Chapter 2
Optimization Models

To model and solve a bi-level or multi-level optimization problem, we have to first understand basic single-level optimization models and related solution methods. This chapter introduces related concepts, models and solution methods of basic single-level optimization including linear programming, non-linear programming, multi-objective programming, goal programming, Stackelberg game theory, and particle swarm optimization. These knowledge will be used in the rest of the book.

This chapter is organized as follows. Section 2.1 introduces basic single-level optimization concepts and models. Section 2.2 presents the solution method of linear programming. Section 2.3 addresses non-linear programming by its definition, classification, theories, and solution methods. Section 2.4 gives the models and solution methods of multi-objective programming. Section 2.5 introduces goal programming and its solution process. In Sect. 2.6, we present the principles, theorems and applications of Stackelberg game theory. Particle swarm optimization, which will be used as a solution method for non-linear optimization problem, is then introduced in Sect. 2.7. Section 2.8 presents a summary.

2.1 Concepts

The core of the decision process is to formulate an identified decision problem and then find an optimal solution. Many decision models have been developed and different types of decision models require different kinds of decision-making methods to obtain solutions. Popular decision models include (1) *Analytic Hierarchy Process* (AHP), which allows consideration of both the qualitative and quantitative aspects of a decision problem and reduces a complex decision to a series of pairwise comparisons; (2) Grid Analysis, also known as decision matrix analysis or multi-attribute utility theory, in which the decision matrices are the most effective, and multiple alternatives and criteria will be taken into account in the decision process; (3) Decision Tree, which is a graph of decisions and their possible consequences, is used to create a plan to reach a goal; and (4) Optimization model which is a more sophisticated approach to solve decision problems and is the main focus of this book.

G. Zhang et al., *Multi-Level Decision Making*,
Intelligent Systems Reference Library 82, DOI 10.1007/978-3-662-46059-7_2

Optimization, also called mathematical programming, refers to the study of decision problems in which one seeks to minimize (min) or maximize (max) a function by systematically choosing the values of variables within their allowed sets. Many real-world decision problems can be modeled by an optimization framework. To model a decision problem as an optimization model, we need, in principle, three sets of basic variables: *decision variables, result variables* and *uncontrollable variables* (*or parameters*).

Decision Variables describe alternative courses of action and are determined by related decision makers. For example, for a product planning problem, the number of products to be produced is a decision variable.

Result Variables are outputs and are often described by objective functions, such as profit (max) and cost (min). The outputs are determined by decision makers, the factors that cannot be controlled by decision makers, and the relationships among the variables.

Uncontrollable Variables (or Parameters) are the factors that affect the result variables but are not under the control of decision makers. These factors can be fixed, in which case they are called parameters, or they can vary. These factors are uncontrollable because they are determined by elements of the system environment. Some of these variables limit decision makers and therefore form what are called the constraints of the problem. For example, each product's cost of production should be less than the total profit, and each product should meet marketing requirements and so on in a product planning problem.

There are many types of optimization models such as linear programming, non-linear programming, multi-objective programming, and bi-level programming.

Linear Programming (LP) is an important type of optimization in which the objective function and constraints are all linear. Linear programming problems include specialized algorithms for their solutions and for other types of optimization problems by solving linear programming problems as sub-problems. Linear programming is heavily used in various management activities, either to maximize the profit or minimize the cost. It is also the key technique of other optimization problems.

Now, we re-consider Example 1.1 discussed in Chap. 1 to explain how to build a model for an LP practical decision problem.

Example 2.1 A company produces two kinds of products: *A* and *B*. We know that the profit of one unit of *A* and B is $40 and $70, respectively. However, the company has limitations in its labor (a total of 501 labor hours available per time slot; each *A* needs 4 h and *B* 3 h), machine (a total of 401 machine hours available, each *A* needs 2 h and *B* 5 h), and marketing requirements (the need to produce 10 units of *A* and 20 units of *B* respectively). The decision problem is how many *A* and *B* should be produced to obtain the maximum profit. Using these settings and requirements, we can establish a linear programming model:

Decision variables:

$$x_1 = \text{units of } A_1 \text{ to be produced;}$$
$$x_2 = \text{units of } A_2 \text{ to be produced.}$$

Result variable (objective function):
Maximize total profit: $40x_1 + 70x_2$,
Labor constraint (hours): $4x_1 + 3x_2 \leq 501$,
Machine constraint (hours): $2x_1 + 5x_2 \leq 401$,
Marketing requirement for x_1 (units): $x_1 \geq 10$,
Marketing requirement for x_2 (units): $x_2 \geq 20$.

This is a linear programming problem and can be modeled by linear programming (see Sect. 2.2).

Non-linear Programming (NLP) is the process of solving a programming problem subject to certain constraints, over a set of unknown real variables, along with an objective function to be maximized or minimized, as with linear programming, but where some of the constraints or the objective function are non-linear. For example,

$$\begin{aligned} \min_{x_1, x_2} \quad & 40x_1^2 + 70x_2^3 \\ \text{s.t.} \quad & x_1^2 + 20x_2 \leq 100, \\ & 2x_1 + 3\sqrt{x_2} \leq 140, \\ & x_1 \geq 10, x_2 \geq 20. \end{aligned}$$

Multi-objective Programming (MOP) is the process of simultaneously optimizing two or more conflicting objectives subject to certain constraints. MOP problems can be found in a variety of fields, such as product and process design, aircraft design, automobile design, or wherever optimal decisions need to be made in the presence of trade-offs between two or more conflicting objectives. Maximizing profit and minimizing the cost of a product; maximizing performance and minimizing the fuel consumption of a vehicle; and minimizing weight while maximizing the strength of a particular component are all examples of multi-objective optimization problems.

