ORSA Journal on Computing

Feature Article—Visualization and Optimization

Christopher V. Jones,

ORSA Journal on Computing
Vol. 6, No. 3, Summer 1994

0899-1499/94/0603-0221 $01.25
© 1994 Operations Research Society of America

FEATURE ARTICLE

Visualization and Optimization

CHRISTOPHER V. JONES / *Faculty of Business Administration, Simon Fraser University, Burnaby, BC, Canada V5A 1S6; Email: chris_jones@sfu.ca*

(Received: February 1993; revised: July 1993, October 1993; accepted: April 1994)

Research in optimization has developed faster and faster algorithms that have analyzed very complicated problems, thereby saving billions of dollars. Solving problems using optimization techniques involves more than just developing clever algorithms, however. Building, debugging, validating, and understanding models, algorithms, data, and solutions require appropriate representations of the models, the algorithms, the data, and the solutions. This paper explores current practice and ongoing research in different representations or visualizations useful for representing optimization models, understanding optimization algorithms, validating and exploring optimal solutions, and summarizing the results.

In the spring of 1940, with Hitler's armies advancing on France, Churchill faced a difficult decision: Should Britain deploy 10 more squadrons of fighter aircraft to France, or should the squadrons stay in Britain?[175] (The author is indebted to Peter Bell[24] for this anecdote.) Churchill felt a strong loyalty to France, but the head of Fighter Command, Hugh Dowding, was convinced that it would be a mistake since the loss of airplanes would leave Britain weakly defended. Dowding asked for an analysis of the situation from the Operational Research Section of Fighter Command (this group coined the term operational research). The presentation of that analysis persuaded Churchill that the Germans would quickly destroy the British squadrons, exposing Britain to great danger. Not only were the 10 squadrons retained in Britain, most of the British squadrons deployed in France were withdrawn. Given the near victory of Germany during the Battle of Britain (which started just two months later), it was clearly the correct decision. History is filled with similarly difficult, although less momentous, decisions. Operations research (OR) seeks to provide techniques to help understand complicated problems so that better decisions can be made.

Since Churchill's time, OR has grown into a mature field, with a rich suite of techniques and applications—from analyzing the effectiveness of HIV needle exchange programs to designing telecommunication networks. One of those techniques involves *optimization*. Given a carefully specified description or formulation of a problem, including the objective of the decision to be made, optimization techniques attempt to determine the best decision that satisfies the limitations or constraints of a problem. Applications to routing vehicles, staffing airlines, managing hydroelectric dams, blending gasoline, among many others, have been quite successful.

Yet algorithms by themselves solve only a part of any problem. The formulation must be constructed. It may have errors, which will have to be uncovered. Data must also be collected and organized. They, too, may also have errors, which will have to be uncovered. And most of the data may not be known with certainty: it may be necessary to explore how the optimal solution may be affected by changes in the data. The results of the analysis usually must be organized or summarized into some representation for the benefit or approval of other parties. We can view the process of problem solving as a series of stages, from problem identification, through model formulation, data collection, algorithm development and testing, solution validation, and final results preparation.

Each of these stages involves different *representations* of the problem at hand (see Table I). Problem solving generally starts with some idea of the general direction to follow. The initial stages of the process concentrate on gathering information to focus the problem further. Informal descriptions of the problem begin to emerge in textual or graphical form. These informal representations are then translated into a formulation that is represented in some form. One possibility is algebraic notation, although other possibilities certainly exist. Data must also be collected and stored, perhaps in some type of database. And then those data are used to create a formulation specific to the actual problem, replacing the algebraic representation with a formulation for the particular problem instance.

Since many algorithms require very restricted formats for input, this formulation might need to undergo further

Subject classifications: Optimization/visualization.

Table I. Stages in the Modeling Life Cycle and Some Visualization Techniques That Have Been or Could Be Used

Stages in the Modeling Life Cycle	Visualization Techniques
Conceptualization	Natural language Formal and informal diagrams Visual interactive modeling High level languages
Formulation	Algebraic languages Block-structured Visual languages Object-oriented programming Spreadsheets
Data Collection	Relational databases Spreadsheets
Solution	Interactive optimization Algorithm animation Minimum cost network flow Neural networks
Solution Analysis	Sensitivity analysis Objective function plots Matrix images Natural language
Results Presentation	Animation Hypertext and hypermedia Natural language Presentation graphics

translation, for example, into the common Mathematical Programming System (MPS) format. The algorithm may also need to be developed. For particularly difficult problems, this will require extensive debugging. Once the algorithm is run, it produces a representation of the solution. The solution may indicate problems in the formulation or problems in the data, whose source must be uncovered and fixed. Frequently, however, the output from the algorithm does not make the key elements of the solution obvious; therefore, it must be translated into another representation that might be more comprehensible. For example, the restriction imposed by MPS that variable names have no more than eight characters has led to the development of elaborate, but impenetrable, naming schemes for variables. At a minimum, the variables must be translated into a more intelligible representation for the final report.

In its history, the field of operations research has explored not just building clever algorithms, but also has employed a variety of representations as part of the problem-solving process. Returning to the story about the British fighters, the analysis had to be presented to Churchill in a convincing way. The analysts took the time to plot the results because they believed they would be more convincing. As Larnder[175] wrote,

> Knowing from experience that people can be persuaded through their eyes when it is impossible to do so through their ears, he [Dowding] laid his graphs in front of the Prime Minister. In Dowding's considered view, "That did the trick." ... Perhaps the real value of this OR study was not so much in presenting a Commander-in-Chief with facts concerning his own forces—facts of which he himself had shrewd knowledge—but, rather that by presenting them in graphical form OR had provided him with the means to oppose successfully what he knew would have been a fatal decision.

In a later example, Dantzig, Fulkerson, and Johnson[79] in 1954 exploited a variety of representations in their seminal paper on the use of cutting plane techniques to solve an instance of the traveling salesman problem. They sought to solve a 49-city traveling salesman problem, visiting one city in each of the lower 48 states in the US plus Washington, DC, using linear programming techniques. The original solutions contained subtours, so additional constraints —cutting planes—had to be added in order to find the optimal solution. Although this early work was essentially anecdotal, the ideas formed the basis of cutting plane techniques that have now been used to solve much larger traveling salesman problems.[136]

In the late 1950s and early 1960s, Ford and Fulkerson,[96] expanding on earlier work by Hitchcock[139] and Koopmans,[171] developed algorithms for the minimum cost network flow problem, which has application to problems in transportation and inventory planning, among many others. Minimum cost network flow problems have a natural graphical representation. Glover, Klingman, and McMillan[118] have proposed a generalized graphical framework for such problems called NETFORMs.

Around the same time as Ford and Fulkerson, project management techniques such as Project Evaluation and Review Technique (PERT) and the Critical Path Method (CPM)[165, 187, 198] were developed to manage large projects consisting of thousands of interrelated tasks. Although specialized algorithms are usually employed to analyze these problems, they can easily be formulated as linear programs. Again, the problem is conveniently represented graphically, which is part of its appeal. Many companies now sell PC-based project management systems that allow project managers to draw network representations of their projects. They can then analyze the criticality of each task to the timely completion of the entire project.

In the 1970s, vehicle routing algorithms began to be used widely in industry. The solutions to these problems are frequently represented graphically as routes of vehicles on a map; for example, see Cullen, Jarvis, and Ratliff,[77] Babin et al.,[11] and Savelsbergh.[231] Users can change the routes interactively, with an evaluation provided of the change. In work that won the Edelman Prize in 1983, Bell et al.[27] developed novel optimization techniques for vehicle routing and relied on an interactive computer graphics implementation (see [93]).

Today, the development of interactive vehicle routing systems continues,[37] spurred on in part by the emergence of a *Geographic Information System* (GIS).[176] As Bodin, Fagan, and Levy[37] state, GISs "support the capture, management, manipulation, analysis, modeling and display of spatially referenced data." GISs have allowed for much more realistic problems to be solved, since they provide detailed, accurate data concerning travel times and routes between destinations.

In the 1980s, in another Edelman Prize winning application, Lembersky and Chi[180] applied dynamic programming and three-dimensional interactive computer graphics to train loggers how to cut raw logs into lumber to yield higher profit. Loggers see a three-dimensional representation of a particular log to be cut. The loggers can then, by hand, specify how the log should be cut as well as compare their cutting pattern to the optimal pattern. Through this interaction, loggers are trained to cut their logs to yield higher profits.

The relevance of representation seems even more relevant to operations research today. Mathematical programs having hundreds of thousands of decision variables and constraints are now routinely solved. The solution to such a large problem would require many pages of output, if it were all printed. A decision maker may be interested in the behavior of a solution over a range of input values. The algorithm may experience numerical instabilities that will have to be understood to be overcome. As Greenberg[127] has stated, "we can solve far larger problems than we can understand."

Luckily, our ability to construct representations has improved. With the proliferation of interactive computer graphics, an entire field has emerged called *scientific visualization* or just *visualization*.[192, 211] Visualization has seen application to problems in physics, fluid dynamics, meteorology, medicine, and molecular biology, among others. Optimization techniques have recently begun to be used to create visualizations.[71, 248] Visualization seeks to provide insightful representations for difficult, complex problems, just like OR problems, except that the tools employed are different. In short:

1. Visualization helps solve problems. Numerous examples exist to support this contention, some of which we have already cited. Exactly what problems and what visualizations are most helpful for particular tasks, and for particular people (including researchers, practitioners, and consumers of optimization) remains difficult to assess and, hence, an area for ongoing research.
2. Visualization will be increasingly expected and required by consumers of optimization. Multi-window, mouse-driven user interfaces are now standard. The field of OR has typically been slow in delivering its tools with good user interfaces. If the field wants to see its tools actually used, it must deliver them in a form that can be easily learned and easily applied.
3. Emerging technologies hold promise for expanding the range of visualization techniques employed in optimization. Computers are becoming less expensive and more powerful. Software tools for building interactive, graphical programs are becoming easier to use. New visualization techniques involving sound, animation, three-dimensional imagery, and tactile feedback are emerging. Some have even been applied to optimization problems, and have demonstrated their effectiveness.
4. If visualization helps solve problems and is expected by users of OR software, then it should be of concern to researchers and practitioners of OR.

The fourth point runs counter to prevailing trends in the field. For example, as stated at the beginning of a recent feature article in the *ORSA Journal on Computing*,[17] "Algorithm development is at the heart of mathematical programming research, wherein more efficient algorithms are prized." It is doubtful that the two prize-winning examples previously cited would have won if they had achieved the same successful results using visualization alone. Should OR be solely about algorithm efficiency or should it be about helping people solve problems? If the OR community chooses to develop efficient algorithms exclusively, clever, powerful, rigorous techniques will certainly be generated. These techniques may even be used to solve real problems. But will OR get the credit? OR techniques may increasingly be seen as a specialized backroom enterprise, exacerbating the lack of recognition mentioned as the largest concern in a recent survey of ORSA and TIMS members.[253]

The development and evolution of computerized spreadsheets provides a cautionary tale. Spreadsheets have proven to be a tremendous aid in solving actual problems. Millions of spreadsheets have been purchased. In fact, spreadsheets are a beautiful example of visualization. The calculation abilities provided by spreadsheets and the underlying algorithms are not particularly novel. One could write a FORTRAN program to duplicate any such calculation. The brilliance of spreadsheets lies in their combining powerful calculation techniques and a common representation format—tables—within a simple user interface.

Spreadsheets have been widely adopted by OR practitioners.[36] In fact, optimization algorithms are now incorporated into the leading spreadsheets. In an interesting juxtaposition, one of the founding partners of the company that marketed the original spreadsheet, VisiCalc,[101] now runs a company that puts optimization capabilities into Microsoft Excel and Quattro Pro.

The great success of spreadsheets could be construed as a "win" for OR, since more and more people build and use mathematical models. Most people using the built-in optimization features have no idea that they are using OR techniques, nor do they care.

Of course, some have said that spreadsheets have been as much a bane as a boon to OR.[107] Putting modeling "elephant guns" into the hands of "children" increases the likelihood of fatal modeling accidents. Modeling gun control is not likely in the near future, so the concerns, though real, are moot. The marketplace has weighed in with its decision: spreadsheets help solve problems.

It seems ironic that such a powerful problem-solving tool (the spreadsheet) was not invented as part of OR—a field that claims to be about solving problems. By concentrating almost solely on algorithm efficiency, OR has neglected how those algorithms are to be delivered. The prominence of the field might be quite different if OR had been the originator of the spreadsheet instead of a Harvard MBA student named Dan Bricklin.

Instead of focusing on sophisticated models or more efficient algorithms, Bricklin focused on the representations people actually used to solve their problems. If OR truly wants to help solve problems, given the success of spreadsheets, creating more efficient algorithms is perhaps not the best path to follow. As Harvey Wagner[264] writes, "Easy-to-obtain, easy-to-use computer programs are the most effective communication link between high-powered research teams and the average professional hoping to apply the research to other real-life situations."

In short, we assert that the field of OR should be concerned about how its techniques are delivered, not just about the techniques themselves. Furthermore, we take the view that delivering optimization technology primarily involves providing a variety of appropriate representations—visualizations—to model builders, algorithm designers, as well as non-technical specialists. This paper surveys and comments upon existing research and practice in visualization and optimization and, moreover, suggests directions for future research.

The paper is organized into four sections. The first two sections decompose the types of representations used and needed to support optimization. Section 1 discusses appropriate representations and visualizations for the different parts of the modeling life cycle (see Table I). Section 2 reverses the perspective by considering different visualization formats and exploring how they have been or might be applied to mathematical programming. In the future, however, a variety of representations will be needed—and will be expected. The subsections in each of these sections are organized into two parts. The first part surveys existing research and practice. The second part offers commentary on the future direction of the particular topic. We do not claim to survey all systems that are being sold or have been mentioned in the literature; the reader is referred to a recent survey that is much more comprehensive in this regard.[236] Rather, we concentrate on what we believe are major trends and highlights. Providing the integration of multiple representations for multiple users is discussed in Section 3. Section 4 presents a research agenda for visualization in mathematical programming. Section 5 concludes by predicting the state of visualization and optimization in the next five years.

1. Visualization and the Modeling Life Cycle

What could be usefully represented in support of mathematical programming? The answer is simple: everything. One might need representations of problems, formal mathematical programming formulations, data, algorithms, ways of debugging algorithms, ways of debugging formulations, algorithm output, and summaries of algorithm output for presentation to decision makers.

We do not restrict our discussion to graphics. For example, a spreadsheet (except for its presentation graphics) consists of tables of numbers and text. We do not restrict ourselves to representations used exclusively for output. In the presentation graphics provided by a spreadsheet, for example, a bar chart encoding some data from the spreadsheet is only an output. The user cannot change the data by manipulating the bar chart with a mouse. (At least one mathematical programming system provides such a capability.[59]) In short, we believe that representation also involves input. How then will a user input the conceptual model, the formulation, and the data? How will a user interact with the algorithm and understand the solution? In discussing the different representations that seem appropriate to different phases of the modeling life cycle in this section, we also discuss how the user would input and interact with those representations.

The *key* challenge, then, is actually to provide an environment that can support the plethora of representation formats that are needed. We discuss possible solutions to this challenge in Section 3.

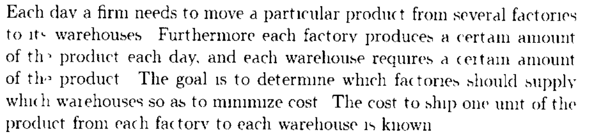
This section is organized according to the stages of the modeling life cycle discussed in the introduction, that is, conceptual models (Section 1.1), formulation (Section 1.2), data collection (Section 1.3), execution (Section 1.4), and solution analysis through to final results presentation (Section 1.5). Many authors have presented different variations of the stages of the modeling life cycle.[107] The study of how to support all the stages in the modeling life cycle has been called *model management*.[87, 239]

1.1. Conceptual Models

The start of a problem-solving process involves translating ill-focused goals and ideas into more coherent, although still imprecise, descriptions of the problem to be attacked. The problem description must be understandable by the decision maker, who frequently has little training in optimization or modeling. The representations may be simple text describing the problem in natural language or they may involve formal or informal diagrams as well. An example of a conceptual description of a classic optimization problem, the Hitchcock-Koopmans transportation problem,[139, 171] is given in Figure 1.

Several authors (Hurrion,[146] Hurrion and Secker,[147] and Bell[21-23]) have proposed a general methodology called *Visual Interactive Modeling* (VIM) for creating appropriate conceptual representations. Similar recommendations are made in [234]. These authors believe that problem solving should concentrate heavily on developing an appropriate conceptual representation, with model formulation delayed until a precise conceptual representation (usually graphical) is developed. In particular, the conceptual representation should be the primary representation of the problem.

Concentrating on the conceptual representation flies in the face of much practice, wherein the formal mathematical formulation is the primary representation of the problem.



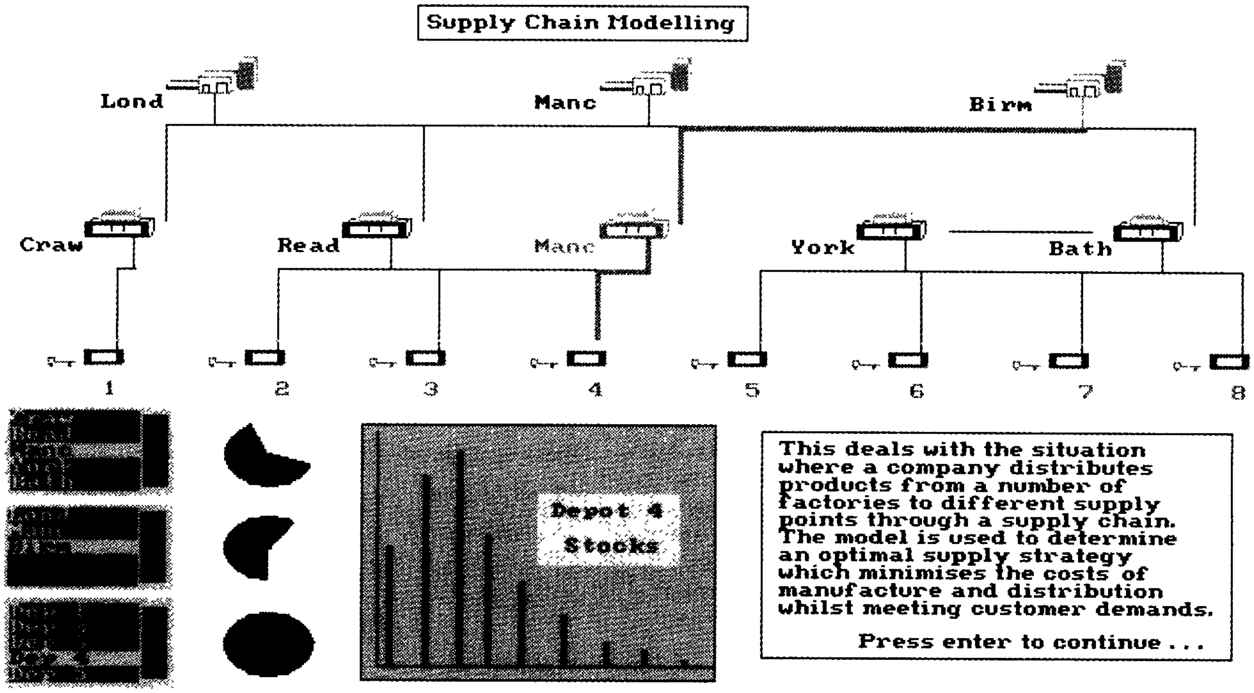
Each dav a firm needs to move a particular product from several factories to its warehouses Furthermore each factorv produces a certain amount of the product each day, and each warehouse requires a certain amount of the product The goal is to determine which factories should supply which warehouses so as to minimize cost The cost to ship one unit of the product from each factorv to each warehouse is known

Figure 1. English-language description of the Hitchcock-Koopmans transportation problem.

By representing the problem in its own terms, one helps ensure that the solution system presented to the decision maker corresponds to the actual problem. As Bell[24] notes, however, the challenge of the approach involves reconciling the conceptual representation with the mathematical representation. For VIM, commercial systems have been developed (for example, GENETIK[151]) that allow one to draw pictures for the conceptual representation, and then link algorithms to those pictures. These systems now provide links to mathematical programming systems as well. For example, an extension to GENETIK called INCEPTA[152] allows problems to be analyzed using linear programming. An example of the application of INCEPTA to a multi-commodity, multi-stage transportation problem is shown in Figure 2.

The VIM approach, which gives conceptual representations such prominence, can be thought of in terms of a larger movement called *User-Centered Design*,[212, 235] which spans the fields of human factors, ergonomics, and industrial design. Advocates of user-centered design believe that designers should begin, continue, and end the design process focused on users' needs. An oft-cited example concerns video cassette recorders. Although many such recorders provide highly developed features, many times those features go unused because users simply cannot figure out how they work. The engineers who developed the recorders became enamored with the bells and whistles, but actual users found them incomprehensible or superfluous. Mathematical programmers, including the author of this paper, have often become so enamored with the powerful tools available that they lose sight of the underlying problem and the ultimate decision makers. Visual interactive modeling attempts to place the emphasis where it should be—on the people who actually have the problem.

Several researchers have proposed specific high-level languages for mathematical programming that represent problems at a conceptual level. The holy grail for this research thread has been the development of modeling tools that are easy to use. In Structured Modeling,[111, 113] Geoffrion proposed a theory of modeling that provides several representations that attempt to represent models not just from optimization, but from the full spectrum of techniques used in OR. Relying on techniques from first-order predicate logic, Krishnan[172] proposed PM*, a high-level language for specifying linear programming problems. Originally developed for production, distribution, and inventory planning problems, models are specified not as a set of decision variables and constraints, but as a set of terms in the decision maker's own vocabulary. A graphical version of PM* has also been proposed.[163]



Supply Chain Modelling

This deals with the situation where a company distributes products from a number of factories to different supply points through a supply chain. The model is used to determine an optimal supply strategy which minimises the costs of manufacture and distribution whilst meeting customer demands.

Press enter to continue . . .

Figure 2. Graphical representation of a multi-commodity, three-state transportation problem developed using GENETIK[151] and INCEPTA.[152] Image courtesy of Insight Logistics.

Comments

Easy-to-use techniques for allowing modeling at a high level of abstraction are difficult to provide. One is caught between demands for generality and specificity—generality in order to model a variety of different problems, and specificity in that the resulting model should fit the problem like a glove. How can one provide powerful yet easy-to-understand modeling constructs? The approaches that we just described represent a start.

The philosophy espoused by the VIM community has merit, because VIM starts with the user and the user's problem, not with the model. Other alternatives, for example, Structured Modeling and PM*, attempt to impose a particular representation style on the problem holder. These two specific proposals lack the vagueness of the guidelines espoused by VIM. The constructs provided by these non-VIM techniques are meant to be sufficiently general and abstract, but one can use *just* those constructs to build models. It is naive to believe that one representation style will satisfy all users and all problems. VIM does not concentrate on a particular solution technique and is not attached to a particular set of modeling constructs, however general. Rather, it first attempts to represent the actual problem, usually visually. At that point, the model and solution technique can be developed. Unfortunately, the VIM technique makes it more difficult to assess the quality of the solution produced. In optimization, one at least can measure how fast an algorithm takes to find an optimal solution (though that is more difficult than might first appear[17]) or measure how far away an approximate solution is from the optimal solution. Taking the user-centered approach espoused by VIM, however, forces one to measure the quality of the solution produced by less objective measures such as user satisfaction.

1.2. Formulation

Once a conceptual model of the problem has been constructed (either formal or informal, visual or not), the next step in mathematical programming usually involves constructing a formal, mathematical representation of the problem, that is, a formulation.

