

OPTIMIZATION CASE STUDIES IN THE NEOS GUIDE

JOSEPH CZYŻYK*, TIMOTHY WISNIEWSKI† AND STEPHEN J. WRIGHT‡

1. Introduction. The point of applied mathematics is that the theoretical and algorithmic developments at the core of the subject are relevant to important applications in the real world. In studying the subject, we learn the usefulness of abstracting individual problem characteristics to a mathematical level. The connection to applications motivates us to tackle many the conceptual difficulties that arise in our study of the mathematics.

In optimization and operations research, case studies have proved to be an effective way to make the connection between theory and algorithms on the one hand and applications on the other. The typical case study encompasses problem description, mathematical formulation, use of algorithms to obtain a solution, and interpretation of the results. The NEOS Guide case studies described in this paper have these features, with the addition of some extra ingredients made possible by the World-Wide Web: accessibility and interactivity. *Accessibility* means simply that the studies can be accessed in seconds (via a few keystrokes and mouse clicks) by anyone with a connection to the Web. The advantage of such an access mode over obvious alternatives (such as obtaining and loading a piece of software from a diskette) may not seem significant, but it can be critical in an age when people have little time to waste. Indeed, the instantaneous nature of access to the Web has been a key element in its success. *Interactivity* gives users the power to define their own problems, within the framework of the case study in question. By relating the performance of the algorithm and properties of the solution to their choice of problem and data, they build intuition about the capabilities and limitations of the algorithm and formulation. Interactivity makes the process active rather than passive, and therefore more fully educational.

Each NEOS case study illustrates one of the standard optimization paradigms: linear programming, convex quadratic programming, integer linear programming, and so on. Each study starts with a brief description of the application and its background, and proceeds to the mathematical formulation of the problem. In some cases, we display the formulation explicitly in terms of AMPL, a mathematical modeling language specifically tailored to optimization problems which can serve as an interface to many different software packages (see Fourer, Gay, and Kernighan [5]). Next, there is a menu or table that allows the users to interact with the case study, for example, by indicating the foods they are prepared to eat (in the case of the diet problem) or by defining their risk tolerance (in the case of portfolio optimization). The solution of the user-defined problem is then computed and displayed, and its significance is explained in terms of the original application. The study concludes with pointers to additional information about the underlying optimization paradigm, algorithm, and software, much of which is found in other areas of the NEOS Guide.

All the case studies can be accessed from the following URL:

* Wrocławska Szkoła Językowa, ul. Kłodnicka 2, 54-217 Wrocław, Poland

† 2515 Plaza Drive, State College, PA 16801, U.S.A.

‡ Optimization Technology Center and Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439, U.S.A. This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

<http://www.mcs.anl.gov/otc/Guide/CaseStudies/>

They are frequently used for class assignments. In some cases, particularly for the simplex applet described in Section 2, they could be used during a lecture or tutorial. Some could even be developed into self-contained course units, the equivalent of one or two classes.

The NEOS case studies are built from a number of tools, including HTML files for the Web pages, perl scripts to drive the interactive aspects, AMPL to interface to optimization software, the graphics package gd 1.2 (a freely available product of Cold Spring Harbor Labs) to display the results, and Adobe Photoshop to add some (strictly nonprofessional) graphic design touches to the Web pages. The underlying solvers include standard optimization packages such as IBM's OSL, together with customized solvers programmed by us in Java.

In the remainder of the paper, we briefly describe three of the NEOS Guide case studies, including the real-world application and the optimization tools (mathematical and computational) that are used to solve it. We show how the user interacts with the studies, illustrating our results with a few screen shots. For more details of the examples we discuss in this paper, see the web site

<http://www.mcs.anl.gov/otc/Guide/SIREV/>

Section 2 describes the diet problem, a case study of a general linear programming problem, and mentions our applet implementation of the simplex method. In Section 3, we discuss the portfolio optimization problem, a case study for convex quadratic programming; and in Section 4, we describe the cutting-stock problem, a particular type of integer linear program.

2. The Diet Problem. Soon after the end of World War II, the U.S. Army commissioned its mathematicians to find a one-day ration for its personnel that met their nutritional requirements while minimizing cost. We know this problem as the diet problem, and it is still in use today as a classic case study for linear programming in many first courses in operations research.

The arrival of optimization as a distinct discipline is usually dated from the late 1940s, when George Dantzig invented the simplex algorithm for linear programming. This enormously successful algorithm, still the most widely used algorithm for linear programming, provided economists with a new tool for solving problems previously thought of as intractable. Moreover, it was an ideal application for that newly invented wonder, the electronic digital computer. We mention a Java implementation of the simplex method later in this section.

Linear programming is the problem of minimizing a linear function of real variables over a set of linear constraints. It can be stated algebraically as follows:

$$(1) \quad \min_{x \in R^n} c^T x \quad \text{subject to } Ax = b, x \geq 0,$$

where $c \in R^n$ is known as the cost vector, $A \in R^{m \times n}$ is the constraint matrix, and $b \in R^m$ is known as the dual cost vector. The particular formulation of the problem in (1) is by no means the only one possible, but it is a useful one for developing theory and algorithms and is referred to as the *standard form*. Any optimization problem with a linear objective and linear constraints (linear inequality constraints, linear equality constraints, bounds) can be reformulated equivalently as a standard-form problem by a series of simple algebraic manipulations, including the insertion of "slack" and "surplus" variables. Texts on linear programming abound; see, for example, Chvátal [2].

