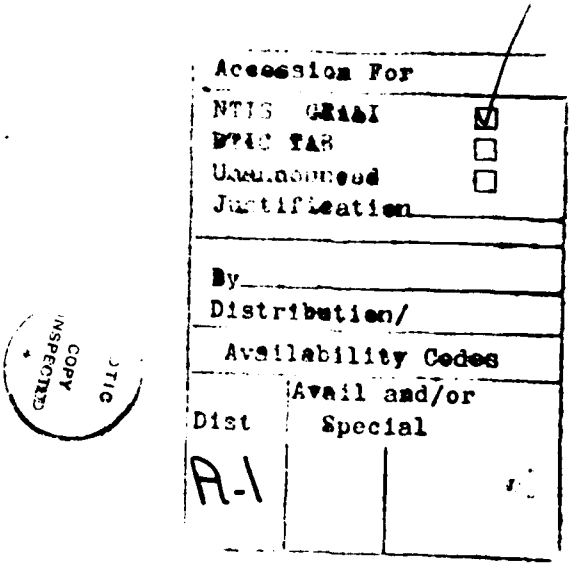
DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

Abstract

Graphical representations popularly thought to be useful for communicating and processing information yield mixed results when tested with real users. Cognitive research suggests that current design methodologies fail to exploit the potentials of graphics for expediting human performance of information-processing tasks: (1) allowing users to substitute less effortful visual procedures in place of more demanding non-visual procedures; and (2) streamlining users' search for information by supporting visual search heuristics. BOZ is a design algorithm that constructively applies cognitive principles of the efficiencies that graphical displays can offer to the problem of discovering novel displays to support specific user tasks. BOZ analyzes formal task descriptions and proposes visual displays and procedures that can help streamline performance of a task. BOZ is used to generate graphical alternatives to a standard tabular display of airline schedule information to support a set of common airline reservation tasks. Reaction time studies done with real users are reported that show that the BOZ-designed displays significantly reduce users' performance time to the task. Regression analyses link the observed efficiency savings to visual procedure substitutions and pruning of research. BOZ is also shown to be useful for analyzing existing displays to discover clever design features that can then be subsequently incorporated into BOZ's design algorithm.



A Cognitive Approach to the Design of Information Graphics

Stephen Casner
Learning Research & Development Center
University of Pittsburgh
Pittsburgh, PA 15260

May 1989

This work is supported by the Office of Naval Research, University Research Initiative, Contract Number N00014-86-K-0678, and in part by Virtual Machine Corporation, Pittsburgh, PA. Reproduction in whole or in part is permitted for any purpose of the United States Government. Approved for public release; distribution unlimited.

Thanks to Stellan Ohlsson, Jill Larkin, Jeffrey Bonar, and Alan Lesgold for helpful comments and criticisms.

A Cognitive Approach to the Design of Information Graphics

STEPHEN CASNER
University of Pittsburgh

Graphical representations popularly thought to be useful for communicating and processing information yield mixed results when tested with real users. Cognitive research suggests that current design methodologies fail to exploit the potentials of graphics for expediting human performance of information-processing tasks: (1) allowing users to substitute less effortful visual procedures in place of more demanding non-visual procedures; and (2) streamlining users' search for information by supporting visual search heuristics. BOZ is a design algorithm that constructively applies cognitive principles of the efficiencies that graphical displays can offer to the problem of discovering novel displays to support specific user tasks. BOZ analyzes formal task descriptions and proposes visual displays and procedures that can help streamline performance of a task. BOZ is used to generate graphical alternatives to a standard tabular display of airline schedule information to support a set of common airline reservation tasks. Reaction time studies done with real users are reported that show that the BOZ-designed displays significantly reduce users' performance time to the task. Regression analyses link the observed efficiency savings to visual procedure substitutions and pruning of search. BOZ is also shown to be useful for analyzing existing displays to discover clever design features that can then be subsequently incorporated into BOZ's design algorithm.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques--*user interfaces*; H.1.2 [**Models and Principles**]: User/Machine Systems--*human information processing*; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems; I.3.6 [**Computer Graphics**]: Methodology and Techniques--*ergonomics*.

General Terms: Design, Human Factors, Algorithms, Theory.

Additional Key Words and Phrases: graphics design, task analysis, perception, visual languages, user interface.

1. THE COGNITIVE FUNCTION OF GRAPHICAL DISPLAYS

A striking conclusion of recent studies concerned with understanding how and why graphical representations are useful is that it is a false assumption that graphical displays are inherently better than other representations, or that perceptual

inferences are in general made more efficiently or accurately than non-perceptual inferences [13, 19, 20]. Rather, these studies suggest that the usefulness of a graphical display is a function of the *task* that the user is performing. Twenty-nine independent empirical studies surveyed in Jarvenpaa and Dickson [16] found graphical displays superior to tabular displays for a restricted set of information-processing tasks, and observed no benefits or poorer performance for other tasks. The implication is that effective graphic design should begin with the task that a to-be-designed graphic is intended to support, and be focused on finding those parts of a task, if any, that might be performed more efficiently within the context of a graphical display.

Larkin and Simon's [20] theoretical analysis points out two ways in which graphical displays can facilitate human performance of information-processing tasks:

- **Substituting visual operators for logical operators:** Graphical displays often allow users to substitute quick and easy perceptual judgements or graphical manipulations (*visual operators*) in place of more effortful non-visual reasoning steps (*logical operators*) that comprise a particular task. Visual operators such as distance and size determinations, spatial coincidence judgements, and color comparisons, sometimes give users the same information as more demanding logical operators such as mental arithmetic, logical reasoning steps, or feature comparisons.

- **Reducing search:** Good graphics often reduce the time that the user must spend searching for information they need. This is accomplished either by grouping together information required to draw a particular inference into one spatial locality, or by employing techniques such as shading and spatial arrangement that help guide the eye toward relevant information and away from irrelevant information.

To illustrate the two ways that graphical displays can help users, Figure 1 shows a graphic used for train schedules in France in the late 1800's [22].

****
Figure 1 here
****

Marey's train schedule can be viewed as a 2-dimensional visual data structure that indexes time and place information along the horizontal and vertical axes, respectively. To retrieve departure and arrival times for a train, the user must perform coincidence judgements along the horizontal axis. Note that for this simple task in isolation, the train schedule does not result in any savings for the user. Searching for departure and arrival times in a tabular presentation such as Figure 2 would seem to progress as quickly and perhaps more accurately. Consequently, with respect to the task of retrieving departure and arrival times, allowing users to substitute spatial coincidence judgements for table lookup seems to be of no use in that it may be both inefficient to use and prone to errors due to the imprecise representation of times. The empirical studies generalize this to show that in most cases tabular representations are best for "information extraction" tasks [16], and thus there seem to be no *inherent* advantages of representing information graphically.

****
Figure 2 here
****

When used for other tasks, the visual train schedule offers several advantages by allowing the user to substitute visual operators in place of more difficult logical operators that would ordinarily require logical reasoning and mental arithmetic, and reducing the time the user must spend searching for information. For example, to

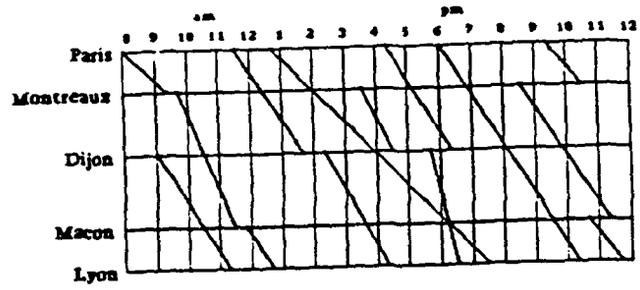

am
pm
8 9 10 11 12 1 2 3 4 5 6 7 8 9 10 11 12
Paris
Montreaux
Dijon
Macon
Lyon

Figure 1

Origin/Destination	Departs	Arrives
Paris-Montreaux	8:00	9:15
Dijon-Lyon	9:00	11:15
Montreaux-Macon	9:45	11:30
Paris-Dijon	11:30	1:30
Macon-Lyon	11:45	12:45
Paris-Lyon	12:30	6:30
Dijon-Lyon	2:30	4:15
Montreaux-Dijon	3:30	4:30
Paris-Dijon	4:30	6:15
Dijon-Lyon	5:30	6:30
Paris-Lyon	6:00	10:15
Montreaux-Macon	8:30	11:30
Paris-Montreaux	9:30	10:30
Macon-Lyon	10:45	11:45

Figure 2

find a route between Paris and Lyon we can search for a single line that runs directly from Paris to Lyon, or find a series of lines such that each successive line lies to the right of the previous line. The savings in search time is achieved because of the way the graphical display indexes the trains by city and time. Notice that accomplishing this same task with the tabular display requires that we continually search the entire list of trains since they are not indexed by city. We can also determine the speed of a train by judging the slope of the line between cities. Rather than dividing the total number of miles traveled (not represented in either display) by the difference of the departure and arrival times, we can compare the speeds of trains by performing simple slope judgements. We can determine the layover between two trains in an intermediate city by estimating the distance between the end of the line depicting the first train and the beginning of the line that depicts the second train. This convention allows the user to substitute a simple distance judgement in place of subtracting the departure and arrival times.

Note that a different user who wishes to understand the route structure of French trains would find both of the displays shown in Figures 1 and 2 cumbersome and would benefit most from the graphic shown in Figure 3. This structuring of the data trades away the departure/arrival time capabilities to allow users to find a city quickly (indexed by geographical location), and to determine routes by performing connectivity judgements between the city names.

****

Figure 3 here

****

In summary, the three alternative ways of presenting the same information show that effective graphical displays are not likely to follow from a design methodology concerned primarily with the information to be displayed but rather from a careful analysis of the *tasks* that manipulate the information. That is, it is unlikely that we

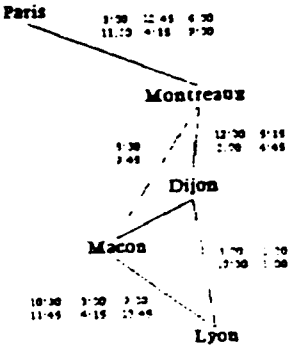


Figure 3

can design graphical displays to support tasks without explicitly considering the nature of those tasks.

