THE ROLE OF SOFTWARE IN OPTIMIZATION AND OPERATIONS RESEARCH

Harvey J. Greenberg
University of Colorado at Denver, USA

Keywords: optimization, mathematical programming, computational economics, mathematical programming systems

Contents

Summary

This chapter presents the many roles of software in optimization. Obtaining a solution has been the focus of mathematical programming, and now one must consider different architectures and algorithm environments. But the role goes beyond the numerical computation, integrating concerns for supporting optimization modeling and analysis. Beginning with a historical perspective this chapter describes modern developments, including that of an *intelligent mathematical programming system.*

1. Introduction

The promise of operations research is to solve decision-making problems, and a large part uses optimization. The *mathematical program* is given by the following:

$$\text{optimize } f(x) : x \in X, g(x) \le 0, h(x) = 0, \qquad (1)$$

where $X \subseteq \mathbb{R}^n$, $f : X \mapsto \mathbb{R}$, $g :\mapsto \mathbb{R}^m$, $h : X \mapsto \mathbb{R}^M$, and *optimize* is either *minimize* or *maximize*.

In words, we seek to find a minimum or maximum of a real-valued function, possibly subject to some mixture of inequality and equality constraints. There are variations of this form, such as *multiple objectives*, *goals*, *uncertainty*, and *logical expressions*. Each variation is approached by using the above standard form, and generally, this can be done in more than one way. More discussion of this and particular mathematical programs are given in the *Mathematical Programming Glossary*.

The basic role of software is to enable an expression of a mathematical program, obtain a solution and provide tools to support model management and analysis of the results. In this chapter, we begin with some historical perspectives about software and optimization, followed by overviews of software designed to obtain a solution. Software for modeling has its own history, but the focus here is on current systems and needs for future designs. Then, *computer-assisted analysis* is explained, drawing from two bodies of work and pointing towards opportunities for enhanced roles of software. This leads naturally into a description of *intelligent mathematical programming systems*, marking new roles for future systems. The concluding section discusses current developments and the role of software in the broader field of operations research.

2. Historical Perspectives

Orchard-Hays gave an excellent historical account of the early years of mathematical programming and its inseparable ties to the computing field. (Also see his companion papers, as well as others, in the same proceedings.) Mathematical programming was treated as a *process*, rather than an object. As such, people needed a way to express their model, often with a general language like FORTRAN or with what were then called matrix generators, which evolved into *modeling languages*. Some evolved further into *modeling systems*.

This emerged in the early years, when *system* was used to denote the fact that it contained complete optimization, taking data as an input file and producing a solution file, rather than being a subroutine library. Currently, both the mathematical programming system (MPS) and the library approach (e.g., IBM OSL) are used. With the growth of object-oriented programming, there is a third paradigm: a shell that serves as a main program with generic classes; the programmer adds particular classes within this framework to represent a particular problem and algorithm. Current systems that use this paradigm are ABACUS, MINTO and PICO.

There were significant developments in the 1980s, not the least of which was the explosive growth of affordable computers. Waren, et al., give the status of nonlinear programming (NLP), with the emergence of PCs cited as a significant change that affects how algorithms are implemented and what has changed in importance, e.g., space (disk and memory) is no longer an issue (also see *Fourer's guide* in Bibliography). This is in some contrast to most of the more recent surveys, which still emphasize the algorithms apart from their implementation.

In integer programming, a recent plenary talk by Ralph Gomory suggested that modern computers can make some of the group-theoretic ideas, which had been abandoned in the 1970s, more practical. Following this note, Karla Hoffman gave a perceptive insight using *dinosaurs flying jets* as a metaphor to emphasize the role of computing power running old algorithms. Her primary assessment, however, was that it is the formulation that has the most impact on solvability, and this is the theme to consider as the modern role for software. In fact, this was a major goal of the project to develop an *Intelligent Mathematical Programming System* (IMPS).

