Preprint ANL/MCS-P8xx-0899, August, 1999
Mathematics and Computer Science Division
Argonne National Laboratory

Stephen J. Wright*

Optimization Software Packages

August 19, 1999

Abstract. We discuss software packages for solving optimization problems, focusing on fundamental software that assumes that the problem has already been formulated in mathematical terms. Such packages can be used directly in many applications, linked to modeling languages or to graphical user interfaces, or embedded in complex software systems such as logistics and supply chain management systems.

1. Introduction

Optimization software is used in a wide variety of applications, including the fundamental sciences, engineering, operations research, finance, and economics. This plethora of applications has led to the development of a large number of fundamental, general purpose software packages that can be used efficiently in a variety of contexts.

This chapter discusses software packages that solve optimization problems which have been specified precisely in a mathematical sense. The specification may take the form of a data file, user-supplied code that fills out given data structures or evaluates functions at given values of the argument, or a model defined via a modeling language such as AMPL, AIMMS, GAMS, or MPL. We do not discuss optimization software that forms a part of larger applications, such as spreadsheets or supply chain management systems, unless the optimization code is also available separately.

Optimization software development has gone hand-in-hand with the theoretical development and analysis of algorithms during the past thirty years. In many instances, the software authors themselves contributed heavily to the theoretical analysis. This fact is hardly surprising, since the experience gained in applying algorithms to practical problems reveals many features of their performance that are not obvious from the theory, thereby driving the modification and invention of new algorithms.

The growing commercial market, particularly in linear and integer programming, has yielded rapid improvements in the quality of the software in some

Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439; wright@mcs.anl.gov

* Research supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

areas and sometimes in the underlying algorithms as well. In other areas, improvements have been more gradual, due in some cases to the lack of demand and in other cases (such as nonlinear programming) to the inherent difficulty of the problem class.

The state of the art in optimization software is discussed in the book of Moré and Wright [2]. Information from this book was used as the initial basis of the NEOS Guide to Optimization Software, which can be found on the Web at

`www.mcs.anl.gov/otc/Guide/SoftwareGuide/`

This site, which is updated continuously, contains more detailed descriptions of most of the packages mentioned in this article. We refrain from giving bibliographic references or URLs in the text below in cases where this information can be found easily in the NEOS Guide. Other notable online resources also can be accessed through the NEOS Guide; go to

`www.mcs.anl.gov/otc/Guide/OtherSites/`

We mention in particular the Decision Tree for Optimization Software at

`plato.la.asu.edu/guide.html`

and Michael Trick's Operations Research Page at

`mat.gsia.cmu.edu`

Surveys of optimization software are carried out regularly in *OR/MS Today*, the magazine published by INFORMS, the leading professional society in operations research.

Some of the codes mentioned in this article can be executed remotely on the NEOS Server at

`www.mcs.anl.gov/neos/Server/`

Users submit data to the Server and receive results through an email or web interface to the Internet. The Server is a convenient way for users to benchmark solvers and to test their suitability for a specific application.

In the following sections, we discuss the availability of software tools in several important areas, including linear programming, integer programming, and nonlinear programming. We mention a few of the major packages in each area, discuss the underlying algorithms, and outline recent developments and current research that is likely to have an impact on future software. In the final section, we present some impressions of the possible future directions of the field of optimization software.

2. Linear Programming

The birth of modern optimization dates to George Dantzig's invention of the simplex algorithm for linear programming in the late 1940s. This extremely effective method is still the basis of most linear programming codes today. Linear programming codes consume far more computer cycles than codes for any

other class of optimization problems. The reasons are partly historical and partly practical. Generations of modelers in economics and finance have been trained to build linear models. Even when the underlying application is actually nonlinear, as is often the case, a linear model suffices to yield useful results, and the unavailability of much of the data associated with the problem makes the use of a nonlinear model unnecessarily sophisticated in any case. Finally, the global solutions of linear programs are comparatively easy to find, while such is not the case with general nonlinear programs.

The linear programming problem can be formulated mathematically as follows:

$$\min_x c^T x \quad \text{subject to } Ax = b,\ x \geq 0, \tag{1}$$

where $x \in \mathsf{R}^n$, $c \in \mathsf{R}^n$, $b \in \mathsf{R}^m$, and $A \in \mathsf{R}^{m \times n}$, for some integers m and n with $m \leq n$. The set of feasible points for (1) is a polyhedral subset of R^n.