In general, a multi-objective programming problem should not have a single solution that simultaneously minimizes or maximizes each objective to its fullest. In each case an objective must have reached a point such that, when attempting to optimize the objective further, other objectives suffer as a result. Finding such a solution, and quantifying how much better this solution is compared to other solutions, is the goal when setting up and solving a multi-objective optimization problem. For example,

$$\begin{aligned} \min_{x_1,x_2} \quad & \begin{pmatrix} 40x_1 + 70x_2 \\ 50x_1 + 60x_2 \end{pmatrix} \\ \text{s.t.} \quad & 10x_1 + 20x_2 \leq 109, \\ & 20x_1 + 30x_2 \leq 419. \end{aligned}$$

Bi-level programming (BLP) and *multi-level programming* (MLP) are complex optimization situations where one optimization problem is embedded in another one. A bi-level programming problem is a multi-level programming problem having two levels. Below is an example of bi-level programming. More detail will be presented in Chap. 3.

$$\begin{aligned} \min_{x_1} \quad & 40x_1 + 70x_2 \\ \text{s.t.} \quad & 10x_1 + 20x_2 \leq 119, \\ & 20x_1 + 30x_2 \leq 409, \\ & \min_{x_2} \quad 50x_1 + 60x_2 \\ & \text{s.t.} \quad 10x_1 + 8x_2 \leq 109, \\ & \qquad x_1 \geq 10, x_2 \geq 2. \end{aligned}$$

We can see that optimization is an ideal model for decision making. The single limitation is that it works only if the problem is structured and, for the most part, deterministic. An optimization model defines the required input data, the desired output, and the mathematical relationships in a precise manner.

2.2 Linear Programming

Linear programming is a mathematical approach to determining a means to achieve the best outcome (such as maximum profit or minimum cost) in a given mathematical model. This model is defined by an objective function and one or more constraints which have linear formats. A typical example would be taking the limitations of materials and labor described by linear inequalities, and then determining the "best" production levels for the maximal profit defined by a linear formula, under those limitations.

LP problem can be written as:

$$\begin{aligned} \max_{x} \quad & f(x) = cx \\ \text{s.t.} \quad & A_x \leq b, \end{aligned} \tag{2.1}$$

where x represents the vector of decision variables, c and b are vectors of known coefficients, and A is a known matrix of coefficients. The expression $f(x)$ to be

maximized (in other cases, it may be minimized) is called the objective function. The equations $Ax \leq b$ are the constraints which specify a convex polytope over which the objective function is to be optimized. Both $f(x)$ and Ax have linear formats.

Linear programming has a tremendous number of application fields. It has been used extensively in business and engineering, in the areas of transportation, energy, telecommunications, and manufacturing. It has been proved to be useful in modeling diverse types of problems in planning, routing, scheduling, assignment, and design.

Just as with standard maximization problems, the method most frequently used to solve LP problems is the simplex method (Charnes and Cooper 1957). This method provides us with a systematic way of examining the vertices of the feasible region to determine the optimal value of the objective function. As is well-known, the simplex method has proven remarkably efficient in practice.

2.3 Non-linear Programming

Non-linear programming is the process of solving a problem of equalities and inequalities, collectively termed constraints, over a set of unknown real variables, along with an objective function to be maximized or minimized, where some of the constraints or the objective function are non-linear. Formally, an NLP problem can be written as:

$$\min_{x} \quad f(x) \tag{2.2a}$$

$$\text{s.t.} \quad h(x) = 0, \tag{2.2b}$$

$$g(x) \geq 0, \tag{2.2c}$$

where $x \in R^n$, $f : R^n \rightarrow R$, $h : R^n \rightarrow R^p$, $g : R^n \rightarrow R^q$. A point x that satisfies the constraints given by (2.2b) and (2.2c) is called a feasible solution to problem (2.2a)–(2.2c). A collection of all such feasible solutions forms the feasible region. NLP is then use to search a feasible solution $\bar{x}$ such that $f(\bar{x}) \leq f(x)$ for any feasible solution x. $\bar{x}$ is called an optimal solution to the problem (2.2a–2.2c). In special cases when the objective function of (2.2a) and constraints (2.2b) and (2.2c) all have linear forms, the problem (2.2a–2.2c) reduces to a linear programming problem (2.1).

2.3.1 Varieties of Non-linear Programming

Based on the mathematical characteristics of the objective function (2.2a) and the constraints (2.2b) and (2.2c), NLP can be in many different formats. For an objective function or a constraint, the format can be linear, sum of squares of linear

functions, quadratic functions, sum of squares of non-linear functions, sparse non-linear functions, or non-smooth non-linear functions.

Based on combinations of the above formats of the objective and constraints, an NLP problem can be a specific type (such as linear objective function, but the constraint is a quadratic function) and thus have particular properties.

2.3.2 Theories and Optimality Conditions of Non-linear Programming

In this section, we introduce the most important and widely used theories and optimality conditions of NLP. We first denote the feasible region of problem (2.2a–2.2c) by S. The following definitions and results can be found in Bazaraa et al. (2013).

Definition 2.1 A point $x^* \in S$ is called a *relative* or *local minimum* of $f(x)$ over S if there is an $\varepsilon > 0$ such that $f(x) \leq f(x^*)$ for all $x \in S$ within a distance ε of x^*. If $f(x) > f(x^*)$ for all $x \in S$, $x \neq x^*$ within a distance ε of x^*, then x^* is called a *strict relative minimum* of $f(x)$ over S.

Definition 2.2 A point $x^* \in X$ is called a *global minimum* of $f(x)$ over S if $f(x) \geq f(x^*)$ for all $x \in S$. If $f(x) > f(x^*)$ for all $x \in S$, $x \neq x^*$, then x^* is called a *strict global minimum* of $f(x)$ over S.

For situations where constraints are absent, the following two theorems hold.