3. Obtaining a Solution

A *solution* is a global optimum, or a declaration that no optimum exists, preferably with a reason, such as *infeasible* or *unbounded*. Software cannot guarantee that a correct conclusion is reached. One reason is computational error, causing inaccurate results, even for linear programs. Some problems use only integer arithmetic, where there is no computational error, but they are typically hard problems that cannot guarantee an accurate result due to how long it would take to consider the large number of possible solutions. It is this latter source of inaccuracy, that is the cutting edge of optimization software.

Several recent surveys provide details on the current state of the art in obtaining a global optimum, drawing from classical methods based on local search (e.g., on calculus). Because we cannot obtain exact solutions to hard problems in a practical amount of time, two basic approaches have been taken: *metaheuristics*, such as a *Tabu Search*, and *approximation algorithms* with guaranteed bounds. Software to implement these can make a difference in their performance (see *Global Optimization*).

In discrete optimization some exact methods exist for special situations, like dynamic programming, , but the *branch and bound/cut* remains the primary exact method. It is also used in continuous (global) optimization, so branch and bound is a general exact method. These typically use linear programming to solve *relaxation* problems for bounds, cut generation, and search guidance. The relaxations drop integer restrictions, and linear functions are used to approximate nonlinear ones (typically using the Taylor expansion). A software issue is how fast the linear programs are solved when used this way. The linear program (LP) relaxation of a combinatorial optimization problem has very different characteristics than a true LP, such as one that represents product distribution. This implies, for example, that the LP algorithm control parameters need adjustments from their default values since they are typically tuned with test problems that are true LPs (see *Linear Programming, Dynamic Programming* and *Bellman's Principle , Combinatorial Optimization and Integer Programming*).

3.1. Data structures, Controls and Interfaces

Wirth's charming equation, *Algorithms + Data Structures = Programs*, is not quite enough to be a system capable of solving optimization problems with computing paradigms available in 2002. To it, we must add:

Programs + Controls + Interfaces = Systems

There have always been algorithm controls, but interfaces have become much more sophisticated, especially with graphics.

Most publications in mathematical programming present algorithms, but very few describe the data structures. Fundamentally, an *algorithm* is a precise sequence of steps. We often refer to an *algorithm family*, or *algorithm strategy*, to mean some steps are not completely defined, leaving room for tactical variations. For example, Figure 1 gives the *simplex algorithm strategy* that uses the 2-phase method of getting an initial feasible solution (or ascertaining that none exists), leaving such steps as pricing open to different algorithm tactics.

Step 0 (Initialize). Assume the LP is in standard form: minimize $cx : Ax = b, x \geq 0$, where A has full row rank. Setup data structure and read LP data: A, b, c. Find initial basis and initialize Phase I objective, $\hat{c}$.

Step 1 (Price). Given basis, B, solve $\pi B = \hat{c}_B$, and set $J = \emptyset$. For each j, compute $d_j = \hat{c}_j - \pi A_j$; if $d_j < 0$, put j into J. Stop when $|J|$ is large enough, or when all j has been interrogated. In the latter case, go to Step 3 if $J = \emptyset$.

Step 2 (Select). Compute $B\beta = b$. For each $j \in J$, solve $B\alpha = A_j$. If $\alpha \leq 0$, EXIT WITH LP UNBOUNDED. Else, choose $q_j \in \text{argmin}\{\frac{\beta_i}{\alpha_i} : \alpha_i > 0\}$, and set $\theta_j = \frac{\beta_q}{\alpha_q}$. Select $p \in J$ and pivot to replace q_p in the basis with p. Return to step 1.

Step 3 (Optimal). If in Phase II, EXIT WITH LP OPTIMAL. Else (in Phase I), and the objective value is positive, EXIT WITH LP INFEASIBLE. Else, set objective to c, and go to step 1 to begin Phase II.

