

# CHAPTER 1

## INTRODUCTION TO OPTIMIZATION

Jasbir S. Arora

*Department of Civil and Environmental Engineering  
Department of Mechanical and Industrial Engineering  
Center for Computer Aided Design  
The University of Iowa  
Iowa City, Iowa, U.S.A.  
E-mail: Jasbir-Arora@uiowa.edu*

Basic concepts of optimization are described in this chapter. Optimization models for engineering and other applications are described and discussed. These include continuous variable and discrete variable problems. Optimality conditions for the continuous unconstrained and constrained problems are presented. Basic concepts of algorithms for continuous and discrete variable problems are described. An introduction to the topics of multiobjective and global optimization is also presented.

### 1. Introduction

Optimization is a mature field due to the extensive research that has been conducted over the last about 60 years. Many types of problems have been addressed and many different types of algorithms have been investigated. The methodology has been used in different practical applications and the range of applications is continuously growing. Some of the applications are described in various chapters of this book. *The purpose of this chapter is to give an overview of the basic concepts and methods for optimization of structural and mechanical systems.* Various optimization models are defined and discussed. Optimality conditions for continuous variable optimization problems are presented and discussed. Basic concepts of algorithms for continuous variable and discrete variable optimization problems are described. Topics of multiobjective and global optimization are also introduced. The material of the chapter is available in many textbooks on optimization.<sup>1-7</sup> It is derived from several recent publications of the author and his co-workers.<sup>7-34</sup>

## 2. Optimization Models

Transcription of an optimization problem into a mathematical formulation is a critical step in the process of solving the problem. If the formulation of the problem as an optimization problem is improper, the solution for the problem is most likely going to be unacceptable. For example, if a critical constraint is not included in the formulation, then most likely, that constraint is going to be violated at the optimum point. Therefore special attention needs to be given to the formulation of the optimization problem.

Any optimization problem has three basic ingredients:

- *Optimization variables*, also called *design variables* denoted as vector  $\mathbf{x}$ .
- *Cost function*, also called the objective function, denoted as  $f(\mathbf{x})$ .
- *Constraints* expressed as equalities or inequalities denoted as  $g_i(\mathbf{x})$ .

The variables for the problem can be continuous or discrete. Depending on the types of variables and functions, we obtain continuous variable, discrete variable, differentiable and nondifferentiable problems. These models are described next; for more details and practical applications of the models, various references can be consulted.<sup>7,9-12,14,16,25-30</sup>

### 2.1. Optimization Models: Continuous Variables

Any continuous variables optimization problem can be transcribed into a *standard nonlinear programming (NLP) model* defined as minimization of a cost function subject to equality constraints and inequality constraints expressed in a " $\leq$ " form as Problem P.<sup>7</sup>

**Problem P.** Find the optimization variable vector  $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^T$  to minimize a cost function  $f(\mathbf{x})$  subject to equality and inequality constraints:

$$g_j(\mathbf{x}) = 0, \quad j = 1 \text{ to } p \quad (1)$$

$$g_j(\mathbf{x}) \leq 0, \quad j = p + 1 \text{ to } m \quad (2)$$

where  $n$  is the number of variables,  $p$  is the number of equality constraints, and  $m$  is the total number of constraints. Note that the *explicit lower and upper bounds* on the variables are included in Eq. (2). However, for efficient numerical calculations the simple form of these constraints is exploited.

The *feasible set* for the problem is defined as a collection of all the points that satisfy the constraints of Eqs. (1) and (2). It is also called the *constraint set*, and is denoted as  $S$ :

$$S = \left\{ \mathbf{x} \mid g_j(\mathbf{x}) = 0, j = 1 \text{ to } p; g_j(\mathbf{x}) \leq 0, j = p + 1 \text{ to } m \right\} \quad (3)$$

Thus the Problem P can be written simply as

$$\underset{\mathbf{x} \in S}{\text{minimize}} f(\mathbf{x}) \quad (4)$$

It is important to note that the feasible set for a problem may be empty if there are too many constraints on the problem or if there are conflicting constraints. In general, this is difficult to determine before the problem is solved. Only after a numerical algorithm fails to find a feasible point for the problem, we can conclude that the set  $S$  is empty.<sup>21</sup> In that case the problem formulation needs to be examined to relax some of the constraints, or eliminate conflict in the constraints. In addition, it is difficult to know, in general, if there is a solution to the Problem P. However, the question of *existence of a solution* can be answered with certain assumptions about the problem functions. It turns out that if  $f(\mathbf{x})$  is *continuous* on a *nonempty feasible set*  $S$ , all constraint functions are continuous, and all inequalities contain their boundary points (i.e., expressed as “ $\leq$ ” and not simply as “ $<$ ”), then there is a solution for Problem P. When these requirements are satisfied, a robust numerical algorithm is guaranteed to converge to a solution point.

If there are no constraints on the variables, the set  $S$  is the entire design space and the problem is called an *unconstrained optimization problem*. If all the functions are linear in terms of the variables, the Problem P is called a *linear programming* (LP) problem. If the cost function is quadratic and the constraints are linear, the problem is called a *quadratic programming* (QP) problem.

An inequality constraint  $g_i(\mathbf{x}) \leq 0$  is said to be *active* at a point  $\mathbf{x}$  if it is satisfied as an equality at that point, i.e.,  $g_i(\mathbf{x}) = 0$ . It is said to be *inactive* if it has negative value at that point, and *violated* if it has positive value. An *equality* constraint is always either *active* or *violated* at any point.

In some applications, several objective functions need to be optimized simultaneously. These are called *multiobjective* optimization problems. They are usually transformed into Problem P by combining all the objective functions to form a composite scalar objective function. Several approaches to accomplish this objective are summarized in a later section.<sup>7,32,35-37</sup>

When a gradient-based optimization method (discussed in a later section) is used to solve Problem P, the cost and constraint functions are assumed to be twice differentiable.

## 2.2. Optimization Models: Mixed Variables

In many practical applications of optimization, discrete variables occur naturally in the problem formulation. For example,

- plate thickness must be selected from the available dimensions,<sup>7</sup>
- material properties must correspond to the available materials,<sup>7,25</sup>
- structural members must be selected from a catalog,<sup>14,26,29</sup>
- number of reinforcing bars in a concrete member must be an integer,<sup>28</sup>
- diameter of rods must be selected from the available sizes,<sup>7,28</sup>
- number of bolts must be an integer,<sup>27</sup>
- number of strands in a prestressed member must be an integer.<sup>28</sup>

Discrete variables must be treated properly in numerical optimization procedures. A mixed continuous-discrete variable optimization problem is defined next as Problem MP.

**Problem MP.** A general mixed discrete-continuous variable nonlinear optimization problem is defined by modifying Problem P to minimize the cost function  $f(\mathbf{x})$  subject to the constraints of Eqs. (1) and (2) with the additional requirement that each discrete variable be selected from a specified set:

$$x_i \in D_i, D_i = (d_{i1}, d_{i2}, \dots, d_{iq_i}); i = 1 \text{ to } n_d \quad (5)$$

where  $n_d$  is the number of *discrete design variables*,  $D_i$  is the set of discrete values for the  $i$ th variable,  $q_i$  is the number of available discrete values for the  $i$ th variable, and  $d_{ik}$  is the  $k$ th discrete value for the  $i$ th variable. Note that the foregoing problem definition includes *integer variable* as well as *0-1 variable* (on-off variables, binary variables) problems. If the problem has only *continuous variables*, and the functions  $f$  and  $g_j$  are twice continuously differentiable, we obtain the Problem P. Many discrete variable optimization problems have nondifferentiable functions; therefore gradient-based methods cannot be used to solve such problems. However, methods that do not require gradients of functions are available to solve such problems.

It is also important to note that the discrete variable optimization problems usually require considerably more computational effort compared to the continuous variable problems. This is true even though the number of feasible points with discrete variables is finite and they are infinite with continuous variables.

### 3. Optimality Conditions for Problem P

#### 3.1. Definitions and General Concepts

The optimality conditions are the mathematical conditions that characterize a minimum point for the problem. Let us first define what is meant by a *minimum* point for the cost function  $f(\mathbf{x})$  before discussing the optimality conditions.

**Local Minimum.** The cost function  $f(\mathbf{x})$  has a *local minimum (relative minimum)* at a point  $\mathbf{x}^*$  in the *feasible set*  $S$  if the function value is the smallest at the point  $\mathbf{x}^*$  compared to all other points  $\mathbf{x}$  in a small *feasible neighborhood* of  $\mathbf{x}^*$ , i.e.,

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad (6)$$

If *strict inequality* (i.e.,  $f(\mathbf{x}^*) < f(\mathbf{x})$ ) holds, then  $\mathbf{x}^*$  is called the *strict* or *isolated local minimum*.

**Global Minimum.** The cost function  $f(\mathbf{x})$  has a *global minimum* (also called an *absolute minimum*) at a point  $\mathbf{x}^*$  if Inequality (6) holds for *all*  $\mathbf{x}$  in the feasible set  $S$ . If strict inequality holds, then  $\mathbf{x}^*$  is called the *strict* or *unique global minimum*.

These definitions show that for the local minimum, we test the inequality in Eq. (6) only for a small *feasible domain* around the point  $\mathbf{x}^*$ , and test it over the *entire* feasible set  $S$  for the global minimum point. Note that the cost function  $f(\mathbf{x})$  can have many global minimum points as long as the function values are the same at all the points. Similarly, there can be multiple local minima in the small feasible domain.

