Introduction to the IBM Optimization Subroutine Library

by D. G. Wilson
B. D. Rudin

This essay introduces the IBM Optimization Subroutine Library (OSL) and seven OSL-related papers that appear in this issue. Developed as a result of a partnership between several IBM research and development groups, OSL provides a suite of tools for manipulating the models and solving the resulting minimization and maximization problems of mathematical optimization. The problems that OSL addresses include: linear, quadratic, mixed-integer, and pure network programming problems. OSL includes solvers based on the classical simplex method and on newer interior point methods. Because a user-supplied driver program coordinates the problem solution, and because of the "mix and match" philosophy of OSL, a user may, within rather wide limits, individually tailor a technique to solve a particular problem. We conclude that OSL is something new in optimization software.

Mathematical optimization is broadly applicable to the problems of minimizing costs, maximizing profits, and scheduling projects subject to time and financial constraints. Computer software implementing the techniques of mathematical optimization has been much in demand, and IBM has been a leader in providing such software. For many years, IBM's Mathematical Programming System Extended/370 (MPSX/370)[1] program product has been a standard tool used by optimization specialists for applications in a wide variety of business and government areas. However, the optimization field has grown in size, variety, and complexity of applications. New methods are needed to solve problems accurately and quickly.

The Optimization Center in the Mathematical Sciences Department of IBM Research has been continuously active in developing needed techniques, and there has been an ongoing partnership between the Mathematical Sciences Department and the Application Technology Center of the IBM Kingston High Performance Computing Solutions Development organization devoted to integrating Research's contributions into IBM engineering and scientific products. This partnership had already produced the IBM Engineering and Scientific Subroutine Library (ESSL). Thus, when it became clear that a new kind of product was needed to support use of IBM systems for optimization applications, it was natural to turn to this partnership, and the Optimization Subroutine Library (OSL) is the result.

OSL was developed to provide practitioners with a suite of tools for manipulating the mathematical models and solving the resulting minimization and maximization problems of mathematical optimization. OSL is a departure from the optimization software products of the past. It has algorithmic breadth: having linear programming via both simplex and interior point approaches, mixed-integer programming, quadratic programming, and network problems. It has platform

breadth: being available on most IBM workstation and mainframe systems with most of the operating system environments, including a low-end version running under the popular disk operating system from Microsoft Corporation. It is extensible, and it is not subject to problem size limitations other than those imposed by the platforms. Finally, it is readily adapted to provide solutions for vendors' products.

In this issue of the *IBM Systems Journal* the reader is introduced to the depth and breadth of OSL. In the remainder of this essay, we identify the problems that OSL addresses and give a moderately technical overview of the content of OSL. The papers by Forrest and Tomlin take the reader through the simplex and interior point algorithms of OSL for solving linear programming problems.[2,3] The paper by Jensen and King that then follows explains the simplex-based algorithm used to solve quadratic programming problems.[4] Next, Minkoff shows how to make the transition from a symbolic representation of a mathematical programming problem to a functional driver program that will use OSL modules to solve the problem.[5] In the following paper, Jensen and King illustrate use of the quadratic programming algorithms of the library in portfolio optimization applications.[6] We then turn to a contribution from one of the customers who helped us as an early user of OSL for their applications. Anbil and Tanga of American Airlines Decision Technologies join with Johnson of IBM to describe their application of OSL to airline crew scheduling problems.[7] Finally, Johnson and Nemhauser of the Georgia Institute of Technology give us some insight into where the mathematical programming and optimization field is headed.[8]

We have claimed that OSL is something new to optimization. We urge the reader, after reading this issue, to work with the product and with the development group to make it even better, thus maintaining OSL as something new to optimization.

The problems addressed by OSL

OSL provides the capability to solve linear, quadratic, and mixed-integer programming problems. For linear programming (LP) problems, both the objective function to be minimized (or maximized) and the constraints that define the domain of the problem are linear. LP problems are the simplest mathematical optimization problems, and we assume that the reader is at least moderately well-acquainted with them. The ability to solve LP problems provides a necessary foundation for solving other types of optimization problems. Nonlinear problems that OSL addresses include quadratic programming (QP) problems, characterized by quadratic terms in the objective function, and mixed-integer programming (MIP) problems, characterized by integrality constraints on some of the variables.