Figure1: Two-Phase Simplex Algorithm Strategy

A *data structure* is how we store the information. Even a specific algorithm, such as Dantzig's Specific Simplex Method, does not specify the type of arithmetic or other things that depend upon the data structure (and the computing environment). In the case of a simplex algorithm, a typical data structure is to store the data in a *sparse format*. For example, the LP matrix is stored by columns with each column having a pair of data entries for each nonzero:

row index | value

In particular, suppose we have a transportation problem with *s* sources and *d* destinations. The dimensions of the matrix are $m = s + d$ rows (one per node) and $n = sd$ columns (one per arc). Each column of the node-arc incidence matrix has exactly two non-zeroes, -1 for the source and +1 for the destination. The total number of nonzeroes, therefore, is 2*n*, and its density is $\frac{2n}{mn}$, which reduces to $\frac{2}{m}$. The greater the number of nodes (*m*), the greater the sparsity, because the density becomes small as *m* becomes large. To illustrate, consider the following node-arc incidence matrix, which represents a transportation problem that has 2 sources and 3 destinations.

$$A = \begin{bmatrix} -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

There are 12 nonzero entries in this 5 × 6 matrix, so its density is $\frac{12}{30}$, which is 0.4. For more realistic sizes, where $m \gg 1000$, the density is less than 1%. Many large LPs have an embedded network structure, which is one reason they tend to be very sparse.

Nonlinear functions are represented by their parse tree, or something very similar. Figure 2 shows an example.

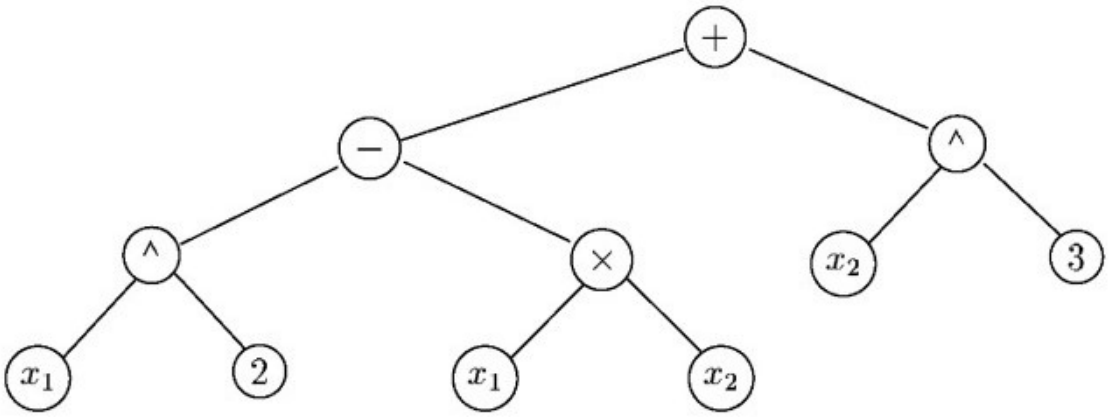

Figure 2: Parse Tree for $f(x_1;\ x_2) = x_1^2 - x_1 x_2\ + x_2^3$

Solvers call upon this tree for function evaluation. Modern methods use *automatic differentiation*, which stores nonlinear functions in a similar data structure, designed for rapid calculations of the function and its derivatives (if they exist).

3.2. Parallel Algorithms

One of the current topics of research is how to design parallel algorithms. An early use of parallel architecture was for Dantzig-Wolfe decomposition in LP. Many types of architectures and algorithms are in the text by Bertsekas and Tsitsiklis, and more recent ideas are in proceedings, notably those edited by Heath, *et al.* and by Pardalos, *et al.*