In the diet problem, the variable x_i represents the number of servings of food i to be eaten in a single day. The corresponding cost vector component c_i represents the cost per serving of food i . The bound constraints $x \geq 0$ enforce the obvious condition that the quantities of each food to be eaten must be nonnegative. (The alternative does not bear thinking about.) The constraints arise from nutritional considerations, and possibly also matters of taste and digestibility. For example, if Q_i^A denotes the amount of Vitamin A per serving of food i , then the total Vitamin A consumed in one day would be $\sum_{i=1}^n Q_i^A x_i$. A requirement that this quantity be at least equal to some value q_i^A is modeled as the following inequality constraint:

$$(2) \quad \sum_{i=1}^n Q_i^A x_i \geq q_i^A.$$

To fit this constraint into the standard form (1) (where all constraints with the exception of the bounds $x \geq 0$ are equality constraints), we define a slack variable s^A to be the amount by which the left-hand side in (2) exceeds the right-hand side. The formula (2) can then be stated equivalently as

$$(3) \quad \sum_{i=1}^n Q_i^A x_i - s^A = q_i^A, \quad s^A \geq 0.$$

Other nutritional constraints also enter the problem, including upper and lower limits on the intake of other vitamins, salt, calories, and fat. The objective function is simply the total cost of the day's ration, which is

$$(4) \quad \sum_{i=1}^n c_i x_i,$$

where c_i is the cost per unit of food i .

The diet problem case study is accessible at

<http://www.mcs.anl.gov/otc/Guide/CaseStudies/diet/>

In this case study, users select from a large menu of possible food choices the items they are prepared to include in their diet. The number of items they choose determines the number of variables n in the problem. (Because of the use of slack variables, the value of n in the actual problem formulation usually is larger than the number of items chosen.) Information on nutritional content of each item was gleaned from the U.S. Department of Agriculture Web site. Prices per unit serving (that is, components of the cost vector c) were in most cases obtained from the same source, though we visited the local supermarket to obtain some of the prices. Government pricing for some items (most notably, butter) is curiously low, leading to a prevalence of these items in many of the calculated diets.

After selecting the food items they are prepared to eat, users are offered the opportunity to modify the nutritional constraints and to impose their own constraints. This process implicitly modifies the problem data (A , b , and the value of m) in (1). For instance, a user may decide that they are not prepared to eat more than two bananas per day, or may decide that at least one tomato per day should be included in their diet. They could also decide to dispense altogether with the upper limit on fat intake—a useful feature for ice-cream fans! This constraint adjustment process implicitly modifies the problem data (A , b , and the value of m) in (1).

The user now clicks a “solve” button. The next Web page now displays either the optimal diet, or else a message indicating that the selection of foods provided by the user was insufficient to generate a feasible diet. In the latter case, the selection of foods and/or the constraints imposed are such that no combination of food quantities meets all the constraints. In this situation, the case study suggests some food items that can be added to the roster of possibilities to yield a feasible problem. The user can then go back to the menu selection page, click on these foods (or some others), and try again.

If the optimal diet was found, a table indicates the quantities and costs of each food item and the total cost. Usually, not all foods chosen by the user appear in the final diet; some of the variables x_i in the model (1) are zero at the solution. In addition, there are graphical depictions (in the form of pie charts, constructed with the graphics utility package `gd 1.2`) of the contributions of each food in the optimal diet to the nutritional constraints: the amount of Vitamin C, the total fat intake, and so on. Finally, there is a page of sensitivity information obtained from the solution of the dual problem.

Often, the optimal diet is not particularly appetizing. The user may not find ten servings of wheat bread per day an exciting prospect, for instance. A few clicks of the “back” button on the Web browser takes one back to the constraints page, where the upper bounds and lower bounds on the food items may be modified and a new optimal diet calculated. This process of adjusting the model and reoptimizing is itself instructive, since it mirrors the way in which optimization software is used in real-world problem solving.

Figure 1 shows sample output for a diet problem trial in which we selected the first 25 foods from column 1 of the roster as foods that we are prepared to include in our diet. Four of these 25 are selected in the optimal diet, which has a daily cost of \$1.04. The figure shows the calories attributable to each of these foods. For further details on the solution of this sample problem, see the web site mentioned earlier.

Duality. The problem (1) gives rise to a rich body of theory, which derives mainly from two sources: the relationship between the algebraic and geometric properties of the problem (1) and the algorithms for solving it, and the relationship between the problem (1) and another linear program known as the *dual*. The dual takes the data objects A , b , and c that define (1) and uses them to define a new linear program as follows:

$$(5) \quad \max_{\pi \in R^m} b^T \pi \quad \text{subject to } A^T \pi \leq c.$$

An alternative (and equivalent) statement of the dual is obtained by introducing slack variables s for the constraints $A^T \pi \leq c$ and writing

$$(6) \quad \max_{\pi \in R^m, s \in R^n} b^T \pi \quad \text{subject to } A^T \pi + s = c, s \geq 0.$$