In problems related to stock trading, both the trend of a stock price and its volatility are considered. Modeling volatility introduces quadratic terms into the objective function, and hence, QP problems arise when volatility is taken into account. In general, QP problems may include quadratic terms in the constraints as well as in the objective function, but in its present form, OSL only handles QP problems with linear constraints. MIP problems may arise in applications in which the variables represent indivisible units, such as railroad cars, aircraft, etc. However, in many MIP problems, the integer variables represent decision alternatives whose possible values are limited to zero and one. For example, assigning a value of one to such a variable might correspond to making a decision to assign a particular crew member to a particular flight, or to build a facility at a particular location, whereas assigning a value of zero to the variable would correspond to the decision not to assign that crew member to that flight, or not to build the facility at that location. Such decision variables can only be "true" or "false." Fractional values for the corresponding problem variables are not meaningful, and rounding a solution in which such a variable did not have a value of either one or zero might give a nonoptimal answer or even one that failed to satisfy some constraint.

Network programming problems are a special subclass of LP problems that can represent flows through a network or a schedule of events. These problems can be represented pictorially by directed graphs with the nodes representing cities, reservoirs, etc., and the interconnecting arcs representing roads, canals, etc. The values of the variables correspond to flows along the arcs. Capacities of the arcs define bounds on these variables, and unit costs of moving items along an arc give the coefficients of the objective function. Material balance conditions at the nodes, including

sources and sinks, define the constraints of the problem.

For pure network programming problems, a strict conservation principle is imposed that greatly restricts the form of the linear constraints, and thus greatly affects the implementation of a solution technique. This principle implies that there be no addition to or diminution of a quantity along an arc. If, for example, a truck travels from one city to another, and it departs with 120 chickens, the conservation principle requires that it arrive with 120 chickens. Similar but somewhat less-restricted problems, called generalized network programming problems, allow for possible linear changes in variable values along arcs, such as changes of units on the variables or fixed percentage losses en route. For example, 2 percent of the chickens may escape along the road.

The user must supply a main program that coordinates the problem solution using subroutines from OSL.

OSL includes solvers for LP, QP, MIP, and pure network programming problems. For LP problems (other than MIP and pure network programming problems), OSL provides solvers based on the classical simplex method and on newer interior point methods. Any of the interior point solvers can also be run on the initial relaxed version of a MIP problem, without its integrality constraints, to get an initial bound on the value of the objective function for the integer solution.

The content of OSL

Organization and philosophy. The OSL development philosophy was to provide a set of building blocks that a user could assemble as required to solve different kinds of optimization problems. OSL includes user-callable subroutines in the following seven categories: (1) solvers for LP, QP, MIP, and pure network programming problems, (2) initialization and setup, (3) input and output, (4) matrix manipulation, (5) message handling, (6) control variable querying and setting, and (7) sensitivity and parametric analyses. In addition, there is a full complement of user exit routines, which the user may accept or replace. (Control variables and user exits are discussed in a later subsection within this section.) The OSL subroutines use state-of-the-art algorithms that have been implemented in codes that take advantage of the architectures of the IBM platforms on which they execute. They are written primarily in portable FORTRAN, with a few assembler language or (on platforms running Advanced Interactive Executive*, or AIX*) C language routines included.

The user must supply a main program that coordinates the problem solution using subroutines from OSL. To solve a particular problem, a user may develop a new program or adapt one of the sample programs supplied with OSL. The main program may be written in FORTRAN, C, PL/I, or APL2. It may be as simple as four subroutine calls, or as complicated as an expert user may care to make it. Various OSL subroutines may be included or omitted, and they may be called in various orders. The user may also include in the main program calls to subroutines written to supplement or even replace certain OSL modules. Thus, within rather wide limits the user may tailor the technique used to solve a particular problem.