Theorem 2.1 *Let $f : R^n \to R$ be twice continuously differentiable throughout a neighborhood of x^*. If f has a relative minimum at x^*, then it necessarily follows that*

1. *The gradient vector $\nabla f(x^*) = 0$.*
2. *$F(x^*)$ is positive semi-definite, where $F(x^*)$ is the Hessian matrix of $f(x)$ at x^*.*

Theorem 2.2 *Let $f : R^n \to R$ be twice continuously differentiable throughout a neighborhood of x^*. Then a sufficient condition for $f(x)$ to have a strict relative minimum at x^*, where $\nabla f(x^*) = 0$ holds, is that $F(x^*)$ is positive definite.*

For NLP problems involving only equality constraints, the following definition and theories hold.

Definition 2.3 A point x^* satisfying the constraints $h(x^*) = 0$ is called a *regular point* of the constraints if the gradient vectors $\nabla h_1(x^*), \ldots, \nabla h_m(x^*)$ are linearly independent.

Theorem 2.3 *At a regular point x^* of the surfaces $S = \{x|h(x) = 0\}$, the tangent plane is equal to $T = \{y|\nabla h(x)y = 0\}$.*

Theorem 2.4 *Suppose that x^* is a local minimum of $f(x)$ subject to $h(x) = 0$ as well as a regular point of these constraints. There then exists a vector $\lambda \in R^m$ such that $\nabla f(x^*) - \lambda \nabla h(x^*) = 0$.*

The following definitions and theories are used for the general NLP problem (2.2a–2.2c).

Definition 2.4 Let x^* be a point satisfying the constraints $h(x^*) = 0$ and $g(x^*) \geq 0$; and let J be the set of indices j such that $g_j(x^*) = 0$. Then x^* is called a *regular point* of these constraints if the gradient vectors $\nabla h_i(x^*)(1 \leq i \leq m)$, $\nabla g_j(x^*)(j \in J)$ are linear independent.

Theorem 2.5 (Kuhn-Tucker Conditions) *Let x^* be a relative minimum for the problem (2.2a–2.2c) and suppose that x^* is a regular point for the constraints. Then there exists a vector $\lambda \in R^m$ and a vector $\mu \in R^q$ such that*

$$\nabla f(x^*) - \lambda \nabla h(x^*) - \mu \nabla g(x^*) = 0, \tag{2.3a}$$

$$\mu g(x^*) = 0, \tag{2.3b}$$

$$\mu \geq 0, \tag{2.3c}$$

$$h(x^*) = 0, g(x^*) \geq 0. \tag{2.3d}$$

2.3.3 Methods for Solving Non-linear Programming Problems

For an NLP problem in which the objective function and constraints have linear forms, the problem becomes an LP problem which can be solved using the well-known simple algorithm.

If the objective function of an NLP problem is convex (for the minimization problem), or concave (for the maximization problem), and the constraint set is convex, then the programming problem is called a *convex programming* problem and general methods from convex optimization can be used.

Several methods are available for solving non-convex problems. One method is to use special formulations of LP problems. Another involves the use of the branch-and-bound technique, where the programming problem is divided into subclasses to be solved with convex (minimization problem) or linear approximations that form a lower bound on the overall cost within the subdivision. With subsequent divisions, an actual solution will be obtained at some point whose cost is equal to the best lower bound obtained for any of the approximate solutions. This solution is optimal, although possibly not unique. The method may also be terminated early, with the assurance that the best feasible solution is within a tolerance of the best point

found; such points are called ε-optimal solution. Terminating to ε-optimal solution is typically necessary to ensure finite termination. This is especially useful for large, difficult problems, and problems with uncertain costs or values where the uncertainty can be estimated with appropriate reliability estimation.

Under differentiability and constraint qualifications, the Kuhn–Tucker conditions provide the necessary conditions for a solution to be optimal. Under convexity, these conditions are also sufficient.

The most popular methods for NLP problems include Zoutendijk's feasible direction method, the gradient projection method, the penalty method, and the Lagrangian method (Bazaraa et al. 2013).

The above-mentioned methods depend on certain mathematical properties of the NLP problems to be solved. Sometimes, these properties are difficult to satisfy. In such situations, these methods become invalid. Heuristics-based methods such as *Genetic Algorithms* (Tang et al. 2011), *Particle Swarm Optimization* (Nezhad et al. 2013), on the other hand, do not have this limitation and are thus another direction for NLP problems.

2.4 Multi-objective Programming

The main characteristics of *Multi-objective Programming* (MOP) are that decision makers need to achieve multiple objectives simultaneously while these multiple objectives are non-commensurable and conflict with each other.

2.4.1 Multi-objective Programming Model

An MOP model considers a vector of variables, objective functions, and constraints. It attempts to maximize (or minimize) the objective functions. Since this problem rarely has a unique solution, we expect to choose a solution from among the set of feasible solutions, which will be explained later in this section. Generally, a MOP problem can be formulated as follows:

$$\begin{aligned} &\max_{x} \quad f(x) \\ &\text{s.t.} \quad x \in X = \{x | g(x) \leq 0\} \end{aligned} \tag{2.4}$$

where $f(x)$ represents k conflicting objective functions, $g(x) \leq 0$ represents m constraints, and $x \in R^n$ is a n-dimensional vector of decision variables.

Multi-objective linear programming (MOLP) is one of the most important forms of MOP problems, which are specified by linear objective functions subject to a set of linear constraints. The standard form of a MOLP problem can be written as follows:

$$\begin{aligned} &\max_{x} \quad Cx \\ &\text{s.t.} \quad x \in X = \{x | Ax \leq b\} \end{aligned} \tag{2.5}$$

where C is a $k \times n$ objective function matrix, A is an $m \times n$ constraint matrix, b is a m-dimensional vector, and x is a n-dimensional vector of decision variable.

We have the following notion for a complete optimal solution.

Definition 2.5 (Sakawa 1993) x^* is said to be a complete optimal solution, if and only if there exists a $x^* \in X$ such that $f_i(x^*) \geq f_i(x)(i = 1, \ldots, k))$ for all $x \in X$.