To illustrate how a parallel architecture can be used effectively, consider a parallelization of the *conjugate gradient method* (see *Nonlinear Programming*). We are given a quadratic form, $\frac{1}{2}x^T Qx - cx$, where Q is an $n \times n$ symmetric, positive-definite matrix. We know its (unique) minimum occurs at the solution to the equation, $Qx = c$, and one way to compute this is by the following iterations:

$$x^{t+1} = x^t + \alpha^t d^t, \tag{3}$$

where x^0 is given, $d^0 = -g^0 = c - Qx^0$, and

$$\alpha^t = -\frac{g^t d^t}{(d^t)^T Q d^t} \tag{4}$$

$$d^t = -(g^t)^T + \beta^t d^{t-1} \tag{5}$$

$$\beta^t = \frac{(g^{t-1})^T Q d^{t-1}}{(d^{t-1})^T Q d^{t-1}} \tag{6}$$

$$g^t = (x^t)^T Q - c = \nabla\left(\frac{1}{2}x^T Qx - cx\right)\Big|_{x=x^t} \tag{7}$$

Let us assess the computations at an iteration. At the beginning of iteration t, we have already computed x^t, d^{t-1} , and g^{t-1} $(g^{t-1})^T$. We must compute $g^t = Qx^t - c$ and $g^t(g^t)^T$; from these, we use equations (6) and (5) to compute β^t and d^t, respectively. Finally, we use equation (3) to compute x^{t+1}, thereby completing the iteration.

Suppose $p \geq n$, so we can assign a processor to compute results pertaining to one coordinate of the vectors. Let processor i be given the i^{th} row of Q, denoted $Q_{i\bullet}$. The vector Qx^t - c is done completely in parallel, with processor i computing $Q_{i\bullet}\, x^t - c_i$. These are communicated to a designated processor, which thus obtains g^t. This processor could compute $g^t(g^t)^T$, or it could broadcast g^t to the other processors and have them compute $(g_i^t)^2$ and send its value back for accumulation. Which is better depends on communication time versus the time to compute an inner product on the main processor. Alternative schemes use hierarchies to accumulate inner products, and much of the design depends not only upon the particular architecture, but also upon whether the matrix and vectors are sparse.

This illustrates the general notion that one way to design a parallel algorithm is to use parallel linear algebra. Many optimization algorithms can be parallelized this way, but some do not use linear algebra centrally. In particular, combinatorial optimization algorithms can take other advantage of parallel computation.

Consider the dynamic programming recursion for the 0-1 knapsack problem:

$$f_j(\beta) = \max\{f_{j-1}(\beta), c_j + f_{j-1}(\beta - a_j)\} \text{ for } \beta \in \{a_j, a_j + 1, ..., b\}. \tag{8}$$

for $j = 1, \ldots, n$, where $f_0 \equiv 0$ (see *Discrete Optimization, Dynamic Programming and Bellman's Principle*). Suppose we decompose each iteration by having each processor compute f_j for an equal number of state values. For notational convenience, let $b = kp$ for some positive integer, k. In words, let the total size of the knapsack be an integer multiple of the number of processors. This means that each processor has k state values

to compute, and each takes the same amount of time. This is clearly a logical decomposition, but its merit depends on communication time, which depends upon the particular parallel architecture.

Scheduling tasks to processors is the key problem in designing a parallel algorithm. *Load balancing* is a proxy goal for maximizing the *speedup*, whose reciprocal is the fraction of the best possible serial computation time. Let $T^*(s)$ be the best time it takes to solve a problem in some class of size s on a serial computer. (The "size" is frequently taken to mean the number of variables, but it could account for other data values.) Let $T_p(s)$ be the time it takes a parallel algorithm to solve the same problem with p processors. Then,

$$Speedup = S_p(s) = \frac{T^*(s)}{T_p(s)}. \tag{9}$$

The best one can hope for is $S_p(s) = p$ — equivalently, $T_p(s) = T^*(s)/p$. This occurs if the best serial algorithm can have its computations decomposed perfectly into p parts, each taking the same amount of time. This could never be fully achieved, even with perfect load balancing, due to *message passing* that is needed – that is, the information produced by a processor must be communicated to at least one other processor.