One can distinguish between formulations for a large class of problems, and formulations for a specific problem with all data values specified. For example, one could specify a formulation for all Hitchcock-Koopmans transportation problems (see Figure 3), or for one specific Hitchcock-Koopmans transportation problem. We shall call the former a *generic model*, and the latter a *model instance*. A generic model usually can be represented far more compactly than a model instance. Most authors recommend that a clean separation between the generic model and corresponding model instances be established. Such a separation facilitates changing either part without adversely affecting the other. The question is how to represent generic models and model instances, and moreover, how to establish a linkage between them.

- Parameters
  - Let $\mathbf{N}^+$ represent the set of positive natural numbers
  - Let $N \in \mathbf{N}^+$ indicate the number of *factories*
  - Let $M \in \mathbf{N}^+$ indicate the number of *warehouses*
  - Let $i \in \{1, \quad , N\}$ represent a particular factory
  - Let $j \in \{1, \quad , M\}$ represent a particular warehouse
  - Let $s_i \in \Re^+$ represent the *supply* at factory i
  - Let $d_j \in \Re^+$ represent the *demand* at warehouse j
  - Let $c_{ij} \in \Re^+$ represent the *cost* to ship one unit of product from factory i to warehouse j
- Decision Variables
  - Let $x_{ij} \in \Re^+$ represent the quantity shipped from factory i to warehouse j

$$\text{minimize total cost} = \sum_{i=1}^{N} \sum_{j=1}^{M} x_{ij}$$

subject to

$$\begin{array}{rcll} \sum_{j=1}^{M} x_{ij} & = & s_i & \forall i \quad \text{Supply Limitation} \\ \sum_{i=1}^{N} x_{ij} & = & d_j & \forall j \quad \text{Demand Satisfaction} \\ x_{ij} & \geq & 0 & \forall i, j \end{array}$$

Figure 3. Algebraic representation of the transportation problem. Note that a feasible solution only exists when $\sum_{i=1}^{N} s_i = \sum_{j=1}^{M} d_j$.

Several different styles of formulation languages have been proposed. These include algebraic, block-structured, object-oriented, spreadsheet, and graphical formulations. Table II surveys some of the relevant references and systems. We discuss each in turn.

Algebraic languages attempt to duplicate the algebraic notation already well known by many formulators of mathematical programs. Formulations of the transportation problem written in AMPL[100] and SML[114] appear in Figures 4 and 5, respectively. Note that these and other algebraic languages such as GAMS,[41] MODLER,[129] and MPL[191] all provide tools for manipulating sets and tables, for indexing over sets, and for expressing summations. Algebraic representations have often been called *row-wise,*

Table II. Relevant Literature on Formulation

Formulation Type	References
Matrix Generator	See [97, 132] for a survey
Algebraic	GAMS,[41] AMPL,[99, 100] MPL,[191] MODLER,[129, 131] LINGO,[181] SML[114]
Block-structured	MIMI,[58, 60] MathPro,[189] PAM[266]
Object-oriented	See [142, 199, 218, 219, 220]
Spreadsheet	Microsoft Excel,[194] Lotus Improv[182, 230]
Graphical	GIN,[246] LPForm,[184] MIMI/G,[59] gLPS[73]
Matrix Images	Alvarado,[3] MATVU,[256] MIMI/G,[59] MPL,[191] Mathematica[267]

```
set PLANT;   # plants
set CUST;   # customers

param sup {PLANT} >= 0;   # amounts available at plants
param dem {CUST} >= 0;   # amounts required at customers

   check: sum {i in PLANT} sup[i] = sum {j in CUST} dem[j];

param cost {PLANT,CUST} >= 0;   # shipment costs per unit
var Flow {PLANT,CUST} >= 0;    # units to be shipped

minimize total_cost:
   sum {i in PLANT, j in CUST} cost[i,j] * Flow[i,j];

subject to Supply {i in PLANT}:
   sum {j in CUST} Flow[i,j] = sup[i],

subject to Demand {j in CUST}.
   sum {i in PLANT} Flow[i,j] = dem[j];
```

Figure 4. AMPL model[100] of the Hitchcock-Koopmans transportation problem. The check statement performs a preliminary feasibility test for this model.

since each constraint, or collection of constraints, is specified directly. At least one algebraic language, AMPL,[98, 100] has been extended to allow model specifications in a column-wise fashion. AMPL has also been extended to include special constructs for problems such as minimum cost network flow models.[98]

Another style of formulation involves a block-structured or process-oriented representation of the problem. Figure 6 shows a block-structured formulation of the transportation problem. A block-structured formulation organizes the linear programming matrix into blocks, where each block represents the intersection of a collection of decision variables and constraints. In Figure 6, we show a model of a transportation problem that contains 1 column and 2 rows. The single column (labeled T SOURCE SINK) represents decision variables transporting material from each source to each sink. The first row (labeled SUPPLY at SOURCE) constrains the amount shipped from each source. It has a matrix coefficient of +1 for all valid entries. The second row (labeled DEMAND at SINK) constrains the amount shipped to each sink to be larger than its demand. Other tables define the objective function coefficients and right-hand-side values. Greenberg and Murphy[133] described how the block-structured and algebraic representations can be integrated, since both attempt to represent the same fundamental object.

Although block-structured languages represent primarily generic models, model instances can also be represented in tabular or matrix form. Moreover, the values in the table can be color-coded. For example, negative values can be colored red, and positive values can be colored green. Such *matrix images* (see Figure 7) can represent very large matrices.

More recently, we have seen the development of object-oriented languages for optimization.[142, 218] Wegner[265] provides a good survey of the basic concepts of object-oriented programming. Object-orientation allows existing model components to be reused in a controlled fashion, by organizing them into a hierarchy. This style of organization seems natural for optimization since, for example, linear programming can be considered to be a special case of nonlinear programming, minimum cost network flow models are a special case of linear programming models, and the transportation problem is a special case of a minimum cost network flow problem.

Perhaps the most ubiquitous model formulation style can be found in spreadsheets. Most recent spreadsheets incorporate linear and nonlinear programming algorithms. Once a spreadsheet is constructed, users need only indicate the cell whose value is to be optimized, the cells whose values can be changed (the decision variables), and the bounds to assign to other cells (the constraints), and the built-in algorithm will attempt to set the decision variables to the values that optimize the indicated cell. Note that the formulation need not be specified as a matrix of constraints, but can look exactly like a traditional spreadsheet (Figure 8 shows a spreadsheet model in Microsoft Excel for the transportation problem). Users generally are not even aware that they are using mathematical programming to solve their problems.

Spreadsheets have recently begun to provide some of the higher-level indexing capabilities found in algebraic languages. For example, Microsoft Excel[194] allows spreadsheet cells to be organized into an outline. Lotus 1-2-3 allows three-dimensional spreadsheets, and Lotus Improv[182, 230] supports multi-dimensional spreadsheets with algebraic formulas defining the values of cells.

Various authors have proposed the use of graphical or *visual languages* to represent mathematical programs.[200] For general surveys on visual programming languages, see [53, 54, 242]. For mathematical programming, most visual languages rely on graphs as the fundamental modeling construct. A variety of graphs and networks have been proposed (see Table 2), at various conceptual levels.

In most of these languages, each constraint and decision variable is represented either as a node or an arc. For example, in an activity-constraint diagram (see Figure 9), square nodes represent decision variables, circular nodes represent constraints, and arcs represent the coefficients of decision variables in constraints. An activity-constraint diagram can model problems at both the generic (see Figure 9a) and instance (see Figure 9b) levels since a node can represent an individual decision variable (or constraint) or a collection of decision variables (or constraints). At the generic level, an activity-constraint diagram essentially translates a block-structured representation into a graph. Collaud's system (see [73]) uses a variation on activity-constraint diagrams as does MIMI/G.[59]

In typical graph-based representations for network flow problems (see Figure 10), constraints are represented by nodes and decision variables are represented by arcs connecting the nodes. Perhaps the most ambitious proposal to support such representations was made by Steiger, Sharda, and LeClaire.[246] Their system, GIN, adopts the NETFORM style of representation proposed by Glover, Klingman, and McMillan.[118]

&SDATA *SOURCE DATA*

PLANT$_i$ /pe/ *There is a list of PLANTS*

SUP(PLANT$_i$) /a/ {PLANT} $\Re^+$ *Every PLANT has a non-negative SUPPLY CAPACITY measured in tons*

&CDATA *CUSTOMER DATA*

CUST$_j$ /pe/ *There is a list of CUSTOMERS*

DEM(CUST$_j$) /a/ {CUST} $\Re^+$ *Every CUSTOMER has a nonnegative DEMAND measured in tons*

&TDATA TRANSPORTATION DATA

LINK(PLANT$_i$,CUST$_j$) /ce/ {PLANT} × {CUST} *There are some transportation LINKS from PLANTS to CUSTOMERS There must be at least one LINK incident to each PLANT. and at least one LINK incident to each CUSTOMER*

FLOW(LINK$_{ij}$) /va/ {LINK} $\Re^+$ *There can be a nonnegative transportation FLOW (in tons) over each LINK*

COST(LINK$_{ij}$) /a/ {LINK} *Every LINK has a* TRANSPORTATION COST RATE *associated with all FLOWS*

\$(COST FLOW) /f/ 1, @SUM$_{ij}$(COST$_{ij}$ * FLOW$_{ij}$) *There is a TOTAL COST associated with all FLOWS*

T SUP(FLOW$_i$,SUP$_i$) /t/ {PLANT}, @SUM$_j$(FLOW$_{ij}$) <= SUP$_i$ *Is the total FLOW leaving a PLANT less than or equal to its SUPPLY CAPACITY? This is called the SUPPLY TEST*

T DEM(FLOW $_j$,DEM$_j$) /t/ {CUST}. @SUM$_i$(FLOW$_{ij}$)=DEM$_j$ *Is the total FLOW arriving at a CUSTOMER exactly equal to its DEMAND? This is called the DEMAND TEST*

Figure 5. Schema in SML[114] for the transportation problem.

COEF	T
	SOURCE
ROW by COL	SINK
SUPPLY at SOURCE	+1
DEMAND at SINK	+1

Figure 6. A block-structured representation of the transportation problem. This example is adapted from MIMI.[60]

Some graph-based languages attempt to represent mathematical programs at a higher conceptual level than constraints and decision variables; see, for example, [164, 184]. Notable among those languages is LPFORM.[184] In LPFORM (see Figure 11), nodes represent decision variables, and arcs represent either decision variables (such as transportation activities) or other logical constraints not directly represented in the linear program. Nodes can also represent existing sub-models that are being used in the larger model.

If one compares the graphical representations for the transportation problem in Figures 9 (activity-constraint), 10 (NETFORM), and 11 (LPFORM), one will note that they all involve a bipartite graph. This reflects the fact that half of the nodes are mainly associated with the sources of supply and the other half are associated with the sinks. At least for the simple transportation problem, these representations are sufficiently similar to suggest that a general form might exist. Murphy, Stohr, and Asthana,[200] in fact, have shown that LPFORM generalizes both the NETFORM and activity-constraint representations.

Since models can grow quite large, their corresponding graph-based representations, particularly model instances, can become quite unwieldy. Even the small graphs in Figures 9 and 10 cannot be easily drawn so that they are clear. A variety of schemes for reducing the tangle of spaghetti that large (and even small) graphs can exhibit have been proposed.[73, 184, 246] These schemes often involve some form of hierarchical decomposition such as collecting a group of related nodes and arcs together into a single node. Unlike the well-established conventions for organizing text, wherein books are organized into chapters, chapters into sections, and sections into paragraphs, generally accepted conventions for organizing large graphs into helpful hierarchies do not exist.

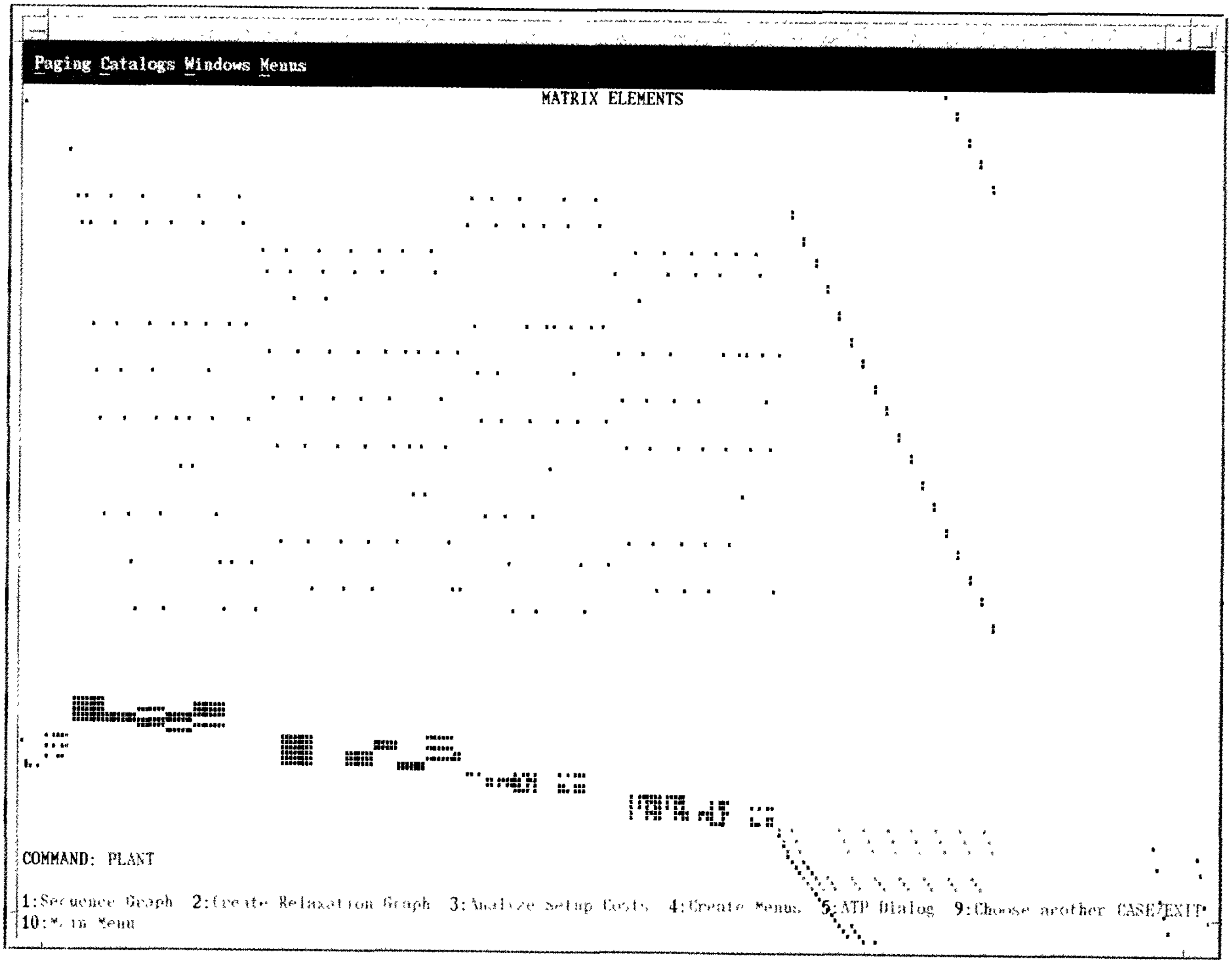
Structured Modeling,[112, 113] in addition to specifying a standard graph-based representation for models (see Figure 12), provides a formal mechanism for creating such hierarchies. The *genus graph* groups together individual model elements based on formal similarity properties. Structured modeling represents an ambitious attempt to provide a standard formalism for modeling.

Comments

Given the variety of representation styles for the actual formulation, the natural question arises: How do they compare? Great attention has been paid to this question by some authors.[97, 115, 132, 200] Although these comparisons have provided much food for thought, none of these representation formats are dominant. The authors of these representation formats almost certainly find their representation format easiest to use. Many authors built the representation specifically for their own use. Perhaps spreadsheets, given their widespread use, can be said to be the dominant representation format. But current spreadsheets do not provide the sophisticated indexing abilities needed by mathematical programming models; nor do they provide graphical formulation tools now becoming available for mathematical programming. Spreadsheet technology is moving to provide such capabilities quickly, however.

In short, if all of these representation formats have their admirers, it would seem prudent to allow users to model their problems in whatever form they find easiest. As stated by Greenberg and Murphy,[132] in comparing three different modeling languages in detail, "Many model builders from academic backgrounds think readily in algebra, while people with process industry backgrounds think in terms of processes [column-wise] . . . The next generation of modeling systems should be able to support all views. We found ourselves switching among the various languages with the same formulation when we wanted to learn different things about the model." The challenge is to provide an ability to translate seamlessly from one form to another.

For graphical languages in particular, although many have been proposed, none can be said to dominate. In fact, graphical languages remain curiosities. Although this may be explained by their recent emergence, it may reflect fundamental problems with the graphical approach. Most


Paging Catalogs Windows Menus
MATRIX ELEMENTS
1:Sequence Graph 2:Create Relaxation Graph 3:Analyze Setup Costs 4:Create Menus 5:ATP Dialog 9:Choose another CASE/EXIT
10:Main Menu

COMMAND: PLANT

Figure 7. Matrix image of a large linear programming model. Non-zero elements are represented by colored dots. Negative and positive values have a different color. Users can also zoom in on various parts of the image. Note that the matrix image extends to the bottom of the figure, with superimposed text. Image courtesy of Chesapeake Decision Sciences.

mathematical programmers were taught how to formulate problems using one particular representation style, usually algebraic. After several courses and much practice, experienced mathematical programmers find their favorite representation scheme quite natural.

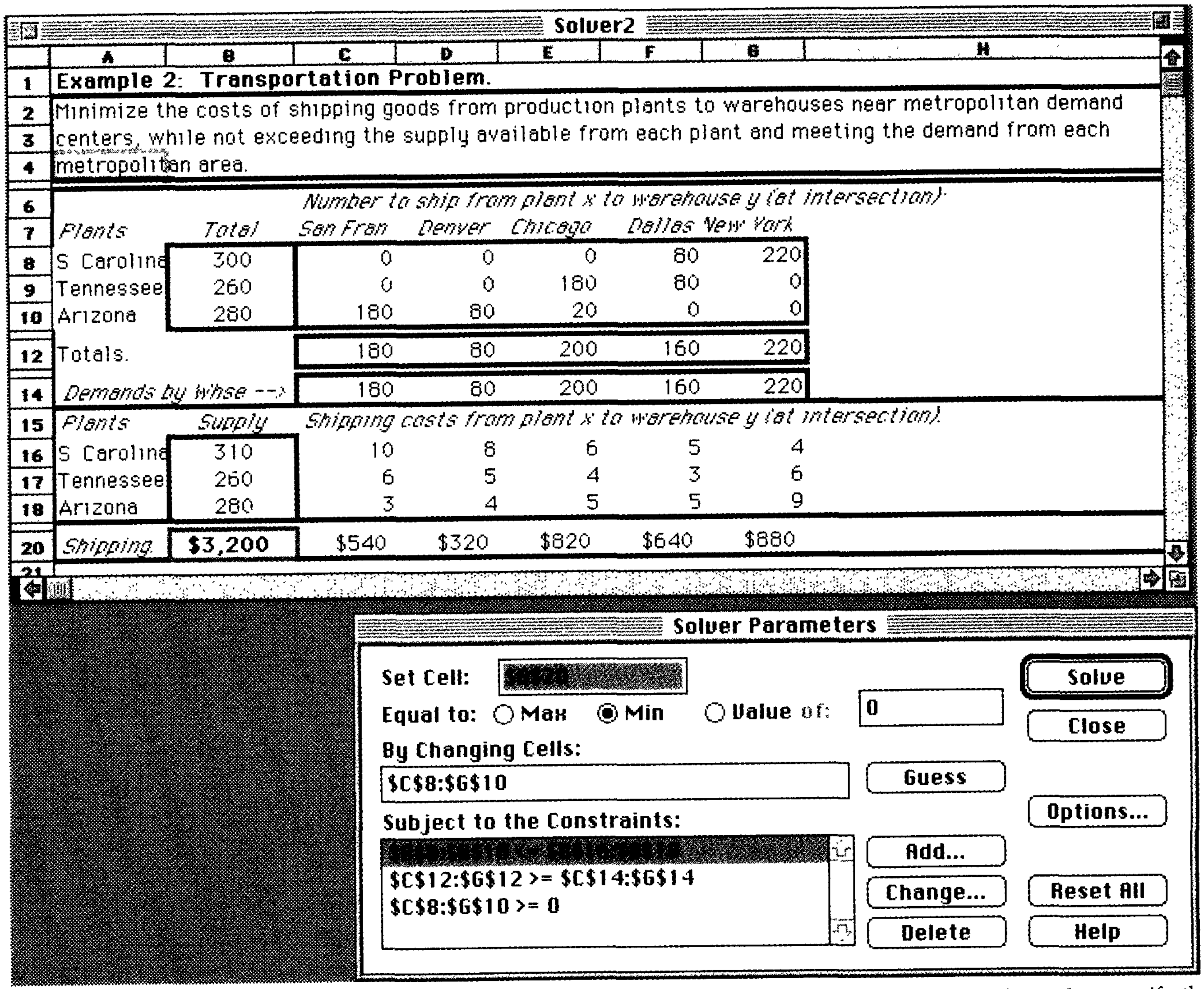
Unlike the algebraic style, no graphical representation scheme is commonly taught or commonly used. If one already feels comfortable with a non-graphical format, there may be no advantage to using any graphical format. Non-technical people, however, often find mathematical notation painfully opaque, so perhaps graphical representations will be helpful for them.

Given both the variety of proposals for graph-based languages for mathematical programming and the wide variety of other graph-based representations used for modeling such as project management and vehicle routing, instead of trying to invent the perfect graph-based language for all optimization problems, a more fruitful approach might seek to provide tools for working with a variety of different types of graphs. Many researchers have been pursuing this idea, and we will discuss that work further in Section 2.3.

1.3. Data Collection

A formulation without data cannot be solved. In many real problems, the data may already exist in corporate databases, spreadsheets, or in someone's head. Somehow the data and formulation must be combined so that an algorithm can be applied. Increasingly, commercial mathematical programming systems are providing linkages to relational databases and spreadsheets. This linkage is facilitated by internationally recognized or defacto standards such as Structured Query Language (SQL) for relational databases and the various file formats provided by spreadsheets.

Database theory has become a respected academic field, and relational databases are now ubiquitous in practice.[81]



Solver2

	A	B	C	D	E	F	G	H
1	**Example 2: Transportation Problem.**							
2	Minimize the costs of shipping goods from production plants to warehouses near metropolitan demand							
3	centers, while not exceeding the supply available from each plant and meeting the demand from each							
4	metropolitan area.							
6			*Number to ship from plant x to warehouse y (at intersection)*					
7	*Plants*	*Total*	*San Fran*	*Denver*	*Chicago*	*Dallas*	*New York*	
8	S Carolina	300	0	0	0	80	220	
9	Tennessee	260	0	0	180	80	0	
10	Arizona	280	180	80	20	0	0	
12	Totals.		180	80	200	160	220	
14	*Demands by Whse -->*		180	80	200	160	220	
15	*Plants*	*Supply*	*Shipping costs from plant x to warehouse y (at intersection).*					
16	S Carolina	310	10	8	6	5	4	
17	Tennessee	260	6	5	4	3	6	
18	Arizona	280	3	4	5	5	9	
20	*Shipping*	$3,200	$540	$320	$820	$640	$880	

Solver Parameters

Set Cell: [illegible]

Equal to: ○ Max ● Min ○ Value of: 0

By Changing Cells:
$C$8:$G$10

Subject to the Constraints:
[illegible]
$C$12:$G$12 >= $C$14:$G$14
$C$8:$G$10 >= 0

Figure 8. Microsoft Excel spreadsheet model of the transportation problem. The dialog box at bottom is used to specify the objective function, decision variables, and constraints for this problem.