The research described in this paper explores a cognitive approach to the design of interactive graphical displays based on an analysis of the tasks for which they are intended to support. Section 2 reviews previous work related to the problem of designing graphical displays. Section 3 states the task-based design problem for interactive graphical displays. In Section 4, a formalism is described that allows designers to characterize the set of logical operators that comprise a user task. Section 5 presents a catalog of visual operators describing information-processing activities that occur within the context of a graphical display. Section 6 describes a design algorithm called BOZ that analyzes a task description and searches the visual operator catalog for ways to substitute visual for logical operators and take advantage of visual search heuristics. BOZ proposes specifications (but not implementations or renditions) of interactive graphical displays that support human performance of the visual operators and search heuristics it finds. In Section 7, BOZ is used to design alternative graphical displays to support a set of airline reservation tasks. Section 8 discusses an experiment in which participants used the BOZ-designed airline reservation graphics. Results show significant decreases in users' performance times when using the BOZ-designed airline reservation displays, and suggest that users obtained the efficiency savings through visual operator substitutions and heuristic visual search.

2. PREVIOUS WORK

The following surveys theoretical and empirical work concerned with the problem of designing effective graphics.

2.1 Designing by intuition

Tufte [27], Bertin [2], Cleveland [9], and Schmid [25] describe design principles concerned with designing graphical displays for the purpose of storage, communication, and manipulation of information. An important limitation of these principles is that they are limited to recognizing and correcting bad practices used in existing graphics. That is, given that the designer has set out to create a graphic using his or her own design intuitions, these principles help the designer to avoid practices that are known to make graphics ambiguous, confusing, or generally less usable. The consequence of this limitation is that they do not help the novice designers how to begin designing a display from scratch. A second limitation of the intuitionalist approaches is that they focus entirely on the information to be presented in a graphic and do not include a concern for the tasks for which the graphics are designed to support.

2.2 Mackinlay's APT

Mackinlay [21] describes APT, a tool that designs static presentations of relational information. A significant contribution of Mackinlay's work is to formally characterize something that many of the intuitionalists informally alluded to: that graphical presentations can be expressed as sentences in a formal graphical language that have the same precise syntax and semantics as propositional formalisms. The advantage of having a formalism for graphical presentations is that it provides a set of criteria for deciding the role of each visible sign or symbol placed in a graphic. A second contribution of APT is that, unlike intuitionalist proposals, APT designs graphics with a minimum amount of intervention on the part of the designer. That is, APT embodies a genuinely *prescriptive* theory of how to design a graphic.

Limitations of APT include the following. First, APT's analysis of formal graphical languages does not extend to *actions* that allow users to manipulate the graphical objects in a display. For instance, graphical interfaces such as those used in intelligent tutoring systems and visual programming languages [4] require that user

be able to graphically manipulate the objects in a display, and that these manipulations have meanings of their own independent of the meanings of the graphical symbols themselves. Second, APT's design algorithm is based on an analysis of the information to be presented and doesnot consider the tasks that the user is to perform. This prevents APT from directly addressing the issue of task-related advantages of graphical representations. Third, when deciding which of a set of alternatives representations to choose, APT generalizes an experimental result that shows that some designs allow users to more accurately extract information from a graphic [10]. More recent experimental work [16] shows that this criterion is an unreliable indicator of how accurately other tasks are performed with the same or different graphic. In general, unreliable differences between users, tasks, and situations suggest that determining the usability of a graphic is an empirical question not decidable by algorithm.

2.3 Cognitive Approaches

Larkin and Simon's work [20] was first to pinpoint the two ways in which graphical displays can help users when performing complex information-processing tasks: (1) allow users to substitute quick visual operators for more effortful logical operators; and (2) reduce the amount of time that users spend searching for information. Cognitive research has discovered that graphical representations can also effect the way that users manage heavy information workloads in short-term memory [15], and provide users with effective strategies for organizing their knowledge about a task in a form readily applicable to problem-solving situations [18].

Cognitive research has pinpointed several design parameters that can be tuned during the process of engineering good displays and has thereby enabled a new generation of approaches that fill the gaps in previous approaches. The research described below is the first design methodology to attempt to apply principles of how graphics support tasks to the problem of designing novel displays.

3. THE DESIGN PROBLEM FOR INTERACTIVE GRAPHICAL DISPLAYS

The problem of designing graphical displays to support an information-processing task can be stated as follows:

> *Given a description of a task, analyze the description looking for ways to substitute visual operators in place of logical operators. Create a graphical display that presents the information relevant to the task in a form that allows the user to perform the visual operators. The visual operators and display chosen should demonstrably offer some efficiency advantage to the user by either reducing the time required to execute each operator, or by facilitating search techniques that reduce the overall time the user must spend searching for information in the display.*

The approach described here solves the design problem for interactive graphical displays using the following four components.

A *task description language* is used to formally specify the individual logical operators that comprise a task.

A catalog of *perceptual* and *graphical operators* (*POPs* and *GOPs*) describes a set of information-processing activities that can be performed in the context of a graphical display.

A *design algorithm* tractably searches the design space of visual operators and data structures suitable for a given task. The design algorithm uses theorem-proving techniques [12] when attempting to locate a set of perceptual and graphical operators that can be shown to be formally equivalent to a set of corresponding logical operators in a task description. Performing the visual procedure can then be shown to always yield the same result as performing the procedure given in the task description. Visual data structures are defined by collecting all information required to perform each visual operator and creating a structure that attempts to insure that all

necessary information is presented in the same spatial locality and in a form that supports the visual operators.

4. TASK DESCRIPTION LANGUAGE

The most important component of a task-based design methodology is a means of making explicit the information-processing activities that the to-be-designed representation is intended to support. Several task description languages exist in the literature, two of them targeted specifically for use in designing user interfaces [24, 5]. Given the particular use of a task description language intended here, there are two constraints that guided the decision to build a new task language over choosing one of the existing techniques. First, it is important that the task description language can be easily used by designers of graphical interfaces. Consequently, despite the many psychological distinctions that more sophisticated task languages allows us to make, it seemed impractical to adopt a language that would require extensive background knowledge, skill, or experience on the part of the designer. Second, it became clear after studying existing task languages that the representation schemes used by those languages would not easily lend themselves to the search algorithms required to match graphics to tasks efficiently. Existing task languages were designed to increase our understanding of the psychological nature of information-processing tasks and not necessarily for use in efficient computer algorithms.

The task description language used here adopts a representation scheme similar to first-order logic. Tasks are described using a collection of *logical operators*. A logical operator is composed of one or more *statements*. Statements use the three meta-commands: ASK, TELL, and RETRACT to query, assert, and remove facts from a simple database of facts. Facts are expressed using *predicates* that describe relations between two or more data variables or literals. Predicates are either pre-defined or user-defined. Pre-defined predicates include arithmetic and logical relations such as

PLUS, DIFFERENCE, AND, OR, NOT, etc. User-defined predicates can be any finite ordered n-tuples of variables or values that describe n-place relations. Logical operators can return values of arbitrary type.

For example, the following logical operator describes computing the layover between two airline flights:

```
(LAMBDA compute-layover (flightA flightB)
  (ASK (Departure flightB) D)
  (ASK (Arrival flightA) A)
  (ASK (DIFFERENCE D A) LAYOVER)
  (RETURN LAYOVER))
```

The keyword LAMBDA is used to denote a logical operator. The list (`flightA flightB`) is the set of arguments that `compute-layover` receives as input. The ASK meta-command states that the database of facts should be checked to see if the predicates that *follow can be shown to* be true: namely if there exist facts expressing the departure and arrival times of the two flights. The statement (`DIFFERENCE D A`) specifies that the pre-defined subtraction predicate is to be computed given the values D and A, and instantiate the variable LAYOVER with the result. The RETURN statement is used to specify the list of values that the operator is to return.

Domain sets are used to specify the types of data variables and literals that occur in user-defined predicates. Domain sets define the universe of discourse for the types of information that the graphical representation will allow the user to manipulate. Three types of domain sets are allowed in the present model: *quantitative*, *nominal* and *ordinal* [21]. For instance, if we were to assert a relation between airline flight numbers and times of the day: (`Departs FLIGHT1 T`), we would declare the variable `FLIGHT1` as type nominal and the variable `T` as type ordinal. The task of determining the name of the oldest individual in a group contains two domain sets, a

nominal domain set of names, and an ordinal domain set of integers that are associated with the names.

Figure 11 shows a complete task description for the activities involved in finding an airline flight satisfying time and cost constraints.

5. PERCEPTUAL AND GRAPHICAL OPERATORS (POPS AND GOPS)

The following describes a set of *perceptual and graphical operators (POPs and GOPs)*, or information-processing activities that are performed within the context of a graphical display and whose performance depends on the use of a graphical display. Perceptual and graphical operators are organized around a set of *primitive graphical languages* available to the designer of a graphical display [21]. Primitive graphical languages comprise the designer's resources for representing information visually. The set of primitive graphical languages used in the present model are shown in Figure 4. The primitive languages shown on left side of Figure 4 have the following common characteristic: they all rely on placing visible graphical symbols in a graphic in order to convey meaning. That is, in order to use these primitive languages to convey meaning in a graphic, we must place some sort of symbol in the graphic that has one or more of the graphical features. In the work described here, primitive graphical languages that use visible graphical symbols are called *notational* languages.

Figure 4 here

Earlier observations of the graphical notations used for problem solving and communicating information in various problem domains [6, 8], and of the graphical conventions used in many computer interfaces suggest two additional categories that relate meaning to actions performed on graphical symbols. A *primitive action*

Notational Primitive Graphical Languages

- Horizontal Position
- Vertical Position
- Cartesian Position
- Height
- Width
- Area
- Connectivity
- Density
- Shading
- Thickness
- Dashing
- Slope
- Shape
- Marks
- Tabular
- Part Whole

Deictic and Action Primitive Graphical Languages

- Overlay
- Pop Up
- Annotate
- Reference

Figure 4

language is used to describe manipulations performed on graphical symbols that can be associated with a meaning independent of the meaning of the graphical symbol itself. A *primitive deictic language* is used to describe references made to graphical symbols, groups of symbols, locations, or regions in a graphic. Football diagrams are an example of a domain that uses a rich set of action and deictic conventions. Casner [6] analyzes and implements a set of action and deictic conventions used in football diagramming. The complete set of action and deictic languages used in the present analysis is shown in the right side of Figure 4.

Associated with each of the primitive graphical languages is a set of perceptual and graphical operators (POPs and GOPs) that are admitted when the designer of a graphical representation elects to use one or more of the primitive languages in a graphic. For example, if we elect to use the "horizontal position" language we admit a family of perceptual operators (POPs) such as determining the horizontal position of a graphical object, comparing two or more horizontal positions, and finding the midpoint of an interval defined by two horizontal positions. Horizontal position also admits a set of graphical operators (GOPs) such as moving a graphical object from one position to another.