The speedup of the dynamic programming parallelization is nearly linear. We have

$$S_p(n,b) = \frac{T^*(n,b)}{T^*(n,k) + M(n,k)}, \tag{10}$$

where $M(n, k)$ is the communication time for n variables and k states. The size is given as the number of variables (n) and the number of states (b), but we can consider the cost per iteration that is, divide by n, since both algorithms perform n iterations. Then,

$$S_p(n,kp) = \frac{\tau(kp)}{\tau(k) + \mu(k)}. \tag{11}$$

Further, each state requires a simple comparison of two values plus a storage of the result, so $\tau(kp) = p\tau(k)$. Substituting this into the above expression simplifies the speedup, which is now independent of the number of variables:

$$S_p(n,b) = \frac{p}{1 + \dfrac{\mu(k)}{\tau(k)}}, \tag{12}$$

The speedup depends on the communication time relative to the computation time, which is the ratio: $\frac{\mu(k)}{\tau(k)}$. We can see that $\tau(K) \approx \alpha k$ for some constant α, but $\mu(k)$ could be $O(k)$ or $O(\log k)$, depending upon the architecture. A worst case has

$S_p(n,kp) \approx \gamma p$ for some constant, γ, which is still very good for large state values. Doubling the number of processors doubles the speedup. The more favorable case is when $\mu(k) = O(\log k)$, for then $\lim_{k\to\infty} S_p(n, kp) \to p$. Currently, one of the most active places for such research is Sandia National Laboratories, with the ASCI red machine that consists of more than 9000 tightly-coupled Intel processors.

They have an object-oriented Parallel Integer and Combinatorial Optimization code, called PICO. Despite its title, PICO is a framework designed for general parallel branch and bound, including nonlinear programs with continuous-valued variables. While there have been some efforts to use parallel algorithms for hard problems, this is still a frontier with many design decisions and experiments ahead. One class of algorithms that have a natural decomposition to balance the load among the processors is the class of stochastic, population-based, evolutionary algorithms.

Some classes of hard nonlinear problems, such as in design optimization, involve expensive functional evaluations. Often, $f(x)$ is the solution to a differential equation or a simulation, where x is a vector of design parameters. Although x is not large, obtaining $f(x)$ can take hours. Parallel algorithms are being developed for such problems, some with a speedup on the order of $\sqrt{p}$.

-
-
-

Bibliography

Ashford R.W. and Daniel R.C. (1986). LP-MODEL: XPRESS-LP's model builder. *IMA Journal of Mathematics in Management* **1**(163), 176. [Downloads are available at <http:// www.dash.co.uk.>]

Bentley P.J. ed. (1992, 1999). *Evolutionary Design by Computers*, 446 pp. San Francisco, CA: Morgan Kaufman. [This is a basic reference for evolutionary algorithms/programming.]

Bertsekas D.P. and Tsitsiklis J.N. (1989). *Parallel and Distributed Computation*, 715 pp. Englewood Cliffs, NJ: Prentice-Hall. [This presents the fundamentals of parallel algorithms, with special attention to optimization.]

Chandru V. and Hooker J.N. (1953, 1999). *Optimization Methods for Logical Inference*, 365 pp. New York: John Wiley & Sons. [This organizes the developments in applying integer programming to the sort of logical inference done in artificial intelligence. Emphasis is given to the authors' work during the past decade.]

Chinneck J.W. (1997). Computer codes for the analysis of infeasible linear programs. *Journal of Operational Research Society* **47**(1), 61-72. [This describes the author's development of computing IISs in various optimization programs.]

Chinneck J.W. (1997). Feasibility and viability. *Advances in Sensitivity Analysis and Parametric Programming*, (ed. T. Gal and Greenberg H.J.), pp. 14-1-14-41, Boston, MA: Kluwer Academic Publishers. [This surveys the state-of-the-art in practical methods for helping an analyst understand why a

problem is infeasible, including non-linear programs. This addresses the related question of whether some variables must be zero in every feasible solution.]