The foregoing definitions of local and global minima cannot be used directly to find the minimum points for the Problem P. However, they can be used to derive the *optimality conditions* that characterize a local minimum point. Note that they cannot be used to derive optimality conditions for a global minimum point for the function  $f(\mathbf{x})$ . The reason is that the global optimality conditions require knowledge about the global behavior of  $f(\mathbf{x})$ . For a discrete variable problem, the definitions are useful because there are only a finite number of points to be checked for optimality of a given point. In fact most stochastic methods for optimization of discrete variable problems, described in a later section, use the definitions to check optimality of a point in its neighborhood.

The optimality conditions can be divided into two categories: *necessary* and *sufficient*. The necessary conditions must be satisfied for a point to be a candidate minimum point. The points that satisfy the necessary conditions are called *stationary points*. Note however that a point satisfying the necessary conditions

need not be a minimum point, i.e., points that are not minima may also satisfy the necessary conditions. The *sufficient condition* if satisfied determines the stationary point to be a local minimum point. If the sufficient condition is not satisfied, no conclusion about the optimality of the stationary point can be drawn. We shall describe both the necessary and sufficient conditions. Sample problems showing the use of these conditions can be found in many textbooks.<sup>2-7,38</sup>

The optimality conditions are used in two ways: (i) they are used to develop numerical methods for finding minimum points, and (ii) they are used to check optimality of a given point; i.e., using them, a stopping criterion for the iterative numerical algorithm can be defined. We shall first present the optimality conditions for the unconstrained problem and then for the general constrained Problem P.

### 3.2. Optimality Conditions for the Unconstrained Problem

When there are no constraints, the problem is to minimize just the cost function  $f(\mathbf{x})$ . The conditions for  $\mathbf{x}^*$  to be a minimum point for the function  $f(\mathbf{x})$  are derived by analyzing the local behavior of the function at the point  $\mathbf{x}^*$ ; i.e., Taylor's expansion for the function.

**First Order Necessary Condition.** If  $\mathbf{x}^*$  is a local minimum for the cost function  $f(\mathbf{x})$ , then the gradient (*first derivatives*) of  $f(\mathbf{x})$  at  $\mathbf{x}^*$  must vanish, that is,  $\partial f / \partial x_i = 0$ ,  $i = 1$  to  $n$ .

**Second Order Necessary Condition.** If  $\mathbf{x}^*$  is a local minimum for the function  $f(\mathbf{x})$ , then its Hessian  $\mathbf{H} = [\partial^2 f / \partial x_i \partial x_j]$  at  $\mathbf{x}^*$  must be at least *positive semidefinite*; i.e., all its eigenvalues must be nonnegative.

**Second Order Sufficient Condition.** If the matrix  $\mathbf{H}(\mathbf{x}^*)$  is *positive definite* at the stationary point  $\mathbf{x}^*$ , then  $\mathbf{x}^*$  is an isolated local minimum point. (A matrix is called positive definite if all its eigenvalues are positive).

Any point  $\mathbf{x}^*$  satisfying the necessary conditions of optimality is called a *stationary point*. If a stationary point is neither a minimum nor a maximum, then it is called an *inflection point*. It should be noted that the optimality conditions are based on derivatives of  $f(\mathbf{x})$  and not the function value. Therefore, the minimum point is not changed if a constant is added to the function, or the function is scaled by a positive constant. The optimum value of the cost function does, however, change in the process.

### 3.3. Optimality Conditions for the Constrained Problem

We now present the optimality conditions for a constrained problem which involve constraints in addition to the cost function. Although a constrained problem can have minimum points where no constraints are *active*, this usually does not happen in practical applications. The case where no constraints are active, the optimum point is inside the feasible set  $S$  and the foregoing optimality conditions for the unconstrained problem apply; i.e., the optimality conditions for the unconstrained problem are a special case of those for the constrained problem. The conditions for the constrained problem can be expressed in several alternate but equivalent ways. We shall present the conditions that are most commonly used in the modern literature. These are known as *Karush-Kuhn-Tucker* or *KKT conditions*.

**Regular Point.** An assumption in the derivation of the KKT necessary conditions is that the minimum point be a regular point of the feasible set  $S$ . A point  $\mathbf{x}$  is called a *regular point* of the feasible set  $S$  if the cost function is continuous and the gradients of all the active constraints are linearly independent at the point. The number of linearly independent vectors cannot be more than  $n$ , the number of variables, i.e., dimension of each vector. Thus the total number of active constraints cannot be more than the number of variables at the regular point; i.e., at a minimum point.

**Karush-Kuhn-Tucker Necessary Conditions.** Let the *Lagrangian* for the Problem P be defined as

$$L(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}) + \mathbf{u} \bullet \mathbf{g}(\mathbf{x}) \quad (7)$$

where  $\mathbf{u}$  is a vector of *Lagrange multipliers* for the constraints  $\mathbf{g}$  that needs to be determined and a “ $\bullet$ ” implies scalar product of vectors. Let  $\mathbf{x}^* \in S$  be a local minimum for  $f(\mathbf{x})$ . Also, let the gradients of the active constraints at  $\mathbf{x}^*$  be linearly independent (i.e., point  $\mathbf{x}^*$  is a *regular point* of the feasible set). Then there exist *unique* Lagrange multipliers  $u_i^*$  such that

$$\nabla L(\mathbf{x}^*) = \mathbf{0}, \quad \text{or} \quad \nabla f(\mathbf{x}^*) + \nabla \mathbf{g}(\mathbf{x}^*) \mathbf{u}^* = \mathbf{0} \quad (8)$$

$$u_i^* g_i(\mathbf{x}^*) = 0, \quad i = (p+1) \text{ to } m \quad (9)$$

$$u_i^* \geq 0, \quad i = (p+1) \text{ to } m \quad (10)$$

where  $\nabla \mathbf{g}(\mathbf{x}^*)$  is an  $n \times m$  matrix. Equations (8) show that the Lagrangian  $L$  is stationary with respect to  $\mathbf{x}$  since the gradient of the Lagrangian is zero, Eq. (9) shows that either the Lagrange multiplier for the  $i$ th inequality is zero or the

constraint is active at the minimum point (if  $u_i = 0$ ,  $g_i$  must be  $\leq 0$  for feasibility of  $\mathbf{x}^*$ ), and Eq. (10) shows that the Lagrange multipliers for the inequality constraints must be nonnegative. In addition,  $\mathbf{x}^*$  must be a feasible point. Equation (9) is called the *switching condition* or *complementary slackness condition* because it identifies active and inactive inequality constraints. Note that there are  $n$  variables and  $m$  Lagrange multipliers, and thus  $(n+m)$  unknowns. There are  $(n+m)$  equations ( $n$  equations in conditions (8),  $p$  equalities, and  $m-p$  equations in condition (9)). Therefore, the necessary conditions give a determinate system of equations, though it is usually nonlinear.

The gradient condition of Eq. (8) can be re-arranged as:

$$-\nabla f(\mathbf{x}^*) = \nabla g(\mathbf{x}^*)\mathbf{u}^* \quad (11)$$

This form of the equation brings out the physical meaning of the gradient condition. It shows that at the minimum point, the steepest descent direction (negative of the cost function gradient) is in the range of gradients of the active constraints; i.e., a linear combination of them with Lagrange multipliers as the scalars of the linear combination.

The regularity check for  $\mathbf{x}^*$  is an important part of KKT conditions. If this check is not satisfied, all other KKT conditions may or may not be satisfied at  $\mathbf{x}^*$ . For example, the Lagrange multipliers may not be unique at  $\mathbf{x}^*$ . However, the irregular points can also be local minimum points where KKT conditions may be actually violated.

Second order optimality conditions can be used to distinguish the minimum points from others. These conditions also involve Hessians of the functions, as for the unconstrained problems; e.g., Hessians of the active constraints at  $\mathbf{x}^*$ . We briefly discuss these conditions next.<sup>1-7</sup>

**Second-order Necessary Condition.** Let  $\mathbf{x}^*$  satisfy the first order KKT necessary conditions for Problem P. Let the Hessian of the Lagrange function  $L$  at  $\mathbf{x}^*$  be defined as

$$\nabla^2 L = \nabla^2 f + \sum_{i=1}^m u_i^* \nabla^2 g_i \quad (12)$$

Let there be nonzero feasible directions,  $\mathbf{d} \neq \mathbf{0}$ , as solutions of the following linear system at  $\mathbf{x}^*$ :

$$(\nabla g_i \cdot \mathbf{d}) = 0, \quad i = 1 \text{ to } p \text{ and for those } i > p \text{ with } g_i(\mathbf{x}^*) = 0 \quad (13)$$

That is, the vectors  $\mathbf{d}$  are in the null space of the gradients of the active constraints. Then if  $\mathbf{x}^*$  is a local minimum point for the optimum design problem, it must be true that

$$Q \geq 0 \text{ where } Q = (\mathbf{d} \bullet \nabla^2 L(\mathbf{x}^*)\mathbf{d}) \quad (14)$$

Note that any point that does not satisfy the second-order necessary condition cannot be a local minimum point.