Short descriptions of the solver modules. OSL includes several solvers based on the well-known simplex method for solving LP problems. In this method, a computationally efficient, systematic search for the optimal value of the linear objective function is performed among the vertices of the feasible region determined by the linear constraints of the problem. (For an LP problem, if there is a nonempty subset of the solution space where all constraint relations can be satisfied, this subset must be a simply connected, convex, polyhedral region, called the feasible region. Furthermore, if there exists an optimal solution of an LP problem, the set of variable values where the optimal solution is attained must include at least one of the vertices of this polyhedral region.)

In any implementation of the simplex method, considerable execution time is spent "pricing," i.e., selecting a particular variable (based on its current importance in the objective function) whose value is to be changed in determining the

next candidate vertex in the systematic search for the optimum. Thus, a good pricing strategy can significantly improve the performance of the solver. In all solvers of OSL that are based on the simplex method, special provision has been made to allow a knowledgeable user to influence the pricing strategy to enhance performance.

In textbook explanations of the simplex method, the work is divided into two phases. In phase one an auxiliary objective function is minimized to obtain a feasible solution, and in phase two the true objective function is minimized (or maximized) starting from this feasible solution. In the state-of-the-art variant of the method implemented as the standard LP solver in OSL, the work is not divided into two phases, but instead a composite objective function is used throughout. This composite objective function is a linear combination of the phase one and phase two objective functions. Thus, the variables are simultaneously moved toward their optimum values as feasibility is approached. A detailed explanation of the implementation of the simplex method for OSL is presented by Forrest and Tomlin elsewhere in this issue.[2]

Over the last twenty years, numerous algorithms for solving LP problems have been developed that produce a sequence of points that converge to the optimum along a trajectory through the interior of the feasible region instead of skirting its periphery, as the simplex method does. Methods that use this approach have come to be known as interior point methods. Three new algorithms of this type are included in OSL. All three are based on the "logarithmic barrier" method, which minimizes a sequence of functions, each of which is a linear objective function augmented by a sum of logarithmic terms multiplied by a barrier parameter. The additional terms form a barrier that keeps successive approximations to the solution inside the feasible region because they grow very large in magnitude as the values of the variables approach their upper and lower bounds. As the minimization progresses, the barrier parameter is progressively reduced, and a sequence of successive approximations to the optimal point is generated.

The three interior point algorithms included in OSL are a primal barrier method and two primal-dual barrier methods, one with a predictor-corrector scheme and one without. In the primal barrier method, the linear part of the objective function is the primal objective function, and successive step directions are the projections of the direction of steepest descent of the augmented objective function onto the space in which all of the constraints of the problem are satisfied. In the two primal-dual barrier methods, solution of the primal and dual problems are addressed simultaneously. Differences in the two methods arise from the techniques used to solve the mildly nonlinear system of equations defined by the necessary conditions for both problems to have a solution. The predictor-corrector method uses an inner-outer iteration to solve the system directly, whereas the other method uses a Newton iteration scheme. All the interior point solvers provide the option to switch over to the simplex method solver at (or near) the completion of the interior point iterations. As an initialization tactic, the primal algorithm may first solve a phase one problem to locate a feasible point as described above. A much more comprehensive explanation of these algorithms is presented by Forrest and Tomlin in a second paper elsewhere in this issue.[3]

The pure network solver included in OSL is an implementation of the simplex method that takes great advantage of the form and content of the coefficient matrix of the set of linear constraint relations, called the constraint matrix of the problem. Constraint matrices for pure network programming problems take a particularly simple form because of the conservation principle mentioned earlier. Each column has only two nonzero entries, and the values of these entries are plus one and minus one. As a result, many of the intermediate computations of the simplex method, such as matrix multiplications and factoring of certain coefficient matrices, can be performed implicitly or avoided altogether. In addition, storage requirements are significantly reduced. Therefore, an implementation of the simplex method that is specialized for pure network programming problems can execute much faster than an implementation that is not so specialized.