Also, ideal solution, superior solution, or utopia point are equivalent terms indicating a complete optimal solution (Lu et al. 2007).

In general, a complete optimal solution that simultaneously maximizes (or minimizes) all objective functions does not always exist when the objective functions conflict with each other. Thus, a concept of Pareto optimal solution is introduced into MOLP.

Definition 2.6 (Sakawa 1993) x^* is said to be a Pareto optimal solution, if and only if there does not exist another $x \in X$ such that $f_i(x) \geq f_i(x^*)$ for all i and $f_i(x) \neq f_i(x^*)$ for at least one i.

The Pareto optimal solution is also called a non-dominated solution, non-inferior solution, efficient solution, and non-dominate solution.

In addition to the Pareto optimal solution, the following weak Pareto optimal solution is defined as a slightly weaker solution concept than the Pareto optimal solution.

Definition 2.7 (Sakawa 1993) x^* is said to be a weak Pareto optimal solution, if and only if there does not exist another $x \in X$ such that $f_i(x) > f_i(x^*), i = 1, \ldots, k$.

Here, let X^{CO}, X^P and X^{WP} denote complete optimal, Pareto optimal, and weak Pareto optimal solution sets, respectively. Then from above definitions, we can easily obtain the following relations:

$$X^{CO} \subseteq X^P \subseteq X^{WP}. \tag{2.6}$$

A satisfactory solution belongs to a reduced subset of the feasible set that exceeds all of the aspiration levels of each objective. A set of satisfactory solutions is composed of acceptable alternatives. Satisfactory solutions do not need to be non-dominated, and a preferred solution is a non-dominated solution selected as the final choice through decision makers' involvement in the information processing stage.

The rest of this chapter focuses mainly on MOLP, the linear form of MOP.

2.4.2 Multi-objective Linear Programming Methods

The methods for solving MOLP problems have been well developed and classified into four classes by Hwang and Masud (1979) and Lai and Hwang (1994). We list them in Table 2.1.

As shown in Table 2.1, the first class of MOLP methods basically does not require any more information nor interaction with decision makers once the objective functions and constraints have been defined. The solution to a MOLP problem is presented on the basis of assumptions made about decision makers' preferences.

The second class of MOLP methods assumes that decision makers have a set of goals to achieve and that these goals will be established before formulation of a mathematical programming model. The multi-objective goal programming (MOGP) assumes that decision makers can specify goals for the objective functions. The key idea behind goal programming is to minimize deviation from the goals or aspiration levels set by decision makers. In most cases, therefore, MOGP seems to yield a satisfactory solution rather than an optimal one. More details about MOGP problem will be discussed later.

Table 2.1 A classification of MOLP methods

	Stage at which information is needed	Type of information	Typical methods
1	No articulation of preference information		• Global criteria method (Hwang and Masud 1979, Salukvadze 1974)
2	A priori articulation of preference information	Cardinal	• Weighting method (Hwang and Masud 1979, Sakawa 1993)
		Ordinal and cardinal	• (Multi-objective) goal programming (GP) (Ignizio 1976)
3	Progressive articulation of preference information (interactive method)	Explicit trade-off	• Efficient solution via goal programming (ESGP) (Ignizio 1981)
			• Interactive multiple objective linear programming (IMOLP) (Quaddus and Holzman 1986)
			• Interactive sequential goal programming (ISGP) (Hwang and Masud 1979)
			• ZW method (Zionts and Wallenius 1983)
		Implicit trade-off	• STEP method (STEM) (Benayoun et al. 1971)
			• STEUER (Steuer 1977)
4	A posteriori articulation of preference information (non-dominated solutions generation method)	Implicit/ explicit trade-off	• Parametric method (Hwang and Masud 1979)
			• Constraint method (Hwang and Masud 1979; Sakawa 1993)

The third class of MOLP, interactive methods, requires more involvement and interaction with decision makers in the solving process. The interaction takes place through decision makers' computer interface at each iteration. Trade-off or preference information from decision makers at each iteration is used to determine a new solution, therefore decision makers actually gain insights into the problem. Interactive programming was first initiated by Geoffrion et al. (1972) and further developed by many researchers. The STEP method (Benayoun et al. 1971) in particular is known to be one of the first interactive MOLP techniques, to which there have been a number of modifications and extensions. The interactive MOGP method was also proposed (Dyer 1972), which attempts to provide a link between MOGP and interactive methods.

Lastly, the purpose of the fourth class is to determine a subset of the complete set of non-dominated solutions to a MOLP problem. It deals strictly with constraints and does not consider the decision makers' preferences. The desired outcome is to narrow the possible courses of actions and select the preferred course of action more easily.

Interaction is one of the most important features for solving MOLP problems. There are three types of interaction in the MOLP problem solving process: pre-interaction (before the solution process), pro-interaction (during the solution process), and post-interaction (after the solution process). The seven MOLP methods selected from Table 2.1, ESGP, IMOLP, ISGP, MOGP, STEM, STEUER, and ZW, have differences in the interaction processes with decision makers. The MOGP, IMOLP and ISGP methods involve pre-interaction with users prior to the solution process through the collection of weights, goals, and priorities of objectives from users. The STEM method engages in pro-interaction during the solution process. Its principle is to require decision makers to nominate the amounts to be sacrificed of satisfactory objectives until all objectives become satisfactory. It first displays a solution and the ideal value of each objective. It then asks decision makers to accept or reject this solution. If it is accepted, the solution is taken as the final satisfactory solution. However, decision makers often make further searches so that more alternative solutions can be generated. If the current solution is rejected, a relaxation process starts. Decision makers will accept a certain level of relaxation of a satisfactory objective to allow the improvement of unsatisfactory objectives. When the relaxation fails, the system enables decision makers to continue re-entering a set of relaxation values and a new solution is then found. If decision makers accept this solution, it becomes the final satisfactory solution. Otherwise the system repeats the above process. Post-interaction is used in all seven methods. After a set of candidate solutions has been generated, decision makers are required to choose the most satisfactory solution.

