MATHEMATICAL PROGRAMMING

Mathematical programming is an interdisciplinary branch of mathematical science, computational science, and operations research that seeks to answer the question, "What is best?," for problems in which the quality of any answer can be expressed as a computable value. In the context of *mathematical programming,* the term *programming* does not denote a particular type of computer programming, but is synonymous with the word *optimization* contemporarily or *optimal planning* originally. In the 1940s, the term programming was used to describe the planning or scheduling of activities within some large organization. Programmers found that they could represent the amount or level of each activity as a variable whose value was to be determined. They then could mathematically describe the restrictions inherent in the planning or scheduling problem as a set of equations or inequalities involving the variables. A solution to all of these constraints would be considered an acceptable or feasible plan or schedule.

It was soon found that modeling a complex operation simply by specifying constraints is hard. If there were too few

constraints, many inferior solutions could satisfy them; if there were too many constraints, desirable solutions were ruled out, or in the worst case no solutions were possible. The success of programming ultimately depends on a key insight that provided a way around this difficulty. One could specify, in additional to the constraints, a measure of performance or objective that is a function of the variables or activities, such as cost or profit, that could be used to decide whether one solution was better than another. Then, a best or an optimal solution is the one that gives the best possible value, that is minimum or maximum value, of the objective function while satisfying all constraints. The term *mathematical programming,* which is interchangeable with *mathematical optimization,* came to be used to describe the minimization or maximization of an objective function of many variables, subject to constraints on the variables.

Not much was known about this field before 1940. For one thing, computers are necessary since applications usually require extensive numerical computation. However, there were some very early theoretical contributions; for instance, in the last century Cauchy described the method of steepest ascent (up a mountain) in connection with a system of equations (derivatives equated to zero). The field began to flourish in the 1940s and 1950s with the introduction and development of the very important branch of the subject known as *linear programming*—the case where all the costs, requirements, and other quantities of interest are terms strictly proportional to the levels of the activities or sums of such terms. In mathematical terminology, the objective function is a linear function, and the constraints are linear equations and inequalities. Such a problem is called a linear program. The term linear programming is referred to as the process of setting up a linear program and solving it.

Linear programming is without doubt the most natural mechanism for formulating a vast array of problems with modest effort and today a standard tool that has saved many thousands or millions of dollars for most companies or businesses of even moderate size in the various industrialized countries of the world. Its huge success is partially due to the facts that the mathematics involved is simple and easy to understand and that its first computational method, known as the simplex method, has been extremely successful in practice. But it seems that the popularity of linear programming lies primarily with the formulation phase of analysis rather than the solution phase. In fact, most existing optimization applications can be categorized more or less as some sort of optimal allocation of scarce resources in which a great number of constraints and objectives that arise in the real world are indisputably linear, especially in the area of managerial economics.

In spite of the broad applicability of linear programming, the linearity assumption is sometimes too unrealistic since many physical and economic phenomena in the world surrounding us are nonlinear. For many real-world problems in which functional relationships are nonlinear and that involve interactions between the problem variables, linear programming models are not sufficient to describe the relevant problem complexities. If instead some nonlinear functions of the variables are used in the objective or constraints, the problem is then called a nonlinear program. Most optimization problems encountered in engineering are of this nature. Solving such problems is harder, but in practice often achievable. Although the study of computational methods for solving nonlinear programs began in the 1960s, many effective algorithms that are able to solve problems with thousands of variables have been developed.

The solution of a linear or nonlinear program can be fractional. For some applications a fractional solution makes perfect sense; a financial investment decision expressed as a fraction of a large unit, say $1.3 million in portfolio selections is such an example. There are applications, however, in which fractional solutions do not make much sense. For example, the optimal solution of an airline scheduling model may be to fly 1.3 airplanes from city A to city B, which, while mathematically correct, is in reality utter nonsense. When we impose the extra restriction on a linear or nonlinear program that some or all variables must take on integral values, we obtain a mixed or pure integer linear or nonlinear program that in general is much harder to solve than its continuous counterpart. Nevertheless, a combination of faster computer technology and more sophisticated methods has made large integer programs increasingly tractable in recent years.

Mathematical programming has undergone rapid development in recent years and grown into a subject of many branches. This article aims at providing general background information in mathematical programming and presenting some basic notions and results in this subject. The emphases are linear, nonlinear, and integer programs. We shall begin in the first section with the so-called diet problem, a classical real-world optimization problem and discuss various ways to model this problem. The second section is concerned with the classification of optimization problems and their standard forms. The section entitled "Applications and Practicalities" gives a brief discussion on some aspects of solving optimization problems in practice and provides some information necessary to those who want to use software to solve optimization problems. A brief overview of important, well-known, and heavily used results in two typical classes of optimization problems, the smooth nonlinear programming problem and integer linear programming problem, is given in the section titled "Basic Theory." In the next section, key algorithms for solving these two classes of problems are reviewed. Finally, in the last section standard textbooks and references on further reading in this subject are provided, including several informative Internet resources related to this subject.

THE DIET PROBLEM

In this section we shall discuss the diet problem, a classical real-world problem that falls into the category of optimal allocation of scarce resources. In order to be solved by computerized optimization algorithms, a real-world problem must be stated in a very rigid algebraic form. We shall analyze the various characteristics of problem situations to formulate the diet problem in several different forms.

Problem Definition

Suppose that prepared foods of the kinds shown in Table 1 at the market are available at the prices per package indicated. These foods provide percentages, per package, of the minimum daily requirements for vitamins A, B, and C for an average individual, as displayed in Table 2. The diet problem is to find the most economical combination of packages that will

Table 1. Food Prices Per Package

Identifier	Food	Price
B	Beef	\$3.49
F	Fish	\$2.99
C	Cheese	\$1.49
M	Meatloaf	\$1.99
S	Spaghetti	\$2.49

meet the basic minimum nutritional requirements for good health on a weekly basis—that is, at least 700% of the daily requirement for each nutrient. Such a problem might, for example, be faced by the dietician of a large army.

Linear Programming Model

The diet problem can be formulated as a linear program. Let us denote by x_B the number of packages of beef to be purchased, x_F the number of packages of fish, and so forth. Then the total cost of diet is

$$\text{Total cost} = 3.49x_B + 2.99x_F + 1.49x_C + 1.99x_M + 2.49x_S$$

The total percentage of requirement of vitamin A is given by a similar formula, except that x_B, x_F, and so forth are multiplied by the percentage instead of the cost per package:

$$\text{Total percentage of vitamin A weekly requirement met} = 60x_B + 40x_F + 5x_C + 70x_M + 25x_S$$

This amount needs to be greater than or equal to 700%. Similar formulas are needed for other vitamins, and each of these is required to be greater than or equal to 700%. Putting these all together, we have the following linear program for the diet problem:

Minimize total cost
Subject to
  Total percentage of vitamin A weekly requirement met $\geq 700\%$
  Total percentage of vitamin B weekly requirement met $\geq 700\%$
  Total percentage of vitamin C weekly requirement met $\geq 700\%$

or in mathematical terms,

Table 2. Vitamin Requirements

	A	B	C
Beef	60%	25%	20%
Fish	40%	45%	40%
Cheese	5%	20%	20%
Meatloaf	70%	29%	30%
Spaghetti	25%	40%	49%

$$\begin{aligned}&\text{Minimize } 3.49x_B + 2.99x_F + 1.49x_C + 1.99x_M + 2.49x_S\\&\text{Subject to}\\&60x_B + 40x_F + 5x_C + 70x_M + 25x_S \geq 700\\&25x_B + 45x_F + 20x_C + 29x_M + 40x_S \geq 700\\&20x_B + 40x_F + 20x_C + 30x_M + 49x_S \geq 700\\&x_B \geq 0,\ x_F \geq 0,\ x_C \geq 0,\ x_M \geq 0,\ x_S \geq 0\end{aligned} \tag{1}$$

Note that we have added lower bounds on the variables at the end in order to have an adequate description of the problem since it does not make sense to purchase fewer than zero packages of a food.

Solving a linear program with less than thousands of variables is considered rather trivial nowadays. The solution of the above linear program is $x_B = 0$, $x_F = 0$, $x_C = 0$, $x_M \approx 5.06$, $x_S \approx 13.83$ with the total cost $\approx$ \$44.51, found in less than 0.01 s on a personal computer (PC) using a commercial linear programming software package. Thus the cost is minimized by a diet of about 5.06 packages of meatloaf and 13.83 packages of spaghetti. But this solution does not seem very much balanced. You can check that it neatly provides about 700% of the requirement for vitamins A and B, but about 830% for vitamin C, a bit more than necessary.

Alternative Linear Programming Models

One might guess that a solution for a more balanced diet would be generated by improving the model Eq. (1). There are at least two quick tricks. The first one is to limit the total percentage of requirement of all vitamins, say, in the range of 700% to 770%, which results in the following model with additional upper bounds on the constraints:

$$\begin{aligned}&\text{Minimize } 3.49x_B + 2.99x_F + 1.49x_C + 1.99x_M + 2.49x_S\\&\text{Subject to}\\&770 \geq 60x_B + 40x_F + 5x_C + 70x_M + 25x_S \geq 700\\&770 \geq 25x_B + 45x_F + 20x_C + 29x_M + 40x_S \geq 700\\&770 \geq 20x_B + 40x_F + 20x_C + 30x_M + 49x_S \geq 700\\&x_B \geq 0, x_F \geq 0,\ x_C \geq 0,\ x_M \geq 0,\ x_S \geq 0\end{aligned} \tag{2}$$

The second is to require the amount of each vitamin to equal 700% exactly. The resultant model is simply to replace each $\geq$ sign with an $=$ sign in the constraints of Eq. (1), that is,

$$\begin{aligned}&\text{Minimize } 3.49x_B + 2.99x_F + 1.49x_C + 1.99x_M + 2.49x_S\\&\text{Subject to}\\&60x_B + 40x_F + 5x_C + 70x_M + 25x_S = 700\\&25x_B + 45x_F + 20x_C + 29x_M + 40x_S = 700\\&20x_B + 40x_F + 20x_C + 30x_M + 49x_S = 700\\&x_B \geq 0, x_F \geq 0,\ x_C \geq 0,\ x_M \geq 0,\ x_S \geq 0\end{aligned} \tag{3}$$

Solving Eqs. (2) and (3), respectively, gives the following solutions. For Eq. (2), the total cost $\approx$ \$45.32, and the solution $x = (x_B = 0, x_F \approx 4.30, x_C = 0, x_M \approx 4.08, x_S \approx 9.71)$; for Eq. (3), the total cost $\approx$ \$45.88, and the solution $x = (x_B = 0, x_F \approx 9.35, x_C = 0, x_M \approx 2.92, x_S \approx 4.87)$. The optimal solutions for diet do become more balanced now as one can see the requirements for vitamins A, B, and C provided by the solution of Eq. (2) are at the level of 700%, 700%, and 770%, respectively. As for the solution of Eq. (3), it is in fact required by the

constraints to be an exactly balanced diet. One can check that, however, since the constraints became more and more restrictive, the total cost went up from \$44.51 for model Eq. (1) to \$45.32 for model Eq. (2) and further to \$45.88 for model Eq. (3).