The follow is a sample of the set of perceptual and graphical operators (POPs and GOPs) admitted by the Horizontal Position primitive graphical language. It is unlikely that we can enumerate an exhaustive set of all possible perceptual operators since these are likely to vary with the experience and skill of individual users. The goal here is to arrive at a basis set that is sufficient to generate useful visual procedures and displays. When the designer of a graphical representation elects to attach meaning to the deliberate placement of a graphical object along a horizontal axis, the following perceptual and graphical operators become available for defining perceptual and graphical information-processing tasks:

Horizontal Position:

Perceptual Operations (POPs):
- determine-horz-position
- find-horz-positioned-object
- search-object-at-horz-pos
- confirm-object-at-horz-pos
- horz-coincidence?
- right-of?
- left-of?
- determine-horz-interval
- bigger-horz-interval?
- smaller-horz-interval?
- equal-horz-intervals?
- find-endpoint-of-horz-interval
- find-midpoint-of-horz-interval
- horz-weighted-interpolation
- horz-containment?
- determine-horz-projection
- determine-horz-distance

Graphical Operations (GOPs):
- horz-move
- scale
- overlay

Perceptual and graphical operators (POPs and GOPs) are formalized using the same logic-based representation scheme used to describe tasks. Figure 5 shows an excerpt from the set of formalized POPs and GOPs that are associated with the Horizontal Position language.

••••••••••

Figure 5 here

••••••••••

6. THE DESIGN ALGORITHM

BOZ is a design program that analyzes task descriptions and attempts to locate perceptual and graphical operators and accompanying visual data structures that can offer the user the two potential advantages of graphics that were described above. Section 6.1 describes how BOZ uses the catalog of POPs and GOPs to attempt to locate visual operators that can serve as substitutes for the logical operators given in

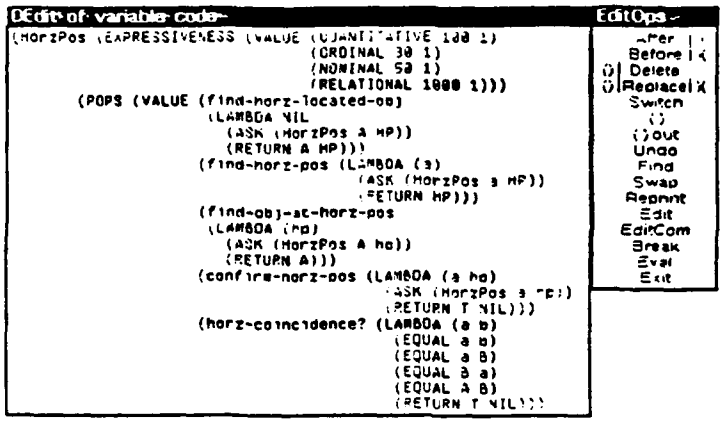

DEdit of variable code
(HorzPos (EXPRESSIVENESS (VALUE (ORDINAL 30 1)
(NOMINAL 50 1)
(RELATIONAL 1000 1)))
(POPS (VALUE (find-horz-located-obj
(LAMBDA NIL
(ASK (HorzPos A HP))
(RETURN A HP)))
(find-horz-pos (LAMBDA (a)
(ASK (HorzPos a HP))
(RETURN HP)))
(find-obj-at-horz-pos
(LAMBDA (hp)
(ASK (HorzPos A hp))
(RETURN A)))
(horz-coincidence? (LAMBDA (a b)
(EQUAL a b)
(EQUAL a B)
(EQUAL B a)
(EQUAL A B)
(RETURN T NIL)))
EditOps
Before
Delete
Replace
Switch
Undo
Find
Swap
Reprint
Edit
EditCom
Break
Eval
Exit

Figure 5

a task description. Section 6.3 shows how BOZ constructs visual data structures that support the prescribed visual operators and attempts to insure that all information necessary to execute a particular operator is available in the same spatial locality. Unlike previous approaches, BOZ *begins* by identifying a set of visual operators to be performed and *then* designs a graphical display for the task information.

6.1 Operator Substitutions: Matching POPs and GOPs to LOPs

BOZ considers each logical operator (LOP) in a task description and exhaustively searches the set of perceptual and graphical operators to locate those POPs and GOPs that compute the same result as the LOP. This is accomplished using theorem-proving techniques that attempt to show that POPs/GOPs and LOPs are isomorphic. Two operators are considered *isomorphic* when they produce the same output when given the same input. This property insures that whatever visual procedures is followed in place of a corresponding non-visual procedures, it is guaranteed that the user will obtain the same results if the visual procedure is performed correctly.

Perceptual and graphical operators can qualify as isomorphs for a logical operator in two ways. *Simple substitutions* are those in which a single perceptual or graphical operator can be shown to be equivalent to a logical operator. *Complex substitutions* are those in which two or more POPs and GOPs can be packaged together using one or more of a set of combination, composition, or repetition rules [3] for building complex operators from a set of primitives. The complex POP/GOP is then substituted for the logical operator in the task description.

6.1.2 Simple substitutions

A single perceptual or graphical operator can be shown to be isomorphic to a logical operator in the following way. Suppose a representation is needed that allows users to easily determine which individuals in a given set are relatives. The single

primitive operator necessary to complete this simple task can be encoded using the task description language as follows:

```
(LAMBDA determine-if-related (person1 person2)
   (ASK (Related person1 person2))
   (RETURN boolean))
```

BOZ then is charged with the job of searching the set of perceptual and graphical operators associated with each of the primitive graphical languages and locating those that are isomorphs of the logical operator in the task description. Suppose BOZ's search mechanism is currently considering the perceptual operators associated with the Connectivity primitive graphical language. The following perceptual operator can be found in Connectivity's collection of POPs:

```
(LAMBDA connected? (obj1 obj2)
   (ASK (Connected obj1 obj2))
   (RETURN boolean))
```

Note that `determine-if-related` and `connected?` are structurally the same. That is, since both operators compute a query operation of a 2-place predicate and return a boolean value, if we substitute the predicate `Connected` for `Related`, and exchange the two argument lists, the operators are the same. We can then conclude that we are entitled to "change the meaning" of the perceptual operator of determining if two graphical objects are connected to ask the question whether or not two persons are related. In other words, any graphic that represents the "related" relation by using connectivity between graphical objects as an encoding scheme will always allow the user to perform the perceptual operator and obtain correct answers. Figure 6 shows a representation that uses the connectivity convention.

****

Figure 6 here

****

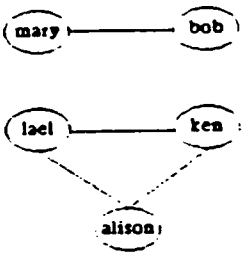


Figure 6

6.1.3 Complex substitutions

More often than not, BOZ exhausts the list of perceptual and graphical operators without finding a POP or GOP that is equivalent to a given LOP. For example, suppose the task described above is changed to include information about the particular kinds of relations between a set of individuals. An example task might be "find the brother of Mary," or "find the cousin of Alison." This task can be represented in using the task description language as follows:

```
(LAMBDA find-certain-relative-of-x? (x relation)
   (ASK (Related x Y relation))
   (RETURN Y))
```

If we consider each of the primitive graphical languages (including Connectivity shown above) in isolation we note that none of them formally qualifies as a substitution of `find-certain-relative-of-x`. From a graphics design perspective this is the same as saying that we cannot simply use graphical objects and links between them to support this task because there will be no way of understanding that the links are to represent different kinds of relations (e.g., mother, father, brother, etc) since there is only one kind of link.

When this occurs BOZ then considers complex perceptual and graphical operators that are constructed from the set of simple POPs and GOPs using one or more of a set rules for combination, composition, and repetition of operators [3] that are defined as follows.

If f and g are operators:

(LAMBDA $f(a_1, a_2, \ldots a_n)$
   (<meta-predicate> (<predicate> arg_1 arg_2 ... arg_a))
   (<meta-predicate> (<predicate> arg_1 arg_2 ... arg_b))
   .
   .

(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_c))
(RETURN Rf_1, Rf_2, ... Rf_d))

and

(LAMBDA g (b_1, b_2, ..., b_m)
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_f))
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_g))
.
.
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_h))
(RETURN Rg_1, Rg_2, ... Rg_i))

then the *combination* of *f* and *g* is the operator:

(LAMBDA f-combination-g (a_1, a_2, ... a_n, b_1, b_2, ..., b_m)
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_a))
.
.
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_c))
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_f))
.
.
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_h))
(RETURN Rf_1, Rf_2, ... Rf_s, Rg_1, Rg_2, ... Rg_t))

The *composition* of *f* and *g* is the operator:

(LAMBDA f-composition-g (arg_1, ... (<predicate> arg_1 arg_2 ... arg_f), ... arg_n)
(<meta-predicate> (<predicate> arg_1 (<predicate> arg_1 arg_2 ... arg_f) ... arg_a))
.
.
(<meta-predicate> (<predicate> arg_1 (<predicate> arg_1 arg_2 ... arg_f) ... arg_c))
(RETURN R_1, R_2, ... R_s))

for some argument, arg_i.

The *repetition* of a single operator *f* is:

(LAMBDA f (a_1, a_2, ... a_n)
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_a))

.
.
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_c))
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_a))
.
.
(<meta-predicate> (<predicate> arg_1 arg_2 ... arg_c))
.
.
.

(RETURN Rf_1, Rf_2, ... Rf_d))

Hence, a *combination* of two operators is a single operator that combines all of the parameters, clauses, and outputs of both individual operators. A *composition* of two operators is a single operator that uses another operator as one or more of its arguments. A *repetition* of a single operator is that operator performed two or more times repetitiously.

By adding the rules for defining more complex perceptual and graphical tasks, BOZ can now design representations to support more complex logical tasks such as the "relatives" example given above. Figure 7 shows how the `connected?` perceptual task can be composed with the `read-label` task associated with the Labels primitive language.

****

Figure 7 here

****

We can see that the composition of the two perceptual operators, one admitted by Connectivity and the other by Labels, yields a complex perceptual operator that is a renaming of the original task operator. The representation that illustrates this convention is shown in Figure 8.

****

Figure 8 here

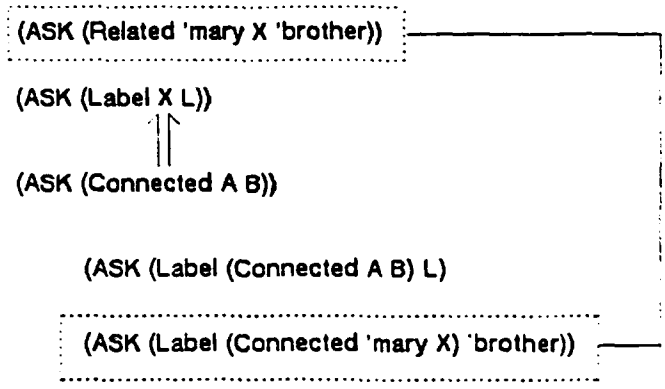

(ASK (Related 'mary X 'brother))
(ASK (Label X L))
(ASK (Connected A B))
(ASK (Label (Connected A B) L)
(ASK (Label (Connected 'mary X) 'brother))

Figure 7

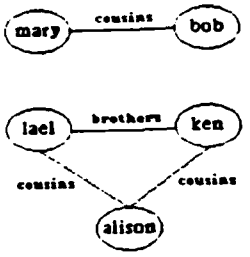


Figure 8

****

Note that if the task is further extended to include directional relations such as "is brother of," BOZ would use the composition of the POPs `determine-directed-relation` and `read-label`.