The simplex method generates a sequence of iterates x^k, $k = 0, 1, 2, \ldots$ that are vertices (extreme points) of the feasible polyhedron. At most m components of each x^k are nonzero, and each iterate is obtained by moving along an edge of the polyhedron from the preceding iterate. The objective value decreases as the iterations proceed, that is, $c^T x^{k+1} \leq c^T x^k$ for all k. Codes based on the simplex method were available from the 1950s onward, but their effiency increased steadily through the 1970s due mainly to advances in the linear algebra techniques and in "pricing," the process of deciding the edge along which to move away from the current iterate x^k. By the late 1980s, commercial simplex codes appeared to have stabilized, but a new spurt of development was motivated by the appearance of codes based on a rival class of algorithms: interior-point methods (see [4]).

Today, interior-point codes coexist with simplex codes in the commercial domain. Commercial simplex codes include CPLEX Simplex, MINOS, and those distributed by XPRESS-MP, OSL, and LINDO. Commercial interior-point codes include CPLEX Barrier and OSL. All these codes are undergoing continual development, and significant improvements are still being reported on many large practical problems. When coupled with the improvements in computer hardware, the resulting solvers are vastly more powerful than was the case in the mid-1980s.

Some codes are inexpensive or freely available to researchers and nonprofit users. The outstanding simplex code in this category is SOPLEX, which is available at the following URL:

`www.zib.de/Optimization/Software/Soplex/`

Academic licensing of the OSL package for Windows platforms is also currently free of charge. Free or inexpensive interior-point codes include BPMPD, PCx, HOPDM, LIPSOL, and LOQO. Interior-point codes are more widely available for the simple reason that they are less complicated to program than simplex codes. However, they frequently outperform even the best simplex codes on some problems, particularly large problems. Simplex codes have the important advantage that they can exploit prior information, such as a good estimate of

the solution of (1), whereas interior-point codes do not derive much benefit from such information. Warm starts are available in one of the most significant uses of linear programming codes, namely, their use in solving continuous relaxations of integer programming problems.

Computational comparisons of the many of the codes mentioned above have been performed by Hans Mittelmann, and the results can be viewed at the following URL:

`http://plato.la.asu.edu/bench.html`

The traditional format for entering data into linear programming codes is the MPS file, a format dating to the 1950s and based on 80-character-wide punch cards. A much more natural way to specify a linear programming problem is via an algebraic modeling language such as AMPL, AIMMS, GAMS, or MPL, which allows variables, constraints, and objectives to be written in much the same way as on the modeler's note pad. Most linear programming codes can be accessed through at least one of these languages. In other cases (for instance LINDO) the linear programming solver and the modeling language are sold as an integrated package. Some packages, notably XPRESS-MP and CPLEX, come with their own algebraic input format.

Another way to interface to the software is through subroutine calls, which are used to build up the model, modify it, solve the problem, and extract the results. This mode is useful when the software must be embedded into a larger system or an existing application, and most commercial codes come with the required "subroutine library" interface.

3. Integer Programming

Integer linear programming problems have the form (1), except that some variables are required to take on integer values. (In the important special case of *binary* variables, we require $x_i \in \{0, 1\}$.) In many problems, there is a subset of variables that is not required to be integral. Such problems are known as "mixed-integer (linear) programs."

Problems with integer constraints are much more difficult to solve than continuous linear programs, due to the fact that the set of feasible points is no longer convex or even connected. The software is considerably more complicated.

Most algorithms proceed by solving a sequence of continuous linear programs of the form

$$\min_x c^T x \quad \text{subject to } Ax = b,\ x \geq 0,\ Cx \leq d,$$

where the additional constraints $Cx \leq d$ are changed as the algorithm proceeds. In these so-called continuous relaxations, the integrality requirements are not imposed on x. The purpose of the additional constraints $Cx \leq d$ is to either fix or bound some of the components of x at integer values or to shrink the feasible set $\{x \mid Ax = b, x \geq 0\}$ by cutting off regions of the continuous feasible set that do not contain integer feasible points. Constraints of the latter type are known as

cutting planes, and were the basis of the first methods developed in the 1950s. Constraints that fix or bound some of the components at integer values arise in the branch-and-bound approach, which has formed the basis of almost all commercial codes. Recent research has focused on combining the cutting-plane and branch-and-bound approaches, and on identifying powerful new classes of cuts. Surprising advances are still being made in this area, as algorithms and heuristics are improved and combined in imaginative ways, and the interaction between the integer programming code and its simplex-method substrate is made more efficient.