**Second-order Sufficient Condition.** Let  $\mathbf{x}^*$  satisfy the first-order KKT necessary conditions for Problem P. Let the Hessian of the Lagrange function  $L$  be defined at  $\mathbf{x}^*$  as in Eq. (12). Let there be nonzero feasible directions,  $\mathbf{d} \neq \mathbf{0}$ , as solutions of the following linear system at  $\mathbf{x}^*$ :

$$(\nabla g_i \bullet \mathbf{d}) = 0, \quad i = 1 \text{ to } p \text{ and for those } i > p \text{ with } g_i(\mathbf{x}^*) = 0 \text{ and } u_i^* > 0 \quad (15)$$

That is, the vectors  $\mathbf{d}$  are in the null space of the gradients of the active constraints with  $u_i^* > 0$  for  $i > p$ . Also let  $(\nabla g_i \bullet \mathbf{d}) \leq 0$  for those active inequalities with  $u_i^* = 0$ . If

$$Q \geq 0 \text{ where } Q = (\mathbf{d} \bullet \nabla^2 L(\mathbf{x}^*)\mathbf{d}) \quad (16)$$

then  $\mathbf{x}^*$  is an isolated local minimum point (isolated means that there are no other local minima in the neighborhood of  $\mathbf{x}^*$ ).

Equations (13) and (15) define vectors that are in the null space of the gradients of the active constraints. There is slight difference in the two null spaces defined by these equations. In Eq. (13), all the active inequalities are included. However in Eq. (15), only the active inequalities with positive multipliers are included. Note that if the Hessian  $\nabla^2 L$  is positive definite at  $\mathbf{x}^*$  then both the second order necessary and sufficient conditions for a local minimum are satisfied. Conversely, if it is negative definite or negative semidefinite, then the second order necessary condition is violated, and the point  $\mathbf{x}^*$  cannot be a minimum point.

### 3.4. Global Optimality and Convexity

Often a question is asked - is the optimum solution a global minimum? Usually the answer to this question is that the solution is only a local minimum. The global solution for the problem can be found by either an exhaustive search of the feasible set  $S$ , or by showing the problem to be convex. Both procedures require extensive computations. If the problem is convex, then any local minimum is also a global minimum and the KKT first order conditions are necessary as well as sufficient. The question of convexity of a problem is briefly addressed here. Methods for finding a global solution are described in a later section.

Problem P is called a *convex programming problem*, if the cost function  $f(\mathbf{x})$  is convex over the convex feasible set  $S$ . Therefore, we need to discuss convexity

of the feasible set  $S$  and the function  $f(\mathbf{x})$ . A set of points  $S$  (vectors  $\mathbf{x}$ ) is called a *convex set* if and only if for any two points A and B in the set  $S$ , the entire line segment AB is also in the set. Graphically this means that a convex set has no re-entrant corners or holes. By this definition, we see that linear equalities and inequalities always define a convex feasible set. Also, a nonlinear equality always defines a nonconvex feasible set. However, in general, the graphical definition is difficult to use to check convexity of a set because an infinite pair of points will have to be considered. Therefore, a better computational procedure is needed to check convexity of a function.

The feasible set for the Problem P is defined by the functions  $g_i(\mathbf{x})$ ,  $i = 1$  to  $m$ . It turns out that if all the functions are convex, then the feasible set  $S$  is convex. Thus we need to know how to check convexity of a function.

A function of  $n$  variables is convex if and only if its Hessian is at least positive semidefinite everywhere over its domain of definition. If a function  $g_i(\mathbf{x})$  is convex, then the set defined by the inequality  $g_i(\mathbf{x}) \leq e_i$  is convex, where  $e_i$  is any constant. Note that this is not an "if and only if" condition; that is if  $g_i(\mathbf{x})$  fails the convexity set, the feasible set defined by it may still be convex. In other words this is only a sufficient condition but it is not a necessary condition. Note that convexity checks for a problem are quite extensive. The Hessian of each nonlinear problem function needs to be evaluated and its form needs to be checked over the entire feasible domain.

The following points should be noted for *convex programming problems*:

- (i) A convex programming problem can have several global minimum points where the cost function has the same numerical value.
- (ii) The convexity check for a constraint function can sometimes fail if the form of the constraint is altered; however, the feasible set defined by the constraint may still be convex. Ref. 7 contains an example that illustrates this point.
- (iii) If the convexity checks fail, the problem can still have a global minimum point in the feasible set. However, it is difficult to conclude that a global solution has been reached.

### 3.5. Lagrange Multipliers

It turns out that the optimum values of the Lagrange multipliers for the constraints represent relative importance of the constraints with respect to the cost function. We discuss this importance here. Also, many times it is useful in practical applications to scale the cost function and constraints to avoid numerical instabilities. We discuss the affect of this scaling on the Lagrange multipliers for the constraints.

### 3.5.1. Changes in Constraint Limit

Let us first study how the optimum value of the cost function is affected if a constraint limit is changed; i.e., a constraint is relaxed or tightened. Assume that Problem P has been solved with the current limit values for the constraints as zero. Let  $e_i$  be a small variation in the right hand side of the  $i$ th constraint. It is clear that the optimum point for the perturbed problem is a function of the vector  $\mathbf{e}$ , i.e.,  $\mathbf{x}^* = \mathbf{x}^*(\mathbf{e})$ . Also  $f = f(\mathbf{e})$ . However, these are implicit functions of  $\mathbf{e}$ , and the following result gives a way of calculating the implicit derivatives  $\partial f / \partial e_i$ .<sup>6,7</sup>

**Sensitivity to Constraint Variations.** Let  $\mathbf{x}^*$  be a regular point that, together with the multipliers  $u_i^*$ , satisfies both the KKT necessary conditions and the sufficient conditions for an isolated local minimum point for the Problem P. If for each  $g_i(\mathbf{x}) = 0$  for  $i > p$ , it is true that  $u_i^* > 0$ , then the solution  $\mathbf{x}^*(\mathbf{e})$  of the modified problem is a continuously differentiable function of  $\mathbf{e}$  in some neighborhood of  $\mathbf{e} = \mathbf{0}$ . Furthermore,

$$\frac{\partial f(\mathbf{x}^*(\mathbf{0}))}{\partial e_i} = -u_i^*; \quad i = 1 \text{ to } m \quad (17)$$

It is useful to note that if the conditions stated in this result are not satisfied, existence of the implicit derivative of Eq. (17) cannot be ruled out. That is, the derivatives may still exist but their existence cannot be guaranteed. Using Eq. (17), we can estimate a change in the cost function due to a change in the right hand side of the  $i$ th constraint. First order Taylor expansion for the cost function about the point  $e_i = 0$  is given as

$$f(e_i) = f(0) + \frac{\partial f(0)}{\partial e_i} e_i \quad (18)$$

where  $f(0)$  is the optimum cost function value obtained with  $e_i = 0$ . Substituting from Eq. (17), we obtain change in the cost function  $\Delta f$  due to the change  $e_i$  as

$$\Delta f = f(e_i) - f(0) = -u_i^* e_i \quad (19)$$

Using the result of Eq. (19), we can show that the Lagrange multiplier corresponding to a " $\leq$  type" constraint must be nonnegative. To see this, let us assume that we want to relax an active inequality constraint  $g_i \leq 0$  by selecting  $e_i > 0$ . This way, the feasible set for the problem gets expanded. Thus the minimum value for the cost function should reduce or stay unchanged with the expanded feasible set. However, Eq. (19) shows that if  $u_i^* < 0$ , relaxation of the constraint ( $e_i > 0$ ) results in an increase in cost (i.e.,  $\Delta f > 0$ ). This is not possible, and therefore, the Lagrange multiplier for a " $\leq$  type" constraint cannot be negative.

### 3.5.2. Scaling of Cost Function.

Some times in practical applications, the cost function for the problem is normalized by multiplying it with a positive constant. Although this scaling does not affect the optimum point, it does change the Lagrange multipliers for all the constraints. Using the KKT conditions of Eq. (8), it can be shown that all the Lagrange multipliers also get multiplied by the same scale factor.<sup>7</sup> Let  $u_i^*$  be the Lagrange multiplier for the  $i$ th constraint with the original cost function. Let the cost function be scaled as  $f^{new} = \alpha f$ , where  $\alpha > 0$  is a given constant, and  $u_i^{new}$  be the new value of the Lagrange multiplier for the  $i$ th constraint at optimum. Then the new and old Lagrange multipliers are related as

$$u_i^{new} = \alpha u_i^*; \quad i = 1 \text{ to } m \quad (20)$$

### 3.5.3. Scaling of Constraints

In numerical calculations, it is useful to normalize all the constraints (normalization of constraints is discussed in the next section). This scaling of a constraint does not change its boundary, so it has no effect on the optimum point or the cost function. However, the Lagrange multiplier for the constraint is affected. Using the KKT conditions of Eq. (8), it can be shown that the Lagrange multiplier for the scaled constraint gets divided by the same scale factor.<sup>7</sup> Let the  $i$ th constraints  $g_i$  be divided by  $\beta_i > 0$  as  $g_i^{new} = g_i/\beta_i$  and  $u_i^*$  and  $u_i^{new}$  be the corresponding Lagrange multipliers for the original and the scaled constraints, respectively. The new and original Lagrange multipliers are related as

$$u_i^{new} = \beta_i u_i^* \quad (21)$$

## 4. Basic Concepts Related to Computational Algorithms

Optimization methods for structural and mechanical systems have matured to the point where they are being used routinely in many practical applications. Many journals dedicated to the field of optimization and many textbooks on the subject can be consulted for the range of applications. Various chapters of this book contain a good sample of practical applications.

Real-world problems are usually quite complex. Each application has its own requirements, simulation methods and constraints to meet. In addition, the desire to solve more complex and larger problems also grows as computer-based computational tools improve. Furthermore, since the methods have matured substantially during the last decade, more nonexperts of optimization techniques

are beginning to use this new methodology in their routine work. These considerations dictate the use of a theoretically sound and numerically reliable algorithm. Use of such an algorithm can remove uncertainty about the algorithm behavior, allowing the users to concentrate on their application. Such theoretically sound algorithms, although computationally more expensive, are more cost-effective in the long run.