6.2 BOZ's Search Complexity

BOZ presently contains about 250 POPs and GOPs descriptions that must be exhaustively searched when considering each LOP in a task description. If a graphical representation is to support n primitive tasks (25-50 for complex representations), in the absence of any mechanism to help prune the search space we have the following worst case search complexity:

$$T_{search} = 200n \text{ (simple substitutions)} + 200^2n \text{ (complex substitutions)}$$

$$= 40{,}200n \text{ (total items searched)}$$

If we later decide to allow 3-place substitutions (e.g., compositions of three POPs or GOPs) we add an additional 8,000,000 x n items to the search space. BOZ circumvents this problem by taking advantage of a formal property of the rules of composition, combination, and repetition. Given a LOP that fails to match any of the simple POPs and GOPs, we can perform a *composition factorization* of the LOP into two component LOPs. For example, the LOP:

```
(LAMBDA find-relative ('joe 'brother)
  (ASK (Related 'joe X 'brother))
  (RETURN X))
```

can be factored into two component LOPs:

```
(LAMBDA x (a)
  (ASK (PredX a B))
```

```
    (RETURN B))

(LAMBDA y (b)
    (ASK (PredY A b))
    (RETURN A))
```

The two component LOPs can now be used to independently search the list of simple POPs and GOPs two times, one time for each component LOP. Any matches that are found for the two LOPs can be shown to have the property that when composed together they result in an isomorph of the original LOP.

Combination factorization and *repetition factorization* are defined analogously and can be used to locate complex matches formed by combination and repetition. Adding the factorization strategies reduces BOZ's worst case search complexity to:

$$t_{search} = 200n + 6 \times 200n = 1400n$$

BOZ is implemented in Interlisp-D on a Xerox 1186 and takes roughly 12 minutes to explore the entire design space of visual operators given a task description containing 50 logical operators.

6.3 Constructing Visual Data Structures

Once BOZ has located a set of perceptual and graphical operators that can potentially serve as substitutes for the logical operators that comprise a task, BOZ must design a visual data structure that supports those visual operators. This is accomplished by examining the information that is required to perform each logical operator and building a graphical display that insures that: (1) information is presented in a form that matches the visual operator description; and, (2) all information needed to perform a visual operator is presented in the same graphical object so it can be accessed without moving the eye over the display. The following describes how BOZ accomplishes this.

Recall that every task description is defined over a finite set of *domain sets*. Domain sets are collections of user-defined variables that can assume exactly one of a set of pre-specified values. When taken together, all of the domain sets used by a task description form a *feature space*. A feature space is formally defined as the cross product of all domain sets spanned by a task description. Figure 9 shows an example of a feature space defined over a collection of domain sets that pertain to making a reservation on a commercial airline.

****

Figure 9 here

****

Each individual operator in a task description defines a relation $(v_i, v_j, \ldots v_k)$, for some i, j, and k less than or equal to the total number of domain sets drawn from a feature space, called a *vector*. Vectors are unordered collections of values such that each value is selected from exactly one domain set. The *length* of a vector is defined as the number of values contained in the vector, or the number of domain sets that the vector spans. Vectors can have lengths that vary from 1 to n, where n is the total number of domain sets.

Vectors are used by BOZ to define visual data structures in the following way. Let T be a set of operators occuring in a task description, and t_i an arbitrary member of T. Let $V(t_i)$ be the vectors implied by an operator t_i. The function $EQV(V(t_i))$ computes the equivalence relation that classifies each vector into exactly one of three equivalence classes. Vectors V1 and V2 are said to be *parallel* when they each contain a value from more than one common domain set. Vectors V1 and V2 are said to be *orthogonal* when each vector contains a value from exactly one common domain

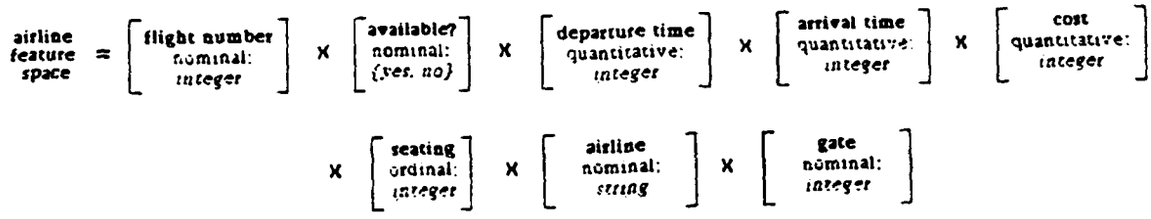

airline feature space
flight number
nominal:
integer
available?
nominal:
{yes, no}
departure time
quantitative:
integer
arrival time
quantitative:
integer
cost
quantitative:
integer
seating
ordinal:
integer
airline
nominal:
string
gate
nominal:
integer

Figure 9

set. Vectors V1 and V2 are *disjoint* when they contain no values from common domain sets. The equivalence relation is the set:

$$EQV(V(t_i)) = \{\{\text{all parallel vectors}\}\ \{\text{all orthogonal vectors}\}\ \{\text{disjoint vectors}\}\}$$

For example, the following tasks define five vectors.

```
(LAMBDA determine-departure-time (flight)
   (ASK (Departure flight DEPARTS))
   (RETURN DEPARTS))

(LAMBDA determine-cost (flight)
   (ASK (Cost flight COST))
   (RETURN COST))

(LAMBDA find-cheapest-leaving-at-T (time)
   (ASK (Departure FLIGHT time))
   (ASK (Cost FLIGHT COST))
   (NOT (Departure FLIGHT2 time)
            (Cost FLIGHT2 COST2)
            (LESS COST2 COST))
   (RETURN FLIGHT COST))

(LAMBDA window-seat-available? (flight)
   (ASK (Seat flight SEAT))
   (ASK (Available SEAT))
   (ASK (Position SEAT 'window))
   (RETURN SEAT))

(LAMBDA determine-gate (airline)
   (ASK (Gate airline GATE))
   (RETURN GATE))
```

The vectors defined by the tasks are the following:

1. determine-departure-time = ({flight-number} X {departure-time})
2. determine-cost = ({flight-number} X {cost})
3. find-cheapest-leaving-at-T = ({flight-number} X {departure-time} X {cost})
4. window-seat-available? = ({flight-number} X {seats} X
   {window, aisle, middle} X {yes, no})

5. determine-gate = ({airline} X {gate})

Vectors 1, 2 and 3 are parellel, vectors 3 and 4 orthogonal, and vector 5 disjoint from all others.

The algorithm that BOZ uses to construct visual data structures given a set of vectors has two basic steps. The first part of the algorithm determines the possible ways in which information from each domain set can be encoded as individual graphical objects on the screen. After analyzing a task description (as described in Section 6), each domain set has associated with it a set of possible primitive graphical languages that can be used to encode that domain set. The set of possible primitive graphical languages are precisely those that are associated with that POPs and GOPs that have been selected in the previous design step (substituting visual for logical operators). Each of these primitive graphical languages has a default presentation object associated with it, either: point, line, or shape. For instance, the default for the Height language is a rectangle. Hence, the possible ways of presenting information manipulated by a task is defined by the set of graphical objects that are associated with those primitive graphical languages that are associated with those POPs and GOPs that have been shown to be useful for performing the task.

The second part of the algorithm uses the vectors to determine how the individual graphical objects created in part 1 can be combined into composite visual data structures such that all information necessary to draw a particular inference is likely to be grouped together in the same spatial locality. This is accomplished in the following way. For each set of parallel vectors, BOZ forms the union of all domain sets that appear in the set of parallel vectors. A *simple* visual data structure is created by grouping together all of the graphical objects that visually encode the domain sets in the union. For each pair of orthogonal vectors, we find the common domain set where the vectors intersect. This domain set is called the *pivot point*. A

multi-dimensional visual data structure is defined to be a graphical object that can be transformed from reflecting a presentation of the domain sets defined by one vector, to presenting the domain sets defined by another vector. Transformation of the multi-dimensional visual data structure is accomplished by a graphical operator (GOP) of the designer's choosing, usually mouse-selection. To further simplify the data structures, when two domain sets occuring in a single vector are of the same type, the graphical objects used to represent them can be composed into a single graphical object. This technique known as *mark composition* and is due to Mackinlay [21]. Similarly, two domain sets of the same type can be represented on the same axis. This is accomplished using *axis composition* [21].

To illustrate how visual data structures are defined using vectors consider the tasks defined for the airline reservation domain. Since vectors 1, 2, and 3 are parallel, a simple visual data structure is created that presents all of the domain sets occuring in each vector the same data structure. Since vector 4 is orthogonal to vectors 1, 2, and 3, a multi-dimensional visual data structure is created that can be transformed from presenting the domain sets contained in vector 4, to the domain sets contained in vectors 1, 2, and 3. The "pivot point" is the domain set {flight-number}. Vector 5 is disjoint from all other vectors and therefore defines its own separate data structure. The abstract specifications for the visual data structures (one multi-dimensional and one simple) defined by the vectors are shown in Figure 10, along with the set of possible primitive languages that can be used to encode each domain set. Figure 10 also shows one possible presentation of the multi-dimensional visual data structure.

.....

Figure 10 here

.....