As in the implementation of the simplex method for LP problems, the pure network solver uses a composite objective function throughout the calculation. Thus, as before, the variables are moved toward their optimum values as feasibility is approached without a division of the work into separate phases. Most of the execution time in the network solver is spent pricing, that is, selecting

a variable whose value is to be changed in determining the next vertex in the systematic search of the simplex method. Thus, a good pricing strategy can significantly improve the performance of the network solver, and special provision has been made to allow a knowledgeable user to select a pricing strategy that is best suited for a particular application.

For a QP problem with linear constraints, as for an LP problem, if there is a nonempty subset of the solution space where all of the linear constraint relations can be satisfied, this subset must be a simply connected, convex, polyhedral region. However, for the QP problem, in contrast to an LP problem, an optimum, if one exists, need not occur at a vertex or even on the boundary of the feasible region. Thus, the simplex method alone cannot be used to solve such problems. The algorithm implemented in OSL for QP problems is a hybrid of two methods, both based on the simplex method. The process is begun by using the simplex method to solve the LP problem obtained by deleting the quadratic terms from the objective function. The optimal solution of this initial linearization of the problem is taken as the initial approximation to the solution of the QP problem.

The first part of the compound algorithm proceeds iteratively with the following steps. The tangent hyperplane to the quadratic objective function at the last previous approximation to the optimal point is generated, and the simplex method is used to solve a new LP problem that has the original constraints and this hyperplane as its objective function. Then, the quadratic objective function is minimized over a multidimensional polyhedral region defined by the optimal points and possible directions of unboundedness for all previously considered LP problems. This multidimensional minimization is a QP problem, but it has a particularly simple form, and it is easily solved. The point where this minimum is attained is taken as the next approximation to the optimal.

When successive approximations are sufficiently close together, the second part of the compound algorithm is used. This subalgorithm is a modification of the simplex method that permits a quadratic objective function, and it converges very rapidly when given a good initial estimate of the solution. This hybrid approach to solving QP problems yields an algorithm that is both fast and robust. The algorithm is explained in more detail by Jensen and King elsewhere in this issue.[4]

For MIP problems, the solution strategy implemented in OSL includes a branch and bound technique and an optional preprocessor that does extensive "probing" of the MIP problem. Briefly, the branch and bound strategy proceeds as follows. As a first approximation, the solution of the LP problem obtained from the original problem by removing the integrality constraints is found. The solution of this problem gives a bound for the optimal value for the MIP problem because the integer solution cannot be better than the solution of the problem without integrality constraints. If the values of all of the variables in the solution of this unrestricted problem that are required to be integers in the MIP problem are indeed integers, this solution is optimal for the original problem. If not, an initial branching of the problem is performed as follows.

A candidate variable with a noninteger value, say X1, is selected as the branching variable, and two new subsidiary problems are considered. This approach is described as "adding two nodes to a branch and bound tree." One subsidiary problem, or branch of the tree, corresponds to a MIP problem in which X1 is constrained to be less than or equal to the largest integer less than X1, and the other branch corresponds to a MIP problem in which X1 is constrained to be greater than or equal to the next higher integer. Each subsidiary problem is a new MIP problem with one strengthened constraint. Ignoring the integrality constraints again gives new LP problems that can be analyzed. If, for any subsidiary problem, all of the solution variables that are required to have integer values do have integer values, this solution is a candidate for the optimum of the original problem, and its optimal value is a bound for all other possible solutions. If not, another candidate variable with a noninteger value, say X2, is selected as the new branching variable, and two new nodes are added to the branch and bound tree. These two nodes (or subsidiary problems) can be solved and recursively split into more nodes until the best integer solution is found. New bounds for the solution are obtained as new integer solutions are found.

After solving some node, K, in the tree, it may be found that the objective value for the LP problem at K, without integrality constraints, is not as

good as the current bound given by the best objective value from among all other known feasible integer solutions. If this occurs, node K can be pruned from the tree, and the time that would have been spent solving children nodes of K can be saved. OSL provides expert users with the capabilities of selecting the next branching variable, selecting the next node to be solved, adding constraints to a node, and fixing the way in which nodes are solved or pruned.