For a brief introduction to methods for integer programming, see Wolsey [3].

High-quality codes for integer programming are available only in the commercial sector, partly because of the need for a highly efficient simplex code to solve the continuous relaxations and partly because of the intrinsic complexity of the heuristics used in the best methods to generate cutting planes and to perform the branching. The CPLEX, XPRESS-MP, and OSL codes are possibly the best known, and all are undergoing continual development. More information on these and other solvers can be found on the NEOS Guide.

Integer nonlinear programs also arise in many applications, but these are very difficult to solve, combining as they do the vagaries of nonconvex nonlinear programming and integer programming. There are a number of research codes for this problem class. The code MIQP for mixed-integer quadratic programming is available on the NEOS Server.

4. Quadratic Programming

Quadratic programming problems have the general form

$$\min_x \; c^T x + \tfrac{1}{2} x^T Q x \quad \text{subject to } \; Ax = b, \; x \geq 0, \tag{2}$$

where Q is a symmetric matrix whose properties and structure are highly application-dependent. When Q is positive semidefinite, the problem (2) is a *convex* quadratic program, and is much easier to solve than in the complementary case in which Q has some eigenvalues negative. Another important special case arises when all the constraints are bounds, that is, the general linear equalities $Ax = b$ are not present.

The simplex method for linear programming, described in Section 2, can be extended to the problem (2). In this setting it is known by the more general term "active-set method," since each step can be characterized by a subset $\mathcal{A} \subset \{1, 2, \ldots, n\}$ of indices of x which are fixed at their bound of 0. As in linear programming, this algorithm generates a sequence of feasible iterates, but it no longer confines its search to the vertices of the feasible polyhedron because the solution may lie in the interior of an edge or face, or even strictly inside the polyhedron. If the component x_i reaches its bound on the current step, the index i is added to $\mathcal{A}$. An index j may be deleted from $\mathcal{A}$ on some iteration if the Lagrange multiplier for the constraint $x_j \geq 0$ indicates that the step generated

by relaxing this constraint is a descent direction for the objective function in (2).

Codes based on the active-set approach appear in OSL and LINDO, while the packages BQPD, QPOPT, and SNOPT also implement this approach.

The interior-point approach can be used as an alternative to the active-set approach in the convex case, that is, Q positive semindefinite. Interior-point solvers are well suited to large-scale problems, and they can often be tailored to exploit special strucure in the matrix Q to improve the efficiency of the calculations. The basic approach is a straightforward extension of Mehrotra's primal-dual algorithm for linear programming. Packages that implement this approach include CPLEX, LOQO, OSL, and BPMPD.

For problems in which only bound constraints are present, algorithms based on the projected gradient approach are useful, particularly when the problem is large. This approach has been investigated in many applications, but no codes are generally available for this specific problem. However, this approach is available in more general codes such as L-BFGS-B or LANCELOT, which are designed for problems with a nonlinear objective.

5. Unconstrained Optimization

In unconstrained optimization, we seek to find the minimizer of a function $f(x)$. Techniques differ according to the whether the function is smooth (at least twice differentiable) or nonsmooth, and to whether we seek a local or a global minimizer.

Algorithms related to Newton's method have proved to be particularly effective when we seek a local minimizer of a smooth function. In general, we form a model of the function f around the current iterate x^k—for instance, the quadratic model

$$q^k(d) \stackrel{\text{def}}{=} f(x^k) + \nabla f(x^k)^T d + \tfrac{1}{2} d^T H_k d \tag{3}$$

where H_k is either the Hessian $\nabla^2 f(x^k)$ or some approximation to it, obtained either from a quasi-Newton update formula or a finite-difference formula. The problem (3) is solved, possibly approximately and possibly with the inclusion of a bound on the size of d. Line-search methods search along the resulting step d to find a new iterate $x^{k+1} = x^k + td$ (for some $t > 0$) with a significantly smaller value of f. Trust-region methods adjust the bound on the size of d until the point $x^{k+1} = x^k + d$ gives an improved f value.