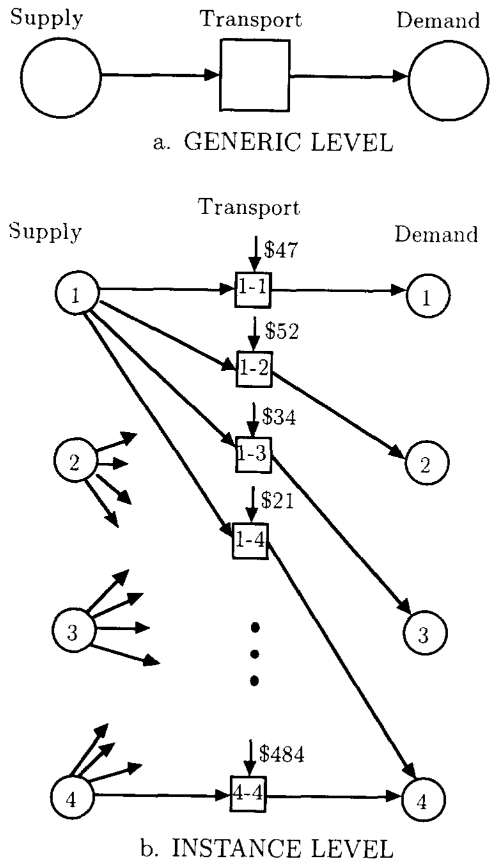
Concepts from relational databases have begun to be incorporated into mathematical programming systems. This takes at least two forms. First, most mathematical programming systems provide relational database-like capabilities for manipulating data. Second, relational database theory provides models for data, and these data models are being extended to cover mathematical programming.

In the first case, Codd[70] proved that three database operators (Projection, Selection, and Join) were sufficient to allow any data manipulation desired. These capabilities are now provided by almost every mathematical programming system. Mathematical programming systems, however, with their rich indexing capabilities, provide very elaborate facilities for accessing and combining data. Several authors, notably Geoffrion,[113, 115] have shown how these capabilities are at least equivalent to relational databases.

In the second case, several authors have proposed extensions to relational databases to cover mathematical programming (see Bhargava, Krishnan, and Mukherjee,[33] and Choobineh[64]). Choobineh[63] extended entity-relationship diagrams,[56] a graph-based model for relational databases, to provide another graph-based language for mathematical programs (see Figure 13). Since entity-relationship diagrams are commonly taught and widely used for database design, Choobineh's approach to providing a graphical language for linear programming may have a greater chance for success than other graphical approaches.

Comments

Users now expect to be able to access data in a mathematical programming system using relational database capabilities. Luckily, standards are now sufficiently developed to



a. GENERIC LEVEL

b. INSTANCE LEVEL

Figure 9. Activity-constraint diagrams for the transportation problem. In (a), the activity-constraint diagram for the generic transportation problem is shown. In (b), the activity-constraint diagram for a four-source, four-sink transportation problem instance is shown. Decision variables are represented by squares, and constraints are represented by circles. Note that for clarity in (b), many of the nodes and arcs are omitted.

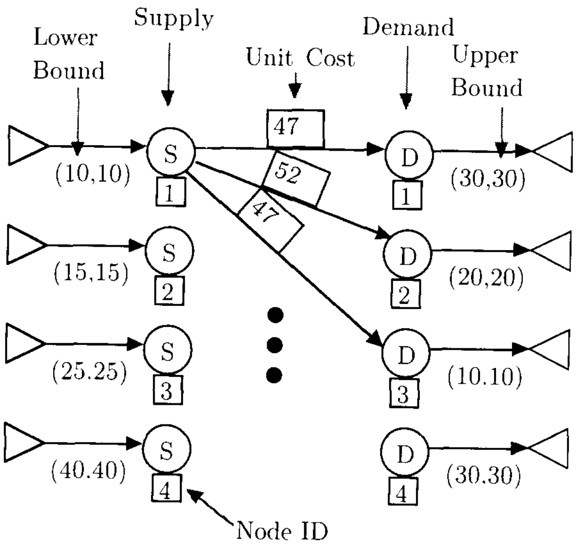


Figure 10. NETFORM[118] representation (after Steiger, Sharda, and LeClaire[246]) of a simple transportation problem. Much of the detail has been suppressed for clarity.

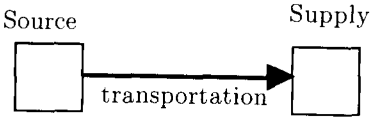


Figure 11. LPFORM[184] graph for the transportation problem. Each square node represents a set of activities. In the example, the left square node represents a collection of sources, and the right square node represents a collection of sinks. The arc represents the flow of material from the sources to the sinks.

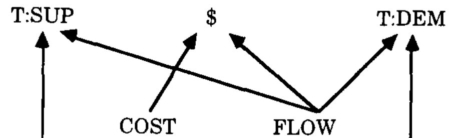


Figure 12. Structured modeling genus graph for the transportation problem. Compare this graph to the textual representation of SML in Figure 5. Each node represents a collection of elements in the model. Each arc $a \rightarrow b$ indicates that genus b is defined, at least in part, by genus a. In short, the genus graph illustrates the interrelationships among genera.

allow easy access to relational databases. Data modeling capabilities are also now incorporated into mathematical programming systems. Like spreadsheets, relational databases provide a useful paradigm for modeling. Databases contain their own particular models of data. The data needed for solution, by definition, are already stored in the database. Relational databases have become part of the basic infrastructure of computing.

1.4. Algorithm Execution

Once the conceptual model has been constructed, the model formulated, and the data collected, the algorithm must then be run. We might believe that the user's involvement only consists of typing the appropriate command to start the algorithm. However, when the algorithm does not converge or when the solution is claimed to be unbounded or infeasible, one often needs to explore the internal execution

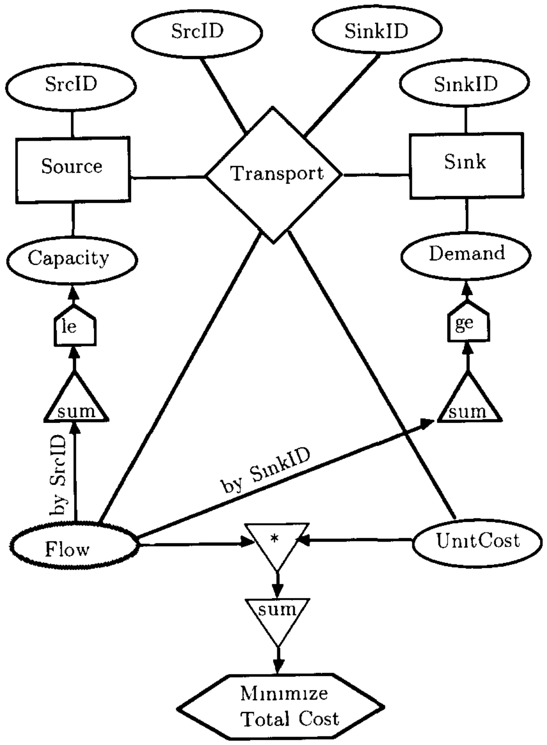


Figure 13. Adapted from work by Choobineh,[63] this graph-based representation is based on entity-relationship diagrams[56] used in database design. An entity-relationship diagram represents entities, such as sources and sinks, as squares. Simple relationships among the entities, such as the transport of material, are represented by diamonds connected to the entities. Entities and relationships can have attributes, represented as ellipses, connected to them. For example, each source has a capacity, and each transport relationship has a unit cost. Choobineh extended this basic model with additional node types to represent constraints, decision variables, and objective functions. Decision variables are represented as ellipses with light borders. For the transportation problem, the ellipse labeled "Flow" represents the decision variables. The objective is represented by a hexagon. Constraints are formed using nodes representing mathematical operators (triangles) and logical relationships (houses) such as greater than or equal to (ge).

details of the algorithm. This again will require appropriate representations.

These representations include human intervention during the solution process, visualizations to help understand the execution of the algorithm, as well as visualizations that provide new and novel insights.

Fisher[92] proposed linking human perceptual abilities with a computer's computational speed to produce a hybrid form of optimization that he called *interactive optimization*. In this style of optimization, the user would intervene at particular points during the execution of the optimization algorithm and provide insights that the algorithm would find difficult to generate by itself. Brady, Rosenthal, and Young[40] provide an example of such a system applied to a facilities location problem. The algorithm drew various circular regions on the screen with the user having to identify a point in the intersection of the regions. The point chosen allows the system to further restrict the size of the region where the facility is to be located. The algorithm terminates when the region contains only a single point, that is, the optimal location of the facility. Another example comes from vehicle routing.[77] Users could specify new routes as starting points. The new routes then appear as new columns in the mathematical program.

Another visualization technique has often been used to help algorithm designers understand the behavior of their algorithms. This visualization technique animates the internal execution of the algorithm, and has been dubbed *algorithm animation*.[30, 44, 45, 244] As an algorithm executes, a picture of the current state of the algorithm, perhaps showing the data structures, is updated when "interesting" events occur. Typical algorithm animations[183] for the simplex method, for example, show a three-dimensional polytope of either a three-variable linear programming problem or a projection of a larger problem into three dimensions. Each pivot of the algorithm is illustrated as a movement from one vertex to another along the polytope. An algorithm animation of an interior point method[110] shows how the algorithm iteratively distorts the polytope. For an example from combinatorial optimization, Boyd, Pulleyblank, and Cornuejols[39] animated several different algorithms for the traveling salesman problem.

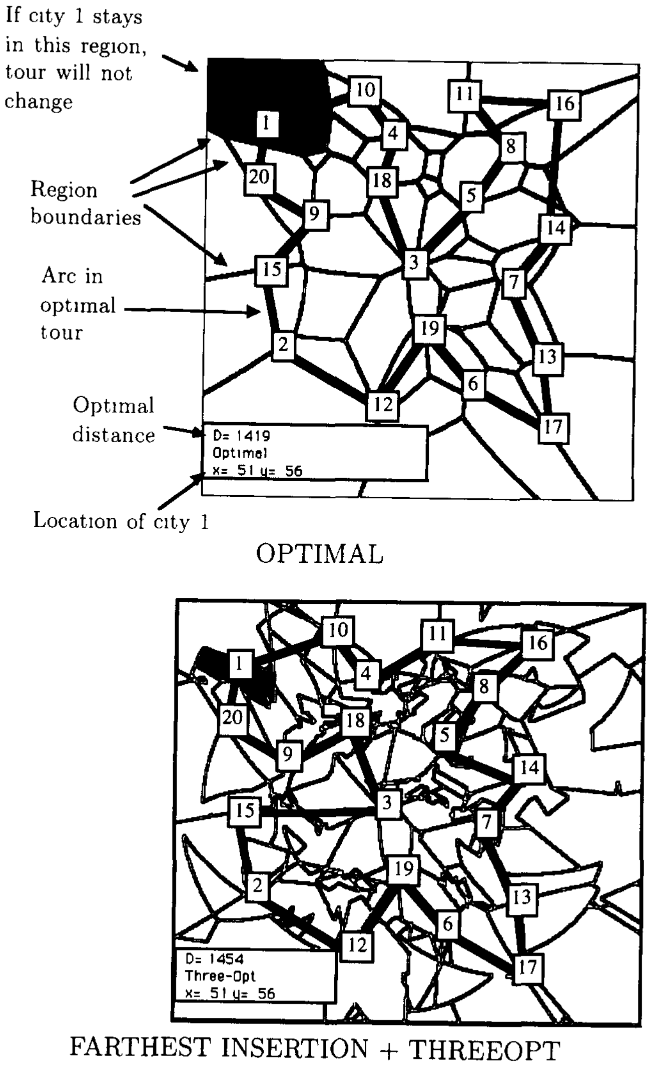
Algorithm animation is essentially the same as animation typically found in discrete event simulation.[26, 124] In such an animation, a picture of the underlying problem changes at interesting events in (simulated) time. Although most authors laud the ability of animation to debug and validate the simulation model, it does not remove the need for careful statistical analysis,[26, 177] since a single animation may not include examples of important behavior.

VIM actually saw its principal first application as Visual Interactive Simulation (VIS). The VIS community takes pains to emphasize the difference between interactive animation and non-interactive animation. In an interactive animation, the model and data can be changed at any time, with the behavior of the model then observed. In non-interactive animation, the user cannot change the model or model data while the animation is proceeding—the animation merely represents the playback of a recorded "movie." For optimization algorithms, especially for large problems, playback of a movie may be all that is possible, since the time between significant events can be large. For example, if one is animating a large linear programming problem, each iteration may require several minutes (or longer), which is far too slow for real-time animation. One may have no alternative but to store a sequence of images, one for each iteration, for later playback after the algorithm is run.

Many solution techniques arguably have enjoyed success because of their natural graphical representation. The classic example for mathematical programming, of course, is a minimum cost network flow model.[96, 119] The proliferation of activity on neural networks,[143, 241] including its use in optimization,[128, 144, 237, 263] has been under intensive investigation in part because of its graphical representation.

Finally, visualization of algorithms can lead to new problems and new theory. Hubbard (as described in [117]) visualized the behavior of Newton's method for finding roots of nonlinear functions. He was interested in how the solution that resulted depended on the starting point. When he color-coded each starting point based on the final solution that was produced, the image that resulted was a fractal. This example illustrates the power of visualization to provide new insights into the behavior of even well-established algorithms.

In another example related to optimization, conventional wisdom has stated that optimal solutions to combinatorial optimization problems are generally more sensitive to changes in the input data than are approximate solutions. As Bartholdi and Platzman[18] wrote about the traveling salesman problem, "optimal solutions are fragile in the sense that they can be exquisitely sensitive to changes in the data." For the planar case, Jones[160] considered how the tour changed when a single city is moved. If the city moves far enough in the correct direction, the tour will eventually change. So, for any tour, there is a region in which the tour will not change, as long as the moving city stays inside the region. Different solution algorithms (optimal and approximate) generally produce different sets of regions. By drawing the regions in which the tour remains unchanged, some insight can be gained on the behavior of different algorithms. In particular, as shown in [160], most of the classic approximation algorithms, with some notable exceptions, produce less stable solutions than optimization algorithms (see Figure 14). It was only through the visualization that this result was uncovered.



OPTIMAL

FARTHEST INSERTION + THREEOPT

Figure 14. The above figures are associated with two different algorithms for the traveling salesman problem, the upper figure with an optimization algorithm, the lower figure with an approximation algorithm (Farthest Insertion plus ThreeOpt). These figures explore the sensitivity of the solution when a city is moved, in this case city 1. Each region in each picture represents an area where the tour computed by the algorithm does not change, as long as city 1 stays inside the region. Note that the regions for the approximation algorithm are far more complex than those for the optimization algorithm.

Comments

Although interactive optimization has often been used, is it useful? No definite answer can yet be given, although some results are beginning to appear. In an experimental comparison of interactive solutions produced by optimization algorithms for the planar traveling salesman problem, Mak, Srikanth, and Morton[186] showed that humans could produce better solutions than even the best heuristic algorithms. Taseen[252] showed that his particular example of the use of VIM for an optimization problem produced consistently better answers than a non-graphical version of the same problem.

As for algorithm animation, although widely touted, experimental evidence has been sorely lacking. The one study that has been conducted[245] actually found that algorithm animation does *not* provide any significant aid to understanding the behavior of algorithms. The authors of the study, who developed a major algorithm animation environment, conjectured that their study did not provide sufficient motivation to subjects, nor enough guidance as to what the subjects should look for in the animation. These negative results may indicate that algorithm animation does not live up to the hype of its promoters, and thus should be abandoned. The wide use of the technique, however, suggests that it does have some useful applications. The challenge is to discover exactly how it is best delivered, to which audiences, and for what tasks.

Visualization techniques have helped to uncover unexpected behavior in algorithms, as witnessed by Hubbard's work (as discussed in [117]), for example. Unfortunately, although the use of visualization has increased, experimental evidence to support its use is rare. We simply do not have clear guidelines that indicate which visualization techniques are best suited to which problems and which users.

1.5. Solution Analysis and Results Presentation

Once an algorithm is run, a solution is produced. For linear programming problems, information useful in sensitivity analysis is also produced, including dual prices and ranges on the objective function coefficients and on the constraints.

In many cases, however, the solution is not correct. It may be that the problem is infeasible or unbounded. The algorithm may fail to converge. For nonlinear programming problems, although a local optimum may have been found, it may not be the global optimum.

The causes of these problems are legion: the input data can contain errors, constraints may be missing, or extraneous constraints may have been included. For nonlinear programs, the initial solution may not have been the best. For integer programs, adjustments to the branching method, the bounding method, or other parameters may need to be performed. For most problems, one is interested not just in an individual solution but in its behavior over a set of possible input parameters or over a range of possible cases.

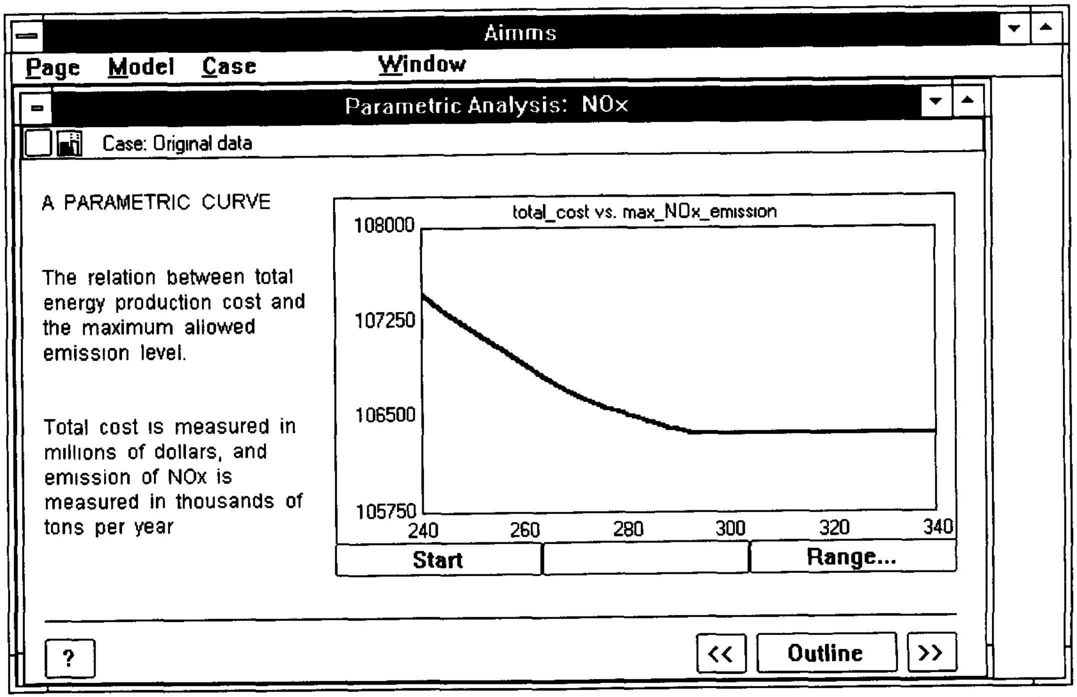
In short, the completion of the algorithm represents the beginning of a long process of debugging and validation, i.e., understanding the problem. Much research has been conducted to provide better analyses of solutions[78, 90, 91, 103, 261] and to diagnose infeasibilities.[61, 62] In this section, we discuss some of the visualization techniques that have been applied.

Several different types of plots are frequently used in sensitivity analysis. For example, plots of the value of the objective as a function of an input parameter (see Figure 15) are commonly seen in textbooks, but are just emerging in commercial systems.[191, 217] In goal programming, one produces plots of efficient frontiers, exploring the trade-offs among multiple objectives.

Perhaps the most well-developed tool for exploring the behavior of linear programming problems is Greenberg's ANALYZE.[126, 130] ANALYZE provides extensive facilities for displaying pieces of the linear programming problem. Since linear programming problems ultimately are transformed into some form of table or matrix, the pieces shown by ANALYZE are typically displayed as submatrices (see Figure 16a). ANALYZE also provides natural language explanations that automatically explain the meaning of variable names and constraints,[125] among others (see Figure 16b). More sophisticated analysis of linear programming models is also provided. ANALYZE, for example, can produce a natural language explanation of the value of a shadow price. Currently, however, all ANALYZE output consists of text, including matrix images (see Figure 16a). At least some of ANALYZE's output could be usefully displayed graphically (for example, see Figure 7).

Comments

Models, like computer programs, are usually not written correctly the first time. Errors in the model itself, in the

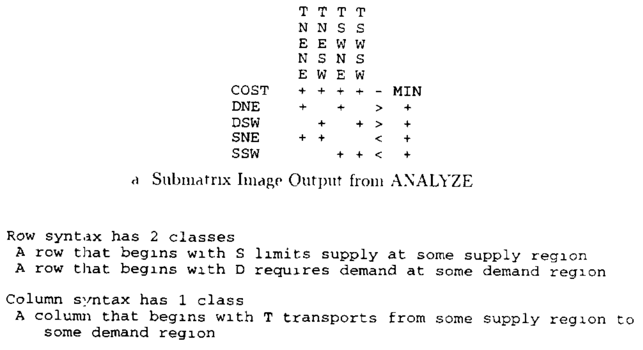


A PARAMETRIC CURVE

The relation between total energy production cost and the maximum allowed emission level.

Total cost is measured in millions of dollars, and emission of NOx is measured in thousands of tons per year

Figure 15. Plot of the objective function value as the value of the right-hand side of a constraint is varied. The image comes from AIMMS.[217] Image courtesy of Paragon Decision Technology.



```
           T T T T
           N N S S
           E E W W
           N S N S
           E W E W
COST       + + + + - MIN
DNE        +   +   >  +
DSW          +   + >  +
SNE        + +     <  +
SSW            + + <  +
```

a Submatrix Image Output from ANALYZE

```
Row syntax has 2 classes
 A row that begins with S limits supply at some supply region
 A row that begins with D requires demand at some demand region

Column syntax has 1 class
 A column that begins with T transports from some supply region to
    some demand region
```

b Natural Language Output from ANALYZE

Figure 16. Two outputs produced by ANALYZE.[126] In (a), the "+"s represent nonnegative matrix entries. The short row and column labels are a consequence of an eight-character limit on identifiers. The column label TNESW represents the transport (T) of material from the northeast (NE) to the southwest (SW) region. Similarly, the row label DNE ensures that the demand (D) by the northeast region (NE) is met. Another output from ANALYZE (b) provides a natural language explanation for the row and column names. Images courtesy of H. Greenberg.

data, and in the basic concepts are usually not revealed until the model is built and solved. With current models often involving hundreds of thousands of decision variables and constraints, it seems imperative to provide "debuggers" to help uncover the problems in a model, similar to debuggers for computer programming.

For example, if an algebraic language is used to specify a problem, then the debugger should report its results in a form consistent with that algebraic language. If the model has been translated into a form required by a solver, much of the semantic information in the algebraic specification is lost. The debugger must somehow maintain linkages to the original formulation, much as a debugger for programming languages maintains linkage between the source code and machine code.

Debugging a model, although extremely important, does not guarantee the accuracy of the model. Models must be validated, that is, their behavior must be judged against a real-world standard. Does the output correspond to our expectations? If not, perhaps the model has uncovered an unexpected, but useful, insight on the real problem. Or perhaps the model is simply inaccurate. In short, does the output make sense when applied to the actual problem? Debugging tools can certainly help probe the behavior of the model, but we seem less able to formalize validation. The model was built to represent a complex real-world problem. How can we be certain that the model represents the real world with sufficient accuracy?

2. Visualization Formats

In the previous section, we explored visualization from the perspective of the tasks involved in a mathematical programming project. In this section, we discuss some of the different formats that are available for representations. We first present them individually. Future optimization systems will allow a variety of representations to be viewed and manipulated simultaneously. The individual representation formats that we consider are text, hypertext, static graphics, animation, sound, touch, and virtual reality.

2.1. Text

Although "visualization" is usually assumed to imply "graphics," text should not be ignored. Ask any poet.