The duality theory of linear programming shows, among other things, that the optimal values of the primal and dual problems are the same. That is, if x^* solves (1) and π^* solves (5), we have

$$(7) \quad c^T x^* = b^T \pi^*.$$

Further, when x is any vector satisfying $Ax = b$, $x \geq 0$ and π is any vector such that $A^T \pi \leq c$, we have that

$$c^T x \geq c^T x^* = b^T \pi^* \geq b^T \pi.$$

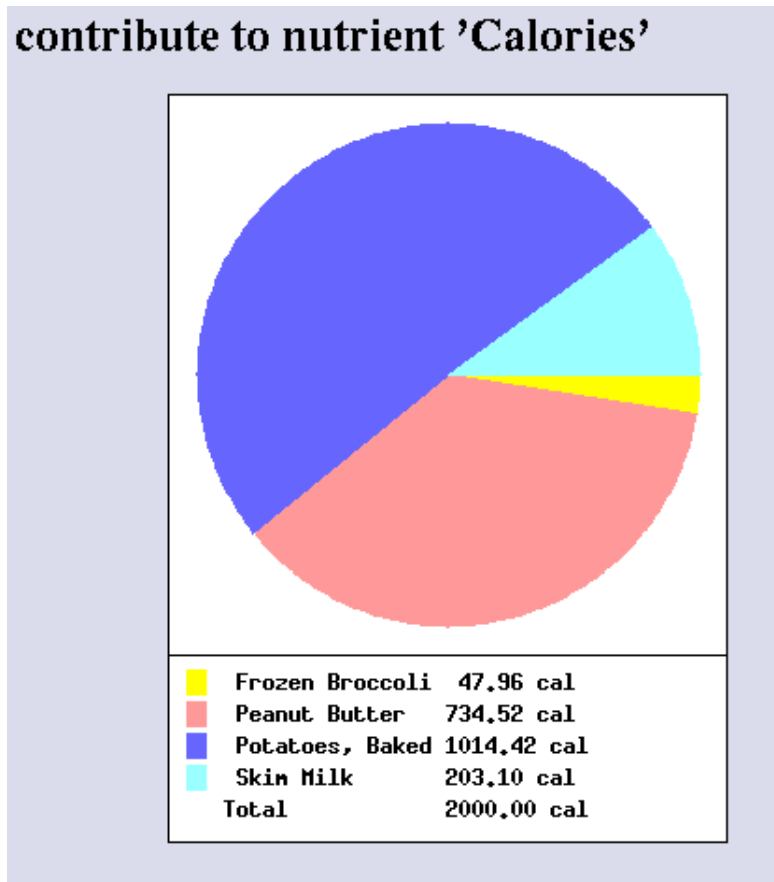


FIG. 1. Sample output: Calorie breakdown for optimal diet when the first 25 items from the menu are selected

The solution of the dual problem (5) is easily recovered once the solution to (1) is known, and vice versa. The dual solution is not just of academic interest; it provides the modeler with important information about the original problem (1). Specifically, it quantifies the sensitivity of the optimal objective value to changes in the elements b_i of the right-hand side of the constraint vector b . If $\phi(A, b, c)$ denotes the optimal objective value $c^T x^*$ for (1), we have that

$$\frac{\partial \phi(A, b, c)}{\partial b_i} = \pi_i^*,$$

where π_i^* is the i th component of the optimal dual vector π^* .

The Simplex Applet. As mentioned above, the simplex method has for many years been the standard tool for solving linear programming problems. For many people, particularly in the financial and economics fields, the simplex method is the only optimization algorithm to which they are ever exposed during their educational and professional lives. Even optimization specialists tend to learn about linear programming and the simplex method well in advance of hearing about other problem classes or algorithms in optimization. Indeed, the ideas behind the simplex method and the techniques used to implement it have had a significant influence on the development of algorithms for nonlinear constrained optimization problems.

The NEOS Guide contains a Java implementation of the simplex method, available as an applet at

<http://www.mcs.anl.gov/otc/Guide/CaseStudies/simplex/>

Users can enter a small linear program and follow the progress of the algorithm in detail via windows that appear on their PC or workstation. The various steps of the algorithm—conversion to standard form, pricing, pivoting and selection of the entering variable—can be viewed individually. The user may even select the variable to enter the basis at each iteration if desired. The main window illustrates current values of the basis components, basis matrix, multipliers, reduced costs, and so on; see Figure 2.

For information about the simplex algorithm, follow the pointers from the Simplex Applet web page, or consult one of the many texts in the area (for example, Chvátal [2]).

3. The Portfolio Optimization Problem. Given a wide range of possible investments, with their (possibly correlated) risks and rewards, how does an investor go about selecting a portfolio that maximizes the expected return while meeting an acceptable standard of risk? Alternatively, what is the portfolio that meets a certain target level of return while minimizing risk? Questions like these are asked daily by both individual and institutional investors, and they lie at the heart of the burgeoning investment advice industry. They were first formulated quantitatively in 1952 by Markowitz [8], also known to computational mathematicians as the inventor of the Markowitz pivot criterion for sparse matrix factorization.