Integer Programming Model

To be really rigorous, one might insist that solutions of the diet problem must be integer-valued as foods are sold in the unit of one package. A straightforward way to obtain integral solutions is to round off the fractional variables of solutions obtained to their nearest integers. While this might be satisfactory in certain situations, a better alternative is to make the use of integer programming technique. To this end, the problem must be formulated as an integer program. For the diet program, this can be easily done by imposing one *additional* constraint that all variables must be integer-valued. The corresponding integer program for linear program Eq. (1) looks like

$$\begin{aligned}
&\text{Minimize } 3.49x_B + 2.99x_F + 1.49x_C + 1.99x_M + 2.49x_S \\
&\text{Subject to} \\
&\quad 60x_B + 40x_F + \ 5x_C + 70x_M + 25x_S \geq 700 \\
&\quad 25x_B + 45x_F + 20x_C + 29x_M + 40x_S \geq 700 \\
&\quad 20x_B + 40x_F + 20x_C + 30x_M + 49x_S \geq 700 \\
&\quad x_B \geq 0, x_F \geq 0, x_C \geq 0, x_M \geq 0, x_S \geq 0 \\
&\quad x_B, x_F, x_C, x_M, x_S \text{ are integers}
\end{aligned} \tag{4}$$

Solving this integer program using an integer programming software yields the total cost $\approx$ \$44.81, and the solution is $x = (x_B = 0, x_F = 0, x_C = 0, x_M = 5, x_S = 14)$. Note that this solution is exactly the result of rounding off the fractional variables (recall that $x_M \approx 5.06$, $x_S \approx 13.83$) of the solution of Eq. (1) to the nearest integers.

The solution for the integer program counterpart of linear program Eq. (2) is the total cost = \$45.32 and $x = (x_B = 0, x_F = 5, x_C = 0, x_M = 4, x_S = 9)$. Note, however, that this solution cannot be obtained by rounding off the fractional variables of solution of Eq. (2) to the nearest integers.

In continuing to solve the corresponding integer program for Eq. (3), we found that this time, however, our software reported an infeasibility of the model, meaning there are no integer-valued variables that would satisfy all the constraints.

Probabilistic Model

Modeling many real-world problems is complicated by the fact that the problem data cannot be known accurately for a variety of reasons. The simplest reason is due to measurement error. Other reasons could be that problem data are stochastic in nature as in many engineering systems and that some data represent information about the uncertain future as in many planning problems. Undoubtedly, the quality of solutions of a model depends not only on the accuracy of functional relationships involved but on the quality of the data in the model. As an example let us revisit the diet problem. One might agree that the amount of vitamin content in food is not a constant but some sort of random variable. The percentage of the minimum vitamin daily requirement per package of a food for an average individual is therefore also a random variable. So is the total percentage of a vitamin weekly requirement met. Then what does constraint satisfaction mean in this instance? The constraints of linear programming formulation Eq. (1) in this context simply say that the means of the total percentages of vitamin requirements met must be greater than or equal to 700%, which, from a statistical point of view, is not adequate. In what follows we will use this example to illustrate briefly how a branch of mathematical programming, so-called *stochastic programming,* can be utilized to deal with the randomness involved in the problem data.

We shall deal with the vitamin A nutrient content first. Other nutrient contents can be treated in the same way. Let p_B, p_F, p_C, p_M, and p_S be random variables representing the percentages, per package, of the minimum weekly requirements for vitamin A of foods beef, fish, cheese, meatloaf, and spaghetti, respectively, for an average individual. Assume that they are normally distributed with means α_B, α_F, α_C, α_M, and α_S and variances σ_B^2, σ_F^2, σ_C^2, σ_M^2, and σ_S^2, respectively; further assume that they are independently distributed for the sake of simplicity of discussion. Now let us denote by x_B the number of packages of beef to be purchased, and so forth, as we did before. Then the total percentage of vitamin A weekly requirement met, denoted by $u_{A,x}$, is

$$u_{A,x}(p) = p_B x_B + p_F x_F + p_C x_C + p_M x_M + p_S x_S$$

which is also a normal random variable with mean

$$\alpha_{A,x} = \alpha_B x_B + \alpha_F x_F + \alpha_C x_C + \alpha_M x_M + \alpha_S x_S$$

and variance

$$\sigma_{A,x}^2 = \sigma_B^2 x_B^2 + \sigma_F^2 x_F^2 + \sigma_C^2 x_C^2 + \sigma_M^2 x_M^2 + \sigma_S^2 x_S^2$$

Obviously, it is too demanding to ask for a solution x to satisfy

$$u_{A,x} \geq 700 \text{ for all possible value of } u_{A,x}$$

In fact, this is impossible as $u_{A,x}$ is a normal distribution that ranges over all possible real values. However, one might soon realize that what is really required practically is that

$$\Pr(u_{A,x} \geq 700) \approx 1$$

where $\Pr(u_{A,x} \geq 700)$ denotes the probability of the event $(u_{A,x} \geq 700)$. This is the basic idea behind what is called the *chance-constrained programming.* In general, the chance-constrained approach is to require that

$$\Pr(u_{A,x} \geq 700) \geq l_A, \tag{5}$$

where l_A is the desired probability that the nutrient constraint be satisfied and often called the *acceptance level.* The quantity l_A is a parameter of the model chosen by the modeler and reflects the modeler's attitude towards how often the nutrient constraint should be satisfied. It should always be, of course, less than 1. The value of 0.95 is often a reasonable pick. It is essential to note that, from a computational point of view, the uncertainty introduced by the randomness of problem data is removed in the above formulation Eq. (5) pro-

vided that the distribution function of problem data is known and computable.

Similarly, the chance constraints for vitamins B and C could be formulated as

$$\Pr(u_{B,x} \geq 700) \geq l_B,$$
$$\Pr(u_{C,x} \geq 700) \geq l_C,$$

respectively. Then based on the chance-constrained programming approach, a *certainty* or *deterministic* counterpart of the earlier diet linear model Eq. (1) while the problem data have uncertainty involved is

$$\begin{aligned} &\text{Minimize } 3.49x_B + 2.99x_F + 1.49x_C + 1.99x_M + 2.49x_S \\ &\text{Subject to} \\ &\quad \Pr(u_{A,x} \geq 700) \geq l_A \\ &\quad \Pr(u_{B,x} \geq 700) \geq l_B \\ &\quad \Pr(u_{C,x} \geq 700) \geq l_C \\ &\quad x_B \geq 0,\ x_F \geq 0,\ x_C \geq 0,\ x_M \geq 0,\ x_S \geq 0 \end{aligned} \tag{6}$$

In general, an optimization model involving uncertainty can be converted to a deterministic nonlinear model, the explicit algebraic form of which can sometimes be obtained without much analytical effort. For instance, assume that the means and variances of the vitamin A contents of different foods are given in Table 3.

Then with some analytical manipulation, the chance-constrained constraint for vitamin A

$$\Pr(u_{A,x} \geq 700) \geq l_A$$

with acceptance level $l_A = 0.95$ is equivalent to the following nonlinear constraint:

$$60x_B + 40x_F + 5x_C + 70x_M + 25x_S - 1.645$$
$$(1.24x_B^2 + 0.19x_F^2 + 0.45x_C^2 + 10.2x_M^2 + 4.25x_S^2)^{1/2} \geq 700$$

Similar equivalent forms can be obtained for other vitamins. Thus Eq. (6) can in fact be converted to a deterministic nonlinear program.

Applicability

The diet problem is one of the first optimization problems studied back in the 1930s and 1940s and was first motivated by the Army's desire to meet the nutrient requirements while minimizing the cost. The original diet problem is essentially the same as the version given here and had 77 foods and 9 nutrients. It was first solved to optimality in the National Bureau of Standards in 1947 using then the newly created simplex method for the linear program. The solution process took nine clerks using hand-operated desk calculators 120 work days although it can be solved now in a few seconds on a PC.

Table 3. Vitamin A Contents

	Beef	Fish	Cheese	Meatloaf	Spaghetti
Mean	60%	40%	5%	70%	25%
Variance	1.24	0.19	0.45	10.2	4.25

The optimal solution turned out to consist of wheat flour, corn meal, evaporated milk, peanut butter, lard, beef liver, cabbage, potatoes, and spinach and does not seem to be tasty at all. This is not surprising as taste is in fact not a concern in the problem definition. It is not expected that people actually choose their foods by solving this model. However, similar models might be of practical use as a way for providing feed for animals. More sophisticated and practical versions of the diet problem taking into account color, taste, and variety as well as the frequency of food consumption have been proposed by dieticians and nutritionists since the original diet problem was published.

CLASSIFICATION, MATHEMATICAL FORMULATIONS, AND STRUCTURE OF OPTIMIZATION PROBLEMS

Although optimization problems arise in all areas of science and engineering, at root they have a remarkably similar form. In general, optimization problems are made up of three basic ingredients: an objective function that we want to minimize or maximize; a set of decision variables that affect the value of the objective function; and a set of constraints that allow the variables to take on certain values but exclude others. In mathematical terms, the most general form of optimization problem may be expressed as follows [*Mathematical program* (MP)]:

$$\begin{aligned} &\text{Minimize } f(x) \\ &\text{Subject to} \\ &\quad h_i(x) = 0,\ i = 1, \ldots, p \\ &\quad g_j(x) \leq 0,\ j = 1, \ldots, r \\ &\quad x \in D \end{aligned} \tag{7}$$

where the decision variable x is a n-component vector $x = (x_1, x_2, \ldots, x_n)$; the objective function f, equation constraint functions h_i and inequality constraint functions g_j are real functions; and D is the domain space where x can take values. A point $x \in D$ that satisfies all constraints is called a feasible point and the set consisting of all feasible points is called a feasible set or feasible region. In the rest of the article, when D is not specified it is always assumed that $D = R^n$, and in such case the last inclusion $x \in D$ is omitted.