Now, we give details of the weighting method for solving MOLP problems.

The key idea of the weighting method is to transform the multiple objective functions in the MOLP problem (2.5) into a weighted single objective function, which is described as follows:

$$\begin{aligned} &\max_{x} \quad wCx \\ &\text{s.t.} \quad x \in X = \{x | Ax \leq b\} \end{aligned} \tag{2.7}$$

where $w = (w_1, w_2, \ldots, w_k) \geq 0$ is a vector of weighting coefficients assigned to the objective functions.

Example 2.2 Let us consider the following example of a MOLP problem.

$$\begin{aligned} \max_{x_1, x_2} \quad & f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix} = \begin{pmatrix} 2x_1 + x_2 \\ -x_1 + 2x_2 \end{pmatrix} \\ \text{s.t.} \quad & -x_1 + 3x_2 \leq 21, \\ & x_1 + 3x_2 \leq 27, \\ & 4x_1 + 3x_2 \leq 45, \\ & 3x_1 + x_2 \leq 30. \end{aligned} \tag{2.8}$$

Let X denote the feasible region of problem (2.8). When $w_1 = 0.5, w_2 = 0.5$, the weighting problem is formulated as

$$\begin{aligned} &\max \quad wf(x) = 0.5x_1 + 1.5x_2 \\ &\text{s.t.} \quad (x_1, x_2) \in X. \end{aligned}$$

The optimal solution is $(x_1^*, x_2^*) = (3, 8)$, and the optimal objective function value is $f^*(x) = (f_1^*(x), f_2^*(x)) = (14, 13)$.

When $w_1 = 1, w_2 = 0$, the optimal solution is $(x_1^*, x_2^*) = (9, 3)$, and the optimal objective function value is $f^*(x) = (f_1^*(x), f_2^*(x)) = (21, -3)$.

When $w_1 = 0, w_2 = 1$, the optimal solution is $(x_1^*, x_2^*) = (0, 7)$, and the optimal objective function value is $f^*(x) = (f_1^*(x), f_2^*(x)) = (7, 14)$.

2.4.3 A Case-Based Example

Example 2.3 A manufacturing company has six types of milling machine, lathe, grinder, jig saw, drill press, and band saw, whose capacities are to be devoted to producing three products x_1, x_2, and x_3. Decision makers have three objectives: maximizing profit, quality, and worker satisfaction. It is assumed that the parameters and objectives of the MOLP problem are defined precisely in this example. For instance, to produce one unit of x_1 requires 12 h of machine milling, as listed in Table 2.2 (Lai 1995).

Table 2.2 Production planning data

Machine	Product x_1 (unit)	Product x_2 (unit)	Product x_3 (unit)	Machine (available hours)
Milling machine	12	17	0	1,400
Lathe	3	9	8	1,000
Grinder	10	13	15	1,750
Jig saw	6	0	16	1,325
Drill press	0	12	7	900
Band saw	9.5	9.5	4	1,075
Profit	50	100	17.5	
Quality	92	75	50	
Worker satisfaction	25	100	75	

This problem can be described by a MOLP model as follows:

$$\begin{aligned} \max_{x_1,x_2,x_3} \quad & f(x) = \begin{pmatrix} 50x_1 + 100x_2 + 17.5x_3 \\ 92x_1 + 75x_2 + 50x_3 \\ 25x_1 + 100x_2 + 75x_3 \end{pmatrix} \\ \text{s.t.} \quad & 12x_1 + 17x_2 \le 1400, \\ & 3x_1 + 9x_2 + 8x_3 \le 1000, \\ & 10x_1 + 13x_2 + 15x_3 \le 1750, \\ & 6x_1 + 16x_3 \le 1325, \\ & 12x_2 + 7x_3 \le 900, \\ & 9.5x_1 + 9.5x_2 + 4x_3 \le 1075, \\ & x_1, x_2, x_3 \ge 0. \end{aligned} \tag{2.9}$$

We can see that this is a typical multi-objective programming problem.

2.5 Goal Programming

Goal programming (GP), originally proposed by Charnes and Cooper (1957), is a great strategy to deal with multi-objective optimization problems by setting multiple goals, as we mentioned before. In some decision situations, a decision maker may have more than one objective, with the improvement on one objective to be achieved only at the expense of others. For example, a coordinator of a multi-division firm considers three objectives in making an aggregate production plan: to maximize the net profit, to maximize the quality of products, and to maximize worker satisfaction (Example 2.3). The three objectives could be in conflict with

each other, but must be considered simultaneously. Any improvement in one objective may be achieved only at the expense of other objectives.

Goal programming takes a 'satisfactory solution' strategy. It requests a decision maker to set a goal or a target for the objective (a set of goals for a MOLP) that the person wishes to attain. A preferred solution is then defined to minimize the deviation from the goal. Therefore, goal programming would appear to yield a satisfactory solution rather than an optimal one. Now we give a formal description of the method adopted by goal programming.