In the remaining sections, some basic concepts related to numerical algorithms for optimization of structural and mechanical systems are presented and discussed. Algorithms for continuous variable problems as well as discrete variable problems are outlined. The ideas of a descent function, constraint normalization, and potential constraint strategy are introduced. Convergence of an algorithm is discussed and attributes of a good algorithm are presented. It is important to note that the gradient-based algorithms converge only to a *local minimum* point for the Problem P. Algorithms for finding a *global solution* require extensive numerical calculations and are outlined in a later section. Multiobjective optimization algorithms are also discussed.

#### 4.1. A Basic Gradient-based Algorithm

Gradient-based optimization algorithms use the following iterative prescription:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}; \quad k = 0, 1, 2, \dots \quad (22)$$

where the superscript  $k$  represents the iteration number,  $\mathbf{x}^{(k)}$  is the current estimate of the optimum design,  $\alpha_k \mathbf{d}^{(k)}$  is a change in design,  $\alpha_k > 0$  is a step size,  $\mathbf{d}^{(k)}$  is a search direction, and  $\mathbf{x}^{(0)}$  is the starting point.

Gradient-based algorithms are broadly classified as *primal methods* and *transformation methods*. In the *primal methods* the direction vector  $\mathbf{d}^{(k)}$  is calculated using the problem functions and their gradients at the point  $\mathbf{x}^{(k)}$ . Then the step size is calculated along  $\mathbf{d}^{(k)}$  that needs only the function values. Different algorithms can be generated depending on how the direction  $\mathbf{d}$  and step size  $\alpha$  are calculated. In many algorithms,  $\mathbf{d}$  is calculated by solving a linear or quadratic programming subproblem. Several philosophies have been used to develop various algorithms. For example, if an intermediate point or the starting point is infeasible, many methods iterate through the infeasible region to reach the final solution; many others correct the constraints to reach the feasible set first and then move along the boundary to reach the solution point. Still others make special calculations not to violate constraints during the iterative process. Some algorithms generate and use second order information for the problem as the iterations progress.

In the *transformation methods* the solution process for Problem P is transformed to a sequence of unconstrained minimization problems. Solutions of the unconstrained problems converge to solution of the original problem. They include *barrier* and *penalty function methods* as well as the *augmented Lagrangian* or *multiplier methods*.<sup>7,15,19,20</sup> In the transformation methods, a transformed function is constructed by adding a penalty term for the constraint violations to the cost function, as  $\Phi(\mathbf{x}, r) = f(\mathbf{x}) + P(\mathbf{g}(\mathbf{x}), r)$ , where  $r$  is a scalar or vector of penalty parameters and  $P$  is a real valued function whose action of imposing the penalty is controlled by  $r$ .

Many methods have been developed and evaluated based on the strategies described in the foregoing. Robust and general algorithms are based on the following four basic steps:

- (i) Linearization of cost and constraint functions about the current point.
- (ii) Definition of a search direction determination subproblem.
- (iii) Solution of the subproblem for the search direction.
- (iv) Calculation of a step size along the search direction.

## 4.2. Constraint Normalization

It is useful to normalize all the constraint functions in numerical calculations because it is not easy to determine which constraint is more severely violated if they are not normalized. Also, in numerical calculations, one value for the parameter to check feasibility of all the constraints cannot be used. As examples, consider a stress constraint as  $\sigma \leq \sigma_a$  and a displacement constraint as  $\delta \leq \delta_a$ , where  $\sigma$  is the calculated stress,  $\sigma_a > 0$  is an allowable stress,  $\delta$  is the calculated deflection, and  $\delta_a > 0$  is an allowable deflection. Since the units for the two constraints are different their values are of widely differing orders of magnitude. If they are violated during the iterative solution process, it is difficult to judge the severity of their violation. However, if they are normalized as  $R - 1.0 \leq 0$ , where  $R = \sigma / \sigma_a$  for the stress constraint, and  $R = \delta / \delta_a$  for the deflection constraint, then it is easy to compare their values.

## 4.3. Potential Constraint Strategy

The optimization methods solve a subproblem to determine the search direction at each iteration. The subproblem is defined using gradients of the constraints. A subproblem that uses gradients of only a subset of the constraints is said to use a *potential constraint strategy*. The potential constraint set is comprised of the indices of active, nearly active and violated constraints, such as the index set  $I_k$

at the  $k$ th point  $\mathbf{x}^{(k)}$ :

$$I_k = \{i \mid i = 1 \text{ to } p \text{ and all } i > p \text{ with } (g_i + \varepsilon) \geq 0\} \quad (23)$$

where  $\varepsilon > 0$  is a small number used to determine nearly active inequalities. Note that the equality constraints are always included in the index set  $I_k$ .

#### 4.4. Descent Function

It is important to monitor progress of the iterative optimization process towards the minimum point. This can be done if a function can be defined that decreases at every iteration. Such a function is called the *descent function* or the *merit function*. The cost function is a descent function for the unconstrained optimization problems because it is required to reduce at each iteration. For constrained problems, many descent functions have been used. These functions must include the effect of constraint violations. The descent function is used in the process of step size determination. The basic idea is to compute a step size along the search direction  $\mathbf{d}^{(k)}$  such that the descent function is decreased. The descent function also has the property that its minimum value is the same as the cost function.

#### 4.5. Convergence of an Algorithm

An algorithm that has been proven to converge starting from an arbitrary point is called a *globally convergent* method, and satisfies two requirements: (i) there is a descent function for the algorithm, and (ii) the search direction  $\mathbf{d}^{(k)}$  is a continuous function of the variables. This requirement implies that the active constraints are not coming in-and-out of the active set. This is called "zigzagging" of constraints.

#### 4.6. Attributes of a Good Algorithm

A good algorithm for practical applications should have the following attributes:

- (i) *Robustness*: The algorithm must be convergent to a local minimum point starting from any initial estimate.
- (ii) *Generality*: The algorithm must be able to treat equality as well as inequality constraints.
- (iii) *Accuracy*: The algorithm must be able to converge to an optimum point as accurately as desired.
- (iv) *Ease of Use*: Implementation of the algorithm must be such that it requires

minimum of input for use of the algorithm by the experienced as well as inexperienced users.

- (v) *Efficiency*: The algorithm must have a faster rate of convergence, i.e., at least superlinear. The algorithm should be able to treat linear constraints efficiently. It should be able to exploit sparsity structure of the problem functions, especially for large-scale problems.

## 5. Overview of Computational Algorithms

Many numerical methods have been developed and evaluated for constrained and unconstrained optimization problems.<sup>1-7,20,33</sup> In addition, algorithms for discrete variable and nondifferentiable problems have been discussed.<sup>16</sup> Many practical applications require optimization of several objective functions, and therefore, procedures to treat multiple objectives in an optimization problem have been developed. In this section, we describe the basic concepts of these algorithms.

### 5.1. Gradient-based Algorithms

The gradient-based methods are suitable for problems with continuous variables and differentiable functions because they utilize gradients of the problem functions. The methods have been thoroughly researched and a considerable body of literature is available on the subject. They include sequential quadratic programming and augmented Lagrangian methods. We discuss the basic concepts related to these methods. The interior point methods, developed initially for linear problems, have also been extended for nonlinear problems.

#### 5.1.1. Linearization and Sequential Linear Programming

All search methods start with an initial estimate for the optimum point and iteratively improve it. The improvement is computed by solving an approximate subproblem which is obtained by writing linear Taylor's expansions for the cost and constraint functions. Let  $\mathbf{x}^{(k)}$  be the estimate for the optimum point at the  $k$ th iteration and  $\Delta\mathbf{x}^{(k)}$  be the desired change. Instead of using  $\Delta\mathbf{x}^{(k)}$  as a change in the current point, usually it is taken as the search direction  $\mathbf{d}^{(k)}$  and a step size is calculated along it to determine the new point. We write Taylor's expansion of the cost and constraint functions about the point  $\mathbf{x}^{(k)}$  to obtain a linearized subproblem as

$$\text{minimize } (\mathbf{c} \bullet \mathbf{d}) \quad (24)$$

subject to the linearized equality constraints

$$(\nabla g_j \bullet \mathbf{d}) = e_j; \quad j = 1 \text{ to } p \quad (25)$$

and the linearized inequality constraints

$$(\nabla g_j \bullet \mathbf{d}) \leq e_j; \quad j > p \text{ and } j \in I_k \quad (26)$$

where  $e_j = -g_j(\mathbf{x}^{(k)})$ , and  $\mathbf{c} = \nabla f(\mathbf{x}^{(k)})$ . Note that a potential constraint strategy for inequality constraints is used in Eq. (26). If it is not to be used,  $\varepsilon$  can be set to a very large number in defining the index set  $I_k$  in Eq. (23).

Since all the functions in Eqs. (24) to (26) are linear in the variables  $d_i$ , linear programming can be used to solve for  $d_i$ . Such procedures are called *Sequential Linear Programming* methods or in short *SLP*. Note, however, that the problem defined in Eqs. (24) to (26) may not have a bounded solution. Therefore, limits must be imposed on changes in the variables. These constraints are called *move limits* in the optimization literature and can be expressed as

$$-\Delta_i \leq d_i \leq \Delta_i; \quad i = 1 \text{ to } n \quad (27)$$

where  $\Delta_i$  is the maximum allowed decrease or increase in the  $i$ th variable, respectively at the  $k$ th iteration. The problem is still linear in terms of  $d_i$ , so LP methods can still be used to solve it. Selection of the move limits at every iteration is important because success of the SLP algorithm depends on them. However, selection of proper move limits is quite difficult in practice.