Early mathematical programming systems could only accept input in textual form (as punched on cards) and produce output in textual form. Perhaps the use of text for building and understanding linear programming models has been fully explored. However, recent developments would suggest otherwise.

For example, the ability of today's computers to display multiple fonts in multiple sizes allows for more expressive representations. Geoffrion[114] in SML showed how different text styles could be used to highlight different aspects of information (see Figure 5).

Whereas textual languages have been able to simulate algebraic syntax, they do not duplicate it. However, it is now possible for users to enter algebraic syntax directly at least for text-editing purposes, including summation notation. This allows users to enter algebraic syntax using special symbols such as Σ and $\forall$ at least in a text editor. Although we are unaware of any optimization system that accepts such formatted text directly, those capabilities should break down the barrier between the traditional algebraic formulation and the actual modeling language.

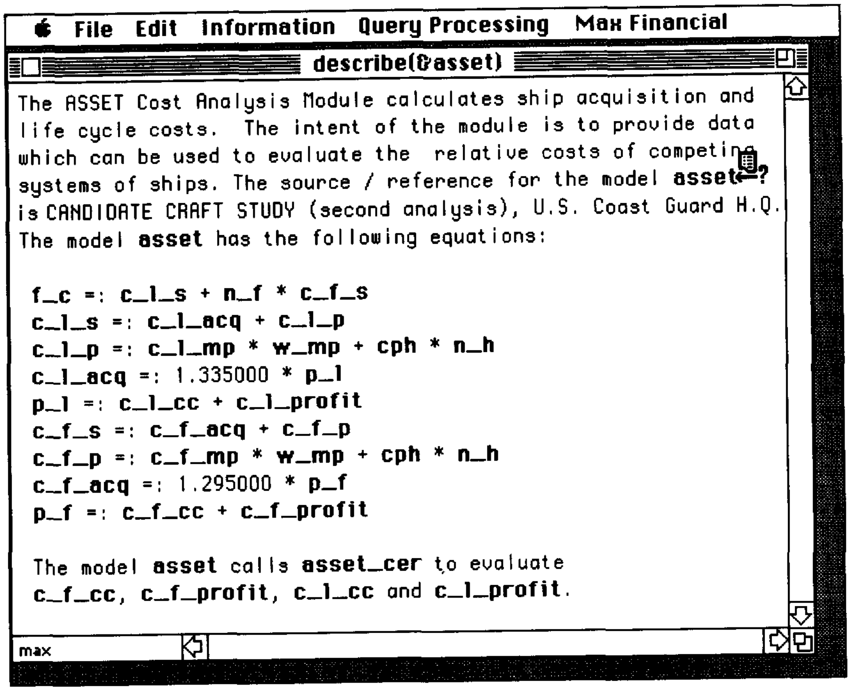
Some authors have developed software that can generate automatically natural language expressions that explain the behavior of mathematical programs. We have already discussed ANALYZE[126] (see Figure 16). The system of Kimbrough et al.[168, 169] also generates text automatically (see Figures 17 and 18).

Textual representations can also be manipulated. Languages for performing symbolic manipulation, for example, MACSYMA,[222] Mathematica,[267] and Maple,[55] provide extensive facilities for performing symbolic differentiation, symbolic integration, and equation solving (for example, Mathematica provides built-in linear and nonlinear programming algorithms). For nonlinear programming, this type of manipulation seems especially useful.

For the input of text, *syntax-directed editors* have been proposed. They provide editing operations tailored specifically to a particular language. For example, the system in [223] provides simple editing operations to add and remove if-then-else and do-while constructs from Pascal, while ensuring that variables are declared before they are used. Such syntax-directed editing techniques have been applied to mathematical programming and other modeling languages.[140, 262]

Comments

Many current modeling languages rely on plain ASCII text created using a text editor. Although this choice greatly enhances portability, it denies users access to more sophis-

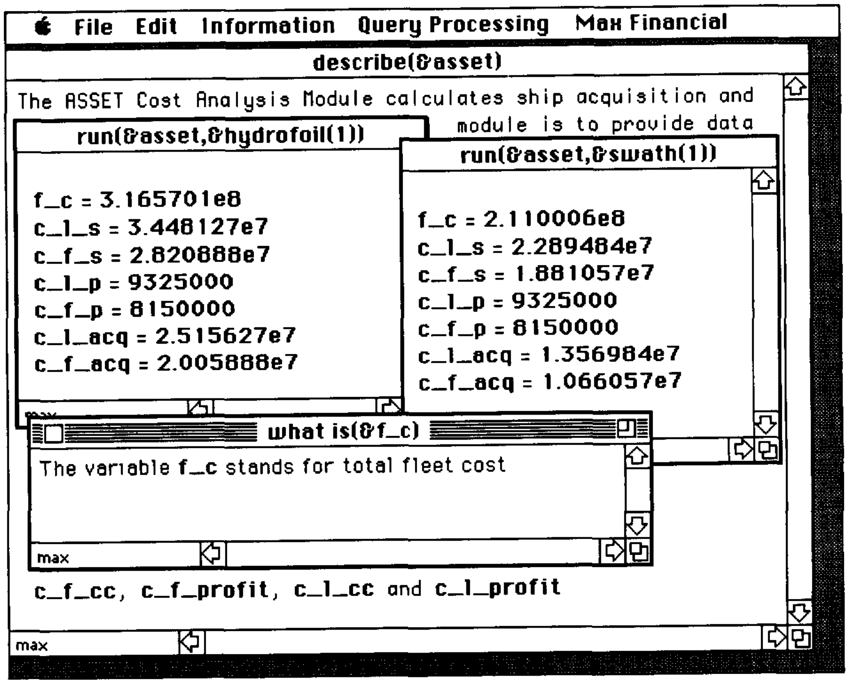


```
The ASSET Cost Analysis Module calculates ship acquisition and
life cycle costs.  The intent of the module is to provide data
which can be used to evaluate the  relative costs of competing
systems of ships. The source / reference for the model asset?
is CANDIDATE CRAFT STUDY (second analysis), U.S. Coast Guard H.Q.
The model asset has the following equations:

 f_c =: c_l_s + n_f * c_f_s
 c_l_s =: c_l_acq + c_l_p
 c_l_p =: c_l_mp * w_mp + cph * n_h
 c_l_acq =: 1.335000 * p_l
 p_l =: c_l_cc + c_l_profit
 c_f_s =: c_f_acq + c_f_p
 c_f_p =: c_f_mp * w_mp + cph * n_h
 c_f_acq =: 1.295000 * p_f
 p_f =: c_f_cc + c_f_profit

 The model asset calls asset_cer to evaluate
 c_f_cc, c_f_profit, c_l_cc and c_l_profit.
```

Figure 17. Clicking on **asset**, the name of a model, in order to display more information about the model. Figure courtesy of S. Kimbrough.



```
The ASSET Cost Analysis Module calculates ship acquisition and
                              module is to provide data
```

run(&asset,&hydrofoil(1))

```
f_c = 3.165701e8
c_l_s = 3.448127e7
c_f_s = 2.820888e7
c_l_p = 9325000
c_f_p = 8150000
c_l_acq = 2.515627e7
c_f_acq = 2.005888e7
```

run(&asset,&swath(1))

```
f_c = 2.110006e8
c_l_s = 2.289484e7
c_f_s = 1.881057e7
c_l_p = 9325000
c_f_p = 8150000
c_l_acq = 1.356984e7
c_f_acq = 1.066057e7
```

what is(&f_c)

```
The variable f_c stands for total fleet cost
```

```
c_f_cc, c_f_profit, c_l_cc and c_l_profit
```

Figure 18. Results of running the **asset** model with two scenarios, **swath** and **hydrofoil**. The user has also asked for more information about the variable **f_c**, which is displayed in the lower window. Figure courtesy of S. Kimbrough.

ticated text formats and text input techniques. The ability to use multiple fonts in multiple sizes is slowly becoming easier. On Unix systems, X Window[232] provides standard mechanisms for working with complex fonts. Microsoft Windows and the Apple Macintosh also provide standard font manipulation capabilities. Adobe Systems,[1] the inventors of the PostScript printer language, has just recently proposed a standard format called Acrobat[226] for interchanging complex multi-font documents across multiple platforms. In short, the ability to work with multiple fonts will become even more widespread.

Syntax-directed editing should help to eliminate syntactic errors in models. Most algebraic languages, for example, assume a standard text editor for building the model. Then, the model is "compiled," with syntax errors uncovered. The modeler must then iterate through a revise-compile loop before the solution algorithm is ever run. A syntax-directed editor would help the modeler uncover syntax errors more quickly, thereby providing valuable assistance in the model-building process.

2.2. Hypertext

One especially interesting extension to text is *hypertext*.[10, 74] Hypertext was foreshadowed almost 50 years ago by Bush's[49] speculative article describing mechanisms to organize complicated information. Briefly, most articles, books, and other printed documents assume that users will read the material from front to back, that is, in a linear fashion. Hypertext-based systems allow users to jump around and explore information in a *nonlinear* fashion. Users need only point to hypertext *links* to jump to display additional information. For example, if the text contains a link labeled "Lagrangean Relaxation," the user can quickly obtain more information about that topic. Hypertext is now often used to provide on-line help, for example, in Microsoft Windows. Hypertext, of course, need not be limited to displaying only text. *Hypermedia*[209] provides nonlinear navigation capabilities for text, graphics, sound, and video.

Given the complexity of mathematical programs, hypertext seems like a promising organizing tool. Kimbrough et al.[168, 169] developed a hypertext-based system for organizing mathematical programming and other types of problem-solving projects. Essentially, any important modeling construct, such as variable names and values, parameter names and values, and model names, can serve as a hypertext link (see Figures 17 and 18). Users need only point to the item of interest in order to obtain more information about that item. When the information is displayed, users can request additional information about items in the new window, and so on. In this fashion, users can navigate through a complicated optimization project. Hypertext style links are now part of MIMI.[59]

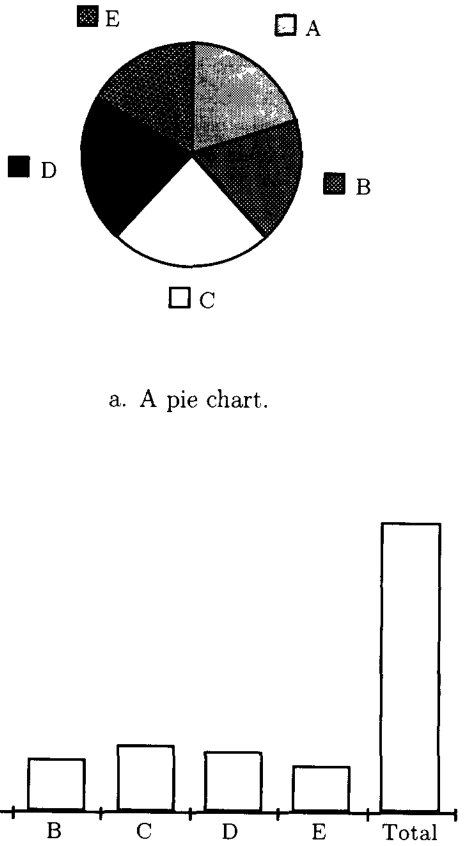
Comments

Hypertext provides a natural, easy-to-use mechanism for navigating through complicated information. Since models are complex, hypertext should become a standard interface style for browsing through optimization models and solutions. The lure of clicking a mouse on a variable, column, or constraint to obtain more information is irresistible.

2.3. Graphics

With the availability of computer graphics hardware, as well as robust graphics standards that allow graphics to be exchanged among applications, presentation graphics such as bar charts and line charts are ubiquitous. They are built into spreadsheets, word processing programs, and databases. Numerous authors have developed general guidelines for their use.[68, 69, 257, 258] Clearly, graphics can be used to present summaries of results, to plot the value of the optimal solution as a function of unit costs, profits, or resource availabilities, or to plot the convergence of the objective function to its optimal value.

Such easy-to-use tools make it easier to misuse presentation graphics as well.[257, 258] For example (as Cleveland and McGill[69] noted), pie charts are a commonly used representation format. Compare the pie chart of Figure 19a to the bar chart of Figure 19b. Both charts plot the same data. In the bar chart, the largest bar represents the sum of the



a. A pie chart.

b. A bar chart.

Figure 19. A pie chart and a bar chart of the same data. According to Cleveland and McGill,[69] most people find it easier to compare the heights of the bars than the sizes of the pie pieces.

other bars. Which of the two formats makes it easier to rank the values from largest to smallest? Most people find that the bar chart representation makes this task much easier. Pie charts, however, are well known and widely used. People may continue to use even harder-to-perceive presentation graphics such as pie charts simply because they are more familiar with them.[116]

Commercial vendors are now providing more and more styles of presentation graphics, often with questionable utility. For example, three-dimensional bar charts might seem alluring, but humans usually find it easier to view a dataset when it is plotted in two dimensions.[69] For machine scheduling, we proposed a three-dimensional Gantt chart[155] that is quite similar to a three-dimensional bar chart. Although this type of Gantt chart is an interesting piece of conceptual art, it has no practical application.

As discussed in Section 1.2, networks and graphs have commonly been used for mathematical programming. Although not as ubiquitous as tables, graphs nevertheless are widely used. In addition to graph-based languages for mathematical programming, many systems have been implemented for specific types of problems, including project management software and interactive vehicle routing systems. Vehicle routing systems make extensive use of different types of graphs and networks[11, 37, 93, 186, 231] since a road map is just a graph.

Just as spreadsheets provide a general, easy-to-use interface for working with tables, perhaps one could provide a general, easy-to-use interface for working with graphs. Several authors have developed systems based on graphs as the fundamental representation format. Dao et al.[80] developed Cabri, which allows users to explore a variety of problems in graph theory. NETPAD[82] includes a modular user interface into which various combinatorial optimization algorithms can be inserted. Linx[213] supports a graph-based representation for production and distribution planning. MIMI/G[59] provides a general graph-based representation capability for mathematical programming and other models.

Jones[156–158, 161–164] proposed a framework for graph-based modeling using *attributed graphs* and *graph-grammars*.[120–122, 204–206] By relying on formal language theory, support can be provided for a variety of graph-based representations, not just in mathematical programming (see Figure 20). Linkages to Structured Modeling[159] and to PM*[163] have also been explored.

In the domain of production scheduling, Gantt charts[65, 104] (see Figure 21) have been used in factories since the early part of the twentieth century and in computer systems since at least the late 1960s.[106] They are ubiquitous in project management software.

For the input of graphics, one can use a *direct manipulation* interface. Direct manipulation interfaces have been defined[148, 240] as interfaces that increase the level of "semantic engagement" between the user and the computer. Rather than having to describe to the computer the desired action (for example, to delete a particular line of text), the user physically performs the operation directly. For graphics input, direct manipulation interfaces typically allow users to draw pictures of their problems using a mouse or stylus.

Angehrn[4] and Angehrn and Lüthi[5] proposed a novel direct manipulation technique for creating and analyzing models. Using their technique, known as *modeling by example*, modelers simply draw a picture of the current problem (actually, a graph). For example, in a telecommunications design problem (see Figures 22 and 23), where the goal is to locate switching centers, a user would draw a picture of the existing network, assign attributes to the nodes and arcs in the network, and suggest locations for the switching centers. The system deduces the basic problem, selects and executes an algorithm, and then presents a solution, that is, a set of switching centers. This technique seems similar to research in computer science on building user interfaces by example.[190, 201–203] In this line of research, a user interface builder draws the desired user interface, provides examples of how the user interface should respond to particular user actions such as moving the mouse, and the interface is then built automatically.

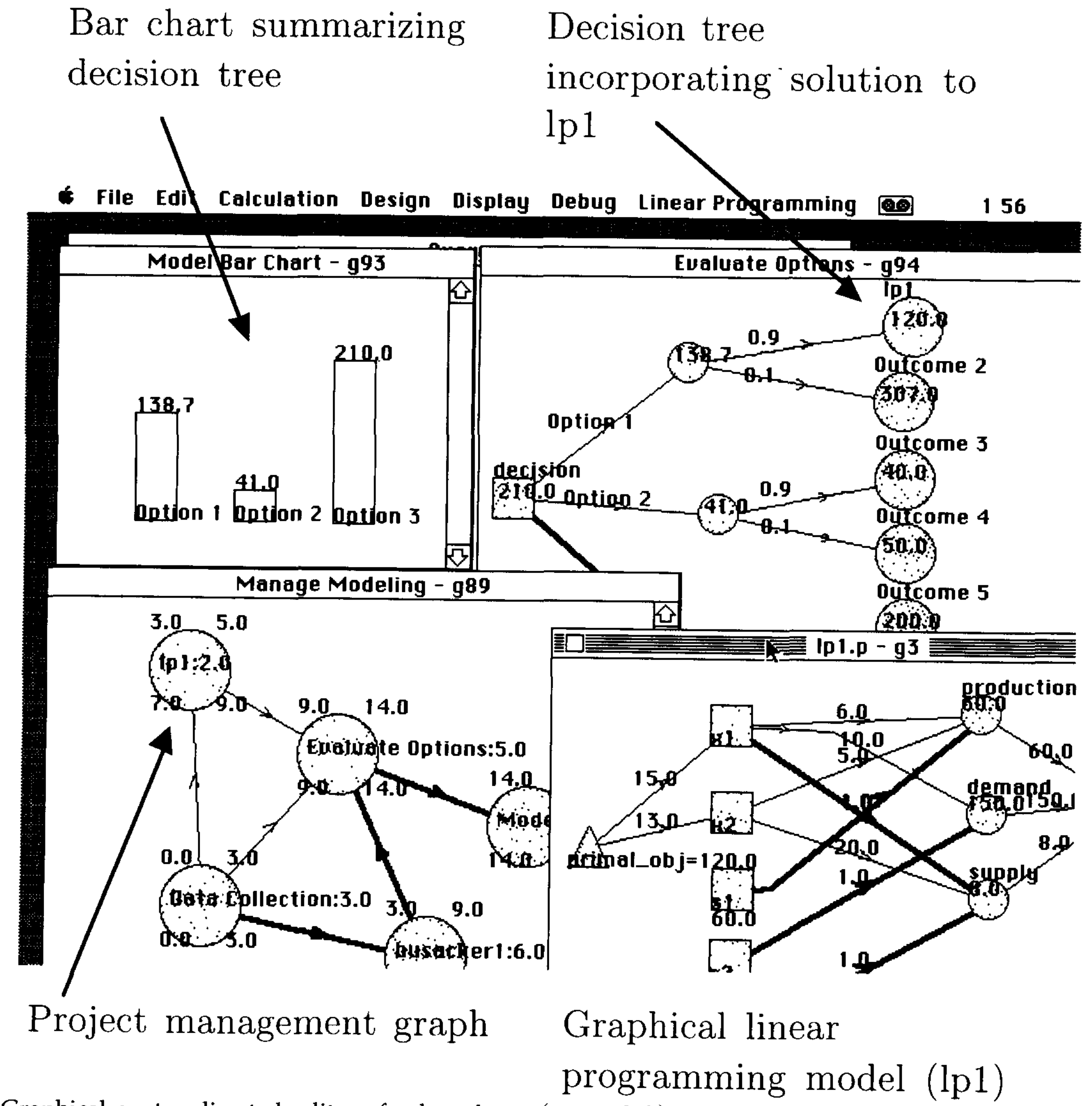
Comments

Even with the wide variety of graphics possible, only a few representation formats are actually used. Traditional presentation graphics are reasonably well covered by today's software. Gantt charts have seen wide use in project management systems, and are also being used in production scheduling systems. Graphs and networks are widely used for modeling, but no single software paradigm analogous to spreadsheets for tables has emerged for graphs and networks. Developing such a paradigm would be a great benefit to modeling practice.

The computer-aided design community has had much greater success developing a general paradigm for their graphical models. Autocad,[196] for example, has established a defacto standard to allow designers to create mechanical and architectural designs. However, these applications seem easier than many of those tackled in OR since mechanical parts and buildings have a natural graphical representation while it is difficult to envision an appropriate representation for an optimal recipe for blending gasoline. Many of the problems that are tackled by OR just do not have an obvious graphical representation.

2.4. Animation

When images are updated quickly enough (ideally over 30 times per second[51]), *animation* occurs. As discussed in Section 1.4, algorithm animation appears useful to help algorithm designers and others understand the behavior of algorithms. Jones[161] proposed a different style of animation intended more for non-technical people. Instead of animating the internals of the algorithm, this style of animation, known as *animated sensitivity analysis*, animates how the solution changes as input parameters are changed continuously. Jones demonstrated applications of this idea to teaching the graphical solution technique for two-variable linear programming problems and understanding solution heuristics for traveling salesman problems and production scheduling problems. For example, in a two-



Bar chart summarizing decision tree

Decision tree incorporating solution to lp1

Project management graph

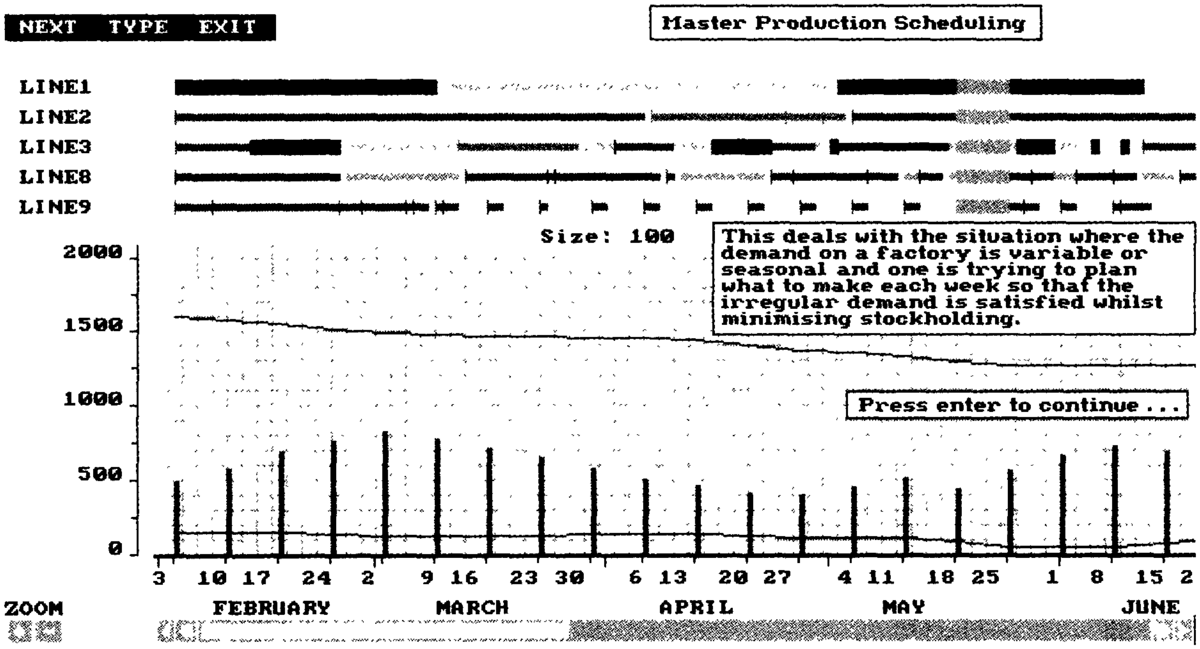
Graphical linear programming model (lp1)

Figure 20. Graphical syntax-directed editors for bar charts (upper left), decision trees (upper right), project management networks (lower left), and linear programs (lower right).[162] Note that the heights of the bars in the bar chart are determined by the values of the probabilistic nodes (smaller circles) in the decision tree. Similarly, the values of the leaves of the decision tree are determined from other graphs. For example, the value of 120.0 for the leaf node labeled lp1 in the decision tree is calculated by the linear program. Finally, the project management network is used to manage the other models.

variable linear programming application, users can change a right-hand-side value by dragging the line representing the constraint with a mouse (see Figure 24). As the line is dragged, the optimal solution, feasible region, and dual values are recalculated and redisplayed simultaneously. Buchanan and McKinnon[48] and Belton, Elder, and Meldrum[28] have also explored animated sensitivity analysis for linear programming.