Suppose that a MOLP problem is defined as follows:

$$\begin{aligned} \max_{x} \quad & f(x) = (\alpha_1 x, \alpha_2 x, \ldots, \alpha_k x) \\ \text{s.t.} \quad & Ax \leq b. \end{aligned} \tag{2.10}$$

For problem (2.10), there are a total of k objectives $(\alpha_1 x, \alpha_2 x, \ldots, \alpha_k x)$ to achieve. We give goals $g_i (i = 1, 2, \ldots, k)$ for the ith objective. Our effort is now focused on making each objective $\alpha_i x$ as close to its goal $g_i (i = 1, 2, \ldots, k)$, as possible. The problem (2.10) is then transformed as follows:

$$\begin{aligned} \min_{x, v_1^-, v_1^+, \ldots, v_k^-, v_k^+} \quad & v_1^- + v_1^+ + \cdots + v_k^- + v_k^+ \\ \text{s.t.} \quad & \alpha_1 x + v_1^- - v_1^+ = g_1, \\ & \alpha_2 x + v_2^- - v_2^+ = g_2, \\ & \vdots \\ & \alpha_k x + v_k^- - v_k^+ = g_k, \\ & v_1^-, v_1^+, \ldots, v_k^-, v_k^+ \geq 0, \\ & Ax \leq b. \end{aligned} \tag{2.11}$$

To give a more clear understanding of the idea adopted, v_i^- and $v_i^+ (i = 1, \ldots, k)$ can be defined as follows:

$$\begin{aligned} v_1^+ &= \frac{1}{2(|\alpha_1 x - g_1| + (\alpha_1 x - g_1))}, \\ v_1^- &= \frac{1}{2(|\alpha_1 x - g_1| - (\alpha_1 x - g_1))}, \\ &\vdots \\ v_k^+ &= \frac{1}{2(|\alpha_k x - g_k| + (\alpha_k x - g_k))}, \\ v_k^- &= \frac{1}{2(|\alpha_k x - g_1| - (\alpha_k x - g_k))}. \end{aligned}$$

In the above formula, v_i^- and v_i^+, are deviation variables representing the under-achievement and over-achievement of the ith goal g_i, for the ith objective $\alpha_i x (i = 1, \ldots, k)$, respectively.

The problem defined by (2.11) is a standard linear programming problem which can be solved by the simplex method.

There are some variants of goal programming. The initial goal programming formulations order the deviations between objectives and goals into a number of priority levels. The minimization of the deviation at a higher priority level are more important than the deviations at lower priority levels. This is called lexicographicor pre-emptive goal programming (Amador and Romero 1989). When clear priority ordering of the goals to be achieved exists, lexicographic goal programming can be used.

Weighted or non pre-emptive goal programming can be used if a decision maker is more interested in making direct comparisons of the objectives. In this situation, all the deviations between objectives and goals are multiplied by weights, which reflect the relative importance of the objectives. We add these weighted deviations together as a single sum to form the objective function. This process is defined by the following formula:

$$\begin{aligned}
\min_{x, v_1^-, v_{1,}^+, \ldots, v_k^-, v_k^+} \quad & v = w_1^- v_1^- + w_1^+ v_1^+ + \cdots + w_k^- v_k^- + w_k^+ v_k^+ \\
\text{s.t.} \quad & \alpha_1 x + v_1^- - v_1^+ = g_1, \\
& \alpha_2 x + v_2^- - v_2^+ = g_2, \\
& \qquad \vdots \\
& \alpha_k x + v_k^- - v_k^+ = g_k, \\
& v_1^-, v_1^+, \ldots, v_k^-, v_k^+ \geq 0, \\
& Ax \leq b,
\end{aligned}$$

where w_i^- and $w_i^+ (i = 1, 2, \ldots, k)$ are non-negative constants representing the relative importance to be assigned to the positive and negative deviations for each of the relevant goals.

Based on goal programming as previously introduced and the MOLP model, MOGP requires that goals are set for each objective, following which a preferred solution is defined as one which minimizes the deviations from those goals.

We assume that the goals $g = (g_1, \ldots, g_k)$ are specified for objective functions $f(x) = (f_1(x), \ldots, f_k(x))$ by decision makers, and a decision variable $x^* \in X$ in the MOLP problem is sought so that the objective functions $f^*(x) = (f_1^*(x), \ldots, f_k^*(x))$ are as close as possible to the goals $g = (g_1, \ldots, g_k)$.

The deviation between $f^*(x) = (f_1^*(x), \ldots, f_k^*(x))$ and $g = (g_1, \ldots, g_k)$ is usually defined as a deviation function $D(f(x), g)$. The MOGP can then be defined as an optimization problem:

$$\begin{aligned} \min_{x \in X} \quad & D(f(x), g) \\ \text{s.t.} \quad & x \in X = \{x \in R^n | Ax \leq b\}, \end{aligned} \tag{2.12}$$

that is, find an $x^* \in X$, which minimizes $D(f(x), g)$ or

$$x^* = \arg\min_{x \in X} D(f(x), g). \tag{2.13}$$

Normally, the deviation function $D(f(x), g)$ is a maximum of deviation of individual goals,

$$D(f(x), g) = \max\{D_1(f_1(x), g_1), \ldots, D_k(f_k(x), g_k)\}. \tag{2.14}$$

From (2.12) and (2.14), the min–max approach is applied to the GP problem:

$$\min_{x \in X} \max\{D_1(f_1(x), g_1), \ldots, D_k(f_k(x), g_k)\}. \tag{2.15}$$

By introducing an auxiliary variable γ, (2.15) can then be transformed into the following linear programming problem:

$$\begin{aligned} \min_{x} \quad & \gamma \\ \text{s.t.} \quad & D_1(f_1(x), g_1) \leq \gamma, \\ & D_2(f_2(x), g_2) \leq \gamma, \\ & \quad \vdots \\ & D_k(f_k(x), g_k) \leq \gamma, \\ & Ax \leq b. \end{aligned} \tag{2.16}$$

Example 2.4 Let us consider the following example of a MOLP problem:

$$\begin{aligned} \max_{x_1, x_2} \quad & f(x) = \begin{pmatrix} 2x_1 + x_2 \\ -x_1 + 2x_2 \end{pmatrix} \\ \text{s.t.} \quad & -x_1 + 3x_2 \leq 21, \\ & x_1 + 3x_2 \leq 27, \\ & 4x_1 + 3x_2 \leq 45, \\ & 3x_1 + x_2 \leq 30, \\ & x_1, x_2 \geq 0. \end{aligned}$$

Suppose the goals are specified as $g = (10, 10)$ for the two objective functions. The original MOLP problem can be converted as the following LP problem with the auxiliary variable γ:

$$\begin{aligned}
\min_{x_1,x_2} \quad & \gamma \\
\text{s.t.} \quad & 2x_1 + x_2 - 10 \le \gamma, \\
& -x_1 + 2x_2 - 10 \le \gamma, \\
& -x_1 + 3x_2 \le 21, \\
& x_1 + 3x_2 \le 27, \\
& 4x_1 + 3x_2 \le 45, \\
& 3x_1 + x_2 \le 30, \\
& x_1, x_2 \ge 0.
\end{aligned}$$

The optimal solution then is $(x_1^*, x_2^*) = (2, 6)$, and the optimal objective function values are $f^*(x) = (f_1^*(x), f_2^*(x)) = (10, 10)$.