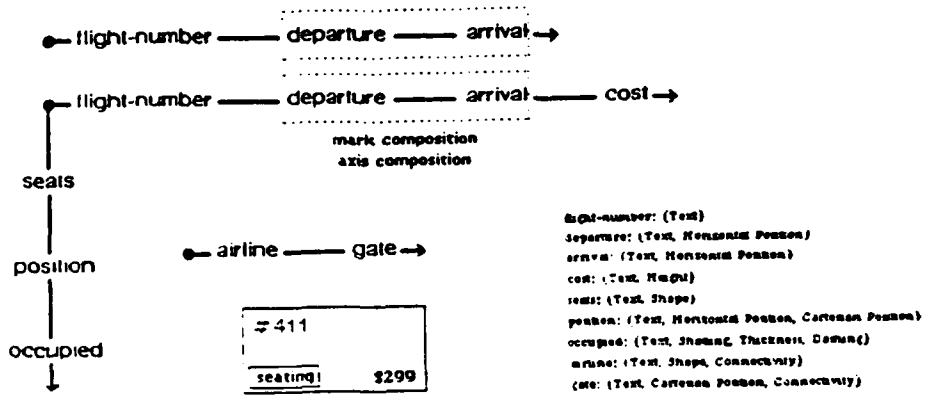

flight-number
departure
arrival
flight-number
departure
arrival
cost
mark composition
axis composition
seats
position
occupied
airline
gate
#411
seating!
$299
flight-number: (Text)
departure: (Text, Horizontal Position)
arrival: (Text, Horizontal Position)
cost: (Text, Height)
seats: (Text, Shape)
position: (Text, Horizontal Position, Cartesian Position)
occupied: (Text, Shading, Thickness, Dashing)
airline: (Text, Shape, Connectivity)
gate: (Text, Cartesian Position, Connectivity)

Figure 10

7. AN EXAMPLE: DESIGNING A GRAPHICAL AIRLINE RESERVATION SYSTEM

The following example illustrates how BOZ can be used to design graphical displays to support user tasks in a real-world domain. The representation designed for the tasks illustrates the two important ways in which graphical representations can help users: (1) supporting the substitution of quick and easy visual operators in place of more effortful logical operators; and (2) reducing the amount of time spent searching for information [20].

7.1 A Simple Airline Reservation Task

The following describes a task in which a user must use airline schedule information to locate a flight or series of flights that travel from Pittsburgh to Mexico City. To make the task both challenging and realistic we impose the following restrictions on the flight that the user may choose:

1. For connecting flights that pass through an intermediate city, the user is free to choose any intermediate city as long as the layover in that city is not more than four hours.

2. The user may choose only those flights that can be shown to have available seats.

3. The user is limited to a $500 budget for the one-way trip. That is, the total price for any series of connecting flights cannot exceed $500.

The task description for the airline task is shown in Figure 11.

Figure 11 here

The task requires that the user perform sophisticated searches and coordinate information about the origin, destination, departure and arrival times, availability, and cost of many flights. The domain sets that describe this information are specified at the top of the task description in Figure 11. The section entitled OPERATORS enumerates the logical operators that are necessary to perform the task correctly.

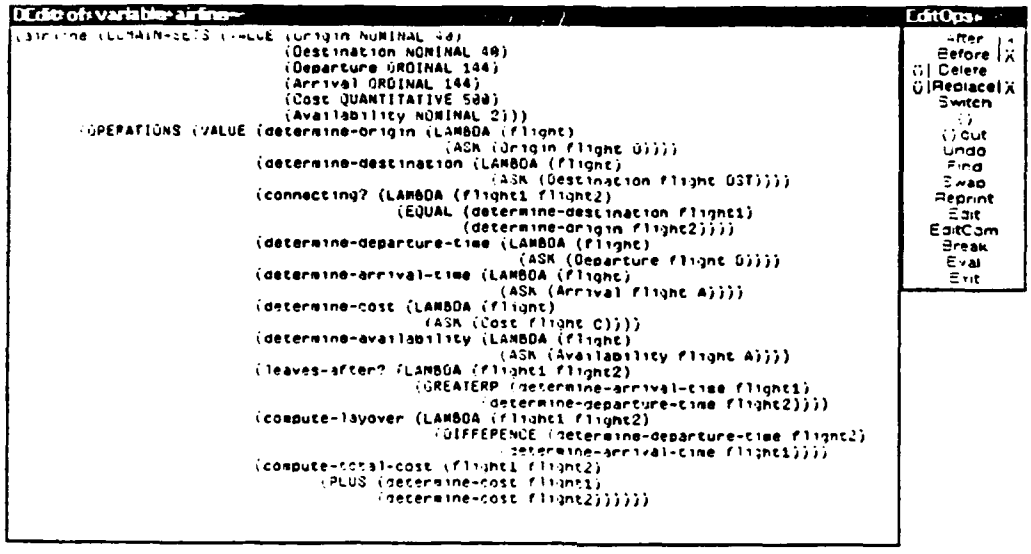

EditOps
(airline (DOMAIN-SETS (VALUE (origin NOMINAL 50)
(Destination NOMINAL 40)
(Departure ORDINAL 144)
(Arrival ORDINAL 144)
(Cost QUANTITATIVE 500)
(Availability NOMINAL 2)))
(OPERATIONS (VALUE (determine-origin (LAMBDA (flight)
(ASK (origin flight O))))
(determine-destination (LAMBDA (flight)
(ASK (Destination flight DST))))
(connecting? (LAMBDA (flight1 flight2)
(EQUAL (determine-destination flight1)
(determine-origin flight2))))
(determine-departure-time (LAMBDA (flight)
(ASK (Departure flight D))))
(determine-arrival-time (LAMBDA (flight)
(ASK (Arrival flight A))))
(determine-cost (LAMBDA (flight)
(ASK (Cost flight C))))
(determine-availability (LAMBDA (flight)
(ASK (Availability flight A))))
(leaves-after? (LAMBDA (flight1 flight2)
(GREATERP (determine-arrival-time flight1)
(determine-departure-time flight2))))
(compute-layover (LAMBDA (flight1 flight2)
(DIFFERENCE (determine-departure-time flight2)
(determine-arrival-time flight1))))
(compute-total-cost (flight1 flight2)
(PLUS (determine-cost flight1)
(determine-cost flight2))))))
After
Before
Delete
Replace
Switch
Cut
Undo
Find
Swap
Reprint
Edit
EditCom
Break
Eval
Exit

Figure 11

When the task description is submitted, BOZ produces a specification summarizing the possible ways of substituting visual operators and displaying information in a form that supports visual operator performance. The specification for the airline reservation task is shown in Figure 12.

****
Figure 12 here
****

In the VISUALOPS section, each of the logical operators in the original task description now has an associated set of visual operators that have been shown to be formally equivalent by BOZ's substitution algorithm. The VECTORS section contains a single vector that specifies that regardless of what visual operators are chosen, all information pertaining to a single flight is to be graphically presented using the same graphical object. The section entitled GRAPHICAL-LANGUAGES summarizes the possible ways of displaying each information type as defined by the set of visual operator substitutions.

The specification in Figure 12 formally describes 756 possible displays for the airline reservation task. The next step is to identify which alternatives are likely to be most effective. Section 10 discusses two important factors that affect the success of a graphical design:

- the extent to which the representation exploits users' real-world knowledge and skills.

- the extent to which representations tap users' prior skills and experience with graphics.

FIGURE 12

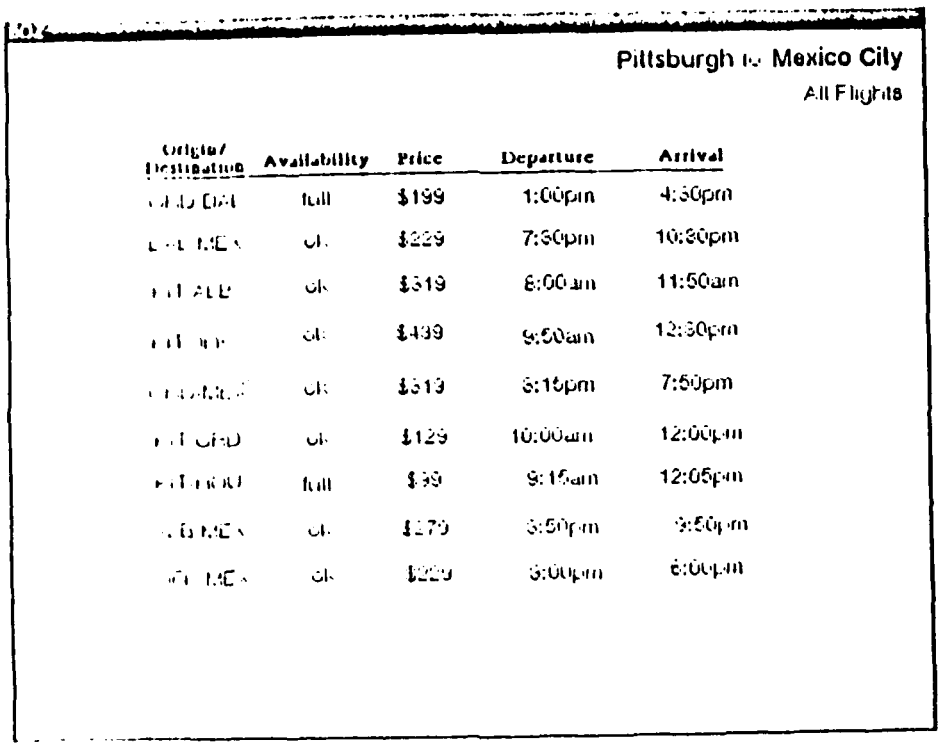
Section 10 also discusses why choosing between alternative designs is necessarily an empirical issue, and why simple strategies that attempt to generalize experimental findings across tasks and users [21] are likely to obtain poor results in practice.

To support efficient exploration of alternative designs, BOZ contains a rapid prototyping tool, implemented using the VisualActive™ interface construction kit [11], that allows designers to quickly mock up candidate designs. The prototyping tool allows designers to create the corresponding default graphical objects that are associated with the perceptual and graphical operators as described in Section 6.3. Graphical objects can have functions attached to them that are called whenever specified mouse selections are performed. The tool also contains a set of prepackaged window, menu, and dialog box facilities that can be called by any function attached to a graphical object. We use the prototyping tool here to help discover which designs summarized in BOZ's output are likely to be the most useful.

The first design produced by BOZ given any task description is one in which relational information is displayed in the rows and columns of a table. Beginning with a tabular display requires that designer have explicit criteria for making decisions to remove information from the table and encoding it graphically. This requirement is consistent with the experimental findings that graphical displays are not always better than tables and that tables should be treated as the default display. Figure 13 shows a tabular display supporting the airline reservation task.

Figure 13 here

Next, we see that we have the opportunity to exploit a real-world notion when encoding departure and arrival times. Times could be represented using Slope such as done in everyday clocks. Choosing this encoding would allow users to perform



Pittsburgh to Mexico City

All Flights

Origin/ Destination	Availability	Price	Departure	Arrival
[illegible]	full	$199	1:00pm	4:30pm
[illegible]	ok	$229	7:30pm	10:30pm
[illegible]	ok	$319	8:00am	11:50am
[illegible]	ok	$439	9:50am	12:30pm
[illegible]	ok	$519	6:15pm	7:50pm
[illegible]	ok	$129	10:00am	12:00pm
[illegible]	full	$99	9:15am	12:05pm
[illegible]	ok	$279	6:50pm	9:50pm
[illegible]	ok	$229	6:00pm	6:00pm

Window Image

departure and arrival time comparisons in a familiar and practiced way. Figure 14 shows an implementation of the clocks idea. The CLOCKS display replaces the numerical presentation of times in the table with clocks supporting slope judgements and comparisons.