The code LANCELOT, though written for more general classes of problems, reduces to a Newton-like method (with approximate Hessians H_k) when applied to unconstrained minimization problems. LBFGS and VE08 are designed for large-scale problems; they use a line-search approach in which H_k is a quasi-Newton aprpoximation that can be stored in $O(n)$ locations (where n is the dimension of the unknown vetcor x), rather than the $n^2/2$ memory locations required by a dense matrix. TN/TNBC and TNPACK, which are also designed

for large problems, take H_k to be the true Hessian, and use an iterative method to find an approximate minimizer of (3). UNCMIN and NMTR (a code from the MINPACK-2 collection) implement a variety of trust-region approaches. The code TENMIN augments the second-order model (3) with third-order terms from the Taylor series for f. It tends to perform more reliably than Newton-type methods when the Hessian $\nabla^2 f$ is singular at the solution. The implicit filtering approach described in Kelley [1, Chapter 7] uses finite-difference approximations to obtain an approximation to the gradient $\nabla f(x^k)$, while a quasi-Newton approximation is used fof H_k. This approach is appealing when the function is noisy, and when derivatives are not available.

Nonlinear conjugate gradient methods are popular for large-scale problems in which gradients ∇f can be supplied by the user. They do not use a quadratic model and require only $O(n)$ storage. Several such methods are implemented in CGplus, which is available through the NEOS Server.

Direct search methods such as Nelder-Mead simplex, Hooke-Jeeves, and multidirectional search are popular because they require only function values to be supplied. These methods, which are designed for small n, evaluate f on a regular pattern of points. This pattern is modified to investigate regions of the space that appear to be promising, and f is reevaluated at any new points so generated. For details on these methods and information on software, see Kelley [1, Chapter 8]. The simulated annealing approach can also be used when only function values are available. For an implementation, see the code ASA.

Often we wish to confine the search to a rectangular box defined by $l \leq x \leq u$, where l and u are vectors of upper and lower bounds, respectively. The most popular algorithms in this class are active-set and gradient-projection methods. Active-set methods hold a subset of the components of x at one or other of their bounds on each iteration, and take steps like those of unconstrained optimization methods in the other components. The active set, which consists of the fixed components of x, generally changes from one iteration to the next. Codes of this type include TN/TNBC. In the gradient projection approach, a search direction d is calculated in the space of all variables, and candidates for the new iterate are ontained by projecting points of the form $x^k + td$ onto the feasible box, where $t > 0$ is the line search parameter. The new point obtained in this way may be improved by a sequence of Newton-like steps. Codes that use this approach include LANCELOT, LBFGS-B, IFFCO, and VE08.

6. Nonlinear Programming

In nonlinear programming problems, both the objective function and constraints can be general nonlinear functions. By adding slack variables to the inequalities, we obtain the following general formulation:

$$\min_x f(x) \quad \text{subject to} \quad c(x) = 0, \;\; l \leq x \leq u, \tag{4}$$

where l and u are vectors of upper and lower bounds, respectively. (Some of the components of l may be $-\infty$, indicating that there is no lower bound on the

corresponding component of x, while similarly some of the components of u may be $+\infty$.)

Algorithms that are currently available in software include sequential quadratic programming, the projected gradient method, and the augmented Lagrangian method, otherwise known as the method of multipliers. Interior-point methods, both primal (log-barrier) and primal-dual, are the subject of much research but are not currently available in production software. Other penalty and barrier techniques, while still investigated by theoreticians from time to time, do not appear in general purpose codes.