Today's investor can choose from a huge range of possible securities—bills and notes from the U.S. Treasury, corporate and municipal bonds, individual stocks from exchanges around the world, mutual funds, and so on—each of which has its own potential risks and rewards. The returns on some securities are unspectacular but steady, while the returns on others are higher over the long term but are subject to wild fluctuations. Moreover, the returns on different securities are often correlated, positively or negatively. For instance, the stock prices of gold mining companies and

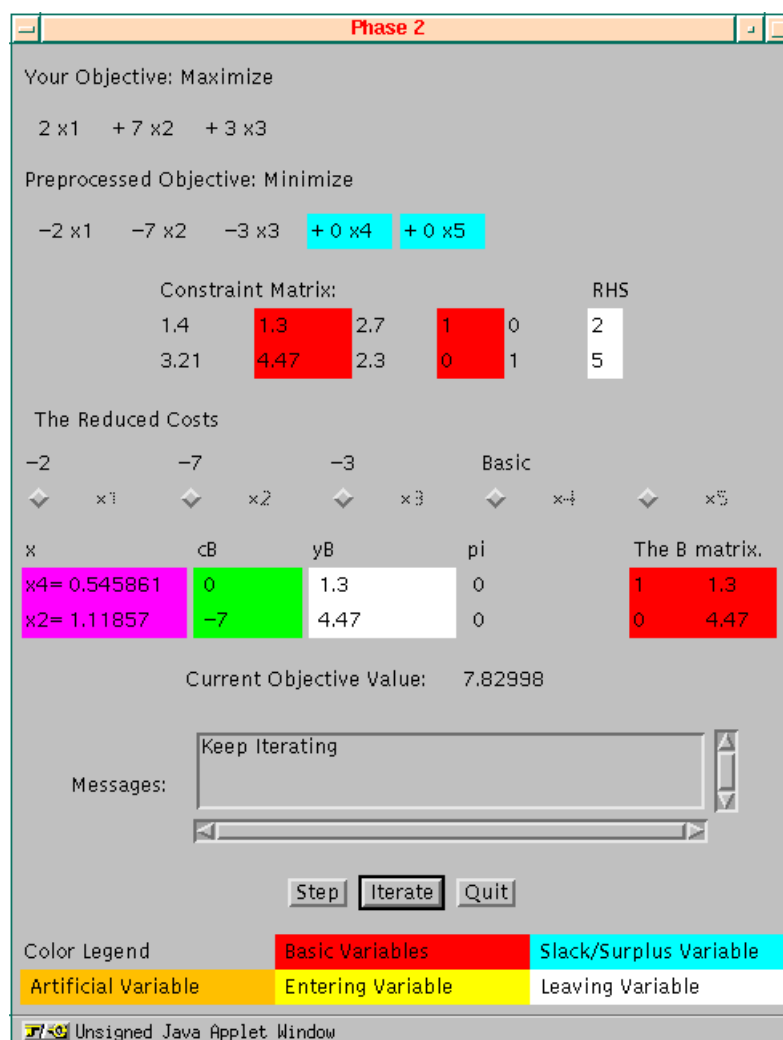


FIG. 2. Main window of the simplex tool

of gold itself tend to increase and decrease more or less synchronously, that is, they are positively correlated. On the other hand, returns on gold investments may be negatively correlated for some reason with returns on the stock of paper manufacturers. By taking advantage of these correlations and of the expected average returns and fluctuations associated with each individual security, we can often design portfolios that achieve expected returns competitive with the best individual securities, but at a lower level of risk.

To see how a portfolio can reduce risk, consider the simple case of two securities—call them A and B—for which the fluctuations on return are negatively correlated. Both investments tend to be profitable over time, but the return on A usually increases whenever the return on B decreases, and vice versa. It is easy to perceive that the returns on a portfolio that mixes A and B will tend to fluctuate less than the returns on an investment in A and B alone. The individual fluctuations in the returns on A and B will tend to cancel each other out, while the long-term average return on the mixed portfolio will (if the mix is chosen properly) be competitive with the long-term average return on both A and B individually. In other words, by mixing A and B appropriately, we can reduce the risk appreciably without affecting the expected return.

More generally, suppose that we have n securities, labeled by the index $i = 1, 2, \dots, n$, and let X_i be a random variable that represents the return on this security during the next month. Naturally, we have no hope of solving the portfolio optimization problem without some information about each X_i , so we assume that each X_i is normally distributed and that values of the expected return $\mu_i = E[X_i]$ and the variance $\sigma_i^2 = E[(X_i - \mu_i)^2]$ are available for each security i . (The variance is a measure of risk, since it quantifies the fluctuation of the random variable about its expected value.) Moreover, we assume that values of the correlations between each pair of securities, defined by

$$\rho_{ij} \stackrel{\text{def}}{=} \frac{E[(X_i - \mu_i)(X_j - \mu_j)]}{\sigma_i \sigma_j}, \quad i, j = 1, 2, \dots, n,$$

are also given. (Note that $\rho_{ii} = 1$ for all i .) We return in a moment to the question of how the μ_i , σ_i , and ρ_{ij} , $i, j = 1, 2, \dots, n$ are determined in practice.