****
Figure 14 here
****

The problem with the clocks convention is that only one departure or arrival time can be represented in a single clock. That is, the clock convention disallows the possibility of grouping time information about many flights in the same spatial locality. The CLOCKS display also fails to gather information about flights into closer spatial proximity where users can perform quick comparisons. Using CLOCKS, users must still search the rows and columns of the table.

An alternative is to exploit users' previous experience with horizontal time scales. Figure 15 shows a display in which we replace the tabular display of departure and arrival times with graphical objects positioned along a horizontal axis. Arrival and departure times can share the same horizontal axis as well as the same graphical object (axis and mark composition).

****
Figure 15 here
****

The time scale convention avoids the problem of clutter when many flights are present.

We can replace the words "ok" and "full" in the flight boxes and allow users to substitute shading judgements when making availability determinations as in the display shown in Figure 16.

Window Image

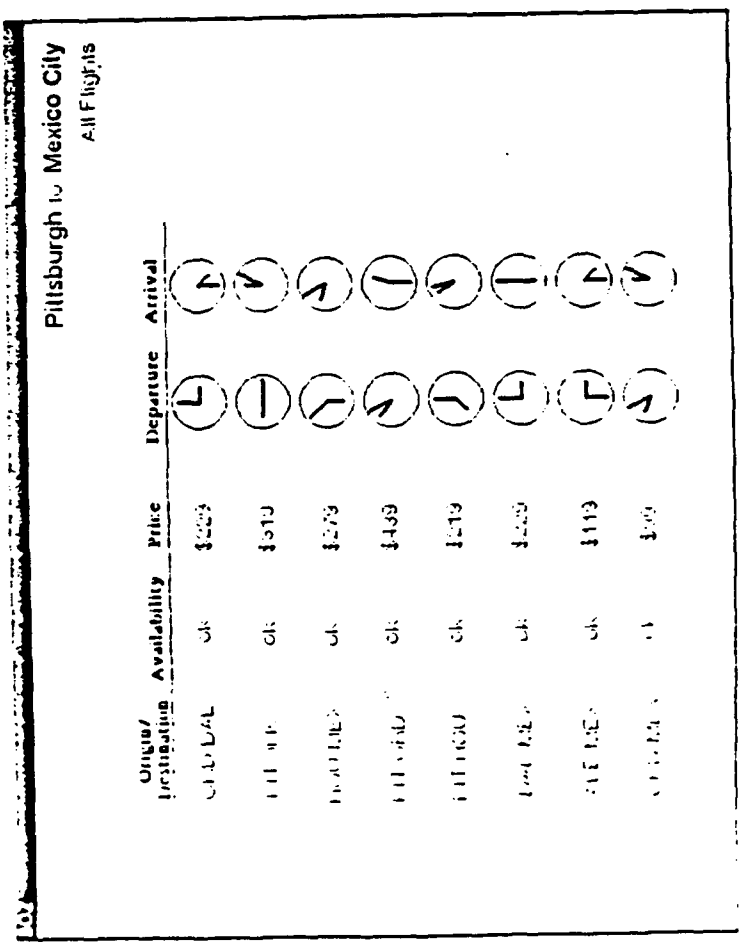
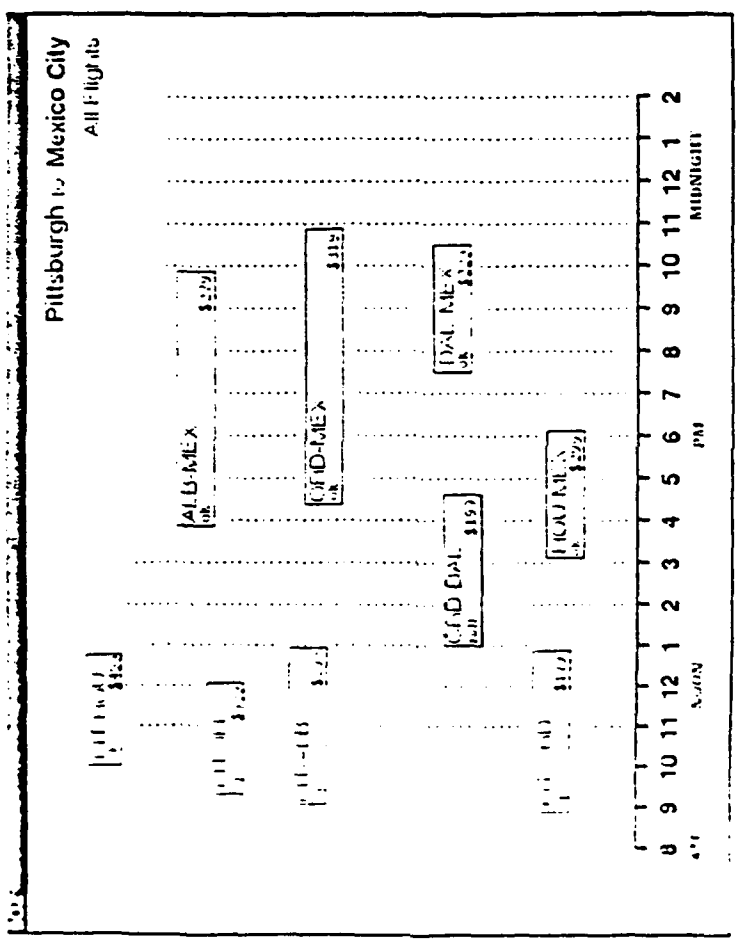


Figure 14

Window Image



****
Figure 16 here
****

It is not obvious that the time to interpret a shade offers any advantages over simply reading the word "available" but this is an empirical question that is addressed in Section 8.

BOZ's output indicates that we can substitute the task of reading and adding numerically expressed cost information by allowing users to perform height judgement as shown in the display in Figure 17.

****
Figure 17 here
****

Again it is not clear that allowing users to judge two or more combined heights yields an advantage over simply adding the numbers, or that it helps users to more easily rule out "tall" (expensive) flights.

The potential efficiencies of the various graphical displays for the airline reservation problem are characterized by the following visual operator substitutions and visual search heuristics.

Operator Substitutions:

DISTANCE JUDGEMENT: The displays in Figure 15, 16, and 17 allow users to substitute a distance judgement in place of subtracting the numerically expressed departure and arrival times.

SHADE JUDGEMENT: Figures 16 and 17 substitute shade judgement for reading the words "ok" and full."

Window Image

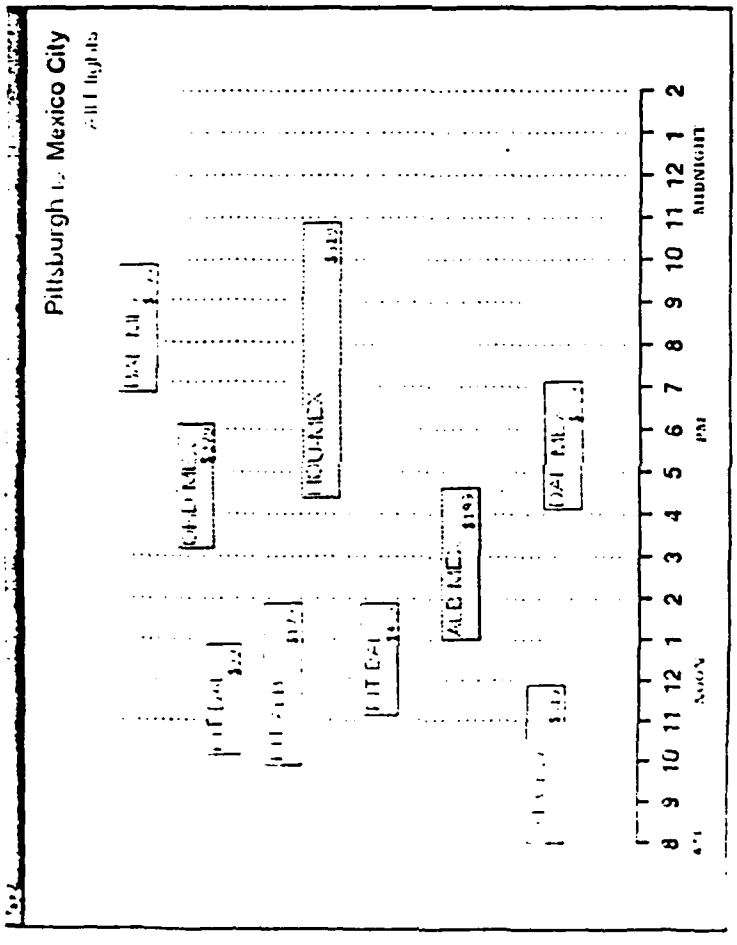
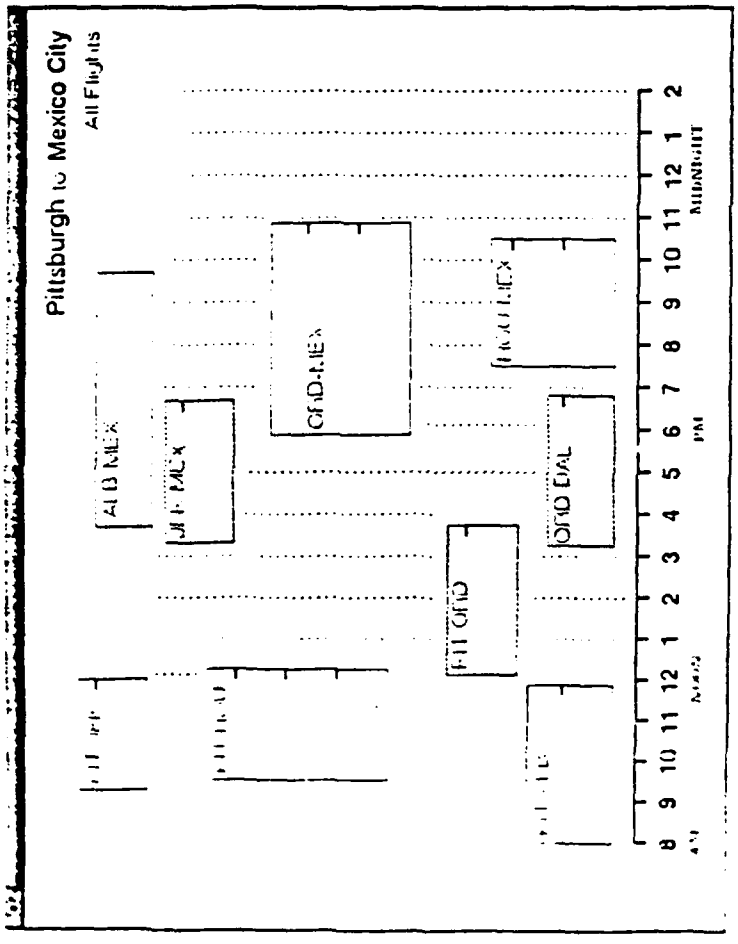


Figure 16

Window Image



STACK HEIGHTS: Figure 17 substitutes judging the combined heights of two flight boxes for adding two numerically expressed costs.