The formulation [Eq. (7)] is a bit too general and further classification is possible based on problem characteristics and structure. The next section is an overview of classification of optimization problems and their standard forms. The subsequent section is concerned with some classes of optimization problems that have special mathematical structure.

Classification and Formulations

Optimization problems are classified into subclasses based on their intrinsic characteristics and the structure of problem functions. Each subclass has its own standard form and has been studied separately in order to develop the most effective algorithms for solving this subclass of problems. Although there is no unified taxonomy for optimization problems, the following considerations seem to lead to a reasonable classification scheme.

Constrainedness. Constrainedness means whether or not a MP has constraints present. Unconstrained programs are those without constraints and $D = R^n$. Constrained programs are those having at least one constraint.

Nonlinearity. This features the nonlinearity of problem functions. If any of the objective function f and constraints h_i and g_j of a MP is not linear, then it is said to be a nonlinear program. Otherwise, it is a linear program. A linear program has to be constrained or it is trivial since in this case, either it has no solution or it has the whole domain R^n as its solutions. An unconstrained nonlinear program is referred to as an unconstrained optimization program. Constrained nonlinear programs can be further classified according to their increasing nonlinearity as quadratic programs, bound-constrained programs, linearly constrained programs, and general nonlinear programs.

Dimensionality. Based on their dimensionality, optimization problems are classified into one-dimensional and multidimensional problems. Unconstrained optimization problems in which the decision variable x is a single-component vector are called one-dimensional optimization problems and form the simplest but nonetheless a very important subclass of optimization problems.

Integrality. Most discrete optimization problems impose integrality on decision variables. If the variables of a MP are required to take integer values, it is called an integer program. In such a case, we often write $D = Z^n$, where Z^n is the set of integral n-dimensional vectors. If some of the variables must be integers but the others can be real numbers, it is called a mixed-integer program. In many models, the integer variables are used to represent logical relationships and therefore are constrained to equal either 0 or 1. Then we obtain a restricted 0-1 or binary integer program. In a binary integer program, we write $D = B^n$, where B^n is the set of n-dimensional binary vectors. Although most integer programs are NP-complete [see Nemhauser and Wolsey (1)], many of them from the real world can be solved, at least close, to optimality by exploiting problem-specific structures.

Size and Sparsity. The size of a MP is measured in terms of the number of variables (components) of x and the number of constraints and is often, though not always, proportional to the difficulty of solving the problem. Traditionally, mathematical programs are grouped into small-scale, intermediate-scale, and large-scale problems. Today, with present computing power small-scale, intermediate-scale, and large-scale linear programs usually mean having from a few to a thousand variables and constraints, a thousand to a few hundred thousands variables and constraints, and more than a million variables and constraints, respectively. For the much harder nonlinear programs, small scale, intermediate scale, and large scale mean having from a few to a dozen variables and constraints, a few hundred to a thousand variables and constraints, and more than a thousand to tens of thousands of variables and constraints, respectively. Data sparsity is also one of the measures of the problem complexity. For most real-world optimization problems, the sparsity increases as the size gets large.

Data Uncertainty. Most practical optimization models include some level of uncertainty about the data or functional relationships involved. In many cases not much is lost by assuming that these "uncertain" quantities are deterministic either because the level of uncertainty is low or because these quantities play an insignificant role in the model. However, there are cases where these uncertain quantities play a substantial role in the analysis and cannot be ignored by the model builder. To deal with the uncertainty involved in optimization problems, stochastic programming has been developed. Stochastic programs are written in the forms of mathematical programs with the extension that the coefficients that are not known with certainty are given a probabilistic representation that could be a distribution function. To solve them with a computer, stochastic programs are in general converted to some certainty equivalents. Much of the study of stochastic programs lies in the phase of uncertainty modeling and how to convert them to deterministic equivalents.

The considerations previously noted can be used to classify optimization problems into subclasses and standard forms of these subclasses have been used to communicate the problem structure to general optimization software packages. We shall list some of the standard forms as follows.

A standard form of a *linear program* (LP) is

$$\begin{aligned} &\text{Minimize } c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ &\text{Subject to} \\ &\quad a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ &a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ &\qquad\qquad \vdots \\ &a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \\ &\quad x_1 \geq 0,\ x_2 \geq 0,\ \ldots,\ x_n \geq 0 \end{aligned} \tag{8}$$

where the objective and all constraints are linear. In more compact vector notation, this standard form becomes

$$\begin{aligned} &\text{Minimize } c^Tx \\ &\text{Subject to} \\ &\quad Ax = b, \\ &\quad x \geq 0 \end{aligned} \tag{9}$$

Note that an inequality constraint such as

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i$$

can be converted into an equivalent equality constraint below by introducing a slack variable x_{n+1}

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n + x_{n+1} = b_i, x_{n+1} \geq 0$$

Quadratic programs (QP) have linear constraints and quadratic objective functions:

$$\begin{aligned} &\text{Minimize } c^Tx + \frac{1}{2}x^TGx \\ &\text{Subject to} \\ &\quad a_i^Tx = b_i,\ i = 1, \ldots, p \\ &\quad a_j^Tx \leq b_j,\ j = 1, \ldots, r \end{aligned} \tag{10}$$

where G is a symmetric matrix.

The *unconstrained optimization problem* (UOP) is a nonlinear program without constraints:

$$\text{Minimize } f(x) \tag{11}$$

When there are some simple bounds on the components of x, it is then called a *bound-constrained problem* (BCP)

$$\begin{aligned} &\text{Minimize } f(x) \\ &\text{Subject to} \\ &\quad l_i \le x \le u_i,\ i = 1, \ldots, n \end{aligned} \tag{12}$$

Problems with nonlinear objective and linear constraints are called *linearly constrained nonlinear programs* (LCNP)

$$\begin{aligned} &\text{Minimize } f(x) \\ &\text{Subject to} \\ &\quad a_i^T x = b_i,\ i = 1, \ldots, p \\ &\quad a_j^T x \le b_j,\ j = 1, \ldots, r \end{aligned} \tag{13}$$

If at least one of the constraints of a MP is nonlinear and no specific structure can be detected, then it falls into the category of general *nonlinear programs* (NLPs), which has the following form:

$$\begin{aligned} &\text{Minimize } f(x) \\ &\text{Subject to} \\ &\quad h_i(x) = 0,\ i = 1, \ldots, p \\ &\quad g_j(x) \le 0,\ j = 1, \ , \ldots, r \end{aligned} \tag{14}$$

These standard forms for continuous optimization problems readily extend to their corresponding *integer programs* by adding one additional constraint $x \in Z^n$ for pure integer programs, or $x \in B^n$ for binary integer programs, etc.

The general formulation of a *stochastic program* (SP) is as follows:

$$\begin{aligned} &\text{Minimize } E\{f(x, \xi)\} \\ &\text{Subject to} \\ &\quad E\{h_i(x, \xi)\} = 0,\ i = 1, \ldots, p \\ &\quad E\{g_j(x, \xi)\} \le 0,\ j = 1, \ldots, r \end{aligned} \tag{15}$$

where ξ is a random vector and E is the expectation functional. This model is rich enough to include a wide range of applications, and in fact, has been further classified.

The above taxonomy is not unique. For instance, a linear program is also a quadratic program, which is also a nonlinear program. In general, an optimization problem should be put in the most restricted class for which it qualities. This helps in accurately communicating the problem structure to software used for solving the problem.

Structures of Optimization Problems

The mathematical structure of an optimization problem has implications for the existence and behavior of solutions, the difficulty of solving the problem, and the speed of convergence of algorithms. The basic mathematical properties of optimization problems are continuity and smoothness. In this article we shall always assume that the problem functions of interest are continuous and smooth as most of them are in the real world. For problems involving nonsmooth functions, we simply comment that they do arise in practical situations and the study of them has in fact formed a branch of mathematical programming, called *nonsmooth optimization* [see Neittaanmaki (2)]. In this subsection we shall present two classes of optimization problems that have very desirable special structure, the convex program and least-square problem.

Convex Programs. Convexity is a very important structure for mathematical programs. A set C in R^n is *convex* if the line segment joining any two points in C is contained in C. A function f defined over C is said to be a *convex* function if the following inequality holds for any two points x_1 and x_2 in C:

$$f(\lambda x_1 + (1 - \lambda)x_2) \le \lambda f(x_1) + (1 - \lambda) f(x_2) \quad \text{for } 0 \le \lambda \le 1$$

A function g defined over C is *concave* if and only if $-g$ is convex. A mathematical program is called a convex (concave) program if the feasible region is a convex set and the objective function is a convex (concave) function. A fundamental property of a convex program is that local solutions are also global solutions. Note that a linear program is a convex program by definition. Detailed information on convexity can be found in Rockafellar (3).

Least-Square Problems. Least-square problems arise from fitting mathematical models to data. Specifically, the assumption is made that the functional relationship between the variable x and function value y is

$$y = f(x, t)$$

where $t \in R^n$ is a vector of parameters that are to be determined, and the form of f is known. Assume that data

$$(x_i, y_i),\ i = 1, \ldots, m$$

have been collected, and we want to select the parameters t in the model $f(x, t)$ so that

$$f(x_i, t) \cong y_i,\ i = 1, \ldots, m$$

It makes sense to choose the "best" estimate of parameters t by solving

$$\underset{t}{\text{Minimize}} \sum_{i=1}^{m} [y_i - f(x_i, t)]^2$$

This unconstrained optimization problem is called a least-square problem. In some situations, it might be necessary for the parameters t of a least-square problem to be subject to certain constraints. Least-square problems can be solved by specifically designed algorithms that take advantage of the structure, namely, the objective is the sum of squares.