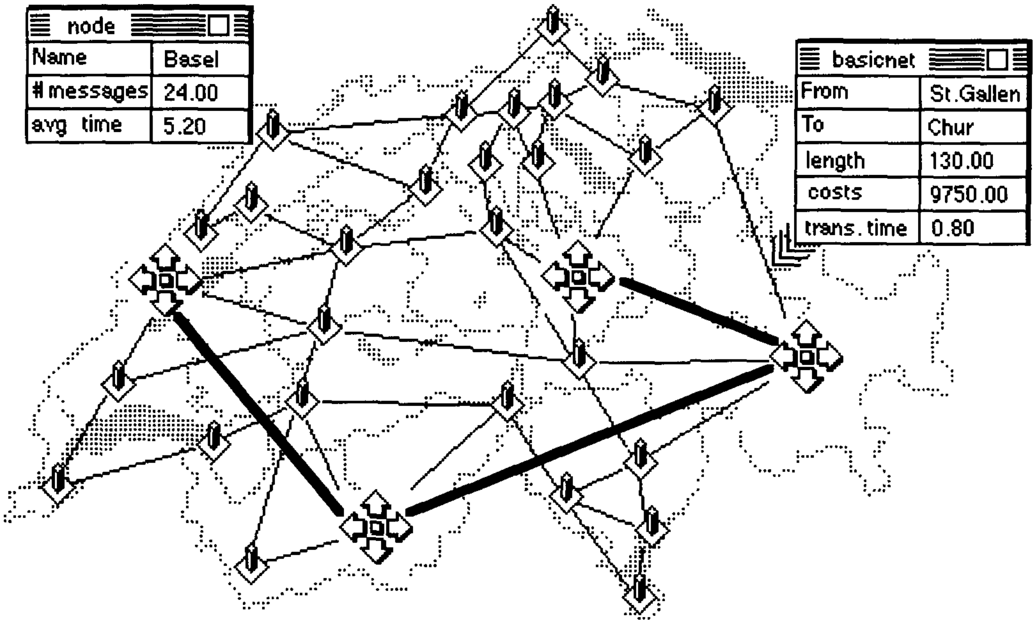
In both algorithm animation and animated sensitivity analysis, it helps to produce a static representation of the history of the animation. For example, in the visualization of the planar traveling salesman problem[161] discussed in Section 1.4, users can move a city and simultaneously see the new tour. As a city moves, the sequence of cities on the tour changes only at discrete points. The regions in which the tour does not change are drawn in "batch" mode.

Animation is now used in a different way, however. In the original version, only a single city moves, say city 1, with the other city locations fixed. What happens if the other cities are allowed to move as well? For each incremental set of city positions, the regions are drawn, assuming city 1 is allowed to move with respect to the other cities. This set of images can then be played back, producing an animation. From viewing the animation, very pre-



This deals with the situation where the demand on a factory is variable or seasonal and one is trying to plan what to make each week so that the irregular demand is satisfied whilst minimising stockholding.

Figure 21. A Gantt chart from INCEPTA.[152] Image courtesy of Insight Logistics.



node	
Name	Basel
# messages	24.00
avg time	5.20

basicnet	
From	St.Gallen
To	Chur
length	130.00
costs	9750.00
trans. time	0.80

Figure 22. A simple model for locating switching centers. Users can point to nodes and arcs to see their attributes. For example, the table on the left-hand side shows the properties of the node Basel (24 messages sent with an average transmission time of 5.2 seconds). In a second table (right), the planner can see the attributes of the arc connecting the two nodes St. Gallen and Chur. Figure courtesy of A. Angehrn.

liminary observations suggest (see Figure 25) that optimal algorithms still produce more stable solutions than most classic approximation algorithms.

Comments

With the advent of multimedia standards such as MPEG[178] and QuickTime,[8] animation is becoming as widespread as graphics. For example, one can now cut and paste animations among spreadsheets, word processors, and even other programs such as Mathematica. Users can then play the animations inside each application.

With the proliferation of animation comes a danger, however. When static computer graphics became cheap and easy, a variety of garish, probably useless graphics

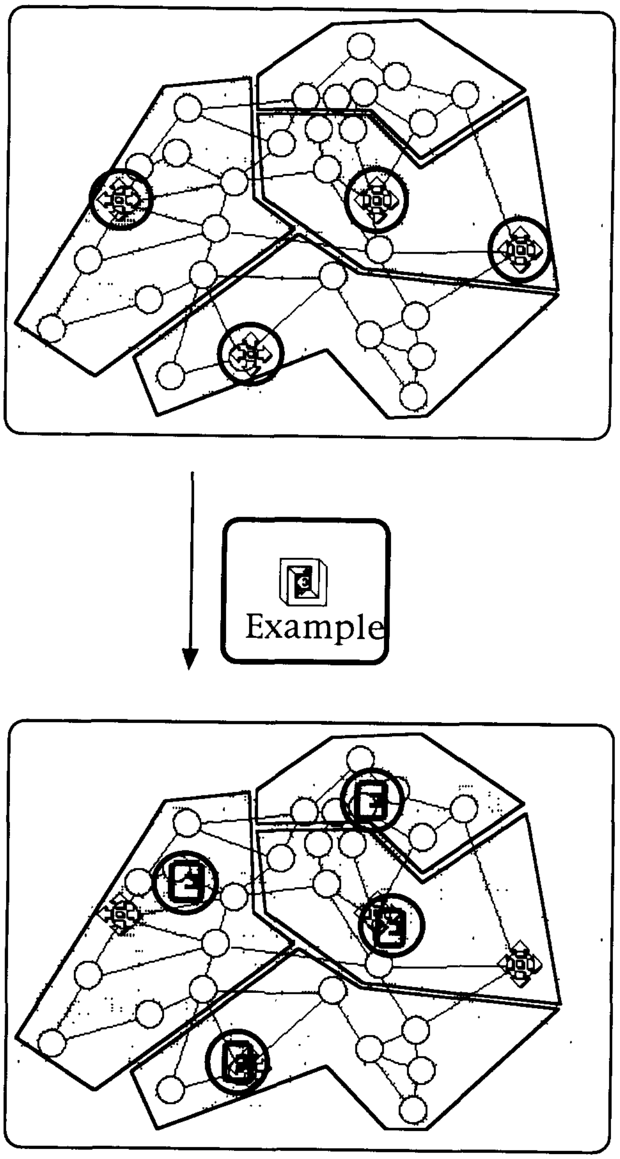


Figure 23. Once users have suggested positions for switching centers (top box), the system infers the general structure of the problem and suggests new locations for the switching centers (bottom box).

were produced.[257] Most people have never been trained in even the most basic principles of graphics design. With the increasing ability to produce animation, we now are faced with the prospect of improper use of this medium.

Guidelines for proper use of animation are rare. Although animators from Walt Disney Studios have produced useful guidelines for producing entertaining animation,[255] how that applies to optimization is unclear. The algorithm animation community[44-46, 244] is providing anecdotal recommendations, but the generalizability of those guidelines is open to question. In short, animation is coming to a computer near you, but how you should use it is just beginning to emerge.

2.5. Sound

More and more computers provide high-quality sound input and output capabilities. Several spreadsheets[182, 194] now allow users to annotate the spreadsheets with recorded sounds. Users just speak into a microphone to add voice annotation. Mathematica[267] allows voice annotation as well, but also allows one-dimensional mathematical functions to be played back as sounds. The user can also plot the sound as a line chart.

Although the use of sound in user interfaces is still emerging, some idea of its potential can be described.[12, 34, 50, 108, 109] Perhaps most importantly, sound provides another means of communication between user and computer. That is, sound provides a communications channel that is independent of the visual channel. Police cars and ambulances exploit this phenomenon: one need only hear the ambulance to know it is nearby. Sound is frequently used in computer programs to signal unusual conditions, if only as a beep when the user commits an error.

Brown and Hershberger[46] explored the use of sound in algorithm animation. They have used sound, for example, in an animation of a sorting algorithm. The magnitude of each item is encoded as the pitch of a sound. Whenever the item is moved, its corresponding note is played. Different sorting algorithms using the encoding scheme produce audibly different patterns.

Comments

Although sound has been used in computer simulation,[76] it has rarely been used for mathematical programming. It has potential because it offers another communication channel between human and computer.

We have informally experimented with the use of sound in an animated sensitivity analysis for the planar traveling salesman problem. As a city is moved, the tour can change. An alert tone can be sounded whenever the tour changes. Also, a tone can be played whose pitch is proportional to the objective function value. Whether these sounds are actually useful or merely produce an annoying cacophony is unclear. The danger of noise pollution can be significant. During the Three Mile Island nuclear plant catastrophe, over 60 audible alarms sounded.[228]

2.6. Touch

Recent research in human-computer interaction has explored exploiting the human sense of touch. Experimental hardware has been developed that gives force-feedback to the user, or simulates the feel of different textured surfaces. Such *haptic* ("of or relating to the sense of touch"[123]) devices are beginning to be applied.

For example, Brooks[42] and Brooks et al.[43] have used a force-feedback robot arm for "molecular docking." Molecular biologists need to understand how molecules interact and how they bond together or "dock." Interactive computer graphics systems that allow users to see and manipulate molecules in three dimensions are widely used. In Brooks's system, the molecular biologists not only see the molecules, but feel the forces exerted on the molecules.

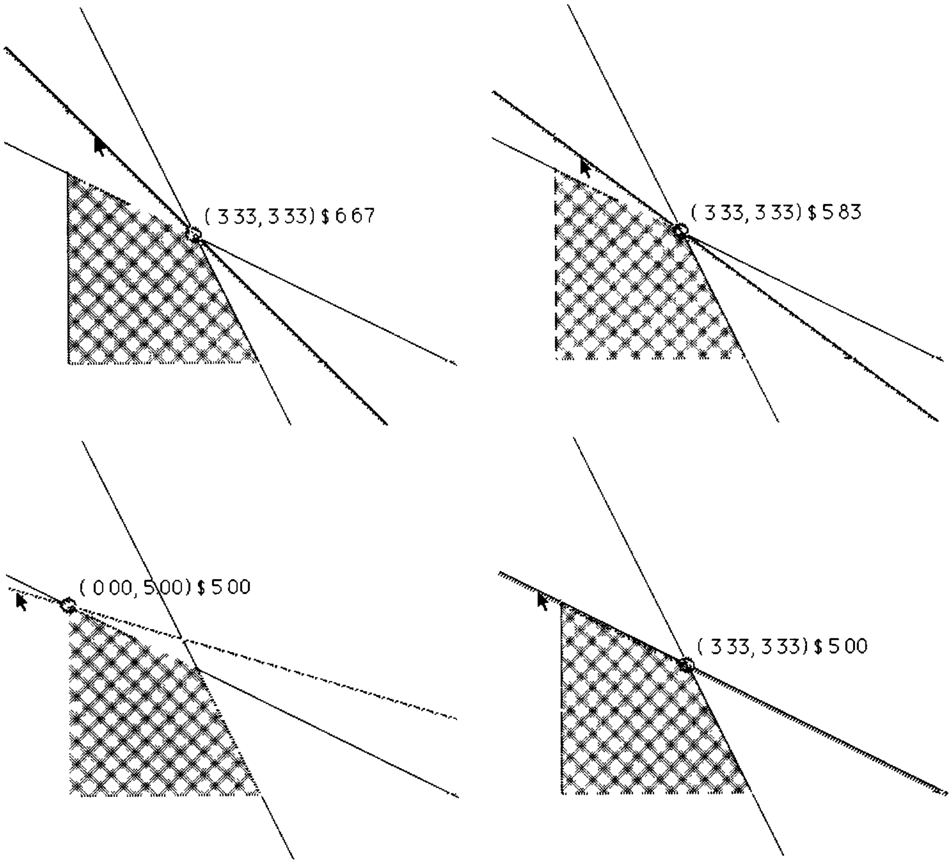


Figure 24. Animating changes to the objective function for a two-variable linear program. The images are arranged clockwise, starting with the figure in the upper left.

Brooks showed that the addition of force-feedback cuts the time to determine the docking configuration by a factor of two. The molecular docking problem can also be modeled as a nonlinear mathematical program, but at the time of Brooks's article, the best mathematical programming approach required over a day of CPU time, whereas molecular biologists could determine the proper configuration in a few minutes.[197]

Comments

At this point, other applications of haptic displays to optimization are only speculative. However, force-feedback might be used to illustrate hard and soft constraints physically. As a soft constraint is violated more and more, the force-feedback would increase proportionately.

2.7. Virtual Reality

In 1965, graphics pioneer Ivan Sutherland[249] described the characteristics of the ultimate computer display. Quite simply, such a display would be able to communicate with users using the full range of human senses available—three-dimensional images using the entire field of view, stereo sound, touch, and perhaps even smell and taste. In short, the goal is to create a *virtual* or *artificial reality*.[173, 174, 224] Since we only can perceive the world through our senses, exploiting those senses to the fullest holds the possibility of more effective human-computer communication.

In recent years, a variety of input and output devices have been developed that are beginning to approach Sutherland's vision. It is now relatively straightforward to add stereoscopic viewing capabilities to traditional computer monitors.[254] But for a more realistic experience, a new output device is emerging, the *head-mounted display*.[225, 250] With current technology, a head-mounted display consists of a special helmet worn by the user. In the helmet are two displays that provide stereoscopic imagery, as well as stereo headphones for playing sounds. A tracking device monitors the position and orientation of the user's head so that the image displayed and sounds generated can be updated appropriately. Other devices (for example, the DataGlove[270]) can track the motion of individual fingers on the user's hands, or even the user's entire body.

Typical head-mounted displays create an *immersive* artificial environment with which the user can interact. Other authors, however, have suggested that users will need to maintain a closer relationship with the real world. For example, a doctor might view a computer-generated image of the insides of a patient projected onto the actual patient. Some have proposed "see-through" head-mounted displays,[89] so that the user sees the real environment and synthetic imagery projected onto that environment.

Virtual reality has perhaps seen widest application in flight simulators[269] and military simulators.[247] Other applications are just beginning to emerge. Although current technology remains limited (the resolution of today's head-mounted displays makes the user legally blind), researchers are actively pursuing applications in space exploration,[94] medical training,[13] and fluid dynamics,[47] among others.

Comments

Given the newness of the field, any discussion of virtual reality for optimization would have to be considered speculative. At present, virtual reality seems to be used mostly for video games. Although people will snicker, one should remember that the largest application of computer graphics in the 1970s was also for video games. Now computer graphics forms an indispensable tool for less frivolous applications.

Recent research appears intriguing. Brooks's research on force-feedback systems for molecular modeling has already been discussed. Feiner and Beshers[88] used a stereoscopic display and a DataGlove to interact with multi-

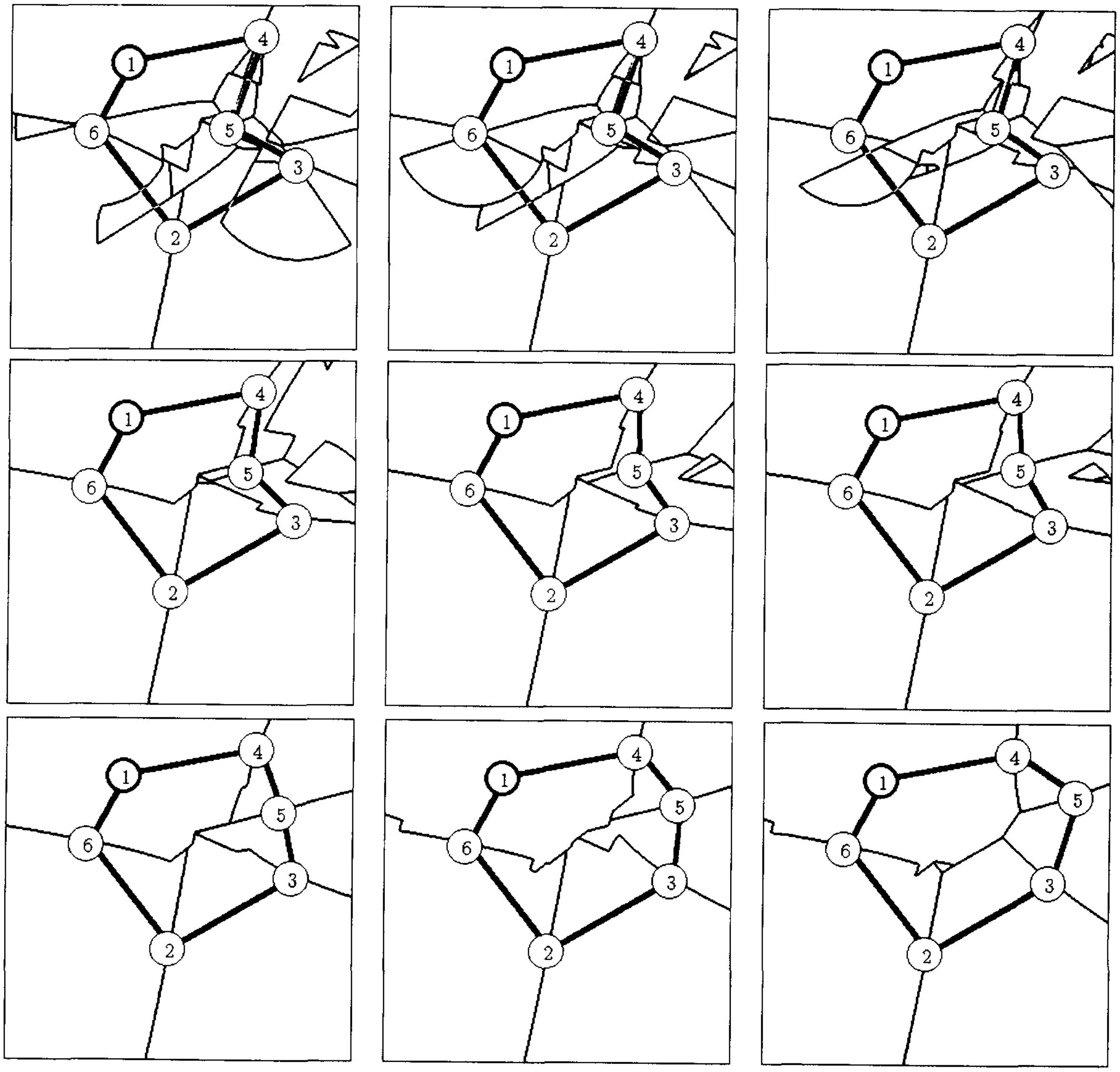


Figure 25. A sequence of images from a movie of the regions for a six-city traveling salesman problem. Each individual image shows the regions for city 1 where the tour calculated by FARTHEST INSERTION remains unchanged. The sequence of images (read from left to right, top to bottom) shows the regions for different locations of city 5 as city 5 moves along a path toward the upper right. Note that as city 5 becomes part of the convex hull of all the cities, the regions become much less complicated.

Published in ORSA Journal on Computing on August 01, 1994 as DOI: 10.1287/ijoc.6.3.221.
This article has not been copyedited or formatted. The final version may differ from this version.

dimensional nonlinear functions. They provide an example of their representation scheme for the standard Black-Sholes options pricing model. Feiner and Besher's proposal involves nested depictions of two-dimensional surfaces where the depth of the nesting depends upon the number of dimensions. This application provides an intriguing example that virtual reality will make it easier for mathematical programmers to view higher dimensional information more easily. In the next section, we discuss other approaches for visualizing multi-dimensional objects.

2.8. Multiple Dimensions

A mathematical program identifies a subset of n-dimensional space that constitutes the set of possible or feasible solutions. Therefore, it seems logical to visualize n-dimensional spaces (see [227] for an exploration of how to visualize four-dimensional space). For linear programming, several authors, for example, [110, 183], have created systems for visualizing three-dimensional polytopes, that is, either three-variable problems or projections of n-dimensional problems into three dimensions.

Mathematicians and statisticians have explored techniques to visualize higher (greater than three) dimensional objects. Banchoff[15, 16] developed graphics and animations illustrating mathematical objects such as a four-dimensional sphere.

Statistics has developed a long tradition of research into visualizing higher dimensional data.[67, 102, 259] Many of the representations are based on projections of multi-dimensional point clouds into two or three dimensions.[19, 20, 52, 145, 259] Several of these projections can, of course, be displayed simultaneously in separate windows. A technique called *brushing*[19, 20] allows users to select points in one view and simultaneously see their positions in all other views. Chernoff[57] and Flury and Riedwyl[95] proposed a novel multi-dimensional representation wherein each data point is encoded as a face, each coordinate of the point encoding a different characteristic of the face such as the size of the nose. Although arguably too cute to be helpful, these representations attempt to exploit our well-developed ability to compare human faces. Inselberg[150] proposed a different type of multi-dimensional plot. Instead of drawing the n coordinate axes perpendicular to one another, they are drawn in parallel. Each data point $(x_1, \ldots, x_n)$ is plotted by locating coordinate x_i along axis i. Then, these locations are connected together. Positive and negative correlations can easily be seen. To be effective, the technique requires that the modeler choose the axes in an appropriate order.

Comments

Most of the attempts to display higher dimensional objects have not even attempted to represent objects having the hundreds, thousands, or even millions of dimensions found in actual mathematical programming problems. One might conclude, therefore, that our inability to visualize n-dimensional space for large n dooms any attempt to visualize complex problems. Displaying multi-dimensional polytopes, although an interesting goal, may be a red herring. Other, more feasible multi-dimensional representations may be useful.

For example, a Gantt chart of a production schedule[65, 104] presents a set of time lines giving the status of resources (say, machines) over time (see Figure 21). Although drawn in two geometric dimensions, this representation encodes several data dimensions including time, machines, operations on machines, status of the operations, and interrelationships among operations. The Gantt chart does not display the entire feasible set of possible schedules as the n-dimensional polytope would, but it still presents useful information.

Another example is a network or graph (as discussed in Section 2.3). For example, Figures 26 and 27 show a graph of the linear programming relaxation for a sequencing problem. The optimal solution should contain no cycles. The LP relaxation, however, contains many cycles (see Figure 26). Users can interactively break the cycle and then generate a new solution that does not contain that cycle.

Similarly, spreadsheets, particularly emerging multi-dimensional spreadsheets such as Lotus Improv,[182] can also be used for multi-dimensional representations. Bertin[31] and Tufte[257, 258] provide articulate discussions, as well as illuminating examples, of multi-dimensional data display in a variety of application contexts. In conclusion, one need not display multi-dimensional polytopes in order to have useful multi-dimensional displays.

3. Integration

In previous sections, we have discussed the use of visualization from the perspective of the different tasks involved, and the different representation formats possible. We now attempt to tie the tasks and representations together. In Section 3.1, we first discuss experimental evidence supporting the use of multiple representations, and then discuss how optimization systems can support multiple representations (Section 3.2). Finally, we discuss how these representations can be built and maintained by multiple users perhaps working in multiple locations at different points in time (Section 3.3).

3.1. Experimental Evidence

Different users require different representations. For example, Orlikowski and Dhar[215] studied expert and novice mathematical programmers and discovered that different groups used different cognitive processes to formulate linear programming models. Exactly which representations will be appropriate for particular types of users and tasks remains poorly understood.