### 5.1.2. Sequential Quadratic Programming - SQP

To overcome drawbacks of SLP, sequential quadratic programming methods (SQP) have been developed where a *quadratic programming (QP) subproblem* is solved to find a search direction and a descent function is used to calculate a step size in that direction.

#### Subproblem QP.

$$\text{Minimize } (\mathbf{c} \bullet \mathbf{d}) + \frac{1}{2}(\mathbf{d} \bullet \mathbf{H} \mathbf{d}) \quad (28)$$

subject to the linearized constraints in Eqs. (25) and (26) where  $\mathbf{H}$  is an  $n \times n$  matrix that is an approximation to the Hessian of the Lagrangian function.

Different definitions of the QP subproblem generate different search directions. Once a direction has been determined, a step size is calculated by minimizing a descent function along it. The descent function for the constrained problems is constructed by adding a penalty for constraint violations to the cost function. One of the properties of the descent function is that its value at the

optimum point be the same as that for the cost function. Also, it must reduce along the search direction at each iteration. In other words, the search direction must be a *descent direction* for the function. Several descent functions have been developed and used with different algorithms. We shall introduce Pshenichny's descent function  $\Phi$  due to its simplicity and success in solving a large number of problems.<sup>4,18,24,33,34</sup> It is the exact penalty function defined as

$$\Phi(\mathbf{x}) = f(\mathbf{x}) + RV(\mathbf{x}) \quad (29)$$

where  $R > 0$  is a penalty parameter and  $V(\mathbf{x}) \geq 0$  is the maximum constraint violation among all the constraints. Note that  $R$  is required to be finite but larger than the sum of the magnitude of all the Lagrange multipliers.

It is important to note that calculation of an exact minimum point for the descent function along the search direction is quite costly. Therefore in most practical implementations of any optimization algorithm, only an approximate step size is determined. This is done using the so-called *inaccurate* or *inexact line search*. In the inaccurate line search procedure, one starts with the trial step size as one. If the descent condition is not satisfied, the trial step is taken as half of the previous trial. If the descent condition is still not satisfied, the trial step size is bisected again. The procedure is continued, until the descent condition is satisfied; i.e., a sufficient reduction in the descent function has been achieved. Performance of several SQP algorithms has been evaluated in Ref. 33.

### 5.1.3. Augmented Lagrangian Method

There is a class of computational methods that transform the constrained problem to an unconstrained problem and solve it by using unconstrained optimization methods. These are called *sequential unconstrained minimization techniques*.<sup>1</sup> The basic idea of these methods is to define an augmented functional by adding a penalty term to the cost function. The penalty term consists of the constraint functions multiplied by the penalty parameters. The penalty parameters are selected and the unconstrained function is minimized. Then the penalty parameters are increased and the unconstrained function is minimized again. The procedure is repeated until there is very little change in the solution. An advantage of the methods is that the unconstrained optimization algorithms and the associated software can be used to solve constrained problems. One drawback of the methods is that the penalty parameters are required to go to infinity to obtain an optimum solution. This can cause instability in numerical calculations.

To overcome difficulty of the foregoing methods, a different class of methods has been developed that do not require the penalty parameters to become infinite. The penalty parameters are required to be sufficiently large but

finite. These are called the augmented Lagrangian methods or the *multiplier methods*. The augmented functional is defined as

$$\Phi = f(\mathbf{x}) + \frac{1}{2} \sum_{i=1}^p r_i (g_i + \theta_i)^2 + \frac{1}{2} \sum_{i=p+1}^m r_i [(g_i + \theta_i)_+]^2 \quad (30)$$

where  $r_i > 0$  are the penalty parameters,  $\theta_i$  for  $i = 1$  to  $p$  are the multipliers for the equality constraints,  $\theta_i \geq 0$  for  $i > p$  are the multipliers for the inequality constraints, and  $(x)_+ = x$  if  $x > 0$ , and  $(x)_+ = 0$  if  $x \leq 0$ . The idea of multiplier methods is to start with some values for the parameters  $r_i$  and  $\theta_i$  and minimize the augmented function of Eq. (30). These parameters are then adjusted using some procedure and the process is repeated until optimality conditions are satisfied. For more detailed discussion and applications of the methods, Refs. 10, 15, 19 and many works cited therein should be consulted.

It is important to note that the augmented functional, such as the one in Eq. (30), have been used as descent functions for many SQP methods to determine an appropriate step size along the search direction.<sup>34</sup>

## 5.2. Algorithms for Discrete Variable Problems

The continuous variable optimization problem has infinite feasible points when the feasible set is nonempty. In contrast, the discrete variable problem has only a finite number of feasible points from which the optimum solution needs to be determined. However, it is more difficult and time consuming to find an optimum solution for the discrete variable problem compared to the continuous variable problem. The reason is that there are no optimality conditions to guide the numerical search process. We usually need to enumerate on the discrete points and use the definition of the minimum point in Eq. (6) to find the best solution. Many methods try to reduce this computational burden by using stochastic ideas or heuristic rules.

The solution algorithm for a mixed-discrete variable optimization problem depends on the type of problem. Five types of mixed variable problems are defined in Refs. 7, 11, 12 and 16 based on the characteristics of variables and problem functions. Also methods to solve the problems are identified. For example, if the problem functions are continuous and the discrete variables can have non-discrete values during the solution process, then gradient-based algorithms can be used to guide the search for a discrete optimum solution. If the problem functions are nondifferentiable and discrete variables must have only discrete values, then implicit or explicit enumeration methods or stochastic methods can be used to solve the problem.

There are basically two classes of methods for solving discrete variable problems: (i) enumeration methods, either implicit or explicit, such as the branch and bound algorithm, and (ii) stochastic or evolutionary methods, such as genetic algorithms and simulated annealing. Detailed review of the methods and their applications are presented in Refs. 7, 11, 12, 14, 16, 25-30. Here we summarize basic concepts and ideas of the methods from these references.

**Branch and Bound Method.** This is one of the most commonly used methods to solve discrete variable problems.<sup>7,12,16</sup> It is also called an implicit enumeration method because one systematically tries to reduce the entire enumeration. It was initially developed for LP problems for which a global solution is obtained. The method has also been applied to nonlinear problems for which there is no guarantee of optimum or even a feasible solution. The method uses the concepts of *branching*, *bounding* and *fathoming* to perform the search for the optimum solution. The solution space for the problem is represented as branches of an inverted tree. Each node of the tree represents a possible discrete solution. If the solution is infeasible, then either the branch is truncated if the cost function is higher than a previously established upper bound, or other branches are searched for a better solution from that node. A node is said to be *fathomed* if no better solution is possible with further branching from that node. When the solution at a node is feasible, it either represents a new upper bound for the optimum if the cost function is smaller than a previously established bound, or the node can be fathomed if no better solution is possible with further branching. The method can be implemented in two different ways. In the first one, non-discrete values for the discrete variables are not allowed during the solution process. Therefore enumeration on the discrete variables needs to be done as explained above. In the second implementation, non-discrete values for the variables are allowed. Forcing a variable to have a discrete value generates each node of the tree. This is done by defining a subproblem with appropriate constraints on the variable to force out a discrete value for the variable. The subproblem is solved using either LP or NLP methods.

**Simulated Annealing.** Simulated annealing (SA) is a stochastic method that can be used to find the global minimum for a mixed variable nonlinear problem.<sup>7</sup> The method does not require continuity or differentiability of the problem functions. The basic idea is to generate random points in a neighborhood of the current best point and evaluate the problem functions there. If the cost function (penalty function for constrained problems) value at any of those points is smaller than the current best value, then the point is accepted, and the best cost function value is

updated. If it is not, then the point is sometimes accepted and sometimes rejected. The acceptance is based on the value of the probability density function of Boltzman-Gibbs distribution. If this probability density function has a value greater than a random number, then the trial point is accepted as the best solution. The probability density function uses a parameter called the temperature. For the optimization problem, this temperature can be the target value for the cost function. Initially, a larger target value is selected. As the trials progress, the target value is reduced (this is called the cooling schedule), and the process is terminated after a large number of trials. The acceptance probability steadily decreases to zero as the temperature is reduced. Thus in the initial stages, the method is likely to accept worse points while in the final stages, the worse points are usually rejected. This strategy avoids getting trapped at local minimizers. The main deficiencies of the method are the unknown rate at which the target level is to be reduced and uncertainty in the total number of trials.

**Genetic Algorithms.** As simulated annealing, these methods are also in the category of stochastic search methods.<sup>35,37,40-44</sup> In the methods, a set of alternative points (called the *population*) at an iteration (called *generation*) is used to generate a new set of points. In this process, combinations of the most desirable characteristics of the current members of the population are used that results in points that are better than the current ones. Thus, the average fitness of successive sets of points improves giving better values for the fitness function. Here fitness is defined using the cost function or the penalty function for constrained problems. The fitness value is calculated for each member of the population. An advantage of this approach is that derivatives of the functions are not needed. One starts with a set of randomly generated points. A finite length string, such as a binary string of 0's and 1's, is usually used to represent each point. Three operators are needed to implement the algorithm: (i) reproduction; (ii) crossover; and (iii) mutation. *Reproduction* is an operator where an old string (point) is copied into the new population according to its fitness. More highly fit strings (those points with smaller fitness values) receive higher numbers of offspring (new points). The crossover operator corresponds to allowing selected members (points) of the population to exchange characteristics of the points among themselves. Crossover entails selection of starting and ending positions on a pair of mating strings (points) at random and simply exchanging the string of 0's and 1's between these positions. *Mutation* corresponds to selection of a few members (points) of the population, determining a location on the strings at random, and switching the 0 to 1 or vice versa. The foregoing three steps are repeated for successive generations of the population until no further

improvement in the fitness is attainable, or the number of generations reaches a specified limit. The member in this generation with the highest level of fitness is taken as the optimum point.