Search Heuristics:

RIGHT OF: Displays in Figures 15, 16, and 17 allow users to limit their search for connecting flights to only those flights that appear to the right of the originating flight.

NO SHADED: Displays in Figures 16 and 17 allow users to immediately exclude from their search any flight square that is shaded since shading indicates that the flight has no available seats.

CHEAPEST FIRST: The display in Figure 17 allows users to immediately rule out "tall" flights from their search since these are likely to exceed the $500 limit.

Whether or not users do or can follow these procedures, and the extent of the efficiency advantages they offer is an empirical question that is addressed in Section 8. The study reported in Section 8 shows that there can be no algorithm that accurately predicts the usability of a graphical representation or even in general decide which is the most promising of a set of alternative designs.

7.2 Extending the Set of Airline Reservation Tasks

To show how BOZ scales up to more complex task I extended the task description to include several other tasks that airline customers frequently perform. Since the entire task description is lengthy (3 pages of code), the tasks are summarized as follows:

Choose that flight of several possibilities that has the best available seat: Some travelers prefer aisle seats since they afford the best mobility. Some travelers prefer window seats for the view. Smart Brazilian travelers pick seats on the right side of the plane when traveling south and the right side when heading north.

Schedule around departure or arrival constraints: Many travelers have time constraints on when they can leave or arrive at their destination.

Schedule a specific layover: Many business travelers keep more than one appointment on the same trip. In this case the customer is concerned with finding a flight that passes through a particular city and lays over during a particular time period.

Find the cheapest flight scheduled in some reasonable time interval: Graduate students are usually most concerned with minimizing the cost of the ticket rather than with arriving at their destination at a specific hour or day.

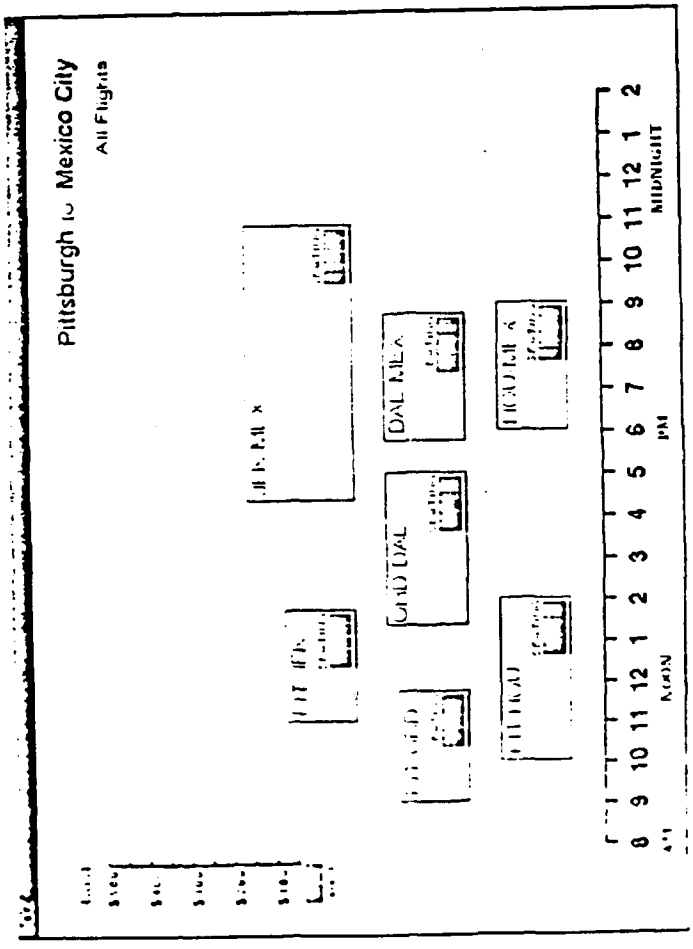
Find a flight that minimizes the number of take-offs and landings: Some customers try to minimize the number of take-offs and landings to minimize the effects of jet-lag, nausea, or ear trouble.

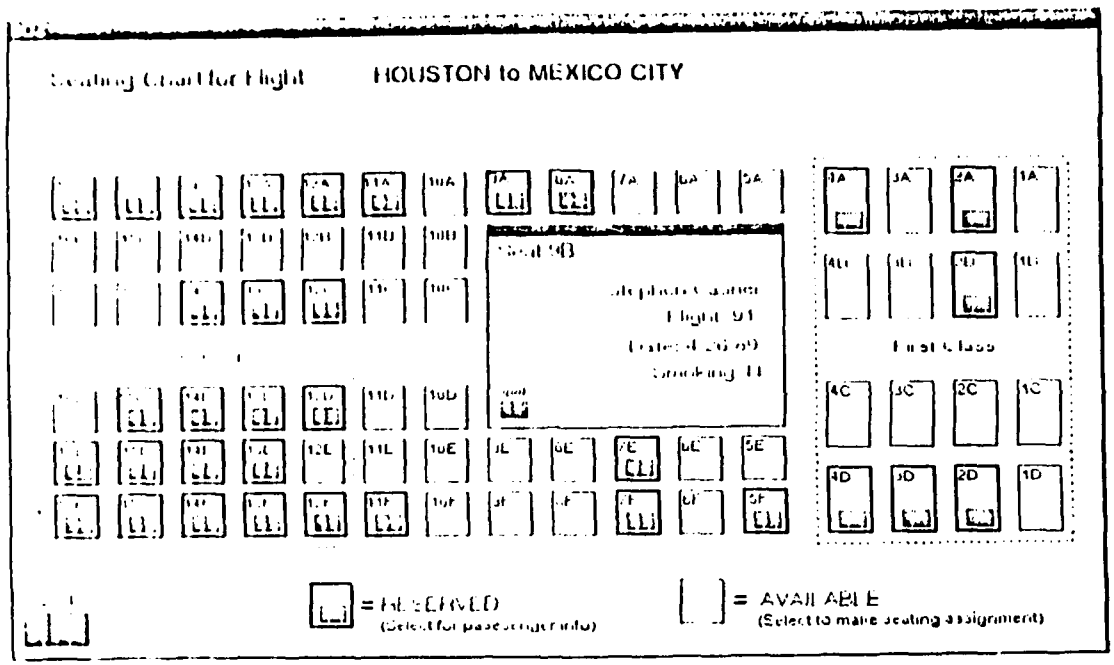
An extended task description was submitted to BOZ that included the tasks above. One candidate design was produced using the prototyping tool and is shown in Figures 18(a) and 18(b).

****
Figures 18(a) and 18(b) here
****

Time constraints are specified by creating a flight square and positioning it along the time scale at the appropriate departure or arrival time. Specific layovers are specified by creating two flight squares and positioning the arrival time of the first flight and the departure time of the second flight at the appropriate points along the scale. Cost constraints are indicated by typing a cost into the lower right region of the flight square. Specifying a particular number of takeoffs and landings is accomplished by creating the appropriate number of flights squares. For example, to search for a flight with two takeoffs and landings, two flight squares are created. Viewing the available seating for a flight is accomplished by mouse-selecting the icon that appears in the lower left corner of a flight square. This causes the seating chart for that flight, shown in Figure 18(b), to be presented. Seats are occupied if they have

Window Image





Window Image

a thick border and available otherwise. To view the status of any seat, or to reserve a particular seat, left mousing on the seat causes a dialog box to appear where the user can view or type the relevant information.

8. USERS' PERFORMANCE WITH THE AIRLINE RESERVATION SYSTEM

Casner and Larkin [8] describes an experimental study in which real users worked with the BOZ-designed airline displays. The purpose of the experiment was to determine the extent to which the hypothesized efficiency advantages of the airline displays were actually reflected in users' performance. Response times were collected from eight participants who performed the simple airline reservation task (described in Section 7.1) using four different versions of the displays a total of ten times each (forty trials total). The four graphic versions used in the experiment were those shown in Figures 13, 15, 16, and 17, herein called Graphics 1, 2, 3, and 4, respectively. Each graphic in turn includes an additional way of substituting a visual operator for a logical operator. The purpose of the experiment was to test the hypothesis that each additional visual operator substitution would reduce the time required to complete the task through either savings gained in visual operator performance, or through reductions in the total number of items in the airline schedule that the user must consider.

Users completed the task after receiving an explanation of the conventions used in each graphic and one practice trial. The order in which the graphics were presented to the users was varied systematically to evenly distribute effects due to learning and practice.

••••••••••••

Figure 19 here

••••••••••••

Airline Graphic Used

	1	2	3	4
Mean Response Time	19.3*	10.1*	7.2*	7.4
Mean Standard Deviation	8.4*	4.7*	2.7*	2.4
Total Number of Errors	6	5	6	8

* Significant at .05 level

Figure 19

The results shown in Figure 19 (Row 1) indicate significant differences in response times between Graphics 1 and 2, and between Graphics 2 and 3, but not between Graphics 3 and 4. The data suggest that graphical encodings used in Graphic 2 (and also in Graphics 3 and 4) reduce the amount of time required to locate two connecting flights, and/or to determine whether or not two flights obey the layover constraint. Allowing users to perform the perceptual operator of determining the shade of a flight box (Graphic 2) surprisingly resulted in a significant savings. The perceptual task of determining whether or not two flights obey the cost constraint by judging the heights of the flight squares did not result in any reliable savings over the task of adding the two numbers, or in narrowing down the search space of flights to consider.

An analysis of the standard deviations in response times shown in Row 2 of Figure 19 suggests that users exhibited significantly more stable performance between Graphics 1, 2, and 3 in that order. Row 3 of Figure 19 shows no observable differences among error rates for the four graphics.

Our next step was to understand how users obtained the efficiency savings we observed. We ran a regression analysis on the number of times each operator (visual or logical) must be performed using each graphic to the observed user response times. We obtained good-fitting models for each of the search heuristics excepting CHEAPEST FIRST, suggesting that users took advantage of *all* of the applicable search heuristics except CHEAPEST FIRST. The model also yielded estimates on performance time for several of the individual visual and logical operators.

- The time required to fix the eye on each item in a display was uniformly about 330 milliseconds for all four displays.

- Visually estimating layovers (`determine-horz-distance`) using Graphics 2, 3, and 4 proceeded about 2 seconds faster than subtracting the numerically expressed times.