Suppose that we construct a portfolio by allocating a fraction w_i of the available resources to security i , for each i . Suppose too that we disallow short selling (which is the practice of selling a stock that you don't currently hold, in the hope of buying it some time later at a lower price). Under these assumptions, the value of w_i must lie in the range $[0, 1]$, with $\sum_{i=1}^n w_i = 1$. The expected return on this portfolio is

$$(8) \quad E \left[\sum_{i=1}^n w_i X_i \right] = \sum_{i=1}^n w_i E[X_i] = w^T \mu,$$

where $w = (w_1, w_2, \dots, w_n)^T$ and $\mu = (\mu_1, \mu_2, \dots, \mu_n)^T$. The variance for the portfolio is

$$(9) \quad \sum_{i=1}^n \sum_{j=1}^n w_i w_j \sigma_i \sigma_j \rho_{ij} = w^T Q w,$$

where Q is the matrix whose (i, j) element is

$$Q_{ij} = \rho_{ij} \sigma_i \sigma_j.$$

In general, we want to choose the allocation vector w so as to keep the expected return (8) large while keeping the variance (9) small. Exactly how these two objective should be traded off is a matter for the investor to decide. Those with a high tolerance for risk will weight the expected return more heavily, while more conservative investors will be more concerned with minimizing variance. We define a “risk tolerance parameter” κ to quantify the relative weighting of the two aims, and we define our optimization problem as follows:

$$(10) \quad \min_w \frac{\kappa}{2} w^T Q w - \mu^T w, \quad \text{subject to } e^T w = 1, w \geq 0.$$

Here, e is the vector $(1, 1, \dots, 1)^T$, so that the constraint $e^T w = 1$ simply means that the sum of the fractions is 1, as discussed earlier. Risk-tolerant investors would typically choose κ near zero, while conservative investors would choose some larger positive value for this parameter.

The portfolio optimization problem can be derived from other perspectives as well. For instance, this “balanced” problem is closely related to two other types of portfolio optimization: one that minimizes risk for a given level of return, that is,

$$(11) \quad \min_w \frac{1}{2} w^T Q w \quad \text{subject to } \mu^T w \geq \mu_*, w \geq 0, e^T w = 1,$$

and another that maximizes return for a given level of risk, that is,

$$(12) \quad \max_w \mu^T w \quad \text{subject to } w^T Q w \leq \sigma_*^2, w \geq 0, e^T w = 1.$$

In (11), μ_* is the minimum acceptable return, while in (12), σ_*^2 is the maximum acceptable variance. When the optimality conditions for (11) and (12) are formulated, one can derive interesting relationships between the Lagrange multipliers for the constraints $\mu^T w \geq \mu_*$ and $w^T Q w \leq \sigma_*^2$, and the parameter κ . Another interesting concept is the “efficient frontier,” which is the set of allocation vectors such that there exists no other vector with a higher return for equivalent variance, or lower variance for an equivalent return. In many cases, the efficient frontier is simply the set of minimizers of (10), for each value of κ in the range $[0, \infty)$. For additional details, see the case study web site, whose address is listed below.

We now return to a critical issue: How do we choose the data for the expected returns, variances, and covariances? Possibly the most important source of information is historical data. If we know the monthly performance of each security for some time past—the return on security i in month t was x_{it} , for $t = 1, 2, \dots, N$ —then we can set

$$\mu_i = \sum_{t=1}^N x_{it}/N, \quad \sigma_i^2 = \sum_{t=1}^N (x_{it} - \mu_i)^2/N, \quad \rho_{ij} = \sum_{t=1}^N (x_{it} - \mu_i)(x_{jt} - \mu_j)/(N\sigma_i\sigma_j).$$

Use of historical data to set the parameters makes the implicit assumption that the random variables X_i will continue to behave in much the same way as they did in the past—an assumption that is shaky, to say the least. Some investment advisors weight the historical data to emphasize data points that were collected at points in past business cycles that correspond to current conditions. In addition, there is plenty of room for subjective input—an investor that expects 2% monthly growth in the price of a particular stock with 3% variance can set the data in his model accordingly. As in all other areas of modeling and prediction, the rule of “garbage in, garbage out” applies.

The portfolio optimization case study can be found at

<http://www.mcs.anl.gov/otc/Guide/CaseStudies/port/>

Our set of securities consists of the 30 stocks that make up the Dow Jones Industrial Average (as of mid-1996), together with a U.S. Treasury bill. The latter is assumed to be a risk-free investment, so that $\sigma_i = 0$ and all entries in the corresponding row and column of Q are zero. We obtained data on monthly total returns (including stock price changes and dividends) for the period 1/1/86–12/31/91 from the CRISPA database at the University of Chicago, and we based our model parameters on this historical data alone.

Users of the case study select the stocks they are willing to include in their portfolio. (They can examine the six-year performance of each stock for background information.) They also input the current annual return on short-term U.S. Treasury bills, and choose the value of their risk parameter κ . The case study returns an output page that indicates the optimal portfolio (in tabular and graphical form), together with its expected return and variance. The user can then experiment with other values of κ to see how the optimal portfolio changes in response to changes in this parameter.

The underlying optimization problem is a convex quadratic program. By definition, the matrix Q in (10) is positive semidefinite and the feasible region is bounded, so the problem certainly has a solution, though not necessarily a uniquely defined one.