**Integer Programming.** The problem is called an integer programming (IP) problem when the variables are required to take on integer values. If all the functions are linear, an integer linear programming (ILP) problem is obtained, otherwise it is nonlinear. The ILP problem can be converted to 0-1 programming problem. Linear problems with discrete variables can also be converted to 0-1 programming problems. Many algorithms are available to solve such problems,<sup>45</sup> such as the branch and bound method discussed earlier.

**Sequential Linearization Methods.** Nonlinear discrete optimization problems can also be solved by sequential linearization procedures. The functions of the problem must be differentiable to use such a procedure. The nonlinear problem is first linearized at the current point. Then an ILP method is used to solve the linearized subproblem. A modification of this approach is to obtain a continuous optimum point first, and then linearize and use IP methods. This process can reduce the number of ILP problems to be solved. Restricting the number of discrete values to a neighborhood of the continuous solution can also reduce the size of the ILP problem.

**Rounding-off Techniques.** Rounding-off is a simple approach where an optimum solution is first obtained by assuming all the variables to be continuous. Then using heuristics, the variables are rounded-off to the nearest available discrete values to obtain a discrete solution. The procedure is applicable to a restricted class of problems where discrete variables can have non-discrete values during the solution process. The process may not result in a feasible point for the discrete variable problem. Note that it is not necessary to round-up all variables to their nearest discrete neighbors. Some of them could be rounded-down while others could be increased. The difficulty with this approach is in the selection of variables to be increased and the variables to be decreased. The strategy may not converge, especially in case of high nonlinearity and widely separated allowable discrete values. In that case, the discrete minimizer need not be in a neighborhood of the continuous solution. As an alternative, a dynamic rounding-off strategy has been used where only one variable is rounded-off to its discrete neighbor at a time. The selected variable is then fixed at the discrete value and the problem is optimized again. This process is repeated until all variables are selected and fixed to discrete values.

**Neighborhood Search Method.** Some times it is reasonable to enumerate on the discrete variables, especially when the number of variables is small. With all the discrete variables fixed at their chosen values, the problem is then optimized for the continuous variables. This approach has some advantages over BBM: it can be implemented easily with an existing NLP solver, the problem to be solved is smaller and the gradient information with respect to the discrete variables is not needed. However, in general, the approach is less efficient than an implicit enumeration method, such as the BBM, as the number of discrete variables and size of the discrete set of values become large. To reduce the number of enumerated cases, a neighborhood search method has been used which first obtains a continuous solution with all the discrete variables considered as continuous. Then only a few discrete values near the continuous solution are selected for explicit enumeration.

### 5.3. Multiobjective Optimization

There are many practical applications where we need to optimize two or more objective functions simultaneously. These are called *multiobjective*, *multi-criteria*, or *vector optimization* problems. Here, we give a brief introduction to the subject by describing some basic concepts, terminology and solution methods. Material for this section is derived from Refs. 7 and 32; for more details, references cited in there and many other sources can be consulted, such as Refs. 35-37, 44.

#### 5.3.1. Terminology and Basic Concepts

The Problem P defined earlier is modified to multiobjective optimization problems as follows: find  $\mathbf{x} \in S$  to minimize

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_k(\mathbf{x})) \quad (31)$$

where  $k$  is the number of objective functions in the vector  $\mathbf{f}(\mathbf{x})$ . A collection of all the objective function vectors is called the *criterion space*. The feasible criterion space  $Z$  is defined as the set of objective function values corresponding to the feasible points in the variable space; i.e.,

$$Z = \{\mathbf{f}(\mathbf{x}) \mid \mathbf{x} \in S\} \quad (32)$$

Algorithms for solution of a single-objective optimization problem give local minima for the cost function in the feasible set. If all local minima are found,

then a global minimum point can be identified. In contrast, the process of solving a multiobjective optimization problem is less definite. Usually this problem does not have a unique solution; i.e., there is no point  $\mathbf{x}$  that minimizes all the objectives simultaneously. Therefore, it is not clear what is meant by the minimum of multiple objective functions. Usually, the objectives have opposing characteristics, since a point that decreases the value of one function may increase the value of another. However, there can be infinite solution points for the problem in the sense of *Pareto optimality*. This is the predominant concept in defining solutions for multiobjective optimization problems that is discussed next.

**Pareto Optimal Points.** A point  $\mathbf{x}^* \in S$  is *Pareto optimal* if and only if there does not exist another point  $\mathbf{x} \in S$  such that  $\mathbf{f}(\mathbf{x}) \leq \mathbf{f}(\mathbf{x}^*)$  with at least one  $f_i(\mathbf{x}) < f_i(\mathbf{x}^*)$ . In other words, a point  $\mathbf{x}^* \in S$  is called Pareto optimal if there is no other point  $\mathbf{x} \in S$  that reduces at least one objective function without increasing another one. Pareto optimal points are also called *efficient points* of the feasible set  $S$ .

**Non-dominated Points.** Another common concept is that of *non-dominated* and *dominated* points of the feasible criterion space  $Z$ . A vector of objective functions  $\mathbf{f}^* = \mathbf{f}(\mathbf{x}^*) \in Z$  is *non-dominated* if and only if there does not exist another vector  $\mathbf{f} \in Z$  such that  $\mathbf{f} \leq \mathbf{f}^*$  with at least one  $f_i < f_i^*$ . Otherwise,  $\mathbf{f}^*$  is *dominated*.

**Utopia Point.** A vector of objective function values  $\mathbf{f}^\circ$  in the criterion space is called the utopia point if  $f_i^\circ = \min \{f_i(\mathbf{x}) \mid \text{for all } \mathbf{x} \in S\}$ ,  $i = 1$  to  $k$ . It is also called the *ideal point*. Utopia point is a unique point in the criterion space that is obtained by minimizing each objective function without regard for other objective functions. Each minimization yields a point in the variable space and the corresponding value for the objective function. It is rare that each minimization will end up at the same point. That is, one point cannot simultaneously minimize all the objective functions. Thus, the utopia point exists only in the criterion space and, in general, is not attainable in the variable space.

**Compromise Solution.** Since the utopia point is not attainable, the next best thing is a solution that is as close as possible to the utopia point. Such a solution is called a *compromise solution*. The methods that seek different compromise solutions are collectively called *compromise programming*.

### 5.3.2. Solution Methods

Since the multiobjective optimization problem has infinite solutions (the *Pareto optimal set*), the user needs to select a solution that suits the requirements of the application. Therefore we may need to generate the entire Pareto set or at least a good representation of it so that the user can select the desired solution. Most solution methods for multiobjective optimization problems combine various objective functions to define a composite scalar function for the problem. This way, a single-objective optimization method can be used to solve the problem. By varying parameters of the composite function, different optimum solutions for the problem can be generated. Some methods always yield Pareto optimal solutions but may skip certain points in the Pareto optimal set; i.e., they may not be able to capture all of the Pareto optimal points. Alternatively, other methods are able to capture all of the points in the Pareto optimal set but may also provide non-Pareto optimal points as well. The former quality is beneficial when one is interested in using a method to obtain just one solution point. The latter quality is useful when the complete Pareto optimal set needs to be generated.

**Weighted Sum Method.** The weighted sum method is the most common approach to multiobjective optimization. Each objective function is scaled by a weighting factor  $w_i > 0$  as  $w_i f_i(\mathbf{x})$ . Then all the objective functions are added together to form a composite objective function to be optimized:

$$U = \sum_{i=1}^k w_i f_i(\mathbf{x}) \quad (33)$$

The objective functions are usually normalized before the weights are assigned to them. The relative value of the weights generally reflects the relative importance of the objectives. This is another common characteristic of the weighted sum methods. If all of the weights are omitted or are set to one, then all objectives are treated equally. The weights can be used in two ways. The user may either set  $w_i$  to reflect preferences before the problem is solved, or systematically alter them to yield different Pareto optimal points (generate the Pareto optimal set). The method is quite easy to use; selection of proper weights is the most difficult part that requires thorough knowledge of the objective functions and their relative importance.

**Weighted Global Criterion.** A broader class of weighted sum methods is based on weighted global criterion which is defined as:

$$U = \left\{ \sum_{i=1}^k [w_i (f_i(\mathbf{x}) - f_i^\circ)]^p \right\}^{1/p} \quad (34)$$

The root  $1/p$  may be omitted because the formulations with and without the root theoretically provide the same solution. The solution with this formulation depends on the values of both  $w_i$  and  $p$ . Generally,  $p$  is proportional to the amount of emphasis placed on minimizing the function with the largest difference between  $f_i(\mathbf{x})$  and  $f_i^\circ$ . Larger  $p$  puts more emphasis on minimizing the largest difference.  $p$  and  $w_i$  typically are not varied or determined in unison. Rather, a fixed value for  $p$  is selected, and then, either  $w_i$  is selected to reflect preferences before the problem is solved, or it is systematically altered to yield different Pareto optimal points. For computational efficiency or in cases where the utopia point  $f_i^\circ$  may be difficult to determine, one may replace  $f_i^\circ$  with an approximate value for it in Eq. (34). The approximation for  $f_i^\circ$  is called an *aspiration point*, *reference point*, *goal*, or *target point*. When this is done,  $U$  is called an *achievement function*.