APPLICATIONS AND PRACTICALITIES

The practical applications of mathematical programming are incredibly vast, reaching into almost every activity in which numerical information is processed. To provide a comprehen-

sive account of all the applications would therefore be impossible, but a selection of primary areas in engineering might include the following.

1. *Operations management.* Applications in this area are often related to allocation of scare resources to maximize some measure of profit, quality, efficiency, effectiveness, etc. Other types of applications are the analysis and tuning of existing operations and development of production plans for multiproduct processes. Representatives are airline crew scheduling problems [Hoffman and Padberg (4)] and inventory control problems [Hillier and Lieberman (5)]. Applications of this sort are often modeled by linear and integer programs. Planning problems concerning the future are usually handled by the stochastic programming technique [Murray (6)].
2. *Design of engineering systems.* Applications in engineering design range from the design of small systems such as trusses [McCormick (7)] and oxygen supply systems [Reklaitis, Ravindran, and Ragsdell (8)] to large systems such as aero-engine and bridges, from the design of individual structural members to the design of separate pieces of equipment to the preliminary design of entire production facilities. Nonlinear programming is often the choice of modeling device in engineering design problems.
3. *Regression and data fitting.* A common problem arising in engineering model development is the need to determine the parameters of some semitheoretical model given a set of experimental data. The regression and data fitting problems can be transformed to nonlinear optimization problems.

It should be noted that in considering the application of optimization methods in design and operations, the optimization step is but one step in the overall process of arriving at a good design or an efficient operation. As a powerful tool, optimization technique has to be well understood by the user in order to be employed effectively.

The process of implementing an optimization application generally consists of the following three major steps:

1. Problem definition and model development.
2. Use of software to solve the model.
3. Assessment of the result.

These steps might have to be repeated several times until a desirable result is obtained. In what follows, we shall comment on each of the three steps.

In step 1, several decisions have to be made, including defining the decision variables, creation of a single criterion (or objective) function, determination of the function forms and constraints representing the underlying cause and effect relationships among the variables, collection and quantification of the data involved, etc.

In step 2, the appropriate optimization software needs to be chosen, information about the problem must be communicated to the software through its user interface, and the solution found needs to be interpreted and understood in the context of the problem domain. Since most optimization packages are developed for solving a particular problem category and there is no universal applicable optimization software existing as of today, selecting the appropriate software that is designed for solving the kind of problem in question is important in this step. The book *Optimization Software Guide* by Moré and Wright (9) published in 1993 provides information on about 75 optimization software packages available then covering many categories of optimization problems. The user interface of most optimization software expects the user to provide two kinds of information. The first kind is required and concerned with the problem description such as the algebraic forms and coefficients of objective and constraint functions and the type of the problem. The second kind is usually optional and related to certain algorithmic controlling parameters that are associated with the implementation of the algorithm. Since optimization algorithms involve many decisions that are inherently problem dependent, the controlling parameters allow the user to tune the algorithm in order to make it most effective for the problem being solved. To ease the use of software for the inexperienced or uninterested user, optimization software tends to provide default settings for these parameters.

Step 3 consists of the validation of solutions found and post-optimality analysis (often called sensitivity analysis [Fiacco (10)]). It is a fact of life that even a good optimization algorithm may claim that a solution is found when it is not. One intrinsic reason is that most optimization algorithms are designed to find points that satisfy necessary conditions that do not guarantee optimality. What is worse is that the problem model itself may be ill-posed. For ill-posed problem models, numerical errors such as round-off errors might drive the computed solution far away from the real one, and in such case model modification and data refinement are necessary.

There have been some serious efforts to define *standard input formats* (SIFs) for describing optimization problems of certain category, for example, the MPS format [see Nazareth (11)] that has become the de facto input format for linear and integer programs, the SMPS format proposed by Birge et al. (12) as a standard input format for multiperiod stochastic linear programs, the LANCELOT specification file developed by Conn, Gould, and Toint (13), and the MINOS specification file used by MINOS [Murtagh and Saunders (14)] for general nonlinear programs. These SIFs tend to be very specific and lengthy and are easy to be understood by computer programs but hard to generate and costly to maintain by humans. To get around this difficulty, algebraic modeling languages for mathematical programming, for example, the powerful AMPL developed by Fourer, Gay, and Kernighan (15), began to surge in the recent years. They provide computer-readable equivalents of notation such as $x_i + y_i$, $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$, $i \in S$, etc., that are commonly seen in algebra and calculus and allow the optimization modeler to use traditional mathematical notation to describe objective and constraint functions. These algebraic forms of problem description will then be converted by computer to formats that are understood by optimization algorithms. The use of modeling language has made optimization model prototyping and development much easier and less error-prone than before.

BASIC THEORY

The theory of mathematical programming is incredibly rich. Specialized theories and algorithms have been developed for

all problem categories. In the subject of linear programming alone, there have been thousands of research papers published and dozens of textbooks available. In this section we shall first give a brief overview of different types of optimal solutions and a fundamental existence result. Since it is impossible to expose even the very basic theoretical results for all problem categories here, we have decided to focus on two important and heavily used problem categories, namely, the smooth nonlinear programming problem and integer linear programming problem.

A general deterministic mathematical programming problem can be written in the following format [*General mathematical program* (GMP)]:

$$\begin{aligned} &\text{Minimize } f(x) \\ &\text{Subject to } x \in \Omega \end{aligned} \tag{16}$$

where f is a real function and $\Omega \subset R^n$ is the feasible region. We shall briefly introduce in rigorous terms what we mean by an optimal solution of Eq. 16, and then present an existence result.

Type of Solutions

In general, there are several kinds of optimal solutions to Eq. 16. A point $x_0 \in \Omega$ is said to be a *strict local* (optimal) solution *or* a *weak local* (optimal) solution if there exists a neighborhood N of x_0 such that $f(x_0) < f(x)$ *or* $f(x_0) \leq f(x)$ for all $x_0 \neq x \in \Omega \cap N$. We say that x_0 is a *strict global* (optimal) solution if $f(x_0) < f(x)$ for all $x_0 \neq x \in \Omega$, and a *weak global* (optimal) solution if $f(x_0) \leq f(x)$ for all $x_0 \neq x \in \Omega$. A global solution is also a local solution by definition but not vice versa. For some mathematical programs with special structure, for example, convex programs, a local solution is also a global solution. For general nonlinear programs, however, a local solution might not be a global solution, and finding a local solution is usually much easier than a global one.

Existence of Solutions

A mathematical program may or may not have a global solution. When there does not exist a global solution, it could be due to the fact that the program is infeasible, that is, the feasible set is empty, or that the program is unbounded, that is, the feasible set is not empty but the objective function value is unbounded from below in the feasible set. A basic result of a mathematical program is the well-known theorem of Weierstrass, which states that if the objective function f is continuous and the feasible set Ω is nonempty and compact, then a global solution exists.

The rest of this section will be devoted to two typical classes of optimization problems, the smooth nonlinear programming problems in next subsection and integer linear programming problems in the subsequent subsection.

Smooth Nonlinear Programming Problems

The emphasis of this subsection is the optimality conditions of solutions of optimization problems. We shall confine ourselves mainly to the consideration of local solutions as they are simpler and more fundamental than global solutions. In fact, for nonlinear optimization problems local solutions are often, though not always, satisfactory enough in practical situations. For simplicity of presentation, we always assume in the article that the problem functions in question have the necessary smoothness required in the context.

Optimality Conditions. As we can see from the definition of optimal solutions, optimality of a local solution point is defined by its relationship with other feasible points in contrast to, say, seeking a point where $f(x) = 0$. The verification of optimality directly by the definition cannot be carried out by computers since it would be necessary to evaluate infinitely many neighboring feasible points of a proposed local solution. Fortunately, if the problem functions are smooth enough, it is possible to derive some practical optimality conditions that can characterize local solutions and involve analytical information only at a proposed solution point.

Optimality conditions have fundamental importance in optimization theory and algorithms since they are essential in understanding solution behavior conceptually and optimization algorithms are often motivated by attempts to find points satisfying them. Optimality conditions are of two types: necessary conditions, which must hold at a local solution, and sufficient conditions, conditions that, if satisfied at a feasible point, guarantee that point to be a local solution. To explain these seemingly abstract conditions, we shall begin with a simple case.

Optimality Conditions in the Unconstrained Case. We shall consider the unconstrained optimization problem:

$$\text{Minimize } f(x)$$

The key to the derivation of optimality conditions is to use models that are simple and easy to manipulate to approximate complicated ones. The mathematical ground of the approximation is the basic Taylor-series expansion of problem functions at a point of interest. When f is once differentiable at a point x_0, it can be expanded in its Taylor series about x_0 up to first-order, which gives

$$f(x) = f(x_0) + \nabla f(x_0)^{\mathrm{T}}(x - x_0) + o(\|x - x_0\|)$$

and when f is twice differentiable, the Taylor series up to second-order is

$$f(x) = f(x_0) + \nabla f(x_0)^{\mathrm{T}}(x - x_0) + \frac{1}{2}(x - x_0)^{\mathrm{T}}\nabla^2 f(x_0)(x - x_0) + o(\|x - x_0\|^2)$$

Another technique often used in the derivation of optimality conditions is to consider movement away from a proposed solution point in some given direction or curve that falls in the feasible region and to examine the behavior of problem functions along this direction or curve. Given a direction d, we say that it is a *descent* or *ascent* direction of f at x_0 if

$$\nabla f(x_0)^{\mathrm{T}} d < 0 \text{ or } \nabla f(x_0)^{\mathrm{T}} d > 0$$

And we say that f has negative or positive curvature in d at x_0 if

$$d^{\mathrm{T}}\nabla^2 f(x_0) d < 0 \text{ or } d^{\mathrm{T}}\nabla^2 f(x_0) d > 0$$

Consider the following Taylor series of f along d at x_0:

$$f(x_0 + td) = f(x_0) + t\nabla f(x_0)^{\mathrm{T}} d + \frac{1}{2}t^2 d^{\mathrm{T}} \nabla^2 f(x_0) d + o(t^2)$$

We can see that whether the sign of $\nabla f(x_0)^{\mathrm{T}} d$ is positive or negative (or equivalently, whether d is a descent or ascent direction) determines whether the value of f increases or decreases initially when x moves away from x_0 along d. When $\nabla f(x_0)^{\mathrm{T}} d = 0$, the curvature of f in d at x_0, that is, $d^{\mathrm{T}} \nabla^2 f(x_0) d$, governs the initial behavior of f at x_0 along d. This observation leads to the following optimality conditions.