Chinneck J.W. Analyzing mathematical programs using MProbe. *Annals of Operations Research*, To appear. Downloads are available at <http://www.sce.carleton.ca/faculty/chinneck/mprobe.html>. [This applies to any mathematical program, but its main use is to provide shape information for nonlinear functions. It probes AMPL source code and uses a sampling technique.]

Chinneck J.W. and Greenberg H.J. (1999). Intelligent mathematical programming software: Past, present, and future. *Canadian Operations Research Society Bulleti*n **33(**2), 14-28. Also appeared in *INFORMS Computing Society Newsletter* **20**(1), 1-9, 1999. [This surveys optimization software that is intelligent in its ability to help users formulate and analyze mathematical programs and generally manage the process of doing so.]

Crescenzi P. and Kann V., ed.. (2000). *A compendium of NP optimization problems*. <http://www.nada.kth.se/~viggo/problemlist/compendium.html>. [The editors maintain this site, giving latest bounds and time complexity of approximation algorithms for NP-hard problems.]

Dutta A., Siegel H.J., and Whinston A.B.. (1983). On the application of parallel architectures to a class of operations research problems. *R.A.I.R.O. Recherche opérationanelle/Operations Research* **17**(4),317-341. [This is one of the first parallel algorithms for linear programming. It applies to Dantzig-Wolfe decomposition.]

Eckstein J., Hart W.E., and Phillips C.A. (2000). *PICO: An object-oriented framework for parallel branch and bound*. Research report, Albuquerque, NM: Sandia National Laboratories. Also available as a RUTCOR technical report. [This is a primer for the entitled object-oriented system where the user supplies the classes for a particular problem and algorithm.]

Fourer R. (1983). Modeling languages versus matrix generators for linear programming. *ACM Transactions On Mathematical Software* **9,** 143-183. [This was an early description that clarified the entitled distinction.]

Fourer R. (1996). Software for optimization: A buyer's guide, Parts 1 & 2. *INFORMS Computing Society Newsletter* **17**(1, 2). [This gives a broad view of what is available in all areas of optimization software.]

Glover F. and Laguna M. (1997). *Tabu Search*, 382 pp. Boston, MA: Kluwer Academic Publishers. [This is the basic book about Tabu Search, written by its pioneers.]

Greenberg H.J. (1993) *A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE*. Boston, MA: Kluwer Academic Publishers. Downloads are available at <http://www.cudenver.edu/~hgreenbe/imps/software.html>. [This describes and illustrates the ANALYZE software system, which is part of the author's intelligent mathematical programming system.]

Greenberg H.J. (1994). Syntax-directed report writing in linear programming. *European Journal of Operational Research* **72**(2), 300-311 [This shows how the rules of composition the syntax of a model can be used to drive the report writing process, including interactive query.]

Greenberg H.J. (1996). A bibliography for the development of an intelligent mathematical programming system. *Annals of Operations Research* **65**, 55-90. Online versions are available at: <http://www.cudenver.edu/~hgreenbe/imps/impsbib/impsbib.html> and <http://orcs.bus.okstate.edu/itorms>. [This gives hundreds of citation from early works through 1996.]

Greenberg H.J.(1996-2000). *Mathematical Programming Glossar*y. <http://www.cudenver.edu/~hgreenbe/glossary/>. [This is an ongoing glossary with about 700 entries and supplements.]

Greenberg H.J. and Maybee J.S. ed. (1981). *Computer-Assisted Analysis and Model Simplification*, 522 pp. New York: Academic Press. [This introduced computer-assisted analysis as a formal foundation for what evolved into an intelligent aid.]