In sequential quadratic programming (SQP), a model problem for (4) is formed by taking Taylor-series approximations about the current iterate x^k. In the case of SQP, this model problem typically has the form

$$\min_d \nabla f(x^k)^T d + \tfrac{1}{2} d^T H_k d \quad \text{s.t. } c(x^k) + \nabla c(x^k)^T d = 0, \;\; l \leq x^k + d \leq u, \quad (5)$$

where H^k represents the Hessian of the Lagrangian function for the problem (4) and is constructed either from exact second derivatives or from a quasi-Newton update formula. If this step d makes progress toward solving (4), as measured by some merit function, we define the new iterate to be $x^{k+1} = x^k + d$. Otherwise, a trust-region or line-search strategy is used to obtain a new candidate step d.

The codes NLPSPR, SPRNLP, and SNOPT use the SQP approach. All of these packages exploit sparsity in the linear algebra calculations, while the codes DONLP2, NLPQL, and NPSOL use dense linear algebra and are therefore suited to problems with fewer variables and constraints. The package FSQP uses a variant of SQP in which all iterates are confined to the feasible region. Such a restriction is useful when the objective function is not defined outside the feasible region, or when it is essential to the user that the code return a feasible point, even when it is not able to converge to full accuracy to a solution of the problem (4).

The method of multipliers makes use of the augmented Lagrangian function for (4), which is defined as

$$\mathcal{L}_a(x, \lambda; \gamma) \stackrel{\text{def}}{=} f(x) + \lambda^T c(x) + \frac{1}{2\gamma} c(x)^T c(x),$$

where $\gamma > 0$ is a positive parameter and λ is a vector of Lagrange multiplier estimates. At iteration k, a new value x^{k+1} is obtained by approximately solving the problem

$$\min_x \mathcal{L}_a(x, \lambda_k; \gamma_k) \quad \text{subject to } \; l \leq x \leq u, \quad (6)$$

and the multiplier estimates are updated by setting

$$\lambda^{k+1} \leftarrow \lambda^k + \frac{1}{\gamma_k} c(x^{k+1}).$$

The parameter γ_k may be either held constant or decreased prior to the next iteration. The subproblem (6) is a bound-constrained optimization problem that can be solved by using the techniques discussed in the previous section.

LANCELOT implements the method of multipliers for general constrained problems. This code also makes use of trust-region techniques, gradient projection techniques for solving (6), efficient iterative linear algebra methods, quasi-Newton updates, and sophisticated heuristics for updating the parameter γ_k (which is actually weighted differently for different components of $c(\cdot)$).

The reduced gradient approach for solving (4) is an extension of the simplex method for linear programming, in which nonbasic variables (those at their upper or lower bound) and basic variables (those whose values are determined by the constraint $c(x) = 0$) take their place alongside superbasic variables, which are allowed to move freely. Techniques from unconstrained minimization are applied to find steps for the superbasic variables. Variables may be reclassified among the three categories when superbasic or basic variables encounter bounds, or when Lagrange multiplier estimates indicate that a nonbasic variable should move off its bound. Codes based on this approach include CONOPT and LSGRG2, both of which exploit sparsity in the Jacobian matrix $\nabla c(x)$.

The sequential-linearly-constrained algorithm used by the popular code MINOS combines the augmented Lagrangian approach with projected gradient and penalty techniques.

It is not currently possible to make general statements about the relative effiency of the techniques discussed above. The paradigm (4) encompasses a wide variety of problems with a wide variety of peculiarities, and a technique that performs relatively well on one type of problem might perform quite poorly on another type.

7. Other Problems

Nonlinear least-squares problems arise often in the context of fitting models to data. In these problems the objective function has the special form

$$f(x) = \frac{1}{2}\sum_{i=1}^{m} r_i^2(x),$$

so that the Hessian $\nabla^2 f(x)$ has the special property that it can be approximated by the term

$$\sum_{i=1}^{m} \nabla r_i(x) \nabla r_i(x)^T, \tag{7}$$

which consists of only first-order information about the functions r_i. Hence, in a sense, we get some information about the second-order quantity $\nabla^2 f(x)$ for free, and algorithms usually exploit this fact by setting H_k in the quadratic model (3) to the approximation (7). MINPACK contains an implementation of a trust-region algorithm in which the approximation (7) is used for the Hessian of f—essentially the Levenberg-Marquardt algorithm. TENSOLVE implements both line-search and trust-region strategies that include second-order information in the Hessian. Like its unconstrained minimization counterpart TENMIN,

this code performs well on degenerate problems. The PORT3 subroutine library contains an implementation of the algorithm NL2SOL, which combines the approximation (7) with a quasi-Newton estimate of the remaining terms of $\nabla^2 f(x)$. VE10 uses a line-search method in which the conjugate gradient method is used to find an approximate minimizer of the quadratic model (3). Other codes that exploit the least-squares structure include DFNLP, MODFIT, PROCNLP, and NLSSOL. The package ODRPACK also implements Levenberg-Marquardt, and additionally addresses an alternative to the least-squares paradigm known as orthogonal distance regression (ODR).