First-Order Necessary Conditions. If x_0 is a local solution, then $\nabla f(x_0)^{\mathrm{T}} d \geq 0$ for all d, or equivalently, $\nabla f(x_0) = 0$.

Second-Order Necessary Conditions. If x_0 is a local solution, then

1. $\nabla f(x_0) = 0$
2. $d^{\mathrm{T}} \nabla^2 f(x_0) d \geq 0$ for all d, that is, $\nabla^2 f(x_0)$ is positive semidefinite

Second-Order Sufficient Conditions. If a point x_0 satisfies $\nabla f(x_0) = 0$ and $d^{\mathrm{T}} \nabla^2 f(x_0) d > 0$ for all $d \neq 0$, that is $\nabla^2 f(x_0)$ is positive definite, then x_0 is a local solution.

Approximations to the Feasible Region and Nonlinear Programs. Having derived the optimality conditions in the unconstrained case, we now turn our attention to the constrained nonlinear program Eq. (14), where the feasible region Ω is defined by

$$\Omega = \{x \in R^n: \quad h_i(x) = 0,\ i = 1, \ldots, p,\quad g_j(x) \leq 0,\ j = 1, \ldots, r\}$$

Note first that at a point x_0 of interest there are two types of inequality constrains: *active* constraints if $g_j(x_0) = 0$, *inactive* constraints otherwise. For a continuous inactive inequality constraint g_j, if $g_j(x_0) < 0$, then g_j stays that way at least locally, that is, $g_j(x) < 0$ for x near x_0 by virtue of continuity. Therefore, the inequality constraint $g_j(x) \leq 0$ is always satisfied for x near x_0 and so can be ignored locally. Thus it is of some importance to know which inequality constraint is active and which is not at a point of interest. Let A(x) denote the index set of active constraints at x.

Given a feasible point x_0, it appears that linearizing the constraint functions by replacing them with their respective first-order approximations would give a good approximation to Ω around x_0:

$$F(x_0) := \{x \in R^n: \quad h_i(x_0) + \nabla h_i(x_0)^{\mathrm{T}}(x - x_0) = 0,\ i = 1, \ldots, p \quad g_j(x_0) + \nabla g_j(x_0)^{\mathrm{T}}(x - x_0) \leq 0,\ j \in A(x_0)\} \tag{17}$$

where the inactive constraints are ignored, and $h_i(x_0) = 0$ for $i = 1, \ldots, p$, and $g_j(x_0) = 0$ for $j \in A(x_0)$ by the feasibility of x_0. However, this is not always true on account of the fact that the boundary of the feasible region may be curved. To ensure the geometry of Ω around x_0 is adequately captured by $F(x_0)$, a *constraint qualification* is required at x_0. A standard constraint qualification requires that the set $\{\nabla h_i(x_0),\ i = 1, \ldots, p;\ \nabla g_j(x_0),\ j \in A(x_0)\}$ be linearly independent.

Better approximations of Ω around x_0 than $F(x_0)$ can be obtained by using higher-order approximations to the problem functions. For example, by replacing the constraint functions with their respective second-order approximations yields a better one:

$$F^2(x_0) := \{x \in R^n: \quad h_i(x_0) + \nabla h_i(x_0)^{\mathrm{T}}(x - x_0) + \frac{1}{2}(x - x_0)^{\mathrm{T}} \nabla^2 h_i(x_0)(x - x_0) = 0,\ i = 1, \ldots, p \quad g_j(x_0) + \nabla g_j(x_0)^{\mathrm{T}}(x - x_0) + \frac{1}{2}(x - x_0)^{\mathrm{T}} \nabla^2 g_j(x_0)(x - x_0) \leq 0,\ j \in A(x_0)\} \tag{18}$$

There are also ways to approximate the nonlinear program Eq. (14) at x_0. Two readily available approximations to Eq. (14) making the use of the approximations $F(x_0)$ and $F^2(x_0)$ are the following: *First-Order Approximation to NLP*

$$\begin{aligned} &\text{Minimize } f^1(x) := f(x_0) + \nabla f(x_0)^T (x - x_0) \\ &\text{Subject to } x \in F(x_0) \end{aligned} \tag{19}$$

and *Second-Order Approximation to NLP*

$$\begin{aligned} &\text{Minimize } f^2(x) := f(x_0) + \nabla f(x_0)^T (x - x_0) + \frac{1}{2}(x - x_0)^{\mathrm{T}} \nabla^2 f(x_0)(x - x_0) \\ &\text{Subject to } x \in F^2(x_0) \end{aligned} \tag{20}$$

Necessary Optimality Conditions of Nonlinear Programs. Intuitively, if a point x_0 is a local location to Eq. (14), then along any feasible smooth curve $c(t)$: $[0, 1] \to R^n$ emanating from x_0, that is, $c(0) = x_0$ and $c(t) \in \Omega$ for any $t \in [0, 1]$, the objective function f cannot decrease initially, which in mathematical terms means that

$$f'_+(c(0)) \geq 0 \tag{21}$$

since otherwise it would contradict the assumption that x_0 is a local solution. Without getting into detailed mathematics, we simply say that Eq. (21) leads to the following.

First-Order Necessary Conditions. If x_0 is a local solution to Eq. (14), and a constraint qualification holds at x_0, then we have

$$\nabla f(x_0)^{\mathrm{T}} d \geq 0 \text{ for any } d = (x - x_0) \text{ such that } x \in F(x_0) \tag{22}$$

These necessary conditions are intuitive, but inconvenient to manipulate. Among the equivalents of Eq. (22), the following system is important:

$$\nabla L(x_0, u, v) = \nabla f(x_0) + \sum_{i=1}^{p} u_i \nabla h_i(x_0) + \sum_{j=1}^{r} v_j \nabla g_j(x_0) = 0 \tag{23}$$

$$h_i(x_0) = 0,\ i = 1, \ldots, p \tag{24}$$

$$g_j(x_0) \leq 0,\ j = 1, \ldots, r \tag{25}$$

$$v_j g_j(x_0) = 0,\ v_j \geq 0, \qquad j = 1, \ldots, r \tag{26}$$

where $L(x, u, v) = f(x) + \sum_{i=1}^{p} u_i h_i(x) + \sum_{j=1}^{r} v_j g_j(x)$ is the famous Lagrange function, u_i's and v_j's are Lagrange multipliers, and

Eqs. (23)–(26) are called the Karush-Kuhn-Tucker (KKT) conditions, which have a fundamental importance in optimization theory. Note that Eqs. (24) and inequalities (25) are actually the feasibility requirement, and Eq. (26) is the so-called complementarity condition. A triple (x, u, v) satisfying Eqs. (23)–(26) is sometimes referred to as a KKT point and x as a *stationary* point.

The complementarity condition Eq. (26) might look a bit strange at the first glimpse. It states that both v_j and $g_j(x_0)$ cannot be nonzero, or equivalently that inactive constraints have a zero multiplier. Note that when g_j is active, v_j could be either positive (in such a case g_j is said to be strongly active) or zero, the intermediate state between being strongly active and inactive. If there is no such j that $g_j(x_0) = v_j = 0$, then strict complementarity is said to hold, and in such a case, dropping all the inactive constraints and forcing all strongly active constraints to equation constraints will not change the behavior of the KKT system locally.

Second- or higher-order necessary conditions are also derivable by taking into account of second- or higher-order derivative information when available and will not be presented here as they are much less useful than the KKT conditions in practice.

Sufficient Optimality Conditions for General Nonlinear Programs. For convex programs, the first-order necessary conditions Eq. (22) are also sufficient for optimality, but for general nonconvex nonlinear programs, a gap exists between the sufficiency and necessity of Eq. (22). Note that, however, sufficient conditions can be obtained by strengthening Eq. (22) by replacing $\geq$ with $>$.

First-Order Sufficient Conditions. Assume x_0 is a feasible point for Eq. (14). If we have

$$\nabla f(x_0)^{\mathrm{T}} d > 0 \text{ for any } d = (x - x_0) \text{ such that } x \in F(x_0) \quad (27)$$

then x_0 is a strict local solution.

Denote $F^{-}(x_0) = \{x \neq x_0 \colon \nabla f(x_0)^{\mathrm{T}}(x - x_0) \leq 0\}$. Then, Eq. (27) can be formulated as $F(x_0) \cap F^{-}(x_0) = \emptyset$. Unfortunately, first-order sufficient conditions Eq. (27) are rather weak since in general $F(x_0) \cap F^{-}(x_0)$ is not empty and in such a case first-order derivative information is not sufficient to characterize the optimality. To complement this, second-order sufficient conditions have been developed that will be presented later.

Assume that x_0 is a stationary point and $x \in F^2(x_0)$. Multiplying the equations in Eq. (18) by u_i and the inequalities in Eq. (18) by v_j and adding them to the objective function of Eq. (20), we then obtain an interesting inequality

$$f^2(x) \geq L(x_0, u, v) + \nabla L(x_0, u, v)^T (x - x_0) + \frac{1}{2}(x - x_0)^T \nabla^2 L(x_0, u, v)(x - x_0)$$

Using the above inequality and the facts that $f(x) = f^2(x) + o(\|x - x_0\|^2)$ and that $L(x_0, u, v) = f(x_0)$ and $\nabla L(x_0, u, v) = 0$, we can conclude the following.

Second-Order Sufficient Conditions. Assume x_0 is a stationary point. If there exist Lagrange multipliers u and v such that for every $x \in F(x_0) \cap F^{-}(x_0)$ we have

$$(x - x_0)^{\mathrm{T}} \nabla^2 L(x_0, u, v)(x - x_0) > 0 \quad (28)$$

then x_0 is a strict local solution. In such a case, we say that (x_0, u, v) satisfies the second-order sufficient conditions.