Experimental evidence that examines the effectiveness of different graphical representations has largely been equivocal, however. Field research suggests that problem solvers like graphical representations,[170] but results from formal experimentation have not shown conclusively that graphical representations consistently produce measurably better performance. Most of the studies have explored the effectiveness of different types of presentation graphics such as

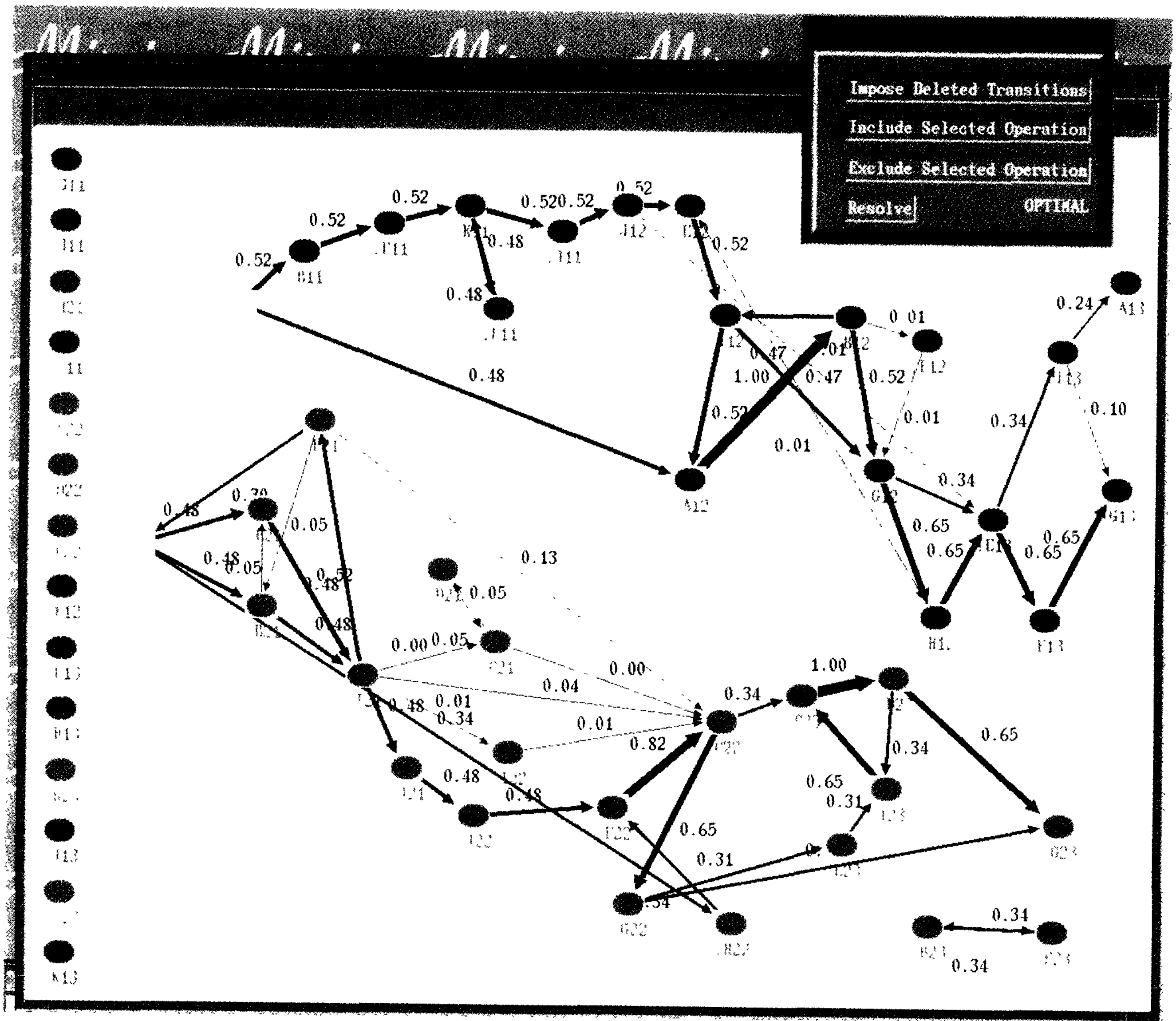


Figure 26. Graph-based representation of the solution to the linear programming relaxation of a sequencing problem. The all-integer solution should contain no cycles. Using the menu at the upper right, users can point to arcs in the graph, and ask that they be forbidden from the solution. Users can then re-solve the model, with the results displayed. Compare to Figure 27. Image courtesy of Chesapeake Decision Sciences.

bar charts versus pie charts versus tabular data. Although most authors now agree that the effectiveness of a particular format depends not only on the format, but also on the users and the tasks at hand, little experimentally based guidance has been developed. In a detailed survey in 1985, DeSanctis[85] stated that "... relatively little is known about the actual utility of graphics as decision aids." Since that survey, several other authors have weighed in with studies that claim to have developed more consistent, applicable guidelines.[29, 153] Mackinlay[185] even attempted to build a system that would select an appropriate representation format automatically.

Guidelines for emerging representation formats such as sound[12, 34, 50] and animation[255] are much less well developed than those for static graphics. Formal experimentation that explores the relative effectiveness of these representation formats is even less developed, although some is emerging.[245] Given the difficulty in producing consistent experimental results even for simple graphics, one might expect that it will be some time before consistent, experimentally verified guidelines are available for the newer representation formats.

To support multiple, simultaneous representations, window-based user interfaces were developed[243] and are now common (Apple Macintosh,[6] Microsoft Windows,[238] and X Window[232]). Any modern interface for mathematical

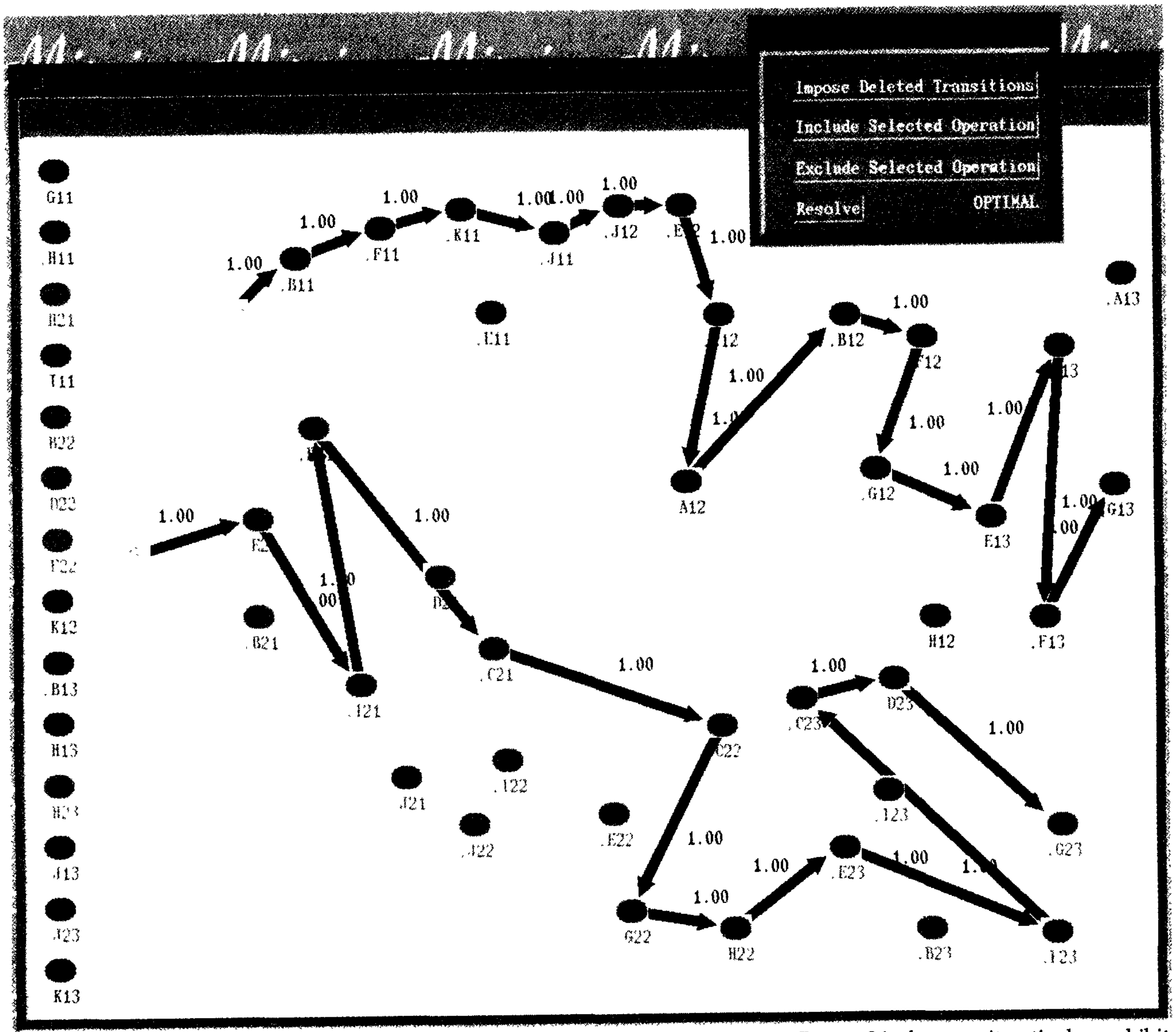


Figure 27. Final solution to the sequencing problem. Starting with the solution in Figure 26, the user iteratively prohibited arcs that produced cycles, and then invoked the optimization algorithm to generate a new solution. Image courtesy of Chesapeake Decision Sciences.

programming will involve a *multiple window user interface.* Sadly, commercial optimization tools that exploit this technology are only now emerging, even though it has been widely available since the mid 1980s. These capabilities provide several opportunities for enhancing the delivery of mathematical programming tools.

In short, no one representation is useful for all tasks and all users.[25, 116, 215] Therefore, we need multiple representations for multiple tasks and users.

3.2. Multiple Representations

If one is to provide multiple representations, two approaches are possible. One approach assumes a base of a single, underlying, formal representation, with alternative representations built on top of the base. A classic example is Structured Modeling.[111] The other approach attempts to link together heterogeneous representations that were originally developed separately, perhaps with vastly different assumptions. We call the former *homogeneous* and the latter *heterogeneous.*

The homogeneous approach provides great economies of scale. Any tool developed, perhaps originally for a specific application, is applicable automatically to all other types of applications. The power of a single representation format has been exploited in widely different settings. Unix, for example, uses text as its single underlying representation. Unix provides extensive facilities to search for patterns of text in files, allowing the text output by one command to be fed (or "piped," in Unix terms) into another command. Relational database systems store not only the data as relations, but also as models of the data, with security information controlling who has access to the data, and the location of the data. Spreadsheets provide another example, since everything represented in a spreadsheet—data, formulas, macros—is stored in the same format.

For optimization (and more general OR models), the homogeneous approach has been advocated by several authors, including [14, 32, 111, 113, 133, 166]. Several implemented, commercial systems[58, 216, 229] also rely on a homogeneous approach. Bhargava and Kimbrough[32] based their approach on the embedded languages approach, which has its roots in formal logic. In Bhargava and Kimbrough's framework, models, data, and information about models are represented in the same way. When a new feature is provided, for example, hypertext navigation, it can be used for the data, the models, and the information about the models. Jones[157, 158, 161] based his approach on attributed graphs and graph-grammars, which are topics of formal language theory.

Given the proliferation of different modeling tools including spreadsheets, symbolic mathematics environments, and word processors, as well as environments specifically targeted to mathematical programming, it seems unlikely that a single environment will completely dominate. Although the homogeneous approach has produced notable successes, even the homogeneous proposals must coexist with other systems. Ultimately, heterogeneous representations must be supported.[105, 112]

Research exploring the heterogeneous approach is just emerging. Derrick and Balci[84] have proposed a heterogeneous environment for simulation. Greenberg and Murphy[133] discuss the different representations required for mathematical programming.

If no single system dominates, then what takes responsibility for linking together heterogeneous representations? The answer is: the computer operating system. All of the different representations must coexist with some operating system anyway. If the operating system manages system resources such as memory, storage, and processor time, it can also provide facilities to promote linking together different representations from different systems.

Features to link together different representations are now being added to operating systems at a rapid pace. For example, interchanging certain types of data such as text, spreadsheet data, and graphics has become increasingly easy. Dynamic Document Exchange (DDE) in Microsoft Windows,[238] Inter-Client Communication Conventions Manager (ICCCM) in X Window,[268] and Publish and Subscribe on the Apple Macintosh[9] allow different representations to be exchanged in real time. For example, data created in a spreadsheet can be inserted into a word processor. When the data in the spreadsheet are changed, the change is reflected in the word processor immediately.

With additional standards such as Object Linking and Embedding (OLE)[195] and Amber,[72] a spreadsheet can be incorporated directly as part of another document, say a presentation. When the user double clicks on the spreadsheet, without leaving the presentation, the user can alter the spreadsheet. Imagine when the boss questions the validity of a model. The presenter need only double click on the spreadsheet in the presentation tool to make the spreadsheet "live" enabling the boss to explore the spreadsheet. Emerging (or existing) object-oriented operating systems such as Pink from Apple and IBM and Cairo from Microsoft[221] as well as NeXTStep from NeXT[210] promise to facilitate linking together a variety of different representations easily.

Linking together multiple, heterogeneous representations requires standard interchange formats. With the development of standards such as DDE, OLE, and ICCCM, as well as multi-media standards such as MIDI,[149] MPEG,[178] QuickTime,[8] and Microsoft Windows Multimedia,[193] it is now possible to exchange animation, video, and sound as easily as text and static graphics. With appropriate interchange standards, perhaps we can build viable mathematical programming environments out of interchangeable components, such as a spreadsheet, a word processor, and a symbolic mathematics system.

Implementing multiple, heterogeneous representations in multiple windows poses several major challenges. If several representations merely represent different views of the same problem, then changes made to one representation should be reflected in the other representations. Current spreadsheets provide such capabilities.

Incorporating such a capability in a multiple window mathematical programming environment remains in its infancy. For example, if an algebraic representation of the problem is displayed in one window, and the instance version is displayed in another window, then when the user deletes a decision variable in the algebraic formulation, the instance window will need to be changed, perhaps radically. An alternative approach would not change the old instance automatically, but would force the user either to ask explicitly for the dependent representations to be updated, or to ask explicitly for new versions of the representations to be created. Kendrick[167] developed a system that linked an algebraic and graphical representation of the transportation problem. Some commercial systems such as AIMMS,[217] MIMI/G,[59] and MPL[191] are beginning to provide such capabilities.

Linking together multiple representations in this fashion is closely related to *constraint-based* programming. Originally proposed by Borning in [38], (see [179, 188, 251] for more recent research) one specifies a set of mathematical relationships or *constraints*, usually mathematical equations. As input values are changed, the set of constraints is solved again. Surprisingly, techniques from mathematical programming have rarely been used to solve the set of constraints. Often, the variables in these equations are tied to properties of graphical objects such as the location or size of an icon. If the value of one or more variables is changed, perhaps through mouse input, the set of mathematical relationships is solved, with the graphics updated as quickly as possible.

This simple idea has been used to help provide sensitivity analysis[66] and to translate a network into a bar chart,[162] and is now seeing application in commercial mathematical programming systems.[59] It could also be used to provide animated sensitivity analysis for representations. By defining constraints across multiple representations and multiple formats, one can provide automatic updating capabilities for linked representations.

The Advanced Visualization System[260] provides perhaps the most advanced commercial implementation of this idea. Essentially, it extends the notion of Unix pipes, which allow for the transmission of text from one Unix command to another, to visualization. Users draw a network of pipes linking data nodes to filters for processing data, and finally to visualization nodes which actually present the information.

For optimization, however, the only common interchange format is the MPS format, which has been showing its age for far too long. MPS can only represent problem instances, not generic structure, and its variable names are limited to eight characters. Many mathematical programming systems provide incredibly rich capabilities for squeezing a variety of information into those eight characters. The row and column names in Figure 16a are restricted to eight characters. The column names begin with the letter "T," because those variables represent transportation activities. The next two characters represent the sources, and the final two characters represent the sinks. For large problems having hundreds of sources and sinks, cryptic code names must be used. Furthermore, MPS can only represent linear programs. Conn, Gould, and Toint[75] have proposed an extended version of the MPS standard that is upwardly compatible with the existing MPS standard. It also allows nonlinear constraints to be specified. It retains for compatibility's sake, however, much of the conventions of MPS, including variable names limited to a maximum of eight characters.

It seems imperative that members of the mathematical programming community join together to develop a more modern standard. The new standard should allow for linear, integer, and nonlinear programming models to be specified. Identifiers should not be limited to eight characters. It should allow the interchange not only of individual problems (instance level), but also problems at the generic level.

More specifically, the new format should contain information about the sets and tables that were used to generate the problem. For example, in MPS, an objective function coefficient for a variable might appear as T1L3L4P8. It might actually represent the transport of product 8 from location 3 to location 4 in time period 1. In other words, this model apparently contains variables that are indexed by time period, product, and location. This type of information should be represented in the interchange standard.

With the proliferation of algebraic languages, it is an appropriate time to begin defining a general interchange standard for these languages. Of course, there may be little commercial incentive to pursue a new standard. After all, a standard interchange format would make it easier for users to migrate from one language to another, which is not necessarily in the best interest of the commercial vendors. However, the success of defacto standards such as Unix, DOS, and Microsoft Windows points to the demand for standards. Upward compatibility with MPS should not be a difficult problem, since a simple conversion program from the old format to the new format should suffice. Without a new standard, the vision of multiple, linked representations from multiple vendors cannot come to pass.

3.3. Multiple Users

Most problem-solving projects are not conducted by one person, but by groups of people. Clearly, groups of people have been using computers for quite some time. A field study of spreadsheet development,[208] for example, found that most spreadsheets were developed in a collaborative process involving several individuals.

The field of *computer-supported cooperative work* (CSCW)[134, 135] attempts to support groups of people in their activities. Many different flavors of CSCW are currently emerging. People might work together simultaneously or *synchronously* on a project; for example, they might edit the same text at the same time.[137] In contrast, people may work on the project at different times, that is, *asynchronously*. They could be in the same room or dispersed geographically. In teleconferencing, meetings are conducted using video to link people from far away (see [35] for an extreme use of teleconferencing).

An increasingly common form of CSCW has been dubbed a *Group Decision Support System* (GDSS).[83, 86, 154, 214] At the University of Arizona,[214] meeting participants work together in a room using a network of computers. Guided by a facilitator, the participants use a variety of tools provided by the GDSS to attack a problem. Tools for brainstorming ideas, organizing ideas into categories, and voting to settle points are some of the many techniques provided.

Comments

Ideas from CSCW have not found their way into the mathematical programming community. This seems somewhat surprising since mathematical programming projects often involve coordinating not only large models and large databases, but also large numbers of people. One could imagine a system for editing mathematical programming formulations that allows multiple users to perform editing simultaneously.

Spreadsheets, as well as some mathematical programming systems,[189, 216] now provide some capabilities for managing multiple versions of models and data. Holsapple, Park, and Whinston[141] proposed an architecture for decision support that could possibly support multiple, simultaneous users. We could also imagine an asynchronous system for mathematical programming formulations that tracks the changes made by each user, much like source code control systems that support computer program development.

4. Research in Visualization and Optimization

The goals of visualization and OR are quite similar: to provide insight into complex problems. OR uses mathematical algorithms whereas visualization studies representations. However, the techniques that OR has developed rest on the solid foundation of mathematics and computer science. We can usually prove that a particular solution to a mathematical program is optimal, that an algorithm will converge, or that the solution, when not optimal, is at least close to the optimal solution. Mathematical proofs of the effects of visualization seem less likely.

At best, we can conduct carefully designed experiments to establish the quality of different forms of visualization. That experimental style of research does not fit into the mathematical traditions of OR. Although reference disciplines, including cognitive psychology, linguistics, human-computer interaction, and formal logic can contribute well-developed theories, many of these disciplines have not traditionally been considered part of the mainstream of OR.

As discussed throughout the previous sections, representation forms a key component of the life cycle of mathematical programming projects. From conceptual models, through formulations, databases, data structures to facilitate solution, outputs of algorithms, debugging information, and final presentation, an OR project is really just a process of transforming one representation into another.

Given the variety of representations needed, the difficulty in constructing them, and most importantly, the need to tie them together, research in this area is essential to the growth of OR. Research on visualization in optimization should explore the following directions.

Empirical Evidence

Several authors have presented articulate, detailed discussions of how to represent complicated datasets and problems. Bertin[31] developed a taxonomy of different graphic representations based on their ability to encode different types of data. Tufte's books[257, 258] developed several recommendations for creating effective, parsimonious graphical representations of data. In Section 3.3 we discussed the lack of experimental evidence concerning the effectiveness of different representations. With new hardware and software capabilities, the constraints on representation capabilities are relaxing—this could lead to better ways of visualizing problems. Although such research in the past has been fraught with inconsistencies, the need to understand which representations work best for which tasks and users remains. Perhaps researchers have considered problems at too high a level when they try to answer a question such as: Do problems represented as bar charts lead to better understanding than the same problems represented as tables? Experimental research that has asked more fundamental questions (such as which encoding schemes are perceived more accurately) seems to have enjoyed greater success. Better experimental evidence, specifically targeted to optimization, is essential.

Formalizations

Several fields, such as formal languages and formal logic, have concentrated on developing theories of representation. These have formed the basis of several homogeneous (as defined in Section 3) representations used for modeling environments. Formal language theory has been used to help develop and describe computer programming languages. Recently, these theories have been applied in the realm of mathematical programming. Syntax-directed editors are one result that have been developed for mathematical programming languages. Kimbrough et al.,[168, 169] Krishnan,[172] Hong, Mannino, and Greenberg,[142] and Muhanna and Pick[199] have relied on formal logic for their approaches. Graph-grammars, a formal theory for visual languages, have been applied to describe the variety of graph-based models that are employed in mathematical programming.[156–158, 161–164] Baldwin[14] and Greenberg and Murphy[133] have proposed the use of a single, central representation format to support a wide variety of different representations or views of the underlying problem. Geoffrion,[111, 113, 114] in developing Structured Modeling, has attempted to provide a single, formal framework of systems supporting multiple representations. In fact, Geoffrion proposes a variety of useful representations based on Structured Modeling. Formalization brings legitimacy to the field, as well as practical tools useful in modeling.

Integration

Many proposals for different representations for mathematical programming exist. The authors of each of these proposals (including the author of this paper) frequently seem to view their representation proposal as the "best," in some way. Yet, given the variety of representations, it does not seem likely that one representation will come to dominate. Rather, different representations will coexist. Mathematical programming systems need to provide a variety of representations, for input as well as output, in multiple formats. Whether this should be accomplished by using a single underlying representation format, or by combining heterogeneous systems, remains an open question. However, emerging operating system technology is making the exchange of information among different systems far easier, so the heterogeneous approach is becoming more possible. Perhaps all that is necessary are a few components such as spreadsheets, databases, algebraic modeling languages, hypertext, and solution algorithms linked together by facilities provided by computer operating systems.

New Visualization Techniques

Newly available visualization techniques, such as animation, sound, computer-supported cooperative work, multimedia,[35, 207] and haptic displays, have yet to be applied widely in mathematical programming. Exploratory research that develops novel and interesting uses of these techniques would be of interest. Such research could bring out the "lunatic fringe," but out of that morass could emerge some useful ideas. Of course, those ideas would need to be carefully tested. Researchers interested in pursuing such a direction would be well advised to consult the work of Brooks.[42, 43] Brooks provided an excellent model for research that combines such exploratory investigation with solid experimental justification.

5. The Future of Visualization and Optimization

In previous sections we have attempted to provide a detailed discussion of existing research and practice applicable to optimization. In this section, we predict the future of visualization and optimization, at least over the next five years.