The global criterion reduces to other common methods with different values of  $p$ . For instance, when  $p = 1$ , Eq. (34) is similar to a weighted sum with the objective functions adjusted with the utopia point. When  $p = 2$  and weights equal to 1, Eq. (34) represents the distance of the current point  $f_i(\mathbf{x})$  from the utopia point, and the solution usually is called *compromise solution* as mentioned earlier. When  $p = \infty$ , Eq. (34) reduces to the well known min-max method.

**Other Methods.** There are other useful methods that reduce the multiobjective optimization problem to a single-objective optimization problem:

- *Lexicographic method* where the objective functions are arranged in the order of their importance and a sequence of optimization problems is solved: minimize  $f_i(\mathbf{x})$  subject to  $f_j(\mathbf{x}) \leq f_j(x_j^*)$ ;  $j=1$  to  $(i-1)$ ;  $i > 1$ ;  $i = 1$  to  $k$ . The process is stopped when two consecutive problems have same solution.
- The  *$\epsilon$ -constraint method* minimizes a single, most important objective function  $f_s(\mathbf{x})$  with other objective functions treated as constraints:  $f_i(\mathbf{x}) \leq \epsilon_i$ ;  $i = 1$  to  $k$ ;  $i \neq s$ , where  $\epsilon_i$  is the upper limit for the objective function  $f_i(\mathbf{x})$ . A systematic variation of  $\epsilon_i$  yields a set of Pareto optimal solutions.
- *Goal programming* approaches set goals  $b_j$  for each objective function  $f_j(\mathbf{x})$ . Then, the total deviation from the goals is minimized. In the absence of any other information, goals may be set to the utopia point, i.e.,  $b_j = f_j^\circ$ . In that case, the method becomes a special case of the global criterion method.

Besides the scalarization methods discussed in the foregoing paragraphs, there are methods that treat all the objective functions at the same time and generate the Pareto optimal set for the problem. A prominent method in this class is the *genetic algorithm* for multiobjective optimization problems.<sup>35-37,44</sup> This method is an extension of the genetic algorithms described earlier for single objective problems. Additional genetic operators are used to generate the new population for the next generation. For each generation, a possible set of Pareto optimal points for the problem is identified. These play a major role in generating new points for the next generation. The iterative process is repeated for a long period of time. At the end, an approximation to the Pareto optimal set is obtained. Since genetic algorithms do not require gradient information, they can be effective regardless of the nature of the problem functions.

#### 5.4. Algorithms for Global Solution

Thus far, we have addressed mainly the problem of finding a local minimum for the cost function. However, in some practical applications, it is important to find globally optimum solutions as opposed to the local ones. The question of when a local solution is also a global optimum is quite difficult to answer because there are no mathematical conditions that characterize a global solution, except for convex programming problems, as discussed earlier. Therefore even when a global solution has been found, it is not possible to recognize it. Due to this reason, it is impossible to define a precise stopping criterion for a computational algorithm for global optimization. Usually, the best solution obtained by an algorithm after it is allowed to run for a long time is accepted as the global solution for the problem. In general, the quality of the solution depends on how long the algorithm is allowed to run. It is important to note that the computational effort to solve a global optimization problem increases enormously as the number of design variables increase. Thus, it remains a challenge to solve the global optimization problem efficiently.

In this section, we present some basic concepts of procedures that can be used to calculate a global solution. We consider the problem with continuous variables and functions. For discrete and nondifferentiable problems, the simulated annealing and genetic algorithms, described earlier, can be used for global optimization. In general, global optimization methods can be divided into two major categories: *deterministic* and *stochastic*. This classification is based on whether or not they incorporate any stochastic procedures to solve the global optimization problem. In the following subsections, we describe basic concepts of some of the methods in both of these categories. The material is derived from

the work of the author and his co-workers.<sup>7,17,22</sup> Numerous other references cited in these articles can be consulted for more details; e.g., Refs. 46-52.

#### 5.4.1. *Deterministic Methods*

An exhaustive search of the feasible set  $S$  is performed in these methods to find the global minimum. The success of the method can be guaranteed for only the functions that satisfy certain conditions. We shall describe basic ideas of four deterministic methods: covering, zooming, generalized descent and tunneling methods.

**Covering Methods.** The basic idea of these methods is to cover the entire feasible set  $S$  by evaluating the cost function at all the points in order to search for a global minimum.<sup>46</sup> This is an enormous calculation and therefore all the covering methods try to implicitly cover the entire set by evaluating the functions at some selected points. Some methods exploit certain properties of the cost function to accomplish this objective. Covering methods have been used mainly to solve two variable problems because for 3 and more variables, the number of computations becomes very large.

**Zooming Method.** This method uses a target value for the global minimum of the cost function which is imposed as a constraint in the solution process.<sup>7,22</sup> Once the target is achieved, it is reduced further to zoom-in on the global minimum. The method combines a local minimization method with successive truncation of the feasible set  $S$ . The basic idea is that once a local minimum point has been found, the problem is redefined in such a way that the current solution is eliminated from any further search by adding the constraint  $f(\mathbf{x}) \leq rf(\mathbf{x}^*)$ , where  $f(\mathbf{x}^*)$  is the cost function value at the current minimum point and  $0 < r < 1$  if  $f(\mathbf{x}^*) > 0$ , and  $r > 1$  if  $f(\mathbf{x}^*) < 0$ . The redefined problem is solved again and the process is continued until no more minimum points can be found. The method has a drawback in that as the target level for the global minimum is lowered, the feasible set for the problem shrinks and may even become disjointed. Therefore as the global minimum is approached, finding even a feasible point for the redefined problem becomes time consuming.<sup>21</sup>

**Methods of Generalized Descent.** These methods are generalization of the descent methods where finite descent steps are taken along the search directions (i.e., straight lines). In those methods, it is sometimes difficult to find a suitable step size along the search direction. Therefore, it may be more effective if we

deliberately follow a curvilinear path (trajectory) in the design space. The curvilinear paths are generated by integrating certain first or second order differential equations. The differential equations use the function values and its gradient along the trajectories. The search for the global minimum is based on solution properties of these differential equations. An important property is that their trajectories pass through majority of the stationary points for the cost function. There are conditions that can determine whether or not the trajectory will pass through all the local minimum points. In that case, the global minimum is guaranteed to be found. The methods have been used for problems with only a few variables.

**Tunneling Method.** The basic idea of the tunneling method is to execute the following two phases iteratively until some stopping criterion is satisfied: the local minimization phase and the tunneling phase. The method was initially developed for unconstrained problems and then extended for constrained problems.<sup>48</sup> A local minimum  $\mathbf{x}^*$  for the problem is calculated in phase one. The tunneling phase determines a new starting point for phase one that is different from  $\mathbf{x}^*$  but has cost function value smaller than or equal to the known minimum value. The tunneling phase is accomplished by finding a root of the nonlinear *tunneling function*,  $T(\mathbf{x})$ . This function is defined in such a way that it avoids previously determined local minima and the starting points. The two phases are repeated until no suitable roots of the tunneling function can be found. This is realized numerically when  $T(\mathbf{x}) \geq 0$  for all  $\mathbf{x}$ . This problem is difficult to solve efficiently because finding a suitable point in the tunneling phase is in itself a global optimization problem.

#### 5.4.2. Stochastic Methods

Most stochastic methods depend on random processes to search for the global minimum point. Some methods are useful for only continuous variable problems while others can be used for all types of problems. These methods are some variation of the pure random search. They try to reduce its computational burden. Pure random search evaluates  $f(\mathbf{x})$  at  $N$  sample points drawn from a random uniform distribution over the feasible set. The smallest function value found is the candidate global minimum for  $f(\mathbf{x})$ . The sample size  $N$  must be quite large in order to get a good estimate of the global solution. Therefore the method is quite inefficient due to the large number of function evaluations. *Single start method* is a simple extension of the method in which a single local search is performed starting from the best point found in the random search.

The stochastic ideas are used in two ways in these methods: (i) to decide stopping criteria for the methods, and (ii) to develop techniques to approximate the *region of attraction* for a local minimum point. The goal of many stochastic methods is to develop good approximations for the regions of attraction for local minima so that the search for that local minimum is performed only once.

Some stochastic methods try to determine all local minima for the function. Then, the best local minimum is claimed as the global minimum point. One difficulty is that the number of local minima for the problem is not known a priori. Therefore it is difficult to determine when to end the search for local minima. Usually a statistical estimate for the number of local minima is used in practice. The methods usually have two phases: a global phase and a local phase. In the *global phase*, the function is evaluated at a number of randomly sampled points. In the *local phase*, local searches are performed from the sample points to yield candidate global minima. The global phase is necessary because just a local strategy cannot give a global minimum. There are many stochastic methods for global optimization, such as multistart, clustering, controlled random search, simulated annealing, acceptance-rejection, stochastic integration, and genetic algorithms. We shall describe only the basic ideas of some of the methods. More details can be found in Refs. 7 and 17 and works cited therein. It is important to note that since some stochastic methods use random processes, an algorithm run at different times can generate different iteration histories and local minima. Therefore, a particular problem needs to be run several times before the solution is accepted as the global optimum.

**Multistart Method.** The basic idea of multistart methods is to perform search for a local minimum from each sample point. The best local minimum point found is taken as the global minimum. The stopping criterion for the method is based on a statistical estimate of the number of local minima for the problem. The method is reliable but it is not efficient since many sample points lead to the same local minimum. Therefore, strategies to eliminate this inefficiency in the algorithm have been developed.