Integer Linear Programming Problems

In this subsection we shall consider the pure integer linear program

$$\begin{aligned} &\text{Minimize } c^T x \\ &\text{Subject to } x \in S = \{x \in Z^n : Ax \leq b,\ x \geq 0\} \end{aligned} \quad (29)$$

Let P denote the polyhedron $\{x \in R^n \colon Ax \leq b,\ x \geq 0\}$ and z_{IP} the optimal value of Eq. (29). Then the feasible set can be rewritten as $S = P \cap Z^n$. For simplicity we shall always assume P is bounded and thus S consists of finitely many points. The focus of this subsection is the relationship between an integer program and its relaxations. The basic concepts and results covered here are those, such as valid inequalities and facets of polyhedron, that are concerned with using continuous objects to describe the discrete feasible set S and how to generate them.

We should stress that for integer programs only global solutions are of interest in general. The primary way of establishing the global optimality of a feasible solution x is to compare $c^T x$ with z_{IP} to check if $c^T x - z_{\mathrm{IP}} = 0$ or more practically $c^T x - z_{\mathrm{IP}} \leq \epsilon$ for some small $\epsilon > 0$. In the latter case, x is a near-optimal solution within the ϵ threshold. One might wonder how the previous verification of optimality can be carried out numerically as z_{IP} is usually unknown in the solution process. The trick is to establish a close enough lower bound w on z_{IP} since $c^T x - w \leq \epsilon$ would imply $c^T x - z_{\mathrm{IP}} \leq c^T x - w \leq \epsilon$. A typical technique for finding a lower bound is to use relaxation. The idea is to replace Eq. (29) by an easier problem that can be solved and whose optimal value is then used as a lower bound. Frequently, it is necessary to refine these problems iteratively to obtain successively tighter bounds.

Relaxation. A relaxation of Eq. (29) is any optimization problem

$$\begin{aligned} &\text{Minimize } z_{\mathrm{RP}}(x) \\ &\text{Subject to } x \in S_{\mathrm{RP}} \end{aligned}$$

where the subscript RP stands for *Relaxed Problem,* with the following two properties:

1. $S \subseteq S_{\mathrm{RP}}$
2. $c^T x \geq z_{\mathrm{RP}}(x)$ for $x \in S$

If the above relaxation has a solution x^* with optimal value z_{RP}, obviously we have $z_{\mathrm{IP}} \geq z_{\mathrm{RP}}$, that is, z_{RP} is a lower bound of z_{IP}. Furthermore, if x^* happens to be feasible for the original integer program, then it is also a solution of the original integer program.

An obvious way to obtain a relaxation is to satisfy property 1 by dropping one or more of the constraints that define S and to satisfy property 2 by setting $z_{\mathrm{RP}}(x) = c^T x$. The *linear programming relaxation* of Eq. (29) is obtained by deleting the integrality constraints $x \in Z^n$ and thus is given by

$$z_{\mathrm{LP}} = \text{Minimize } \{c^T x \colon x \in P\}$$

Solving this linear program results in a lower bound z_{LP} of z_{IP}. Unfortunately, this lower bound is usually not good enough for difficult integer programs and successive improvement is often needed.

Since the solutions of a linear program lie on its vertices, it is not hard to see that extending the feasible set S to its convex hull will result in a relaxation that is equivalent to Eq. (29),

$$\begin{aligned} &\text{Minimize} \quad c^T x \\ &\text{Subject to } x \in \text{conv}(S) \end{aligned} \tag{30}$$

where conv(S) is the convex hull of S, that is, the set of points that can be written as a convex combination of points in S, that is,

$$\text{conv}(S) = \left\{ x: x = \sum_{i=1} \lambda_i x^i,\ \sum_{i=1}^{m} \lambda_i = 1,\ \lambda_i \geq 0, \right.$$
$$\left. \text{where } x^1, \ldots, x^m \text{ is any set of points in } S \right\}$$

This observation, however, does not help us much since it is in general expensive to find a linear inequality description of conv(S). The focus has largely been on the representation and construction of a weaker relaxation

$$\begin{aligned} &\text{Minimize} \quad c^T x \\ &\text{Subject to } x \in Q \end{aligned} \tag{31}$$

where Q is a polyhedron satisfying $\text{conv}(S) \subseteq Q \subseteq P$ such that the linear program Eq. (31) gives an optimal or near-optimal solution to Eq. (29). To this end, the following concept is useful.

Valid Inequality. An inequality $\pi^T x \leq \pi_0$, where π is a vector, is valid for S, or equivalently conv(S), if it is satisfied by all points in S. Given two valid inequalities $\pi^T x \leq \pi_0$ and $\gamma^T x \leq \gamma_0$ that are not scalar multiples of each other, we say that $\pi^T x \leq \pi_0$ is stronger than or dominates $\gamma^T x \leq \gamma_0$ if $\{x: \pi^T x \leq \pi_0, x \geq 0\} \subset \{x\ \gamma^T x \leq \gamma_0, x \geq 0\}$. A maximal valid inequality of S is the one that is not dominated by any other valid inequality of S.

Obviously the set of maximal valid inequalities for S describes conv(S). Thus it would be of considerable interest to know how the valid inequalities, especially, maximal valid inequalities can be generated.

Generating Valid Inequalities. Note that $\text{conv}(S) \subseteq P$ since $S = P \cap Z^n \subseteq P$, and in general $\text{conv}(S) \neq P$. So there might exist valid inequalities for S that are not valid for P. Therefore, the valid inequalities for S cannot be derived only from information about P and have to be obtained using the additional integrality constraint $S \subseteq Z^n$. There are several general methods for generating valid inequalities and the one we shall present here is the so-called *Chvatal-Gomory* (GC) *rounding method*. This approach is based on the simple principle that if a is an integer and $a \leq b$, then $a \leq \lfloor b \rfloor$, where $\lfloor b \rfloor$ is the largest integer less than or equal to b. For $S = \{x: Ax \leq b, x \geq 0\} \cap Z^n$, where $A = (a_1, a_2, \ldots, a_n)$ and $N = (1, \ldots, n)$, the method is a three-step procedure:

1. Choose a nonnegative vector $u = (u_1, \ldots, u_n) \geq 0$, and take a linear combination of the constraints with weights u_i for all i and obtain the following valid inequality
$$\sum_{j \in N} (ua_j) x_j \leq ub$$
2. Since $x \geq 0$ implies $\sum_{j \in N} (ua_j - \lfloor ua_j \rfloor) x_j \geq 0$, subtracting it from the left-hand side of the preceding inequality yields the valid inequality
$$\sum_{j \in N} (\lfloor ua_j \rfloor) x_j \leq ub$$
3. Since $x \in Z^n$ implies $\sum_{j \in N} (\lfloor ua_j \rfloor) x_j$ is an integer, we invoke integrality to round down the right-hand side of the above inequality and obtain the valid inequality
$$\sum_{j \in N} (\lfloor ua_j \rfloor) x_j \leq \lfloor ub \rfloor \tag{32}$$

The valid inequality Eq. (32) can be added to $Ax \leq b$, and then the procedure can be repeated by combining generated inequalities and/or original ones. It can be proved that by repeating the CG procedure a finite number of times, all of the valid inequalities for S can be generated.

In fact, some of the maximal valid inequalities are necessary in the description of conv(S) and others are not and thus can be dropped. To find out which are necessary and which are not, the following notion from the theory of polyhedra is useful.

Facets of Polyhedron. If $\pi^T x \leq \pi_0$ is a valid inequality for the polyhedron conv(S) and $F = \{x \in \text{conv}(S): \pi^T x = \pi_0\}$, F is called a face of conv(S), and we say that $\pi^T x \leq \pi_0$ represents F. If a face $F \neq \text{conv}(S)$, then dim(F), the dimension of F, must be less than dim(conv(S)). A face F of conv(S) is a facet of conv(S) if $\dim(F) = \dim(\text{conv}(S)) - 1$.

It can be shown that for each facet F of conv(S), one of the inequalities representing F is necessary in the description of conv(S). For this reason techniques for finding facets are important in solving integer programs effectively. General methods for generating all valid inequalities such as the CG procedure can be quite inefficient in obtaining facets. The best-known technique for finding facet-defining inequalities of integer programs is to make the use of problem structure and is quite problem-specific. It is indeed more of an art than a formal methodology. Considerable efforts have been devoted to the determination of families of facet-defining inequalities or strong valid inequalities for specific problem classes, and there are many interesting problems for which facet-defining inequalities or strong valid inequalities have been obtained. Interested readers may consult Nemhauser and Wolsey (1) for more information.

ALGORITHMS

An algorithm is in our context a numerical procedure for starting with given initial conditions and calculating a sequence of steps or iterations until some stopping rule is satisfied. A variety of algorithms have been developed for each

class of optimization problems. Similar to what we did in the section entitled "Basic Theory," to give the reader a sense of what optimization algorithms look like we shall mainly focus our attention on two typical classes of optimization problems discussed in the previous section, the smooth nonlinear program and integer linear program, and discuss algorithms for solving these two problem classes. The following subsection covers the Newton-type methods for solving smooth nonlinear programs, while in the subsection thereafter, two general methods for solving integer programs, the branch-and-bound method and cutting-plane method, will be presented.

Solving Smooth Nonlinear Programming Problems

Almost all algorithms for smooth optimization are iterative in the sense that they generate a series of points, each point being calculated on the basis of the points preceding it. An iterative algorithm is initiated by specifying a starting point. If an algorithm is guaranteed to generate a sequence of points converging to a local solution for starting points that are sufficiently close to the solution, then this algorithm is said to be *locally convergent.* If the generated sequence of points is guaranteed to converge to a local solution for *arbitrary* starting points, the algorithm is then said to be *globally convergent.* The focus of this subsection is Newton-type methods. We shall begin with the basic Newton method for solving unconstrained optimization problems, which is known to be only locally convergent, and then briefly review how we can globalize Newton's method so that it converges for any starting point. Finally, a generalization of the basic Newton method to constrained problems is presented. Interestingly enough, it has been noticed that almost all iterative algorithms for smooth nonlinear programming that perform exceptionally well in practice are some variants of Newton's method.

Before introducing Newton's method, we must stress that an algorithm being theoretically convergent does not mean it always converges to a solution in a practically allowed time period. The consensus in nonlinear optimization is that to be considered as practically convergent, an algorithm has to be at least superlinearly convergent, a notion related to the speed of convergence, which we shall briefly present next.