1. All computer software, including software incorporating OR techniques, will use multi-window, mouse-driven user interfaces. Those systems that do not provide such capabilities will not survive. As Schultz and Pulleyblank[233] said in a discussion of trends in optimization, "...people now expect much higher quality user interfaces and presentations of results."
2. Spreadsheets will provide increasingly powerful modeling capabilities, especially in the area of multi-dimensional indexing. Spreadsheets will therefore become the delivery vehicle of choice for optimization, at the expense of algebraic modeling languages.
3. A single algebraic modeling language standard will emerge, either through market forces, or through cooperation on the part of members of the mathematical programming community. That standard might perhaps be based on multi-dimensional spreadsheets.
4. A standard paradigm or system to support drawable models or programs will emerge, perhaps based on graphs or networks. Visual Basic[2] and Hypercard[7] provide examples of some possible directions in this area.
5. Animation will become as widespread as static graphics are today. Applications will be in algorithm animation and animated sensitivity analysis.
6. Hypertext (really hypermedia) will be a standard tool for navigating through spreadsheets, modeling languages, word processors, and any other complex system.
7. Mathematical programming systems will support *multiple* representations including algebraic formulations, block-structured, relational databases, matrix images, graph-based and other graphical representations, spreadsheet, presentation graphics, natural language, hypertext, and animation.
8. More and more optimization systems will be composed of modular building blocks linked together by facilities provided by the computer operating system.
9. Virtual reality will move primarily from the video game stage to see more common application by optimization systems. A promising application to molecular biology was described by Brooks et al.[43]
10. Multiple user modeling systems will emerge. The basic facilities for allowing multiple users to work together on a project, simultaneously or not, geographically separated or not, will be provided by the operating system. People throughout the world will be able to exchange problem instances and generic models quickly and easily.
11. Research in visualization and OR will continue to be dangerous for untenured faculty. Hence, as Hirshfeld[138] predicted, most of the interesting developments in visualization and optimization will come from industry and not academia.
12. Millions more will make use of optimization, often to great benefit. Even so, many of these new users will not know or care about the OR heritage of the technology that they are using. If true, this prediction implies that the prominence of optimization as a field will continue to fade, as the techniques it nurtured, ironically, see greater and greater use.

In conclusion, OR, not just optimization, has made great strides in developing techniques to analyze complicated problems. These sophisticated mathematical and computer-based techniques have yielded large benefits. Problem solving in general, however, involves more than building an algorithm. It consists of a process of transforming and understanding various representations or visualizations of the problem at hand. The solution algorithm shares at least equal importance with the problem-solving process and the associated representations and visualizations. If visualization has such importance, then it needs to be studied, researched, and improved.

Unlike mathematics and computer science, however, visualization does not yet constitute a mature, refined science. It remains difficult to prove or predict the relative quality of a particular visualization for specific tasks or users. Yet, techniques are emerging, based in the fields of cognitive psychology, formal logic, and formal languages that are theoretically grounded and have practical application.

If we improve our ability to visualize our problems as much as we have improved the ability of algorithms to "solve" problems, then we should be able to model and solve our problems much more effectively and efficiently. At the very least, we should see the wider use of the sophisticated techniques that OR has spent a considerable amount of time developing.

Regardless of the success of academic researchers in providing provable results about the effectiveness of visualization, the market will continue to produce new and better visualization techniques for building, solving, and understanding problems that can be analyzed using optimization.

Acknowledgments

I would like to thank Osman Balci, Ed Wasil, and Bruce Golden for inviting me to prepare this article. I would most especially like to thank the five anonymous referees for their detailed, thorough, and extremely helpful comments that greatly improved the article.

References

1. ADOBE SYSTEMS, 1985. *Postscript Language Reference Manual*, Addison-Wesley, Reading, MA.
2. P.G. AITKEN, 1993. *Visual Basic for MS-DOS*, Microsoft Press, Redmond, WA.
3. F.L. ALVARADO, 1990. Manipulation and Visualization of Sparse Matrices, *ORSA Journal on Computing* 2:2, 186–207.
4. A.A. ANGEHRN, 1991. Modeling by Example: A Link Between Users, Models and Methods in DSS, *European Journal of Operational Research 55:3*, 296–308.
5. A.A. ANGEHRN and H.-J. LÜTHI, 1990. Intelligent Decision Support Systems: A Visual Interactive Approach, *Interfaces 20:6*, 17–28.

6. Apple Computer, 1985. *Inside Macintosh*, Volumes 1, 2, and 3, Addison-Wesley, Reading, MA.

7. Apple Computer, 1990. *HyperCard Reference*, Apple Computer, Cupertino, CA.

8. Apple Computer, 1991. *QuickTime Developer's Kit, Version 1.0*, Apple Computer, Cupertino, CA.

9. Apple Computer, 1991. *Inside Macintosh*, Volume 6, Addison-Wesley, Reading, MA.

10. Association for Computing Machinery, 1988. *Hypertext on Hypertext* (Software), ACM Press: Database and Electronic Products Series, New York.

11. A. Babin, M. Florian, M. James-Lefebvre and H. Spiess, 1982. EMME/2: An Interactive Graphic Method for Road and Transit Planning. Publication No. 204, Centre de Recherche sur les Transports, Université de Montréal, Montréal, Canada.

12. R.M. Baecker and W.A.S. Buxton, 1987. *Readings in Human-Computer Interaction*, Morgan Kaufmann, Los Altos, CA.

13. M. Bajura, H. Fuchs and R. Ohbuchi, 1992. Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient, *Computer Graphics 26:2*, 203–210.

14. D. Baldwin, 1989. Principles of Design for a Multiple Viewpoint Problem Formulation Support System, Unpublished Ph.D. Dissertation, School of Business Administration, Texas Tech University, Lubbock, TX.

15. T.F. Banchoff, 1986. Visualizing Two-Dimensional Phenomena in Four-Dimensional Space: A Computer Graphics Approach, in *Statistical Image Processing and Graphics*, J. Wegman and D.J. DePriest (eds.), Marcel Dekker, New York.

16. T.F Banchoff, 1990. *Beyond the Third Dimension: Geometry, Computer Graphics and Higher Dimensions*, W.H. Freeman, New York.

17. R.S. Barr and B.L. Hickman, 1993. Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions, *ORSA Journal on Computing 5:1*, 2–18.

18. J.J. Bartholdi III and L.K. Platzman, 1988. Heuristics Based on Spacefilling Curves for Combinatorial Problems in Euclidean Space, *Management Science 34:3*, 291–305.

19. R.A. Becker and W.S. Cleveland, 1987. Brushing Scatterplots, *Technometrics 29:2*, 127–142.

20. R.A. Becker, W.S. Cleveland and A.R. Wilks, 1987. Dynamic Graphics for Data Analysis, *Statistical Science 2:4*, 355–395.

21. P.C. Bell, 1985. Visual Interactive Modeling as an OR Technique, *Interfaces 15:4*, 26–33.

22. P.C. Bell, 1985. Visual Interactive Modeling in Operational Research: Successes and Opportunities, *J. Opnl. Res. Soc. 36*, 975–982.

23. P.C. Bell, 1986. Visual Interactive Modeling in 1986, pp. 1–12 in *Recent Developments in Operational Research*, V. Belton and R. O'Keefe (eds.), Pergamon Press, Oxford, UK.

24. P.C. Bell, 1992. Visual Aid: Interactive Graphics Come of Age in the OR/MS Community, *OR/MS Today 19:4*, 24–27.

25. P.C. Bell and P.Y.K. Chau, 1992. An Empirical Assessment of Three Types of Simulation Models Used in Developing Decision Support Systems, Technical Report, Western Business School, University of Western Ontario, London, Ontario, Canada.

26. P.C. Bell and R.M. O'Keefe, 1987. Visual Interactive Simulation—History, Recent Developments, and Major Issues, *Simulation 49:3*, 109–116.

27. W.J. Bell, L.M. Dalberto, M.L. Fisher, A.J. Greenfield, R. Jaikumar, P. Kedia, R.G. Mack and P.J. Prutzman, 1983. Improving the Distribution of Industrial Gases with an On-line Computerized Routing and Scheduling Optimizer, *Interfaces 13:6*, 4–23.

28. V. Belton, M. Elder and D. Meldrum, 1993. Using VILP, Technical Report, Strathclyde University, Glasgow, UK.

29. I. Benbasat, A.S. Dexter and P. Todd, 1986. An Experimental Program Investigating Color-Enhanced and Graphical Information Presentation: An Integration of the Findings, *Communications of the ACM 29:11*, 1094–1105.

30. J L. Bentley and B.W. Kernighan, 1987. A System for Algorithm Animation: Tutorial and User Manual, Computer Science Technical Report No. 132, AT & T Bell Laboratories, Murray Hill, NJ.

31. J. Bertin, 1983. *Semiology of Graphics: Diagrams, Networks, Maps*, W.J. Berg (trans.), University of Wisconsin Press, Madison, WI.

32. H.K. Bhargava and S.O. Kimbrough, 1990. On Embedded Languages for Model Management, pp. 443–452 in *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences*, Jay F. Nunamaker (ed.), IEEE Computer Society Press, Los Alamitos, CA.

33 H.K. Bhargava, R. Krishnan and S. Mukherjee, 1992. On the Integration of Data and Mathematical Modeling Languages, *Annals of OR 38*, 69–95.

34. M.M. Blattner, D.A. Sumikawa and R.M. Greenberg, 1989. Earcons and Icons: Their Structure and Common Design Principles, *Human-Computer Interaction 4:1*, 11–44.

35. S.A. Bly, S.R. Harrison and S Irwin, 1993. MediaSpaces: Bringing People Together in a Video, Audio, and Computing Environment, *Communications of the ACM 36:1*, 28–47.

36. S. Bodily, 1986. Spreadsheet Modeling as a Stepping Stone, *Interfaces 16:5*, 34–52.

37. L. Bodin, G. Fagan and L. Levy, 1993. Vehicle Routing and Scheduling Problems over Street Networks, Technical Report, College of Business and Management, University of Maryland, College Park, MD.

38. A. Borning, 1981. ThingLab—A Constraint-Oriented Simulation Laboratory, *ACM Trans. on Programming Languages and Systems 6:4*, 353–387.

39. S.C. Boyd, W R. Pulleyblank and G. Cornuejols, 1987. Travel—An Interactive Traveling Salesman Problem Package for the IBM Personal Computer, *OR Letters 6:3*, 141–143.

40. S.D. Brady, R.E. Rosenthal and D. Young, 1984. Interactive Graphical Minimax Location of Multiple Facilities with General Constraints, *AIIE Transactions 15:3*, 242–254.

41. A. Brooke, D. Kendrick and A. Meeraus, 1992. *GAMS: A User's Guide, Release 2.25*, The Scientific Press, South San Francisco, CA.

42. F. Brooks, 1988. Grasping Reality Through Illusion: Interactive Graphics Serving Science, pp. 1–11 in *Proceedings of the ACM SIGCHI Human Factors in Computer Systems Conference*, ACM Press, New York.

43. F.P. Brooks, M. Ouh-Young, J.J. Batter and P.J. Kilpatrick, 1990. Project GROPE—Haptic Displays for Scientific Visualization, *Computer Graphics 24:4*, 177–185.

44. M.H. Brown, 1988. *Algorithm Animation*, MIT Press, Cambridge, MA.

45. M.H. Brown, 1992. Zeus: A System for Algorithm Animation and Multi-View Editing, Digital Systems Research Center Technical Report 75, Digital Equipment Corporation, Palo Alto, CA.

46. M.H. Brown and J. Hershberger, 1991. Color and Sound in Algorithm Animation, Technical Report 76a, Digital Systems Research Center, Digital Equipment Corporation, Palo Alto, CA.

47. S. Bryson and C Levit, 1992. Virtual Wind Tunnel: An

Environment for the Exploration of Three Dimensional Unsteady Flows, *Computer Graphics and Applications 12:4*, 25–34.

48. J.T. Buchanan and K.I.M. McKinnon, 1987. An Animated Interactive Modeling System for Decision Support, pp. 111–118 in *Operational Research '87*, Elsevier Science Publishers, Amsterdam, The Netherlands.
49. V. Bush, 1945. As We May Think, pp. 101–108 in *The Atlantic Monthly*, July.
50. W. Buxton, S.A. Bly, S.P. Frysinger, D. Lunney, D.L. Mansur, J.J. Mezrich and R.C. Morrison, 1985. Communications with Sound, pp. 115–119 in *Proceedings of the ACM SIGCHI Human Factors in Computing Systems Conference*, L. Borman and B. Curtis (eds.), ACM Press, New York.
51. S. Card, T.P. Moran and A. Newell, 1983. *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ.
52. D.B. Carr, R.J. Littlefield, W.L. Nicholson and J.S. Littlefield, 1987. Scatterplot Matrix Techniques for Large *N*, *Journal of the American Statistical Association 82:398*, 424–436.
53. S.K. Chang (ed.), 1990. *Principles of Visual Language Systems*, Prentice-Hall, Old Tappan, NJ.
54. S.K. Chang, T. Ichikawa and P.A. Ligomenides, 1986. *Visual Languages*, Plenum Press, New York.
55. B.W. Char, 1991. *Maple V Language Reference Manual*, Springer-Verlag, New York.
56. P. Chen, 1976. The Entity-Relationship Model: Toward a Unified View of Data, *ACM Transactions on Database Systems 1:1*, 9–36.
57. H. Chernoff, 1971. The Use of Faces to Represent Points in *k*-Dimensional Space Graphically, *Journal of the American Statistical Association 68*, 361–368.
58. Chesapeake Decision Sciences, 1993. *MIMI: Manager for Interactive Modeling Interfaces: User's Manual*, New Providence, NJ.
59. Chesapeake Decision Sciences, 1993. *MIMI/G User's Manual*, New Providence, NJ.
60. Chesapeake Decision Sciences, 1993. *MIMI/LP User's Manual*, New Providence, NJ.
61. J.W. Chinneck, 1990. Localizing and Diagnosing Infeasibilities in Networks, Technical Report SCE-90-14, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada.
62. J.W. Chinneck and E.W. Dravnieks, 1989. Locating Minimal Infeasible Constraint Sets in Linear Programs, *ORSA Journal on Computing 3:2*, 157–168.
63. J. Choobineh, 1991. A Diagramming Technique for Representation of Linear Programming Models, *Omega 19:1*, 43–51.
64. J. Choobineh, 1991. SQLMP: A Data Sublanguage for Representation and Formulation of Linear Mathematical Models, *ORSA Journal on Computing 3:4*, 358–375.
65. W. Clark, 1957. *The Gantt Chart*, third edition, Sir Isaac Pitman and Sons, London.
66. E. Clemons and A. Greenfield, 1985. The SAGE System Architecture: A System for the Rapid Development of Graphics Interfaces for Decision Support, *IEEE Computer Graphics and Applications 5:11*, 38–50.
67. W.S. Cleveland, 1987. Research in Statistical Graphics, *Journal of the American Statistical Association 82:398*, 419–423.
68. W.S. Cleveland and R. McGill, 1985. Graphical Perception and Graphical Methods for Analyzing Scientific Data, *Science 229*, 828–833.
69. W.S. Cleveland and R. McGill, 1984. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods, *Journal of the American Statistical Association 79:387*, 531–554.
70. E.F. Codd, 1970. A Relational Model for Large Shared Data Banks, *Communications of the ACM 13*, 370–387.
71. M.F. Cohen, 1992. Interactive Spacetime Control for Animation, *Computer Graphics 26:2*, 293–302.
72. R. Cohen and H. Norr, 1993. Amber: Apple's Answer to OLE, *MacWEEK 7:20*, 1ff.
73. G. Collaud and J. Pasquier-Boltuck, 1994. gLPS: A Graphical Tool for the Definition and Manipulation of Linear Problems, *European Journal of Operations Research*, to appear.
74. J. Conklin, 1987. Hypertext: An Introduction and Survey, *IEEE Computer 20:9*, 17–41.
75. A.R. Conn, N. Gould and P.L. Toint, 1994. *LANCELOT: A FORTRAN Package for Large-Scale NonLinear Optimization*, Springer-Verlag, Berlin, to appear.
76. R.W. Conway, W.L. Maxwell and S.L. Worona, 1986. *User's Guide to XCELL Factory Modeling System*, The Scientific Press, Palo Alto, CA.
77. F.H. Cullen, J.J. Jarvis and H.D. Ratliff, 1981. Set Partitioning Based Heuristics for Interactive Routing, *Networks 11*, 125–143.
78. G.B. Dantzig, 1963. *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ.
79. G. Dantzig, R. Fulkerson and S. Johnson, 1954. Solution of a Large-Scale Traveling-Salesman Problem, *Operations Research 2*, 393–410.
80. M. Dao, M. Habib, J.P. Richard and D. Tallot, 1986. Cabri, an Interactive System for Graph Manipulation, pp. 58–67 in *Graph-Theoretic Concepts in Computer Science*, G. Tinhofer and G. Schmidt (eds.), Springer-Verlag, Berlin.
81. C.J. Date, 1986. An Introduction to Database Systems, 4th ed., Addison-Wesley, Reading, MA.
82. N. Dean, M. Mevenkamp and C.L. Monma, 1992. NETPAD: An Interactive Graphics System for Network Modeling and Optimization, pp. 231–243 in *Computer Science and Operations Research: New Developments in their Interfaces*, O. Balci, R. Sharda and S.A. Zenios (eds.), Pergamon Press, Oxford, UK.
83. A.R. Dennis, J.F. George, L.M. Jessup, J. Nunamaker, Jr. and D.R. Vogel, 1988. Information Technology to Support Electronic Meetings. *MIS Quarterly 12*, 591–624.
84. E.J. Derrick and O. Balci, 1992. DOMINO: A Mutifaceted Conceptual Framework for Visual Simulation Modeling, Technical Report TR-92-43, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA.
85. G. DeSanctis, 1984. Computer Graphics as Decision Aids: Directions for Research, *Decision Sciences 15*, 463–487.
86. G.S. DeSanctis and R.B. Gallupe, 1987. A Foundation for the Study of Group Decision Support Systems, *Management Science 33*, 589–609.
87. D.R. Dolk, 1990. A Generalized Model Management System for Mathematical Programming, *ACM Transactions on Mathematical Software 12*, 92–126.
88. S. Feiner and C. Beshers, 1990. Worlds within Worlds: Metaphors for Exploring *n*-Dimensional Virtual Worlds, pp. 76–83 in *Proc. UIST '90 (ACM Symposium on User Interface Software and Technology)*, Snowbird, UT.
89. S. Feiner and A. Shamash, 1991. Hybrid User Interfaces: Breeding Virtually Bigger Interfaces for Physically Smaller Computers, pp. 9–17 in *Proc. UIST '91 (ACM Symposium on User Interface Software and Technology)*, Hilton Head, SC.
90. A.V. Fiacco, 1983. *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*, Academic Press, New York.
91. A.V. Fiacco (ed.), 1990. *Annals of OR 27:1–4*, Special Issue on Optimization with Data Perturbations.

92. M.L. Fisher, 1986. Interactive Optimization, *Annals of Operations Research 5*, 541–556.

93. M.L. Fisher, A. Greenfield and R. Jaikumar, 1982. VERGIN: A Decision Support System for Vehicle Scheduling, Working Paper 82-06-02, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, PA.

94. S.S. Fisher, M. McGreevy, J. Humphries and W. Robinett, 1986. Virtual Environment Display System, *ACM 1986 Workshop in Interactive 3D Graphics*, Chapel Hill, NC.

95. B. Flury and H. Riedwyl, 1981. Graphical Representation of Mutivariate Data by Means of Asymmetrical Faces. *Journal of the American Statistical Association 76:376*, 757–765.

96. L.R. Ford and D.R. Fulkerson, 1962. *Flows in Networks*, Princeton University Press, Princeton, NJ.

97. R. Fourer, 1983. Modeling Languages versus Matrix Generators for Linear Programming. *ACM Transactions on Mathematical Software 9*, 143–183.

98. R. Fourer and D.M. Gay, 1991. Expressing Special Structures in an Algebraic Modeling Language, Technical Report, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL.

99. R. Fourer, D.M. Gay and B. W. Kernighan, 1990. A Mathematical Programming Language, *Management Science 36:5*, 519–554.

100. R. Fourer, D.M. Gay and B.W. Kernighan, 1993. *AMPL: A Modeling Language for Mathematical Programming*, The Scientific Press, South San Francisco, CA.

101. P. Freiberger and M. Swaine, 1984. *Fire in the Valley*, Osborne/McGraw-Hill, Berkeley, CA.

102. H.G. Funkhouser, 1937. Historical Development of the Graphical Representation of Statistical Data, *Osiris 3*, 269–404.

103. T. Gal, 1979. *Postoptimal Analyses, Parametric Programming, and Related Topics*, McGraw-Hill, New York.

104. H.L. Gantt, 1919. *Organizing for Work*, Harcourt, Brace and Howe, New York.

105. D. Garlan, 1987. Views for Tools in Integrated Environments, Unpublished Ph.D. Dissertation, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

106. M. Garman, 1970. Solving Combinatorial Decision Problems via Interactive Computer Graphics with Applications to Job-Shop Scheduling, Unpublished Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA.

107. S.I. Gass, 1990. Model World: Danger, Beware the User as Modeler, *Interfaces 20:3*, 60–64.

108. W.W. Gaver, 1989. The SonicFinder: An Interface that Uses Auditory Icons, *Human-Computer Interaction 4:1*, 67–94.

109. W.W. Gaver, R.B. Smith and T. O'Shea, 1991. Effective Sounds in Complex Systems: The Arkola Simulation, pp. 85–90 in *Reaching Through Technology: Human Factors in Computing Systems; ACM SIGCHI '91 Conference Proceedings*, S.P. Robertson, G.M. Olson and J.S. Olson (eds.), ACM Press, New York.

110. D.M. Gay, 1987. Pictures of Karmarkar's Linear Programming Algorithm, Computer Science Technical Report No. 136, AT & T Bell Laboratories, Murray Hill, NJ.

111. A.M. Geoffrion, 1987. An Introduction to Structured Modeling. *Management Science 33:5*, 547–588.

112. A.M. Geoffrion, 1989. Integrated Modeling Systems, *Computer Science in Economics and Management 2*, 3–15.

113. A.M. Geoffrion, 1989. The Formal Aspects of Structured Modeling, *Operations Research 37:1*, 30–51.

114. A.M. Geoffrion, 1992. The SML Language for Structured Modeling, *Operations Research 40:1*, 38–75.

115. A.M. Geoffrion, 1992. Indexing in Modeling Languages for Mathematical Programming, *Management Science 38:3*, 325–344.

116. J. Ghani, 1981. The Effects of Information Representation and Modification on Decision Performance, Unpublished Ph.D. Dissertation, Decision Sciences Working Paper 81-08-01, The Wharton School, The University of Pennsylvania, Philadelphia, PA.

117. J. Gleick, 1988. *Chaos: Making a New Science*, Viking, New York.

118. F. Glover, D. Klingman and C. McMillan, 1977. The NETFORM Concept: A More Effective Model Form and Solution Procedure for Large Scale Nonlinear Problems, National Technical Information Service, US Department of Commerce, Springfield, VA, 25 pp.

119. F. Glover, D. Klingman and N. Phillips, 1992. *Network Models in Optimization and Their Applications in Practice*, Wiley-Interscience, New York.

120. H. Göttler, 1983. Attributed Graph-Grammars for Graphics, pp. 130–142 in *Graph-Grammars and their Application to Computer Science*, H. Ehrig, M. Nagl and G. Rozenberg (eds.), Springer-Verlag, Berlin.

121. H. Göttler, 1987. Graph-Grammars and Diagram Editing, pp. 216–231 in *Graph-Grammars and their Application to Computer Science*, H. Ehrig, M. Nagl, G. Rozenberg and A. Rosenfeld (eds.), Springer-Verlag, Berlin.

122. H. Göttler, 1989. Graph Grammars, a New Paradigm for Implementing Visual Languages, pp. 152–166 in *Rewriting Techniques and Applications*, N. Dershowitz (ed.), Springer-Verlag, Berlin.

123. P.B. Gove (ed.), 1981. *Webster's Third New International Dictionary of the English Language*, Merriam-Webster, Springfield, MA.

124. J.W. Grant and S.A. Weiner, 1986. Factors to Consider in Choosing a Graphically Animated Simulation System, *Industrial Engineering 18*, 37ff.

125. H.J. Greenberg, 1987. A Natural Language Discourse Model to Explain Linear Programs, *Decision Support Systems 33*, 333–342.