**Clustering Methods.** The basic idea of clustering methods is to remove inefficiency of the multistart method by trying to use the local search procedure only once for each local minimum point.<sup>51</sup> The random sample points are linked into groups to form clusters. Each cluster is considered to represent one region of attraction such that a search initiated from any point in the region converges to the same local minimum point. Four clustering methods have been used for

development of the regions of attraction: density clustering, single linkage, mode analysis, and vector quantization multistart.

**Controlled Random Search.** The controlled random search has both global and local phases in its algorithm. It uses the idea of a *simplex* which is a geometric figure formed by a set of  $n+1$  points in the  $n$ -dimensional space ( $n$  is the number of variables). In two dimensions, the simplex is just a triangle and in three dimensions, it is a tetrahedron. The method does not use gradients of the cost function and so continuity of the functions is not required. In the global phase, one starts with  $n+1$  sample points. The worst point (having the largest value for the cost function) is replaced by a trial point evaluated using the centroid for the  $n$  sample points including the worst point. If the trial point is feasible and has better cost function value, then it replaces the worst point of the selected set. Otherwise, the process is repeated until a better point is found. In the local phase, the worst point among the current  $n+1$  sample points is reflected about the centroid of the simplex. The point is then expanded or contracted to obtain a better point. The worst point is replaced by this point. The two phases are repeated until a stopping criterion is satisfied.

**Acceptance-Rejection Methods.** The acceptance-rejection methods use ideas from statistical mechanics to improve efficiency of the multistart algorithm.<sup>49</sup> The strategy is to start the local minimization procedure only when the randomly generated point has smaller cost function value than that of the local minimum previously obtained. This forces the algorithm to tunnel below the local minima in search for a global minimum. This modification, however, has been shown to be inefficient, and therefore the tunneling process has been pursued only by means of deterministic algorithms, as explained earlier. The acceptance-rejection based methods modify this tunneling procedure which is sometimes called *random tunneling*. The idea of acceptance phase is to some times start local minimization from a randomly generated point even if it has a higher cost function value than that at a previously obtained local minimum. This involves calculation of certain probabilities. If the local minimization procedure started from an accepted point produces a local minimum that has higher cost function value than a previously obtained minimum, then the new minimum point is rejected. This is called the rejection phase.

**Stochastic Integration.** In these methods, a stochastic perturbation of the system of differential equations for the trajectory methods is introduced in order to force the trajectory to a global minimum point. This is achieved by monitoring the cost

function value along the trajectories. By changing some coefficients in the differential equations we get different solution processes starting from the same initial point. This idea is similar to simulated annealing but here a parameter in the differential equation is decreased continuously.

## 6. Concluding Remarks

Basic concepts and terminology used for optimization of structural and mechanical systems are described. Various types of optimization models are presented and discussed. Optimality conditions for continuous variable optimization problems are presented. Concept related to algorithms for continuous variable optimization problems are presented and discussed. Basic concepts of methods for discrete variable, multiobjective and global optimization problems are described. The material is introductory in nature, and so, several references are cited for readers interested in more in-depth study of various topics.

## References

1. A.V. Fiacco and G.P. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques* (Society for Industrial and Applied Mathematics, Philadelphia, 1968).
2. P.E. Gill., W. Murray and M.H. Wright, *Practical Optimization* (Academic Press, New York, 1981).
3. D.G. Luenberger, *Linear and Nonlinear Programming* (Addison-Wesley, MA, 1984).
4. B.N. Pshenichny and Y.M. Danilin, *Numerical Methods in Extremal Problems* (Mir Publishers, Moscow, 1982).
5. A.D. Belegundu and T.R. Chandrupatla, *Optimization Concepts and Applications in Engineering* (Prentice Hall, New Jersey, 1999).
6. E.J. Haug and J.S. Arora, *Applied Optimal Design* (John Wiley, New York, 1979).
7. J.S. Arora, *Introduction to Optimum Design*, (Elsevier Academic Press, Boston, 2004).
8. J.S. Arora, in *Advances in Structural Optimization*, Ed. J. Herskovits (Kluwer Academic Publishers, Boston, 1995), p. 47.
9. J.S. Arora, Ed, *Guide to Structural Optimization* (American Society of Civil Engineering, Reston 1997).
10. J.S. Arora, in *Structural Dynamic Systems: Computational Techniques and Optimization*, Ed. C.T. Leondes (Gordon and Breech Publishers, Newark, 1999), p. 1.
11. J.S. Arora, in *Proceedings of the 14<sup>th</sup> Analysis and Computation Conference* (American Society of Civil Engineers, Reston, 2000).
12. J.S. Arora, in *Recent Advances in Optimal Structural Design*, Ed. S. Burns (American Society of Civil Engineers, Reston, 2002), p. 1.
13. J.S. Arora and E.J. Haug, *AIAA J.*, 17 (1979).

14. J.S. Arora and M.W. Huang, *Struc. Mech. Earth. Eng.*, 113 (1996).
15. J.S. Arora, A.I. Chahande, and J.K. Paeng, *Num. Meth. in Eng.*, 32 (1991).
16. J.S. Arora, M.W. Huang and C.C. Hsieh, *Str. Mult. Opt.*, 8 (1994).
17. J.S. Arora, O.A. Elwakeil, A.I. Chahande and C.C. Hsieh, *Str. Mult. Opt.*, 9 (1995).
18. A.D. Belegundu and J.S. Arora, *Num. Meth. in Eng.*, 20 (1984).
19. A.D. Belegundu and J.S. Arora, *AIAA J.*, 22 (1984).
20. A.D. Belegundu and J.S. Arora, *Num. Meth. in Eng.*, 21 (1985).
21. O.A. Elwakeil and J.S. Arora, *AIAA J.*, 33 (1995).
22. O.A. Elwakeil and J.S. Arora, *Num. Meth. in Eng.*, 39 (1996).
23. C.C. Hsieh and J.S. Arora, *Comp. Meth. Appl. Mech. Eng.*, 43 (1984).
24. M.W. Huang and J.S. Arora, *Num. Meth. in Eng.*, 39 (1996).
25. M.W. Huang and J.S. Arora, *Num. Meth. in Eng.*, 40 (1997).
26. M.W. Huang and J.S. Arora, *Str. Mult. Opt.*, 14 (1997).
27. M.W. Huang, C.C. Hsieh and J.S. Arora, *Num. Meth. in Eng.*, 40 (1997).
28. F.Y. Kocer and J.S. Arora, *J. of Str. Eng.*, 122 (1996), p. 804.
29. F.Y. Kocer and J.S. Arora, *J. of Str. Eng.*, 122 (1996), p. 1347.
30. F.Y. Kocer and J.S. Arora, *J. of Str. Eng.*, 123 (1997).
31. O.K. Lim and J.S. Arora, *Comp. Meth. Appl. Mech. Eng.*, 57 (1986).
32. T.R. Marler and J.S. Arora, *Str. Mult. Opt.*, 26 (2004).
33. P.B. Thanedar, J.S. Arora, C.H. Tseng, O.K. Lim and G.J. Park, *Num. Meth. in Eng.*, 23 (1987).
34. C.H. Tseng and J.S. Arora, *Num. Meth. in Eng.*, 26 (1988).
35. C.A. Coello-Coello, D.A. Van Veldhuizen and G.B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems* (Kluwer Academic Publishers, Boston, 2002).
36. K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms* (John Wiley, New York, 2001).
37. A. Osyczka, *Evolutionary Algorithms for Single and Multicriteria Design Optimization* (Physica-Verlag, Germany, 2002).
38. E.K.P. Chong and S.H. Żak, *An Introduction to Optimization* (John Wiley, New York, 2001).
39. C.H. Tseng, L.W. Wang and S.F. Ling, *J. of Str. Eng.*, 121 (1995).
40. J.H. Holland, *Adaptation in Natural and Artificial System* (University of Michigan Press, Ann Arbor, 1975).
41. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, Reading, 1989).
42. M. Mitchell, *An Introduction to Genetic Algorithms* (MIT Press, Cambridge, 1996).
43. S. Pezeshk, and C.V. Camp, in *Recent Advances in Optimal Structural Design*, Ed. S. Burns (American Society of Civil Engineers, Reston, 2002).
44. M. Ehrgott and X. Gandibleux, Eds., *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys* (Kluwer Academic Publishers, Boston, 2002).
45. A. Schrijver, *Theory of Linear and Integer Programming* (John Wiley, New York, 1986).
46. Y.G. Evtushenko, *Numerical Optimization Techniques* (Optimization Software, New York, 1985).
47. C.A. Floudas, P.M. Pardalos, et al, *Handbook of Test Problems in Local and Global Optimization* (Kluwer Academic Publishers, Boston, 1999).

48. A.V. Levy and S. Gomez, in *Numerical Optimization 1984*, Eds. P.T. Boggs, R.H Byrd and R.B. Schnabel (Society for Industrial and Applied Mathematics, Philadelphia, 1985).
49. S. Lucidi and M. Piccioni, *Opt. Th. Appl.*, 62 (1989).
50. P.M. Pardalos, A. Migdalas and R. Burkard, *Combinatorial and Global Optimization* (World Scientific Publishing, New Jersey, 2002).
51. A.H.G. Rinnooy Kan and G.T. Timmer, *Math. Prog.*, 39 (1987).
52. A. Törn and A. Žilinskas, *Global Optimization* (Springer-Verlag, Germany, 1989).