When the goals are specified as $g = (15, 15)$, the optimal solution is $(x_1^*, x_2^*) = (1.865, 7.622)$, and the optimal objective function values are $f^*(x) = (f_1^*(x), f_2^*(x)) = (11.351, 13.378)$. We learn from the optimal objective function values that the goals are not achieved. The reason is that the goals specified are beyond the feasible constraint area. The point of $\left(x_1^*, x_2^*\right) = (1.865, 7.622)$ is on the boundary of the feasible constraint area.

Goal programming has the advantages of being simple and easy to use. It can handle relatively large numbers of variables, constraints and objectives, which accounts for the large number of goal programming applications in many diverse fields, such as business management, transportation planning, and resource optimization. A limitation of goal programming is that setting the goals for some of the objectives may not be straight forward. In-depth field knowledge might be required to solve a decision problem, and experiments sometimes need to be carried out to set suitable goals.

2.6 Stackelberg Game Model

The Stackelberg game model, which is also called a leader-follower game, was first proposed by Heinrich von Stackelberg in 1952 (Stackelberg 1952). It is based on economic monopolization phenomena. In a Stackelberg game, one player acts as a leader and the rest as followers. The problem is then to find an optimal strategy for the leader, assuming that the followers react in a rational way which will optimize their objective functions, given the leader's actions.

Stackelberg used a hierarchical model to describe a market situation in which decision makers try to optimize their decisions based on individually different objectives but are affected by a certain hierarchy.

2.6.1 Stackelberg Game and Bi-level Programming

The Stackelberg leadership model considers the case of a single leader and follower. Let X and Y be the strategy sets for the leader and follower respectively. Denote their objective function by $F(x, y)$ and $f(x, y)$ respectively. Knowing the selection x of the leader, the follower can select his best strategy $y(x)$ such that his objective function $f(x, y)$ is maximized, i.e.,

$$y(x) \in \Phi(x) = \arg\max_{y \in Y} f(x, y). \tag{2.17}$$

The leader then obtains the best strategy $x \in X$ as

$$x \in \arg\max_{x \in X} \{F(x, y) | y \in \Phi(x)\}. \tag{2.18}$$

Formulae (2.17) and (2.18) can be combined to express the Stackelberg game as follows:

$$\begin{aligned} \max_{x} \quad & F(x, y) \\ \text{s.t.} \quad & x \in X, \\ & y \in \arg\max_{y \in Y} f(x, y). \end{aligned}$$

Bi-level programming (see Chap. 3) is more general than Stackelberg game in the sense that the strategy sets (also called the admissible sets) depend on both x and y. This leads to a general bi-level programming (Candler and Norton 1977) as follows:

$$\begin{aligned} \max_{x} \quad & F(x, y) \\ \text{s.t.} \quad & G(x, y) \leq 0, \\ & y \in \text{argmax}\{f(x, y) | g(x, y) \leq 0\}. \end{aligned} \tag{2.19}$$

Bi-level programming problem (2.19) is a generalization of several well-known optimization problem (Dempe 2002). For example, if $F(x, y) = -f(x, y)$, then it is a classical min–max problem; if $F(x, y) = f(x, y)$, we have a realization of the decomposition approach to optimization problem; if the dependence of both the leader's and the follower's problem on y is dropped, the problem is reduced to a bi-criteria optimization problem.

2.6.2 Stackelberg Game and Nash Game

The Stackelberg game can be considered as an extension of the well-known Nash game (Nash 1951). In the Nash game, we assume that there are k players, and the ith player has a strategy set X_i, and his objective function is $f_i(x)$ for $i = 1, 2, \ldots, k$, where $x = (x_1, x_2, \ldots, x_k)$. Each player chooses a strategy based on the choices of the other players and there is no hierarchy. The unstructured problem is modeled as follows: for $i = 1, 2, \ldots, k$, we have $\max_{x_i \in X_i} f_i(x)$.

This is a Nash game in which all players aim to maximize their corresponding objective functions.

In contrast, there is a hierarchy between the leader and followers in the Stackelberg game. The leader is aware of the choices of the followers, thus the leader, being in a superior position with regard to everyone else, can achieve the best objective while forcing the followers to respond to this choice of strategy by solving the Stackelberg game. Without loss of generality, we now assume that the first player is the leader, and the rest of the players are followers. Let $X_{1-} = X_2 \times X_3 \times \cdots \times X_k, f_{1-}(x) = (f_2(x), \ldots, f_k(x))$, and $x_{1-} = (x_2, \ldots, x_k) \in X_{1-}$. The above Nash game is accordingly transformed into a Stackelberg game, which is given as follows:

$$\begin{aligned} &\max_{x_1 \in X_1} \quad f_1(x) \\ &\text{s.t.} \quad x_{1-} \in \operatorname{argmax}\{f_{1-}(x) | x_{1-} \in X_{1-}\}. \end{aligned}$$

This is a Stackelberg game or leader-follower game.

2.6.3 Applications of Stackelberg Games

The investigation of Stackelberg games is strongly motivated by real world applications, and Stackelberg games techniques have been applied with remarkable success in many domains, such as transportation network design, production planning and logistics.