Speed of Convergence. Assume that the sequence $\{x_k\}$ generated by an algorithm converges to x^*. If we have

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = \beta < 1$$

the sequence is said to converge *linearly* to x^* and the rate of convergence is linear. The case for which $\beta = 0$ is referred to as *superlinear* convergence. If

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} = \beta > 0$$

then the rate of convergence is quadratic. The algorithm is said to be linear, superlinear, or quadratic, according to the convergence rate of the sequence it generates. It is easy to see that quadratic convergence is faster than superlinear convergence, which is faster than linear convergence. A rich theory on speed of convergence, or convergence rates, for measuring the effectiveness of algorithms has been developed [see Ortega and Rheinboldt (16)].

We shall first consider the unconstrained optimization problem, which is central to the development of optimization algorithm. Constrained optimization algorithms are often extensions of unconstrained ones.

Newton's Method. The underlying principle in most iterative algorithms for smooth optimization is to build, at each iteration, a local model of the problem that is valid near the current solution estimate. The next, often improved, solution estimate is obtained at least in part from solving this local model problem. At the current iteration, the basic Newton method solves the local model that is obtained by replacing the original function with its quadratic approximation around the current iterate x_k

$$\underset{s}{\text{Minimize}}\, q_k(s) := f(x_k) + \nabla f(x_k)^{\mathrm{T}} s + \frac{1}{2} s^{\mathrm{T}} \nabla^2 f(x_k) s$$

where $s = x - x_k$. When the Hessian matrix $\nabla^2 f(x_k)$ is positive definite, q_k has a unique minimizer that can be obtained by solving the linear system $\nabla q_k(s) = 0$, that is,

$$\nabla f(x_k) + \nabla^2 f(x_k) s_k = 0 \text{ or } s_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k)$$

The next iterate is then $x_{k+1} = x_k + s_k$. Convergence is guaranteed if the starting point x_0 is sufficiently close to a local solution. The most notable feature of Newton's method is that the rate of convergence is quadratic.

Globalization of Newton's Method. When the starting point x_0 is far away from a local solution, the iterates generated by the basic Newton method may not even converge. A common approach is to use a line search to globalize the basic Newton method so that it converges from any starting point.

Given a descent search direction d_k, a line-search method generates the iterates by setting $x_{k+1} = x_k + \alpha_k d_k$, where α_k is chosen so that $f(x_{k+1}) < f(x_k)$. A practical criterion for a suitable α_k is to require α_k to satisfy the so-called sufficient decrease condition

$$f(x_k + \alpha_k d_k) \le f(x_k) + \mu \alpha_k \nabla f(x_k)^{\mathrm{T}} d_k$$

where μ is a constant with $0 < \mu < 1$.

Most line-search versions of the basic Newton method generate the search direction $d_k = s_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k)$ by occasionally replacing the Hessian matrix $\nabla^2 f(x_k)$ with $\nabla^2 f(x_k) + E_k$ such that the resultant matrix is sufficiently positive definite. This guarantees that the search direction s_k defined by Newton's method is a descent direction since $\nabla f(x_k)^{\mathrm{T}} s_k = -\nabla f(x_k)^{\mathrm{T}} [\nabla^2 f(x_k) + E_k]^{-1} \nabla f(x_k) < 0$.

Constrained Optimization. Many techniques have been proposed for solving the constrained nonlinear program Eq. (14). One of them is the *sequential quadratic programming* (SQP) method, which is a generalization of Newton's method for unconstrained optimization. At the current solution estimate, this method uses a linearly constrained quadratic local model to approximate the original problem. In its purest form, replacing the objective function with its quadratic approximation

$$q_k(s) := f(x_k) + \nabla f(x_k)^{\mathrm{T}} s + \frac{1}{2} s^{\mathrm{T}} \nabla^2 L(x_k, u_k, v_k) s$$

and the constraint functions with their respective linear approximations, SQP solves

$$\begin{aligned}&\text{Minimize } q_k(s)\\&\text{Subject to}\\&\quad h_i(x_k) + \nabla h_i(x_k)^{\mathrm{T}} s = 0,\ i = 1, \ldots, p\\&\quad g_j(x_k) + \nabla g_j(x_k)^{\mathrm{T}} s \le 0,\ j = 1, \ldots, r\end{aligned}$$

where $s = x - x_k$, and sets the new solution estimate $x_{k+1} = x_k + s_k$. As a variant of Newton's method, SQP inherits excellent local convergence property. Given a KKT point (x^*, u^*, v^*) satisfying the second-order sufficient conditions, when SQP starts at a point x_0 sufficiently close to x^* and all the Lagrange multiplier estimates (u_k, v_k) remain sufficiently close to (u^*, v^*), the sequence it generates converges to x^* at a quadratic rate. One complexity of SQP is that the Lagrange multiplier estimates are needed to set up the second-order term in q_k and so must be updated from iteration to iteration. A direct treatment is simply to use the optimal multipliers for the quadratic local problem at the previous iteration. The interested reader may consult Fletcher (17) for details.

Similar to the basic Newton method for unconstrained optimization, the SQP method in its pure form given earlier is not guaranteed to converge for a starting point that is far away from a local location. Again, a line search along the search direction s_k can be used to globalize SQP. Of course, we now want the next iterate not only to decrease the value of the objective function but also to come closer to the feasible region. But often these two aims conflict and so it is necessary to weight their relative importance and consider their joint effect in reaching optimality. One commonly used technique to achieve this is to use a *merit* or *penalty function* to measure the closeness of a point to optimality

$$m(x; c) := f(x) + \sum_{i=1}^{p} c_i |h_i(x)| + \sum_{j=1}^{r} c_{p+j} \max(g_j(x), 0)$$

where $c_k > 0$ are penalty parameters. Then a line search aiming at achieving sufficient decrease of the merit function can be used to choose an α_k for $x_{k+1} = x_k + \alpha_k d_k$, where $d_k = s_k$. The interested reader might consult Fletcher (17) for more information.

Solving Integer Linear Programming Problems

In the section titled "Integer Linear Programming Problems" we have addressed some basic properties of integer programs and discussed the relationship between an integer program and its relaxations and how to generate valid inequalities to improve the relaxations. In general, integer programs are much more complicated and expensive to solve than their continuous relaxations on account of the discrete nature of the variables. A simple-minded way to deal with an integer program is to form its corresponding continuous relaxation by dropping the integrality constraint, and then to solve the relaxation and round off the solution to its nearby integers in certain manner. In fact, this is how many integer programs are handled in practice. It is important to realize that there is no guarantee that a good solution can be obtained in this way, even by examining all integer points in some neighborhood of the solution of a relaxation. General techniques for solving integer programs do exist, though they often need to be customized in order to be most effective. We shall present two general methodologies for solving integer programs, namely, the branch-and-bound method and cutting-plane method. For simplicity, we shall confine ourselves to the integer linear programming problem.

Branch-and-Bound Method. The branch-and-bound method solves an integer program by solving a series of related continuous programs in a systematic way. The basic idea behind it is the familiar *divide and conquer*. In other words, if it is too hard to optimize over the feasible set S, perhaps the problem can be solved by optimizing over smaller sets and then putting the results together. More precisely, we can partition the feasible set S into a set of subsets $\{S^i: i = 1, \ldots, k\}$ such that $\cup_{i=1}^{k} S^i = S$ and $S^i \cap S^j = \emptyset$ for $i, j = 1, \ldots, k$, $i \neq j$, and solve the problem over each of the subsets, i.e., solve (IP^i)

$$\begin{aligned}&\text{Minimize } c^T x\\&\text{Subject to } x \in S^i\end{aligned}$$

for $i = 1, \ldots, k$. Assume their respective solutions are x^i with optimal value z^i_{IP} for $i = 1, \ldots, k$. Then we can easily put the results together since it is obvious that the optimal value of the original problem $z_{\mathrm{IP}} = \min_{i=1,\ldots,k} z^i_{\mathrm{IP}}$. Let j be the one such that $z_{\mathrm{IP}} = z^j_{\mathrm{IP}}$. Then x^j is a solution of the original problem. Note that this scheme can be applied recursively, that is, if a particular subproblem IP^i cannot be easily solved, the divide-and-conquer process can be carried out for the subproblem IP^i, meaning the subset S^i can be further partitioned and the problem can be solved over the furthered partitioned subsets.

In general, partitioning is done by imposing additional bounds on certain components of x. For instance, the original problem can be partitioned into two subproblems by "branching" on some component, say x_1, yielding IP^1

$$\begin{aligned}&\text{Minimize } c^T x\\&\text{Subject to } x \in S,\ x_1 \le 10\end{aligned}$$

and IP^2

$$\begin{aligned}&\text{Minimize } c^T x\\&\text{Subject to } x \in S,\ x_1 \ge 10 + 1\end{aligned}$$

It is possible to repeat the branching process on IP^1 and IP^2, and again on the resulting problems. However, the total number of resultant subproblems increases *exponentially* with the number of levels of branching done, and it is unrealistic to solve all these subproblems when the total gets too high. The branch-and-bound method takes advantage of the fact that many of these subproblems can actually be "pruned" based on information about bounds on the optimal value. Specifically, since the subproblems are solved sequentially, at any stage we can keep track of the best feasible solution obtained so far and its objective function value, which we denote by $\bar{x}_{\mathrm{IP}}$ and $\bar{z}_{\mathrm{IP}}$, respectively. Assume IP^i is the current subproblem we are dealing with. We form a continuous relaxation RP^i of IP^i, solve it, and obtain its global solution x^i_{R} with optimal value z^i_{R}. Now if x^i_{R} is feasible for IP^i, it is then a solution of IP^i and so IP^i is already solved and can be pruned. If $z^i_{\mathrm{R}} < \bar{z}_{\mathrm{IP}}$, x^i_{R} is then a better feasible solution than $\bar{x}_{\mathrm{IP}}$. Thus, $\bar{x}_{\mathrm{IP}}$ and $\bar{z}_{\mathrm{IP}}$

should be updated by setting $\bar{x}_{IP} = x^i_R$ and $\bar{z}_{IP} = z^i_R$. Otherwise, x^i_R is not better than $\bar{x}_{IP}$ and so can be ignored. If x^i_R is not feasible for IP^i, we also compare z^i_R with $\bar{z}_{IP}$. If $z^i_R \geq \bar{z}_{IP}$, we then conclude that there is no hope of finding a better solution than $\bar{x}_{IP}$ by solving subproblem IP^i. The reason is that z^i_R is a lower bound of the optimal value of IP^i due to the fact that RP^i is a relaxation of IP^i and $\bar{z}_{IP}$ is already as good as z^i_R. Thus, in such a case IP^i can be pruned. However, in the case for which $z^i_R < \bar{z}_{IP}$, we cannot rule out the possibility that IP^i could have a solution that is better than $\bar{z}_{IP}$, and so the branching process needs to be carried out further on IP^i.