Greenberg H.J. and Murphy F.H. (1991). Approaches to diagnosing infeasible linear programs. *ORSA Journal on Computing* **3**(3), 253-261. Note: the journal has been renamed *INFORMS Journal on Computing*. [This describes the main approaches to the entitled problem, pointing out that Diagnosis = Isolation + Explanation.]

Greenberg H.J. and Murphy F.H. (1992). A comparison of mathematical programming modeling systems. *Annals of Operations Research* **38**,177-238. [This uses specific examples to compare and contrast three types of systems: process, algebraic and schematic.]

Greenberg H.J. and Murphy F.H. (1995). Views of mathematical programming models and their instances. *Decision Support Systems* **13**(1), 3-34. [This shows many ways to view optimization models, with focus on LP. A central data structure is proposed to enable the computer to produce any view that best meets the cognitive skill of the user.]

Heath M.T., Ranade A., and Schreiber R.S. ed. (1999). *Algorithms for Parallel Processing*. New York: Springer. [This is a modern account, with a variety of subjects by different authors.]

Hochbaum D.S., ed. (1997). *Approximation Algorithms for NP-Hard Problems*. Boston, MA: PWS Publishing Co. [This is the fundamental reference to learn about approximation algorithms and their application to optimization problems.]

Hoffman K.L. Combinatorial optimization: Current successes and directions for the future. J*ournal of Computational and Applied Mathematics*, (In Press). [This gives perceptive insights about the importance of modeling combinatorial optimization problems. It was also a keynote talk at the 1999 SIOPT meeting.]

Hooker J. (2000). *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. New York: John Wiley & Sons. [This combines the strengths of the entitled topics into a coherent theory of problem expression and algorithm design.]

Jones C.V. (1995). *Visualization and Optimization*. Boston, MA: Kluwer Academic Publishers,. An online version is at <http://www.chesapeake2.com/cvj/itorms/>. [This is a complete compendium of the many visualizations that have been developed, including many by the author.]

McAloon K. and Tretko C. (1995). *Logic and Optimization*. New York: John Wiley & Son. [This is an important introduction, taking problem descriptions from the view of artificial intelligence, and solution methodology from integer programming.]

McCarl B. (1996.). *GMSCHK User Documentation*. <http://agrinet.tamu.edu/mccarl/gamssoft.htm>. Lubbock, TX: Texas A&M University. [This is one-of-a-kind software that applies directly to GAMS source code.]

Nemhauser G.L., Savelsbergh M.W.P., and Sigismondi G.C. (1994). MINTO, A Mixed INTeger Optimizer. *Operations Research Letters* **15**, 47-58. Downloads available at <http: //akula.isye.gatech.edu/~mwps/projects/minto.html>. [This is an object-oriented system where the user supplies the classes for a particular problem and algorithm.]

Orchard-Hays W. (1978). History of mathematical programming systems. *Design and Implementation of Optimization Software*, (ed. H.J. Greenberg) 551 pp., The Netherlands: Alphen aan den Rijn : Sijthoff & Noordhoff. [This gives the early history of mathematical programming and how it was intertwined with the history of computer science during the years 1950-75.]

Pardalos P.M., Resende M.G.C., and Ramakrishnan K.G. ed. (1995). Parallel Processing of Discrete Optimization Problems, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **22**. Providence, RI: American Mathematical Society. [This has an important collection of papers and an insightful introduction.]

Robbins T. (1970). *Antenna Design*. Doctoral Thesis, Southern Methodist University, Dallas, TX. [This was an early example of applying optimization to a very difficult objective, requiring significant computer time for each evaluation.]

Rumelhart D.E. and McClelland J.L. ed. (1986). *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vol. 1: Foundations. Cambridge, MA: MIT Press, [This was a very inspiring book that helped to create major research efforts in neural network computers.]

Rutherford T. (1999). *Some GAMS Programming Utilities*. <http://nash. colorado.edu/tomruth/inclib/tools.htm>. Boulder, CO: University of Colorado. [This is a useful suite of tools, including interfaces with building output tables.]