In the probing mentioned earlier, each of the zero-one variables is fixed first to zero and then to one, and the logical consequences of all these assignments are investigated. The effect of this preprocessing is similar to branching in the branch and bound strategy, but the logical analysis does not require the resulting LP problems to be solved. The preprocessor may fix variables, modify coefficients in constraints, add new constraints, determine infeasibility, or even determine an optimal solution. If the number of zero-one variables in the MIP problem is significant, this preprocessing can greatly reduce the solution time required.

Customizing the solver algorithms. OSL includes control variables and calls to user exits that will allow expert users to develop custom solution strategies for particular problems. At the same time, the default control paths allow all users to solve most problems with exemplary speed. Moreover, as typical users gain experience with solving problems of the type in which they are most interested, they will become more able and willing to take advantage of the available options.

User-settable control variables affect many aspects of the execution of the algorithms. These multiposition switches permit defining, among many other things: the number of simplex iterations to be done with one pricing strategy before changing to another, the tolerances for detecting zero values and certain error conditions, the allowed amounts of primal and dual infeasibilities, the initial weight for the phase one part (or feasibility component) of the composite objective function used by the simplex method LP solver, the rate at which the barrier parameter of the interior point solvers is reduced, the maximum number of steps of the simplex method to be done before a matrix refactorization is done, and the maximum number of nodes allowed in the branch and bound tree for the MIP solver. Of course, default values that were determined by running the solvers against a suite of representative test problems are supplied for all such variables.

Embedded calls by each solver to certain "user exit" subroutines make possible even more comprehensive customization of OSL. A suite of (mostly dummy) user exit routines is distributed with OSL. To take advantage of the user exits, the user must write one or more replacement routines (in FORTRAN), compile them, and load them with the main program and the OSL subroutines. At load time, the user's routines will supersede the user exits distributed with OSL, and at run time, the user's routines will gain control in the midst of the execution of the OSL solver routines. This suite provides a very powerful means for the user to monitor and control the execution of the algorithms. For example, a novice user could print out intermediate information as the solution of a problem progressed, or an expert user could change solution tactics in major ways whenever it seemed advantageous to do so.

OSL provides a collection of tools for solving LP, QP, and mixed-integer programming problems.

Summary

OSL, a new IBM product for manipulating and analyzing optimization problems, was developed in response to customer requirements for more powerful and more flexible software. OSL provides a collection of tools for solving linear programming, quadratic programming, and mixed-integer programming problems. Individual OSL components implement state-of-the-art algorithms in code that takes special advantage of characteristics of the IBM platforms on which they run. These algorithms include both interior point algorithms and algorithms based on the simplex method. Collectively, the OSL components provide a versatile array of tools for solving optimization problems. These components can be com-

bined into applications as simple as "input, call solver, output" sequences or into complicated, individualized algorithms that have been highly customized by knowledgeable users.

OSL was developed as a result of a partnership between IBM research and development groups in Yorktown Heights and Kingston, New York, and Almaden, California. Members of these groups are authors or coauthors of several papers in this issue of the *IBM Systems Journal*. We have given brief descriptions of the classes of problems that OSL addresses and a technical overview of the content of OSL. In doing so, we have introduced the contents of several papers in this issue. For more detailed discussions of OSL, and the algorithms implemented therein, the reader is referred to these papers and to the OSL Guide and Reference.[9]

*Trademark or registered trademark of International Business Machines Corporation.