Many strategies are known in the implementation of the branch-and-bound method with respect to how to branch a subproblem, how to pick the next subproblem to consider when the current subproblem is pruned, etc. For the interested reader, Nemhauser and Wolsey (1) is a good book for details. It is easy to see that the quality of produced bounds ($\bar{z}_{IP}$ and z^i_R) is crucial in pruning out subproblems effectively and in fact the primary factor in the efficiency of a branch-and-bound algorithm.

Cutting-Plane Algorithm. The cutting-plane algorithm works with a sequence of successively tighter continuous relaxations of the integer program Eq. (29) until, hopefully, an integer optimal solution is found. The basic idea is simple. Assume that at the current iteration a solution x^* to the current continuous relaxation is found. If x^* is an integer solution, then it is a solution to the integer program and the problem is solved. Otherwise, we try to find a valid inequality for S that is not satisfied by x^* by solving, often approximately, a so-called separation problem. Since this valid inequality cuts off x^* from S, or more appropriately from conv(S), we then add it to the current relaxation to form a tighter relaxation and proceed to the next iteration. In order to have a sufficiently tighter relaxation, it is desirable to generate a facet-defining inequality that cuts of x^* from conv(S). Generating good cuts is often problem specific and details can be found in Nemhauser and Wolsey (1).

Recently, the cutting-plane algorithm has been incorporated into the general branch-and-bound scheme for solving subproblems or at least improving the bounds. The combined method, called the branch-and-cut method, has proved to be quite effective in solving some hard integer programs [see Hoffman and Padberg (4)].

FURTHER READING

In the previous sections we have sketched some basic results in the subject of mathematical programming, which is now on its way to maturity. There exists a vast literature on this subject. In fact, all topics mentioned in the paper have been explored in great detail in the past several decades. In what follows, we shall suggest some general references based on our limited knowledge on this subject. A collection of articles on the historical accounts of many branches of mathematical programming is the interesting book edited by Lenstra, Rinnooy Kan, and Schrijver (18); excellent state-of-the-art expository articles on the most important topics in mathematical programming by leading experts in the field can be found in the handbook edited by Nemhauser, Rinnooy Kan and Todd (19). Standard textbooks or references in the subject are the following: Dantzig (20) in linear programming; Bazaraa, Jarvis, and Sherali (21) in linear programs and network flows; Luenberger (22) in linear and nonlinear programming; McCormick (7) and Fletcher (17) in nonlinear programming; Fiacco (10) in sensitivity analysis; Gill, Murray, and Wright (23) in numerical methods and implementation; Rosen (24) in large-scale optimization; Fiacco and McCormick (25) and Megiddo (26) in interior point and related methods; Nemhauser and Wolsey (1) and Schrijver (27) in integer programming; Ahuja, Magnanti, and Orlin (28) in network flows; Hall and Wallace (29) in stochastic programming; Neittaanmaki (2) in nonsmooth optimization; Anandalingam (30) in multilevel programming; Sawaragi, Nakayama and Tanino (31) in multiobjective optimization; Moré and Wright (9) in evaluation and comparison of optimization software packages; Hocking (32) in optimal control. The introductory books in operations research by Hillier and Lieberman (5), Winston (33) and Winston and Albright (34) also cover many branches of mathematical programming. The *Mathematical Programming Society* has published several volumes of selective tutorial lectures given by leading experts covering many branches of mathematical programming at its triennial international symposiums, and the latest ones are the volume "Mathematical Programming: State of the Art 1994" (35) edited by Birge and Murty and the special issue "Lectures on Mathematical Programming, ISMP97" (36) edited by Liebling and Werra.

Many journals contain articles in mathematical programming. The ones devoted to this subject are *Mathematical Programming, Optimization, Journal of Optimization Theory and Applications, SIAM Journal on Optimization,* and *Journal of Global Optimization.* Some of the most relevant ones are *Mathematics of Operations Research, SIAM Journal on Control and Optimization, Operations Research, Management Science, The European Journal of Operational Research,* and *Operations Research Letters.*

There is also a tremendous amount of information relevant to the subject on the Internet. The Operations Research Page (http://mat.gsia.cmu.edu) of Professor Michael Trick at Carnegie Mellon University is a page for pointers to all aspects of Operations Research. The Optimization Technology Center founded jointly by Argonne National Laboratory and Northwestern University has a home page (http://www.mcs.anl.gov/home/otc) that has a lot information on optimization techniques and also implements the so-called network-enabled optimization system designed for solving optimization problems remotely over the Internet. The Mathematical Programming Glossary page (http://www-math.cudenver.edu/~hgreenbe/glossary/glossary.html) maintained by Professor Harvey Greenberg at University of Colorado at Denver contains many technical terms and links specific to mathematical programming.

BIBLIOGRAPHY

1. G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization,* New York: Wiley, 1988.
2. M. Neittaanmaki, *Nonsmooth Optimization,* London: World Scientific Publishing, 1992.
3. R. T. Rockafellar, *Convex Analysis,* Princeton, NJ: Princeton University Press, 1970.

4. K. L. Hoffman and M. Padberg, Solving airline crew scheduling problems by branch-and-cut, *Management Sci.,* **39** (4): 657–682, 1993.
5. F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research,* New York: McGraw-Hill Publishing, Inc., 1980.
6. W. Murray, Financial planning via multi-stage stochastic programs, In J. R. Berge and K. G. Murty (eds.), *Mathematical Programming: State of the Art 1994,* Ann Arbor, MI: 1994.
7. G. P. McCormick, *Nonlinear Programming: Theory, Algorithms and Applications,* New York: Wiley, 1983.
8. G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell, *Engineering Optimization: Methods and Applications,* New York: Wiley, 1983.
9. J. J. Moré and S. J. Wright, *Optimization Software Guide,* Vol. 14 of Frontiers in Applied Mathematics, Philadelphia: SIAM, 1993.
10. A. V. Fiacco, *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming,* New York: Academic Press, 1983.
11. J. L. Nazareth, *Computer Solution of Linear Programs,* New York: Oxford University Press, Inc., 1987.
12. J. B. Berge et al., A standard input format for multiperiod stochastic linear programs, *COAL Newsletter* **17**: 1–19, 1987.
13. A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *LANCELOT,* Berlin: Springer-Verlag, 1992.
14. B. A. Murtagh and M. A. Saunders, MINOS 5.1 User's Guide, Technical Report No. SOL 83-20R, System Optimization Laboratory, Standard University, Standard, 1983.
15. R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL, A Modeling Language for Mathematical Programming,* San Francisco, CA: The Scientific Press, 1993.
16. J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables,* New York: Academic Press, Inc., 1970.
17. R. Fletcher, *Practical Methods of Optimization,* New York: Wiley, 1987.
18. J .K. Lenstra, A. H. G. Rinnooy Kan, and A. Schrijver (eds.), *History of Mathematical Programming,* Amsterdam: CWI, 1991.
19. G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd (eds.), *Optimization,* Amsterdam: North-Holland, 1989.
20. G. B. Dantzig, *Linear Programming and Extensions,* Princeton, NJ: Princeton University Press, 1963.
21. M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, *Linear Programming and Network Flows,* New York: Wiley, 1990.
22. D. G. Luenberger, *Linear and Nonlinear Programming,* Reading, MA: Addison-Wesley, 1984.
23. P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization,* London: Academic Press, 1981.
24. J. B. Rosen (ed.), *Supercomputers and Large-Scale Optimization: Algorithms, Software, Applications,* Annals of Operations Research, Vol. 22, Switzerland: J. C. Baltzer AG, Science Publishers, 1990.
25. A. V. Fiacco and G. P. McCormick, *Nonlinear Programming, Sequential Unconstrained Minimization Techniques,* New York: Wiley, 1968.
26. N. Megiddo (ed.), *Progress in Mathematical Programming—Interior Point and Related Methods,* New York: Springer, 1989.
27. A. Schrijver, *Theory of Linear and Integer Programming,* New York: Wiley, 1986.
28. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows,* Englewood Cliffs, NJ: Prentice Hall, 1993.
29. P. Hall and S. W. Wallace, *Stochastic Programming,* New York: Wiley, 1994.
30. G. Anandalingam (ed.), *Hierarchical Optimization,* Annals of Operations Research 34, Switzerland: J. C. Baltzer AG, Science Publishers, 1992.
31. Y. Sawaragi, H. Nakayama, and T. Tanino, *Theory of Multiobjective Optimization,* New York: Academic Press, 1985.
32. L. M. Hocking, *Optimal Control: An Introduction to the Theory with Applications,* New York: Oxford University Press, 1991.
33. W. L. Winston, *Operations Research: Applications and Algorithms,* Boston: PWI-Kent Publishing Co., 1991.
34. W. L. Winston and S. C. Albright, *Practical Management Science: Spreadsheet Modeling and Applications,* Belmont, CA: Duxbury Press, 1997.
35. J. B. Birge and K. G. Murty (eds.), *Mathematical Programming: State of the Art 1994,* Ann Arbor, MI: The University of Michigan, 1994.
36. T. M. Liebling and D. de Werra (eds.), *Lectures on Mathematical Programming, ISMP97, Mathematical Programming, Vol. 79,* New York: Elsevier Science, 1997.

JIMING LIU
Lucent Technologies

MATHEMATICAL PROGRAMMING. See GEOMETRIC PROGRAMMING.

MATHEMATICAL THEORY OF COMMUNICATIONS. See INFORMATION THEORY.

MATHEMATICS. See GEOMETRY.

MATLAB. See CIRCUIT STABILITY.

MATRIX PROPERTY. See EIGENVALUES AND EIGENFUNCTIONS.

MATRIX RICCATI EQUATIONS. See KALMAN FILTERS AND OBSERVERS.

MAXIMALLY FLAT GAIN FILTERS. See BUTTERWORTH FILTERS.

MAXIMUM ENTROPY. See MAXIMUM LIKELIHOOD DETECTION.