126. H.J. Greenberg, 1987. ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models, *OR Letters 6.5*, 249–259.

127. H.J. Greenberg, 1988. A Summary of Remaining Goals and Strategies for the Development of an Intelligent Mathematical Programming System, Technical Report, Department of Mathematics, University of Colorado, Denver, CO.

128. H.J. Greenberg, 1990. Neural Networks and Heuristic Search, *Annals of Mathematics and Artificial Intelligence 1*, 75–95.

129. H.J. Greenberg, 1992. MODLER: Modeling by Object-Driven Linear Elemental Relations, *Annals of Operations Research 38*, 239–280.

130. H.J. Greenberg, 1993. *A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE*, Kluwer, Boston, MA.

131. H.J. Greenberg, 1993. *Modeling by Object-Driven Linear Elemental Relations: A User's Guide for MODLER*, Kluwer, Boston, MA.

132. H.J. Greenberg and F.H. Murphy, 1992. A Comparison of Mathematical Programming Modeling Systems, *Annals of Operations Research 38*, 177–238.

133. H.J. Greenberg and F.H. Murphy, 1994. Views of Mathematical Programming Models and their Instances, *Decision Support Systems*, to appear.

134. S. Greenberg (ed.), 1991. *Computer-supported Cooperative Work and Groupware*, Academic Press, London.

135. I. Greif, 1988. *Computer-supported Cooperative Work: A Book of Readings*, Morgan Kaufmann, San Mateo, CA.

136. M. Grötschel and M.W. Padberg, 1985. Polyhedral Theory, pp. 251–305 in *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (eds.), Wiley, Chichester, UK.

137. R.D. Hill, T. Brinck, J.F. Patterson, S.L. Rohall and W.T. Wilner, 1993. The Rendezvous Language and Architecture, *Communications of the Association for Computing Machinery 36:1*, 62–67.

138. D.S. Hirshfeld, 1990. Some Thoughts on Math Programming Practice in the '90s. *Interfaces 20:4*, 158–165.

139. F.L. Hitchcock, 1941. Distribution of a Product from Several Sources to Numerous Localities, *Journal of Mathematical Physics 20*, 224–230.

140. C.W. Holsapple, S. Park and A.B. Whinston, 1989. Generating Structure Editor Interfaces for OR Procedures. Technical Report, Center for Robotics and Manufacturing Systems, University of Kentucky, Lexington, KY.

141. C.W. Holsapple, S. Park and A.B. Whinston, 1991. Framework for DSS Interface Development. Technical Report, Center for Robotics and Manufacturing Systems, University of Kentucky, Lexington, KY.

142. S.N. Hong, M.V. Mannino and B. Greenberg, 1991. Measurement Theoretic Representation of Large, Diverse Model Bases: The Unified Modeling Language: $\mathscr{L}_U$, Technical Report, Department of Management Science and Information Systems, The University of Texas at Austin, Austin, TX.

143. J.J. Hopfield and D.W. Tank, 1986. Computing with Neural Circuits: A Model, *Science 233*, 625–633.

144. J.J. Hopfield and D.W. Tank, 1985. Neural Computation of Decisions in Optimization Problems, *Biological Cybernetics 52*, 141–152.

145. P.J. Huber, 1987. Experiences with Three-Dimensional Scatterplots, *Journal of the American Statistical Association 82:398*, 448–453.

146. R.D. Hurrion, 1986. Visual Interactive Modeling, *European Journal of Operational Research 23*, 281–287.

147. R.D. Hurrion and R.J.R. Secker, 1978. Visual Interactive Simulation: An Aid to Decision Making, *Omega 6*, 419–426.

148. E.L. Hutchins, J.D. Hollan and D.A. Norman, 1986. Direct Manipulation Interfaces, pp. 87–124 in *User Centered System Design: New Perspectives on Human-Computer Interaction*, D.A. Norman and S.W. Draper (eds.), Lawrence Erlbaum, Hillsdale, NJ.

149. IMA, 1983. *MIDI Musical Instrument Digital Interface Specification 1.0*, IMA, North Hollywood, CA.

150. A. Inselberg, 1985. The Plane with Parallel Coordinates, *The Visual Computer 1*, 69–91.

151. Insight International Ltd., 1992. *Genetik Software Description*, Woodstock, Oxon, UK.

152. Insight Logistics Ltd., 1993. *INCEPTA Tutorial*, Woodstock, Oxon, UK.

153. S.L. Jarvenpaa and G.W. Dickson, 1988. Graphics and Managerial Decision Making: Research Based Guidelines, *Communications of the Association for Computing Machinery 31:6*, 764–774.

154. L.M. Jessup and J.S. Valacich (eds.), 1993. *Group Support Systems*, Macmillan, New York.

155. C.V. Jones, 1988. The Three-Dimensional Gantt Chart, *Operations Research 36:6*, 891–903.

156. C.V. Jones, 1990. An Example Based Introduction to Graph-Based Modeling, pp. 433–442 in *Proceedings of the Twenty-Third Annual Hawaii International Conference on the System Sciences*, IEEE Computer Society Press, Los Alamitos, CA.

157. C.V. Jones, 1990. An Introduction to Graph-Based Modeling Systems, Part I: Overview, *ORSA Journal on Computing 2:2*, 136–151.

158. C.V. Jones, 1991. An Introduction to Graph-Based Modeling Systems, Part II: Graph-Grammars and the Implementation, *ORSA Journal on Computing 3:3*, 180–206.

159. C.V. Jones, 1992. Animated Sensitivity Analysis, pp. 177–196 in *Computer Science and Operations Research: New Developments in their Interfaces*, O. Balci, R. Sharda and S.A. Zenios (eds.), Pergamon Press, Oxford, UK.

160. C.V. Jones, 1992. The Stability of Solutions to the Euclidean Traveling Salesman Problem, Technical Report, Faculty of Business Administration, Simon Fraser University, Burnaby, British Columbia, Canada.

161. C.V. Jones, 1994. An Integrated Modeling Environment Based on Attributed Graphs and Graph-Grammars, *Decision Support Systems*, to appear.

162. C.V. Jones, 1992. Attributed Graphs, Graph-Grammars, and Structured Modeling, *Annals of Operations Research 38*, 281–324.

163. C.V. Jones and K. D'Souza, 1992. Graph-Grammars for Minimum Cost Network Flow Modeling, Technical Report, Faculty of Business Administration, Simon Fraser University, Burnaby, British Columbia, Canada.

164. C.V. Jones and R. Krishnan, 1992. A Visual, Syntax-Directed Environment for Automated Model Development, Technical Report, Faculty of Business Administration, Simon Fraser University, Burnaby, British Columbia, Canada.

165. J. Kelley, 1961. Critical Path Planning and Scheduling: Mathematical Basis, *Operations Research 9:3*, 296–321.

166. D.A. Kendrick, 1990. Parallel Model Representations, *Expert Systems with Applications 1:4*, 383–389.

167. D.A. Kendrick, 1991. A Graphical Interface for Production and Transportation System Modeling: PTS, *Computer Science in Economics and Management 4:4*, 229–236.

168. S.O Kimbrough, C.W. Pritchett, M.P. Bieber and H.K. Bhargava, 1990. The Coast Guard's KSS Project, *Interfaces 20:6*, 5–16.

169. S.O. Kimbrough, C.W. Pritchett, M.P. Bieber and H.K. Bhargava, 1990. An Overview of the Coast Guard's KSS Project: DSS concepts and technology, pp. 63–77 in *Transactions of DSS-90: Information Technology for Executives and Managers, Tenth International Conference on Decision Support Systems*, Linda Volonino (ed.), Cambridge, MA.

170. P. Kirkpatrick and P.C. Bell, 1989. Visual Interactive Modeling in Industry: Results from a Survey of Visual Interactive Model Builders, *Interfaces 19:5*, 71–79.

171. T.C. Koopmans, 1949. Optimum Utilization of the Transportation System, *Econometrica 17*, 3–4.

172. R. Krishnan, 1990. A Logic Modeling Language for Model Construction, *Decision Support Systems 6*, 123–152.

173. M. Krueger, 1983. *Artificial Reality*, Addison-Wesley, Reading, MA.

174. M. Krueger, 1991. *Artificial Reality II*, Addison-Wesley, Reading, MA.

175. H.L. Larnder, 1979. The Origin of Operational Research, *Operational Research '78*, K.B. Haley (ed.), North-Holland, Amsterdam.

176. R. Laurini and D. Thompson, 1992. *Fundamentals of Spatial Information Systems*, Academic Press, London.

177. A.M. Law and W.D. Kelton, 1991. *Simulation Modeling and Analysis*, 2nd Ed., McGraw-Hill, New York.

178. D. Le Gall, 1991. MPEG: A Video Compression Standard for Multimedia Applications, *Communications of the ACM 34:4*, 46–58.

179. W. Leler, 1988. *Constraint Programming Languages*, Addison-Wesley, Reading, MA.

180. M.R. Lembersky and U.H. Chi, 1984. Decision Simulators Speed Implementation and Improve Operations, *Interfaces 14:4*, 1–15.

181. Lindo Systems, Inc., 1992. *LINGO Optimization Modeling Language*, Lindo Systems, Chicago, IL.

182. Lotus Development, 1993. *Improv for Windows Release 2.0 Reference Manual*, Lotus Development Corporation, Cambridge, MA.

183. I. Lustig, 1989. Applications of Interactive Computer Graphics to Linear Programming, pp. 183–189 in *Proceedings of the Conference on Impact of Recent Computer Advances in Operations Research*, R. Sharda, B.L. Golden, E. Wasil, O. Balci and W. Stewart (eds.), North-Holland, New York.

184. P.-C. Ma, F.H. Murphy and E.A. Stohr, 1989. A Graphics Interface for Linear Programming, *Communications of the ACM 32·8*, 996–1012.

185. J. Mackinlay, 1986. Automating the Design of Graphical Presentations of Relational Information. *ACM Transactions on Graphics 5:2*, 110–141.

186. K.-T. Mak, K. Srikanth and A. Morton, 1990. Visualization of Routing Problems, CRIM Working Paper 90-03, Department of Information and Decision Sciences, College of Business Administration, University of Illinois at Chicago, Chicago, IL.

187. D.G. Malcolm, J.H. Roseboom, C.E. Clark and W. Fazar, 1959. Applications of a Technique for R and D Program Evaluation (PERT), *Operations Research 7:5*, 646–669.

188. J. Maloney, A. Borning and B. Freeman-Benson, 1989. Constraint Technology for User-Interface Construction in ThingLab II, *SIGPLAN Notes 24:10*, 381–388.

189. MathPro, Inc., 1989. *MathPro Usage Guide: Introduction and Reference*, MathPro, Inc., Washington, DC.

190. D.L. Maulsby, I.H. Witten and K.A. Kittlitz, 1989. Metamouse: Specifying Graphical Procedures by Example, *Computer Graphics 23:3*, 127–136.

191. Maximal Software, Inc., 1992. *MPL User's Guide*, Maximal Software, Arlington, VA.

192. B.H. McCormick, T.A. DeFanti and M D. Brown, 1987. Visualization in Scientific Computing, *Computer Graphics 21:6*, 1–14.

193. Microsoft Corporation, 1991. *Microsoft Windows Multimedia Authoring and Tools Guide*, Microsoft, Redmond, WA.

194. Microsoft Corporation, 1992. *Microsoft Excel User's Guide Version 4.0*, Microsoft, Redmond, WA.

195. Microsoft Corporation, 1992. *Object Linking and Embedding Programmer's Reference*, Version 1, Microsoft Press, Redmond, WA.

196. A.R. Miller, 1988. *The ABC's of AUTOCAD*, Sybex, San Francisco, CA.

197. O. Ming, D.V. Beard and F.P. Brooks, Jr., 1989. Force Display Performs Better than Visual Display in a Simple 6-D Docking Task, pp. 1462–1466 in *Proceedings of the IEEE Robotics and Automation Conference 3*, IEEE Computer Society Press, Washington, DC.

198. J.J. Moder, C.R. Phillips and E.W. Davis, 1983. *Project Management with CPM, PERT and Precedence Diagramming*, 3rd Ed., Van Nostrand Reinhold, New York.

199. W.A. Muhanna and R.A. Pick, 1990. Meta-Modeling Concepts and Tools for Model Management: A Systems Approach, Technical Report, Faculty of Accounting and Management Information Systems, Ohio State University, Columbus, OH.

200. F.H. Murphy, E.A. Stohr and A. Asthana, 1992. Representation Schemes for Linear Programming Models, *Management Science 38:7*, 964–991.

201. B.A. Myers, 1986. Visual Programming, Programming by Example and Program Visualization: A Taxonomy, *SIGCHI Bulletin 17:4*, 59–66.

202. B.A. Myers, 1988. *Creating User Interfaces by Demonstration*, Academic Press, Boston, MA.

203. B.A. Myers, D.A. Guise, R.B. Dannenberg, B. Vander Zanden, D.S. Kosbie, E. Pervin, A. Mickish and P. Marchal, 1990. Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces, *IEEE Computer 23:11*, 71–85.

204. M. Nagl, 1976. Formal Languages of Labeled Graphs. *Computing 16*, 113–137.

205. M. Nagl, 1979. A Tutorial and Bibliographical Survey on Graph-Grammars, pp. 70–126 in *Graph-Grammars and their Application to Computer Science and Biology*, V. Claus, H. Ehrig and G. Rozenberg (eds.), Springer-Verlag, Berlin.

206. M. Nagl, 1987. Set Theoretic Approaches to Graph Grammars, pp. 41–54 in *Graph-Grammars and their Application to Computer Science*, H. Ehrig, M. Nagl, G. Rozenberg and A. Rosenfeld (eds.), Springer-Verlag, Berlin.

207. A.D. Narasimhalu and S. Christodoulakis, 1991. Multimedia Information Systems: The Unfolding of a Reality, *IEEE Computer 24:10*, 6–8.

208. B.A. Nardi and J.R. Miller, 1991. Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development, pp. 29–54 in *Computer-Supported Cooperative Work and Groupware*, S. Greenberg (ed.), Academic Press, London.

209. S.R. Newcomb, N.A. Kipp and V.T. Newcomb, 1991. The "HyTime" Hypermedia/Time-based Document Structuring Language, *Communications of the ACM 34:11*, 52–66.

210. NeXT Computer, Inc., 1992. *NeXTSTEP General Reference*, Addison-Wesley, Reading, MA.

211. G.M. Nielson, B. Shriver and L.J. Rosenblum, 1990. *Visualization in Scientific Computing*, IEEE Computer Society Press, Los Alamitos, CA.

212. D.A. Norman and S.W. Draper (eds.), 1986. *User Centered System Design: New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum, Hillsdale, NJ.

213. Numetrix, Inc., 1992. Linx Production and Distribution Logistics Planning System, Numetrix Limited, Toronto, Ontario, Canada.

214. J.F. Nunamaker, A.R. Dennis, J.S. Valacich, D.R. Vogel and J.F. George, 1991. Electronic Meeting Systems to Support Group Work, *Communications of the ACM 34:7*, 40–61.

215. W. Orlikowski and V. Dhar, 1986. Imposing Structure on Linear Programming Problems: An Empirical Analysis of Expert and Novice Modelers, pp. 308–312 in *Proceedings of the National Conference on Artificial Intelligence 1*, American Association of Artificial Intelligence, Menlo Park, CA.

216. K H. Palmer, N.K. Boudwin, H.A. Patton, A.J. Rowland, J.D. Sammes and D.M. Smith, 1984. *A Model Management Framework for Mathematical Programming*, Wiley, New York.

217. Paragon Decision Technology, 1993. Personal Correspon-

dence, PARAGON Decision Technology, Haarlem, The Netherlands.

218. P. Piela, 1989. ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis. Unpublished Ph.D. Dissertation. Carnegie Mellon University, Pittsburgh, PA.

219. P. Piela and R. McKelvey, 1992. An Introduction to ASCEND: Its Language and Interactive Environment, Technical Report, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

220. P.C. Piela, R.D. McKelvey, B. Katzenberg and B. Mehlenbacher, 1991. Integrating the User into Research on Engineering Design Systems, Technical Report, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

221. T. Quinlan and E. Scannell, 1993. Industry Sees Object System as Inevitable, *InfoWorld 15:4*, 13.

222. R.H. Rand, 1984. *Computer Algebra in Applied Mathematics: An Introduction to MACSYMA*, Pitman, Boston, MA.

223. T.W. Reps, 1986. *Generating Language-Based Environments*, The MIT Press, Cambridge, MA.

224. H. Rheingold, 1991. *Virtual Reality*, Summit Books, New York.

225. W. Robinett and J.P. Rolland, 1992. A Computational Model for the Stereoscopic Optics of a Head-Mounted Display, *Presence 1:1*, 45–62.

226. R.S. Rubin, 1993. Adobe Acrobat Juggles Text, Graphics for Corporate Docs, *MacWEEK 7:24*, 1ff.

227. R. Rucker, 1985. *The Fourth Dimension and How to Get There*, Viking Penguin, New York.

228. M.S. Sanders and E.J. McCormick, 1987. *Human Factors in Engineering and Design*, 6th Ed., McGraw-Hill, New York.

229. SAS Institute, 1989. *SAS/OR User's Guide Version 6*, 1st Ed., SAS Institute, Cary, NC.

230. S. Savage, 1992. Lotus Improv as a Modeling Language, Presented at ORSA/TIMS Joint National Meeting, San Francisco, CA.

231. M.W.P. Savelsbergh, 1988. Computer Aided Routing, Technical Report, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

232. R.W. Scheifler and J. Gettys, 1987. The X Window System, *ACM Transactions on Graphics 5:2*, 79–109.

233. H. Schultz and W. Pulleyblank, 1991. Trends in Optimization, *OR/MS Today, 18:4*, 20–25.

234. M. Scriabin, J. Farlette, D. Kotak and M.A. Matthews, 1991. Airport Capacity Planning Symbiotic System (AIRSYM): Combining Human and Artificial Intelligence, presented at Canadian Operational Research Society Competition on the Practice of OR, Quebec, May 21–23.

235. J. Sedgwick, 1993. The Complexity Problem, *The Atlantic 271:3*, 96–104.

236. R. Sharda, 1992. Linear Programming Software for Personal Computers: 1992 Survey, *OR/MS Today 19:2*, 44–60.

237. R. Sharda, 1994. Neural Networks for the MS/OR Analyst: An Application Bibliography, *Interfaces 24:2*, 116–130.

238. T. Sheldon, 1992. *Windows 3.1 Made Easy*, Osborne McGraw-Hill, Berkeley, CA.

239. B. Shetty, H.K. Bhargava and R. Krishnan (eds.), 1992. Model Management in Operations Research, *Annals of Operations Research 38*, 1–530.

240. B. Shneiderman, 1983. Direct Manipulation: A Step Beyond Programming Languages, *IEEE Computer 16:8*, 57–69.

241. B.D. Shriver (ed.), 1988. Artificial Neural Systems, *IEEE Computer 20:3*, special issue.

242. N.C. Shu, 1988. *Visual Programming*, Van Nostrand Reinhold, New York.

243. D.C. Smith, C. Irby, R. Kimball, B. Verplank and E. Harslem, 1983. Designing the Star User Interface, pp. 297–313 in *Integrated Interactive Computing Systems*, P. Degano and E. Sandwell (eds.), North-Holland, Amsterdam, The Netherlands.

244. J. Stasko, 1990. A Practical Animation Language for Software Development, pp. 1–10 in *Proceedings of the IEEE International Conference on Computer Languages*, IEEE Computer Society Press, Los Alamitos, CA.

245. J. Stasko, A. Badre and C. Lewis, 1993. Do Algorithm Animations Assist Learning? An Empirical Study and Analysis, pp. 61–66 in *Human Factors in Computing Systems, INTERCHI '93 Conference Proceedings*, Conference on Human Factors in Computing Systems, INTERACT '93 and CHI '93, Amsterdam, The Netherlands.

246. D.M. Steiger, R. Sharda and B. LeClaire, 1993. Graphical Interfaces for Network Modeling: A Model Management System Perspective, *ORSA Journal on Computing 5:3*, 275–291.

247. B. Sterling, 1993. War is Virtual Hell, *Wired 1:1*, 46ff.

248. M.C. Surles, 1992. An Algorithm with Linear Complexity for Interactive, Physically-based Modeling of Large Proteins, *Computer Graphics 26:2*, 221–230.

249. I.E. Sutherland, 1965. The Ultimate Display, *Proceedings of the IFIP Congress 2*, 506–508.

250. I.E. Sutherland, 1968. A Head-Mounted Three-Dimensional Display, pp. 757–764 in *Fall Joint Computer Conference, AFIPS Conference Proceedings, I*, Thompson Book Company, Washington, DC.

251. S. Takahashi, S. Matsuoka, A. Yonezawa and T. Kamada, 1991. A General Framework for Bi-Directional Translation between Abstract and Pictorial Data, pp. 165–174 in *Proceedings UIST 1991* (*ACM Symposium on User Interface Software and Technology*), ACM Press, New York.

252. A.A. Taseen, 1993. Visual Interactive Linear Programming: The Concept, an Example and an Empirical Assessment of its Value in Supporting Managerial Decision Making, Unpublished Ph.D. Dissertation, University of Western Ontario, London, Ontario, Canada.

253. R. Teach, 1991. State of the Profession: Lack of Recognition Tops List of Concerns, *OR/MS Today 18:5*, 26–30.

254. Tektronix, Inc., 1987. 3D Stereoscopic Color Graphics Workstation (TEK 4126 Product Literature), Tektronix, Beaverton, OR.

255. F. Thomas and O. Johnston, 1984. *Disney Animation*, Abbeville Press, New York.

256. A. Tuchman and M. Berry, 1990. Matrix Visualization in the Design of Numerical Algorithms, *ORSA Journal on Computing 2:1*, 84–92.

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

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

259. J. Tukey, 1977. *Exploratory Data Analysis*, Addison-Wesley, Reading, MA.

260. C. Upson, T. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz and A. Van Dam, 1989. The Application Visualization System: A Computational Environment for Scientific Visualization, *IEEE Computer Graphics and Applications 9:4*, 30–42.

261. C.P.M. Van Hoesel, A.W.J. Kolen, A.H.G. Rinnooy Kan and A.P.M. Wagelmans, 1989. Sensitivity Analysis in Combinatorial Optimization: A Bibliography, Technical Report, Econometric Institute, Erasmus University, PO Box 1738, 3000 DR, Rotterdam, The Netherlands.

262. F. Vicuña, 1990. Semantic Formalization in Mathematical Modeling Languages, Unpublished Ph.D. Dissertation, Computer Science Department, UCLA, Los Angeles, CA.

263. E. Wacholder, 1989. A Neural Network-Based Optimization Algorithm for the Static Weapon-Target Assignment Problem, *ORSA Journal on Computing 1:4*, 232–246.

264. H. Wagner, 1989. The Next Decade in Operations Research: Comments on the CONDOR Report, *Operations Research 37:4*, 664-672.

265. P. Wegner, 1990. Concepts and Paradigms of Object-Oriented Programming, *OOPS Messenger 1:1*, 7–87.

266. J.S. Welch, 1987. PAM—A Practitioner's Approach to Modeling, *Management Science 33:5*, 610–625.

267. S. Wolfram, 1991. *Mathematica: A System for Doing Mathematics by Computer*, 2nd Ed., Addison-Wesley, Redwood City, CA.

268. X Consortium, 1989. *Inter-Client Communication Conventions Manual*, The X Consortium, Cambridge, MA.

269. J.K. Yan, 1985. Advances in Computer-Generated Imagery for Flight Simulation, *IEEE Computer Graphics and Applications 5:8*, 37–51.

270. T. Zimmerman, J. Lanier, C. Blanchard, S. Bryson and Y. Harvill, 1987. A Hand Gesture Interface Device, *SIGCHI Bulletin*, special issue, 189–192.