Cited references

1. *Mathematical Programming System Extended/370 General Information Manual*, GH19-6549, IBM Corporation; available through IBM branch offices.
2. J. J. H. Forrest and J. A. Tomlin, "Implementing the Simplex Method for the Optimization Subroutine Library," *IBM Systems Journal* **31**, No. 1, 11–25 (1992, this issue).
3. J. J. H. Forrest and J. A. Tomlin, "Implementing Interior Point Linear Programming Methods in the Optimization Subroutine Library," *IBM Systems Journal* **31**, No. 1, 26–38 (1992, this issue).
4. D. L. Jensen and A. J. King, "A Decomposition Method for Quadratic Programming," *IBM Systems Journal* **31**, No. 1, 39–48 (1992, this issue).
5. A. S. Minkoff, "A Systematic Approach to OSL Application Programming," *IBM Systems Journal* **31**, No. 1, 49–61 (1992, this issue).
6. D. L. Jensen and A. J. King, "Frontier: Graphical Interface for Portfolio Optimization in a Piecewise Linear-Quadratic Risk Framework" *IBM Systems Journal* **31**, No. 1, 62–70 (1992, this issue).
7. R. Anbil, R. Tanga, and E. L. Johnson, "A Global Approach to Crew-Pairing Optimization," *IBM Systems Journal* **31**, No. 1, 71–78 (1992, this issue).
8. E. L. Johnson and G. L. Nemhauser, "Recent Developments and Future Directions in Mathematical Programming," *IBM Systems Journal* **31**, No. 1, 79–93 (1992, this issue).
9. *Optimization Subroutine Library Guide and Reference*, SC23-0519, IBM Corporation; available through IBM branch offices.

Accepted for publication September 13, 1991.

D. George Wilson *IBM Application Solutions Division, P.O. Box 100, Kingston, New York 12401.* Dr. Wilson rejoined IBM in 1990 after an absence of nearly 22 years. He first joined IBM in 1962 as an associate programmer in the Federal Systems Division. While there, he worked at various code development assignments for the military and the Federal Aviation Agency until 1968, when he left IBM to complete his graduate education. In the years 1970–1972, he was an assistant professor of mathematics at Virginia Commonwealth University. In 1972 he joined the research staff of the mathematics and statistics research group at Oak Ridge National Laboratory and worked there until 1990 as a researcher and, ultimately, group leader, developing and analyzing progressively more complicated mathematical and computational models of physical systems. His specialty became the formulation of models representing heat transfer processes involving change of phase, and the numerical solution of the resulting system of mildly nonlinear partial differential equations using high-performance vector computing facilities. From 1974 to 1989, he was also an adjunct professor of mathematics at the University of Tennessee at Knoxville. Since January of 1990, Dr. Wilson has been a member of the Optimization Subroutine Library development team in the Application Technology Center at the Kingston Center for High Performance Computing Solutions Development. He holds a B.S. in chemistry from the University of Oklahoma, an M.S. in physics from Iowa State University, and a Ph.D. in mathematics from the University of Maryland.

Bernard D. Rudin *IBM Advanced Workstations Division, 11400 Burnet Road, Austin, Texas 78758.* Dr. Rudin recently accepted a new assignment in Austin. Since 1987, he had been manager of the Application Technology Center at the IBM Kingston Center for High Performance Computing Solutions Development. There he was responsible for enablement of strategic applications for all IBM engineering and scientific computing platforms, development of application building block products, e.g., the Engineering and Scientific Subroutine Library and the Optimization Subroutine Library, and provision of application impetus and development partnership to product divisions to aid in improvement of product designs for engineering and scientific application performance. Dr. Rudin joined IBM in 1966, after holding various positions in management, research, and engineering at the Lockheed Missile and Space Company, the Lockheed California Company, and the Marquardt Aircraft Corporation. In the years 1966–1977, he held managerial and senior technical positions with IBM's Federal Systems Division, Systems Development Division, and Data Processing Product Group. From 1978 to 1983, he was manager of the Los Angeles Scientific Center, where he was responsible for the development and execution of the Scientific Center Systems and Software advanced technology missions and complementary market support activities. From 1983 to 1986, he worked as a senior engineer and manager in Scientific and Engineering Processor Products, Data Systems Division, analyzing applications in support of new product design, and enabling applications for vector and parallel processors. Dr. Rudin holds a B.S. in astronomy from the California Institute of Technology, an M.S. in mathematics from the University of Southern California, and a Ph.D. in mathematics from Stanford University.

Reprint Order No. G321-5456.