For additional information on portfolio optimization, see the books by Markowitz [9, 10] and the article by Perold [12]. A number of enhancements to the basic problem (10) have been proposed by various researchers and practitioners. For one thing, constraints on the allocation vector may be desirable; we may want to limit our exposure to technology stocks to 25% of our portfolio, for example. We can allow short-selling of some securities in the model by moving the lower bound from 0 to some negative number. We may want to limit the number of different securities in our final portfolio to a reasonable number (10 or 20, say), since a solution that requires us to invest in 500 different financial instruments is not of much practical use. We may also want to include transaction costs in the objective function, or to impose a minimum balance on any securities that are included in the final portfolio. Some of these requirements increase the complexity of the problem considerably (see Bienstock [1]) and, in particular, may necessitate the use of integer programming algorithms.

Figure 3 shows the composition for the optimal portfolio when we choose 29 of the 30 stocks as possible investments—all except Philip Morris—and set the bond interest rate to 5.0% and the risk tolerance parameter κ to 15. The optimal portfolio is quite conservative; it suggests 43% of assets in bonds. However, its predicted annual return of about 23% is high, while the conservative nature of the bond investment makes the standard deviation of the portfolio less than half the deviations of any of the three stocks represented in the portfolio.

4. The Cutting-Stock Problem. The cutting-stock problem is a classic linear programming problem, with important applications, that can be solved efficiently with a technique known as column generation. In fact, it is an *integer* linear program: All components of the solution are required to be integers.

An excellent description of the problem and its history is given by Chvátal [2, Chapter 13]; we present just a brief summary here. The problem originated in paper and textile mills that manufacture rolls of material in fixed widths, called “raws.” Customers place orders for quantities of rolls with various widths, which are referred to as “finals.” Each raw is sliced (with a knife) into one or more finals according to

Makeup of Optimal Portfolio			
Name	Avg Return (monthly, pct)	Std Deviation	Pct of Optimal Portfolio
COCA COLA CO	2.885	6.574	13.3
MERCK & CO INC	3.196	6.606	30.6
TEXACO INC	1.869	6.381	13.1
bond	0.407	0.000	43.1

Optimal Portfolio Statistics	
Avg Return (monthly, pct)	1.78
Standard Deviation	3.02
Avg Return (annualized, pct)	23.57

FIG. 3. Sample output: Stocks in optimal portfolio when bond rate is %5 and risk tolerance parameter is $\kappa = 15$

some pattern. Typically, after one or more finals are cut from a raw, a piece is left over and discarded as waste. The optimization problem is to find a scheme for satisfying all customer orders while using the minimum number of raws. In other words, we look for the mix of slicing patterns, and the number of times each pattern is applied, that minimizes the total number of raws used.

We describe the problem by studying the following specific example from [2]. Suppose that the raws are 110 inches wide and that four orders for finals of different widths have been received. The quantities of each width are shown in Table 1.

TABLE 1
Cutting-stock example input

order number (i)	1	2	3	4
order width (W_i)	43	36	29	13
number ordered (b_i)	297	713	147	301

To meet these orders, each 110-inch roll is cut into one of a number of possible patterns. One pattern would consist of two 43-inch widths and a 13-inch width, leaving an 11-inch remainder that is discarded, since it is too narrow to be used for other finals. A second pattern would consist of two 29-inch widths and a 43-inch width, leaving a 9-inch strip of waste. Numerous other patterns are possible.

We can model this problem as a linear program in which m is the number of distinct order widths, while p is the number of the possible patterns. The entry A_{ij} of the $m \times p$ constraint matrix A would be the number of times that the order width W_i is represented in pattern j . For instance, if the two patterns discussed in the previous

paragraph give rise to the first two columns of A , we would have

$$A = \begin{bmatrix} 2 & 1 & \dots \\ 0 & 0 & \dots \\ 0 & 2 & \dots \\ 1 & 0 & \dots \end{bmatrix}.$$

If we let x_j denote the number of raws that are cut into pattern j , we find that the i th order is satisfied provided that

$$\sum_{j=1}^p A_{ij} x_j \geq b_i.$$

The total number of raws used in this process is simply $\sum_{j=1}^p x_j$, which we can write as $e^T x$, where e is the vector $(1, 1, \dots, 1)^T$. Moreover, common sense dictates that each x_j must be an integer and that it cannot be negative. In summary, the cutting-stock problem can be written as follows:

$$(13a) \quad \min \sum_j x_j = e^T x, \quad \text{subject to } Ax \geq b, x \geq 0,$$

$$(13b) \quad \text{where each element of } x \text{ is an integer.}$$

One difficulty in solving the problem (13) is the requirement of integrality of the solution components. In general, optimization problems that contain discrete variables (such as integers) are much more time-consuming to solve than those in which all variables are real numbers. In this particular case, however, solutions of good quality often can be obtained by simply ignoring the integrality requirement (13b) and adjusting the real numbers obtained by solving (13a) by small amounts to ensure that the orders are met.

The second, more significant difficulty is that there are usually a huge number of different ways to slice a raw into a collection of finals. That is, the number of different patterns—the dimension p in the problem (13)—may be enormous. It is known from the theory of linear programming that a solution x^* of (13a) need have no more than m nonzero components, and in our case m is much smaller than p . It seems wasteful, then, to go to the trouble of actually figuring out all the different patterns, since such a small proportion of them have any impact on the solution. In fact, it is not necessary to do so. The column generation approach, due to Gilmore and Gomory [6, 7], does a good job of identifying the “useful” patterns—the ones most likely to contribute to the solution—while verifying that all the patterns not explicitly calculated are not relevant to the solution. We include a brief description of column generation in the subsection that follows.