Sowa J.F. (1984). *Conceptual Information Processing*, 374 pp. Amsterdam; New York: American Elsevier. [This presented new concepts connecting language and intelligence. It led to new

implementations of natural language processing.]

Spedicato E. ed. (1994). *Algorithms for Continuous Optimization: The State of the Art.* NATO ASI Series, 565 pp. Dordrecht, The Netherlands: Kluwer Academic Publishers. [This has an excellent collection of papers, focused on numerical methods to solve nonlinear programs.]

Stasko J.T., Dominque J.B., Brown M.H., and Price B.A., ed. (1998). *Software Visualization; Programming as a Multimedia Experience*, 562 pp. Cambridge, MA: MIT Press. [This is a general book on visualization that includes algorithm animation.]

Thienel S. (1995). *ABACUS A Branch-And-CUt System*. Doctoral Thesis, Universität zu Köln, Germany. More information is available at <http://www.informatik.uni-koeln.de/ls_juenger/ projects/abacus/>. [This is an object-oriented system where the user supplies the classes for a particular problem and algorithm.]

Vanderplaats G. (1995). *DOT Users Manual*. Colorado Springs, CO: Vanderplaats Research and Development, Inc. Downloads available at <http://www.vrand.com>. [This describes a Design Optimization System, which addresses problems that are hard because the objective function requires a significant amount of time for each evaluation. Generally, $f(x)$ is the numerical solution to a PDE, where x is a vector of design parameters.]

Waren A.D., Hung M.S., and Lasdon L.S. (1987). The status of nonlinear programming software: An update. *Operations Research* **35**(4), 489-503. [One of the significant advances addressed in this survey is the emergence of PCs, making computer storage space no longer an issue, as it had been.]

Wirth N. (1976). *Algorithms + Data Structures = Programs*, 366 pp. Englewood Cliffs, N.J.: Prentice-Hall. [This is a classic book about *Algorithms + Data Structures = Programs*.]

Biographical Sketch

Harvey Greenberg is Professor of Mathematics at the University of Colorado at Denver. After receiving his Ph.D. in Operations Research from The Johns Hopkins University in 1968, he joined the Faculty of Computer Science and Operations Research at Southern Methodist University. Dr. Greenberg has pioneered the interfaces between operations research and computer science since that time. He was a founder of what is now the INFORMS Computing Society, having served as its Chairman twice, and he was the founding Editor of the INFORMS Journal of Computing. While at the U.S. Department of Energy, 1976-83, Dr. Greenberg developed Computer Assisted Analysis (CAA), dealing with advanced analysis techniques, including debugging scenarios that might be infeasible or anomalous. Shortly after joining CU-Denver in 1983, he formed a Consortium to develop an Intelligent Mathematical Programming System, which extended CAA in breadth and depth. During that time, the project generated several software systems and about 50 papers. One of the systems, ANALYZE, was recognized with the first "ORSA/CSTS Award for Excellence in the Interfaces Between Operations Research and Computer Science." The system is still used today, and it was a major part of Dr. Greenberg's plenary talk at the Canadian Operational Research Society, when he accepted their Harold Lardner Prize for having "achieved international distinction in Operational Research." Since 1996, Dr. Greenberg has been an active developer of web materials to support teaching. In addition to his own site, which contains the Mathematical Programming Glossary, he developed a site as a service to the Mathematics Department, called "Using the Web to Teach Mathematics." Most recently, Dr. Greenberg established a collaborative relationship with Sandia National Laboratories, helping to develop parallel algorithms for NP-hard problems, notably in computational biology, such as protein folding. Dr. Greenberg has published 104 papers and four books; he has (co)edited seven additional books or proceedings. He serves on six editorial boards, and has been Guest Editor for the *Annals of Mathematics and Artificial Research* on such topics as "Reasoning about Mathematical Models" and "Representations of Uncertainty".