Global optimization problems, in which we seek the global minimizer of a nonconvex function f, are becoming increasingly important in applications, and the increasing power of computers has made it possible to solve them in many interesting cases. Usually it is essential for the algorithm to exploit the special properties of f. While general algorithmic strategies such as branch-and-bound, smoothing, multistart, etc are useful in designing algorithms for specific applications, general-purpose codes are less useful. Nevertheless, several such codes exist. We mention ASA, the adaptive simulated annealing code discussed in Section 5, LGO, and OPTECH. The new code MCS is implemented in Matlab and is freely available from the site

`http://solon.cma.univie.ac.at/~neum/software/mcs/`

This web site contains a link to a web page with much useful information about global optimization techniques.

Optimization problems involving networks arise in many applications, and a great deal of research has gone into devising algorithms that exploit the structure of such problems. Linear network problems (such as linear minimum-cost flow, or maximum flow) are often solved by general simplex solvers, which sometimes make an effort to detect and exploit the network structure in the way they perform their linear algebra calculations. Other solvers such as NETFLOW and LNOS exploit the structure explicitly, often applying special-purpose algorithms. Nonlinear network problems are addressed by LSNNO.

8. Future Directions

Though the current generation of optimization software has been extremely useful in a wide variety of applications, there is plenty of room for improvement in the next generation. Much of the current software does not make use of advanced software concepts, such as object-oriented design and data-structure independence. Much of it is programmed in Fortran and is invoked through subroutine-call interfaces whose flexibility is limited (though the increasing use of modeling languages is improving the ease of use of many codes). Moreover, most of the software is tied to the standard taxonomy of optimization problems that is reflected in the section titles in this article. While the structure of certain problems can often be exploited by using sparse linear algebra routines at each iteration of the optimization algorithm, there are many other problems

that do not fit neatly into the standard taxonomy, and that would benefit from specialized linear algebra routines or new algorithms.

We believe that object-oriented design and component software ideas will become key features of the next generation of optimization software. Components that implement succesful algorithms such as sequential quadratic programming, augmented Lagrangian methods, and interior-point methods can be programmed in a way that is independent of the special features of the underlying problem and the computational platform on which it is to be executed. These components can be reused efficiently in a wide variety of applications when coupled with modules that implement the linear algebra operations specific to the application type and the architecture of the machine. Powerful toolkits for application areas such as statistics, process design and control, scheduling, and inverse problems can be built up from components that implement optimization algorithms, specialized and general linear algebra operations, mesh generation algoriths, discretization algorithms for differential equations, visualization techniques, and graphical user interfaces.

Given the continuing improvements in such long-studied areas as linear programming, it seems certain that algorithmic advances will continue to play a role in improvement of the software tools. Moreover, the scope of optimization software will be widened to handle more difficult and computationally challenging problem classes, particularly those involving uncertainty, discrete decision variables, and matrix variables. Areas such as stochastic programming, stochastic integer programming, semidefinite programming, and second-order cone programming have been widely studied, but are not well represented in the current generation of production software. This is because of the complexity of the algorithms, the sophisticated demands they place on modelers, and their high computational requirements. This situation will change in future generations of optimization software as the benefits of these algorithms are demonstrated in applications.

References

1. C. T. Kelley, *Iterative Methods for Optimization*, Frontiers in Applied Mathematics, SIAM, Philadelphia, Penn., 1999.
2. J. J. Moré and S. J. Wright, *Optimization Software Guide*, no. 15 in Frontiers in Applied Mathematics, SIAM, Philadelphia, Pa, 1993.
3. L. A. Wolsey, *Integer Programming*, John Wiley and Sons, 1998.
4. S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM Publications, Philadelphia, 1997.