- Judging the combined heights of two flight boxes (`stack-heights`) in Graphic 4 was negligibly 100 to 300 milliseconds slower than adding the numerically expressed costs.

The saving gained through substitution of visual for logical operators and use of search heuristics match well with the global reductions observed in overall response times. Overall the results agree with users' comments after using all four graphics: that Graphic 3 was the most effective. The interested reader can find details of the experimental design and methodology in Casner and Larkin [8].

9. GENERAL DISCUSSION

The research described above adopts a task-based approach to the design of graphical representations in which graphics are viewed as perceptually and graphically manipulated data structures that help streamline task performance much in the way that abstract data structures help expedite abstract computational processes. The important distinction made in a task-based design methodology is that the effective use of visual data structures and procedures, as with abstract data structures, depends on choosing the right structure for the right task. That is, it is inappropriate to say that a particular graphical display is the best choice or that it is useful in general. One can only compare graphical displays with respect to a particular task. Consequently, the design methodology proposed here *first* designs visual tasks based on an analysis of the task requirements, and *then* proceeds to select a visual format for the task information with the visual task in mind.

The task-based approach to graphical displays was made concrete by creating a formalism that allows us to specify and compare alternative visual and non-visual procedures, and to show that the alternative procedures always produce the same results. Two specific types of cognitive efficiency advantages of graphical displays were discussed: (1) allowing users to substitute quick and easy perceptual judgements in place of more demanding logical reasoning steps; and, (2) reducing search for needed information by grouping related information and supporting visual search heuristics. The examples and experimental results suggest that the analysis can be successfully applied to designing effective graphical displays that provide the two types of efficiency advantages. Furthermore, the view of graphical displays as visual data structures and procedures that provide efficiency advantages seems to explain the difference between graphical displays that succeed in practice and those that do not.

A necessary weak link in the design process is the inability to predict in advance the relative usability of each of a set of alternative designs. Given that we can already show that each of BOZ's designs can potentially help streamline users' tasks, the remaining factors that most influence usability seem to be the extent to which a display exploits the user's knowledge about "everyday things" in the real world [23]; and the extent to which displays allow users to make use of existing knowledge and skills for using graphics [17]. Real-world knowledge often allows users to quickly understand a graphical convention. For example, the diagram about politicians shown in Figure 20 exploits the user's knowledge that "left means liberal" and "right means conservative."

Figure 20 here

The psychological literature shows that no other design consideration is likely to outweigh the results achieved by exploiting real-world knowledge [23].

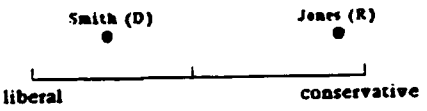


Figure 20

Furthermore, neglecting to use real-world conventions (e.g., putting liberals on the right, conservatives on the left) can lead to increased learning time and cognitive work, and decreased response accuracy [23].

The second most important factor in choosing between alternative designs is the extent to which each design can exploit previous experience, practice, and skill with graphics that the user may have. There are many existing graphical conventions that the designer can use and expect reasonable results. For example, when choosing among designs that encode time as a dimension, those designs that use a horizontal time scale are most likely to tap users' previous experience. The large number of pie charts in the everyday literature suggest that users may have practised skill for substituting size and slope judgements for more complex proportional reasoning tasks.

The two factors affecting the usabililty of a graphic suggest that determining which of a set of candidate designs is largely an empirical question not decidable by any sort of quick-and-easy algorithm. Mackinlay's APT [21] attempted to predict the accuracy of alternative graphic designs using an experimental observation that presentation formats can be grouped into three basic categories that are ordered by the degree of accuracy to which users can extract information from them [10]. However, more recent studies show that users' performance on information extraction tasks are poor indicators of performance on other types of tasks [16]. Other research shows that users' performance is extremely sensitive to small changes in the task, the complexity of the task [26], problem domain, and even the social conditions in which the graphic is used [1]! The large number of psychological factors influencing performance and the variance between users, tasks, and situations suggest that a realistic strategy for deciding between alternative graphic designs involves prototyping designs and performing small amounts of well-planned user testing [14].

BOZ may also be useful for explaining why existing graphical displays are successful, and to help discover clever design properties enjoyed by existing displays that can be later incorporated into BOZ's design algorithm. Existing displays can be analyzed by describing a set of activities for which the displays are claimed to be useful and showing that the design of the display is such that it allows the users to perform computationally interesting sets of visual procedures. Consider the graphical representation used in the calculus for *vectors in the plane*, shown in Figure 21.

****
Figure 21 here
****

Vectors use lines in the plane to represent forces acting on a body. The magnitude of a force (a quantitative value) is represented by the length of a vector line. The direction of a force (also a quantitative value) is represented by the slope of a vector line. The surprising feature of vector representations is that they do not use the spatial position of a vector to encode information. The spatial position primitive graphical languages are the most informationally rich and salient of the primitive language yet they are not used in the vector representation. The decision to keep the spatial position primitive languages "in reserve" (at some cost in understandability to the novice student) pays off when when the task of adding together vectors is introduced. Leaving the spatial position languages free allows us to move vectors around in the plane and be sure that the vectors still represent the same entity. Having this freedom more specifically allows us to arrange vectors such that the beginning of one vector coincides with the end of another vector as shown in Figure 21. We can make of use annotations (a GOP) and also draw a line that connects the beginning of the first vector in a "train" and the end of the last vector. Of course, it can easily be shown that the vector added as an annotation is exactly that vector that represents the sum of all of the vectors in the train. Summing together vectors

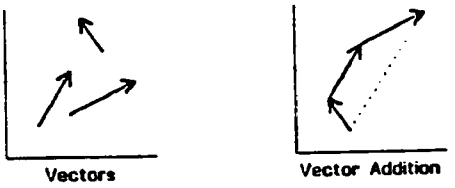


Figure 21

without the benefit of the graphical representation of course requires more sophisticated mathematical knowledge and procedures.

The analysis of the design used for vectors suggests the following general design principle: that sometimes a sacrifice in one aspect of a design can lead to magnificent gains in another aspect of the same design. The analysis of the vector representation has provided an interesting design property that falls outside of BOZ's capabilities, and that suggests a promising new idea to investigate that may be generalizable across many types of graphical displays.

ACKNOWLEDGEMENTS

This work is supported by the Office of Naval Research, University Research Initiative, Contract Number N00014-86-K-0678, and in part by Virtual Machine Corporation, Pittsburgh, PA . I thank Stellan Ohlsson, Jill Larkin, Jeffrey Bonar, and Alan Lesgold for helpful comments and criticisms.

REFERENCES

1. Asch, S.E., Studies of independence and submission to group pressure: A minority of one against a ananimous majority. *Psychological Monographs*, 1956, 70.

2. Bertin, Jacques, *Semiology of graphics*, W. Berg, transl., Madison, WI: University of Wisconsin Press, 1983.

3. Brainerd, W.S., and L.H. Landweber, *Theory of Computation*, New York: John Wiley and Sons, 1974.

4. Bonar, J. and R. Cunningham. Bridge: An inteiligent tutor for thinking about programming, in *Artificial Intelligence and Human Learning*, John Self, ed., London: Chapman and Hall Publishing, 1988.

5. Card, S.K., Moran, T.P., and A. Newell, *The Psychology of Human-Computer Interaction*, Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.

6. Casner, Stephen, Building customized diagramming languages, in *Visual Languages and Visual Programming*, S.-K. Chang, ed., New York: Plenum Press, in press.

7. Casner, Stephen, and Jill H. Larkin, Cognitive efficiency considerations for good graphic design, submitted (also available as Learning Research and Development Center Technical Report, 1989).

8. Casner, Stephen and Jeffrey Bonar, Using the expert's diagrams as a specification of expertise, in *Proceeedings of the IEEE 1988 Workshop on Visual Languages*, Pittsburgh, PA, October 10-12, 1988, 150-157.

9. Cleveland, W.S., *Elements of Graphing Data*, Monterey, CA: Wadsworth Advanced Books and Software, 1895.

10. Cleveland, W. S. and R. McGill, Graphical perception: Theory, experimentation, and application to the development of graphical methods, *Journal of the American Statistics Association* 79 (387), Sept., 1984, 531-554.

11. Cunningham, R.E., Corbett, J.D., and J.G. Bonar, Chips: A tool for developing software interfaces interactively, Learning Research and Development Technical Report No. LSP-4, October 1987.

12. Genesereth, M., and N. Nilsson, *Logical Foundations of Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1987.

13. Goldenberg, E. Paul, Mathematics, metaphors, and human factors: Mathematical, technical, and pedagogical challenges in the educational use of graphical representation of functions, *Journal of Mathematical Behavior* 7, 1988, 135-173.

14. Gould, J.D., and Lewis, C., Designing for usability: Key principles and what designers think, *Communications of the ACM* 28, 3 (March 1985), 300-311.

15. Hegarty, Mary and Marcel Just, Understanding machines from text and diagram, in *Knowledge Acquisition from Text and Picture*, H. Mandl and J. Levin, eds., Amsterdam: North-Holland, 1988.

16. Jarvenpaa, S.L. and G.W. Dickson, Graphics and managerial decision making: Research Based Guidelines, *Communications of the ACM* 31, 6 (June 1988), 764-774.

17. Kieras, D., and P.G. Polson, An approach to the formal analysis of user complexity, *International Journal of Man-Machine Studies* 22, 1985, 365-394.

18. Koedinger, K.R. and J.R. Anderson, Abstract planning and perceptual chunks: Elements of expertise in geometry, to appear in *Cognitive Science*.

19. Larkin, Jill, "Display-based problem-solving," *Complex Information Processing: The Impact of Herbert Simon*, Klahr, D, and K. Kotovsky, eds., Hillsdale, NJ: Erlbaum, 1989.

20. Larkin, Jill and Herbert Simon, "Why a diagram is (sometimes) worth 10,000 words," *Cognitive Science* 11, 1987, 65-99.

21. Mackinlay, Jock, "Automating the design of graphical presentations of relational information," *ACM Transactions on Graphics* 5 (2), April 1986, 110-141.

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

23. Norman, Donald, *The Psychology of Everyday Things*, New York: Basic Books, 1988.

24. Payne, Stephen and T.R.G. Green, Task-action grammars: A model of the mental representation of task languages, *International Journal of Man-Machine Studies* 2, 1986, 93-133.

25. Schmid, Calvin F., *Statistical Graphics: Design Principles and Practices*, New York: John Wiley and Sons, 1983.

26. Schneider, Walter, Training high-performance skills: Fallacies and guidelines, *Human Factors* **27** (3), 1985, 285-300.

27. Tufte, Edward R., *The Visual Display of Quantitative Information*, Cheshire, Connecticut: Graphics Press, 1983.