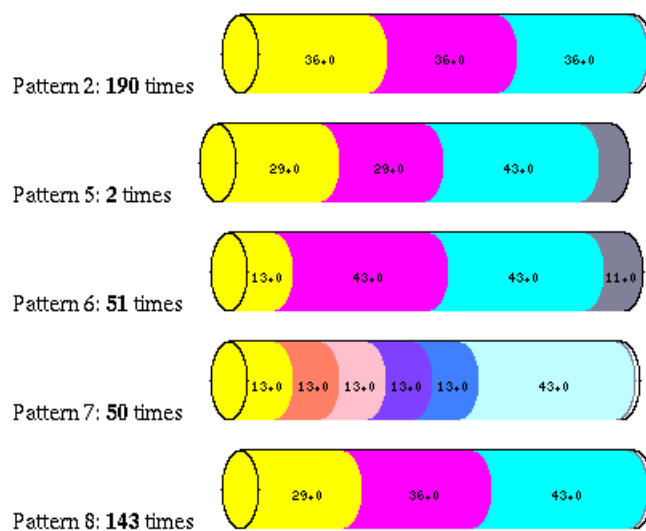
The NEOS cutting stock case study can be found at

<http://www.mcs.anl.gov/otc/Guide/CaseStudies/cutting/>

In the demonstration, users select the number of distinct widths in their proposed order (m , in the notation above) and then enter the widths W_i , the required quantities of each width b_i , and the raw width L . NEOS then solves the problem and responds with the following information (See Figure 4):

- A table showing the patterns considered in generating the final result. These include the initial patterns and the patterns that come from the use of column generation.
- Graphics illustrating the patterns chosen in the final solution and the quantity of each pattern needed to meet the customer orders.

the optimal solution is:



436 raw rolls were used in this scenario.

2.80 % of the paper was wasted during the process.

The data set given by you is:

DATA SET	Width	Order
# 1	43	297
# 2	36	713
# 3	29	147
# 4	13	301

FIG. 4. *Output from the cutting-stock case study*

- The proportion of waste generated in the final pattern.

The user can also examine the AMPL formulation of the column generation procedure. As with most AMPL models, this one is simple and intuitive.

Column Generation. To outline the column generation approach, we need to mention a few points from the duality theory of linear programming (see Section 2). Recalling (5), and accounting for the slightly different form of (13a), we can write its dual as

$$(14) \quad \max b^T \pi, \quad \text{subject to } A^T \pi \leq e, \pi \geq 0.$$

Recall from Section 2 that if x is feasible for (13a) while π is feasible for (14), we have $e^T x \geq b^T \pi$, and that equality of these two objectives is attained at optimality.

Column generation starts by choosing a few patterns, which together correspond to a column submatrix of A . Denoting this submatrix by \bar{A} , and using \bar{x} to denote the corresponding subvector of x , we then solve the following reduced version of (13a):

$$(15) \quad \min_{\bar{x}} \bar{e}^T \bar{x}, \quad \text{subject to } \bar{A} \bar{x} \geq b, \bar{x} \geq 0,$$

for which the dual problem is

$$(16) \quad \max_{\pi} b^T \pi, \quad \text{subject to } \bar{A}^T \pi \leq \bar{e}, \pi \geq 0.$$

(The vector \bar{e} also contains all 1s, but has smaller dimension than the vector e in (13a) and (14).) Denoting the solutions of (15) and (16) by \bar{x}^* and π^* , respectively, we have from the duality theorem that $\bar{e}^T \bar{x}^* = b^T \pi^*$. We now ask the question: Can the solution to the reduced problem be extended to a solution of the original problem (13a) merely by padding out the vector \bar{x}^* with zeros, or must we add some columns to \bar{A} to make it a closer match to the original problem? In the latter case, how do we determine the column to add?

The column generation approach answers these questions by looking for a new pattern that violates the dual feasibility condition $A^T \pi^* \leq e$. That is, it seeks a column of A (which we denote by $z \in \mathbf{R}^m$ and which is not currently represented in the reduced matrix \bar{A}) such that $z^T \pi^* > 1$. To be a valid pattern, the components of z must of course be nonnegative integers and must satisfy the constraint

$$\sum_{i=1}^m W_i z_i \leq L.$$

In other words, the total width of the finals in this pattern do not exceed the width of the raw. To seek the column z of A that actually *maximizes* the violation of the dual feasibility condition, we solve the following problem:

$$(17) \quad \max_z z^T \pi^*, \quad \text{subject to } \sum_{i=1}^m W_i z_i \leq L, \text{ each } z_i \text{ a nonnegative integer,}$$

which is known as a *knapsack problem*. If the solution z of this problem satisfies $z^T \pi^* \leq 1$, we conclude that the dual feasibility condition $A^T \pi^* \leq e$ is satisfied by the original problem. In this case, the reduced solution \bar{x}^* yields (after padding with zeros) a solution of the original problem (13a). Otherwise, when $z^T \pi^* > 1$, the vector z represents the “most troublesome” pattern, that is, the one whose omission is most responsible for the reduced problem (15) being an inadequate substitute for

the full problem (13a). We respond by adding this column z to the matrix \bar{A} (thereby increasing the dimension of the reduced problem by 1), and then repeat the process of solving (15) followed by (17).

The knapsack problem (17) can be solved by a specialized branch-and-bound procedure; see Chvátal [2] for details. The final step in the process is to add the integrality constraints (13b) to the solution of the continuous problem (13a) that arises from the column generation procedure. It usually suffices to make some ad-hoc adjustments, that is, rounding non-integer components of the solution x^* up to the next integer. In the NEOS Guide, we use the more rigorous approach of solving an integer version of the final reduced problem (15), in which the restriction of integrality is included. This problem is much smaller than the full problem (13) (since just a small proportion of the p columns of A are represented in the final \bar{A}) and so is not too time-consuming to solve.

5. Other Features of NEOS. Apart from the case studies, the NEOS Guide contains comprehensive information about optimization software and algorithms. The Software Guide area describes about 120 codes, packages, and modeling languages, outlining their algorithmic capabilities and hardware requirements and giving Web pointers and contact information for the authors and vendors of the software. Classification by problem area makes it easier for users to find the codes with just the right combination of capabilities that they need. (The material in the Software Guide was originally drawn from the 1993 book of Moré and Wright [11], but has been augmented and enhanced.)

Another NEOS Guide feature, the Optimization Tree, contains thumbnail sketches of different areas of optimization. The “leaves” of the tree contain descriptions of a particular problem class, along with a sketch of the main algorithms and pointers to the relevant software packages in the Software Guide. The FAQs for linear and nonlinear programming, maintained for some years by John Gregory on Usenet, are now maintained in the NEOS Guide by Bob Fourer. A repository of test problems is in the early stages of preparation.

The NEOS Guide remains under continuous development. Our intent is for it to be a community resource for optimization, and we always solicit input from our colleagues to help fill the many gaps that remain in our coverage. The Guide can be accessed at

<http://www.mcs.anl.gov/otc/Guide/>

Another branch of NEOS, the NEOS Server, is a facility for solving optimization problems remotely over the Internet. The Server now supports solvers for many classes of problems, including linear and nonlinear constrained optimization, unconstrained and bound-constrained optimization, nonlinear complementarity, and linear network optimization. Users communicate with the Server via email, the Web, or a Tcl/Tk tool running on their own workstation. They submit data and code (written in Fortran, C, or AMPL) that define their optimization problem, and in some cases choose algorithmic parameters for the particular code they are using. The Server then schedules their job for execution on one of its roster of workstations, and finally transmits the results to the user.

For more information on the NEOS Server, visit its Web page at

<http://www.mcs.anl.gov/otc/Server/>

or see the reports of Czyzyk, Mesnier, and Moré [3] and Ferris, Mesnier, and Moré [4].

Acknowledgments. We acknowledge the important contributions of other members of the Optimization Technology Center to this work, including Rob Stubbs (port-

folio optimization study), Yuan Che (cutting-stock study), Bob Fourer, Jorge Moré, and Jorge Nocedal. We are also indebted to our many users—teachers, students, and industrial users of optimization—who provided valuable feedback and suggested many improvements.

We are also pleased to acknowledge the support of the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, and of Northwestern University.

REFERENCES

- [1] D. BIENSTOCK, *Computational study of a family of mixed-integer quadratic programming problems*, *Mathematical Programming*, 74 (1997), pp. 121–140.
- [2] V. CHVÁTAL, *Linear Programming*, W. H. Freeman and Company, New York, 1983.
- [3] J. CZYZYK, M. P. MESNIER, AND J. J. MORÉ, *The network-enabled optimization system (NEOS) server*, Preprint MCS-P615-1096, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., October 1996.
- [4] M. C. FERRIS, M. MESNIER, AND J. J. MORÉ, *NEOS and Condor: Solving optimization problems over the Internet*, tech. rep., MCS Division, Argonne National Laboratory, Argonne, Ill., 1998.
- [5] R. FOURER, D. M. GAY, AND B. W. KERNIGHAN, *AMPL: A Modeling Language for Mathematical Programming*, The Scientific Press, South San Francisco, Calif., 1993.
- [6] P. C. GILMORE AND R. E. GOMORY, *A linear programming approach to the cutting-stock problem*, *Operations Research*, 9 (1961), pp. 849–859.
- [7] ———, *A linear programming approach to the cutting-stock problem—Part II*, *Operations Research*, 11 (1963), pp. 863–888.
- [8] H. MARKOWITZ, *Portfolio selection*, *Journal of Finance*, 8 (1952), pp. 77–91.
- [9] ———, *Mean-Variance Analysis in Portfolio Choice and Capital Markets*, Basil Blackwell, New York, N.Y., 1987.
- [10] ———, *Portfolio Selection: Efficient Diversification of Investments*, Basil Blackwell, Cambridge, Mass., 1991.
- [11] J. J. MORÉ AND S. J. WRIGHT, *Optimization Software Guide*, no. 15 in *Frontiers in Applied Mathematics*, SIAM, Philadelphia, Pa, 1993.
- [12] A. F. PEROLD, *Large-scale portfolio optimization*, *Management Science*, 30 (1984), pp. 1143–1160.