Stackelberg games have been applied to the network design problem (Ben-Ayed 1988) arising in transportation systems. In the accompanying formulation, a central planner controls investment costs at the system level, while operational costs depend on traffic flow, which is determined by the individual user's route selection. Because users are assumed to make decisions to maximize their peculiar utility functions, their choices do not necessarily coincide with the choices that are optimal for the system. Nevertheless, the central planner can influence users' choices by improving certain links, making some relatively more attractive than others. In deciding on these improvements, the central planner tries to influence users' preferences in such a way that total costs are minimized. The

partition of the control variables between the upper and lower levels naturally leads to a bi-level formulation.

Moreover, a fuzzy Stackelberg game model was set up to control traffic flow in a disaster area after an earthquake (Feng and Wen 2005). When a severe earthquake occurs, roadway systems usually suffer various degrees of damage, reducing their capacity and causing traffic congestion. Maintaining viable traffic functions to facilitate the saving of more lives is a crucial mission task following an earthquake. The aim of the commander of the government Emergency-Response Centre at county and city level (the upper level) is to allow traffic to pass through disaster areas to the extent that is possible given the roadway's capacity, while road users (at the lower level) always choose the shortest route to affect emergency rescues. To solve this decision problem, the bi-level technique has been used post-earthquake to provide an efficient traffic control strategy for recovery from chaos.

A Stackelberg game has been formulated for a newsboy problem. The decision makers are the manufacturer and retailers. The former acts as a leader who controls the product price, and the retailers as the followers who decide the quantity of newspapers to order. The relationship between the manufacturer and retailers is a sequential non-cooperative game. The manufacturer first decides the product price, and the retailers then decides the quantity. The manufacturer tries to determine product price and maximize his profit after considering the retailers' behavior. The retailers' decision is to optimize the order quantity so as to maximize his profit at a given product price. Clearly, this newsboy problem can be modeled as a Stackelberg game.

In addition, Stackelberg games are frequently utilized in many other real-world cases, such as resource allocation, network investigation, and engineering. These applications have provided stimulating environments for the development of Stackelberg games.

2.7 Particle Swarm Optimization

In the computational intelligence area, *particle swarm optimization* (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. PSO is a heuristic algorithm proposed by Kennedy and Eberhart (1995), Shi and Eberhart (1998).

Inspired by the social behavior of animals, such as fish schooling and bird flocking, PSO is a kind of population-based algorithm. The population of PSO is called a swarm, and each individual in the swarm is called a particle. The similarity between PSO and other evolutionary algorithms lies in the fact that an individual in the community is moved to a good area according to its fitness for the environment. Unlike other evolutionary computation methods, however, each particle in PSO has an adaptable velocity (position change), according to which it moves in the search space (Parsopoulos and Vrahatis 2002). Moreover, each particle has a memory, remembering the best position it has ever visited in the search space (Kennedy and Eberhart 1995). Thus, its movement is an aggregated acceleration towards its best

previously visited position and towards the best particle of a topological neighborhood.

Suppose the current search space for PSO is n-dimensional, then the ith particle of the swarm can be represented by an n-dimensional vector, $x_i = (x_{i1}, \ldots, x_{in})$. The velocity (position change) of this particle can thus be represented by another n-dimensional vector $v_i = (v_{i1}, \ldots, v_{in})$. The best previously visited position of the ith particle is denoted as $p_i = (p_{i1}, \ldots, p_{in})$. Defining g as the index of the best particle in the swarm (i.e., the gth particle is the best), and letting the superscripts denote the iteration number, the swarm is manipulated according to the following two equations (Eberhart et al. 1996):

$$v_{id}^{k+1} = wv_{id}^{k} + cr_1^k\left(p_{id} - x_{id}^{k}\right) + cr_2^k\left(p_{gd}^{k} - x_{id}^{k}\right),$$
$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{K+1},$$

where $d = 1, \ldots, n$ denotes the d-dimensional vector, $i = 1, 2, \ldots, N$ denotes the ith particle, N is the size of the swarm, w is the inertia weight, c is a positive constant, called the acceleration constant, r_1, r_2 are random numbers, uniformly distributed in [0,1], and k determines the iteration number.

Like many other global optimization methods, whether deterministic or evolutionary, PSO suffers from the problem of local optima. The existence of many local optimal solutions makes it difficult for PSO to detect the global optimal solution. In some cases, sub-optimal solutions are acceptable although not desirable, while in others, a global optimal solution is indispensable. The development of robust and efficient methods for avoiding local solutions is the subject of current PSO research.

Stretching technique (Parsopoulos and Vrahatis 2002) has been shown through simulation experiments and it provide an effective way for the PSO method to escape local optimal solution.

The idea behind the function of Stretching is to perform a two-stage transformation of the original objective function $F(x)$. The two-stage transformation can be applied immediately after a local optimization solution $\bar{x}$ of the function $F(x)$ has been detected. This transformation has been proposed by Parsopoulos and Vrahatis (2002) and is defined as follows:

$$G(x) = F(x) + \gamma_1 |x - \bar{x}|(\text{sign}(F(x) - F(\bar{x})) + 1), \tag{2.20}$$

$$H(x) = G(x) + \gamma_2 \frac{\text{sign}(F(x) - F(\bar{x})) + 1}{\tanh(\mu(G(x) - G(\bar{x})))}, \tag{2.21}$$

where γ_1, γ_2 and μ are arbitrary chosen positive constants, and $\text{sign}(x)$ is defined by:

$$\text{sign}(x) = \begin{cases} 1, & \text{if } x < 0; \\ 0, & \text{if } x = 0; \\ -1, & \text{if } x < 0. \end{cases}$$

The first transformation stage, defined in (2.20), elevates the function $F(x)$ and eliminates all the local optimization solutions that are less optimal than the result of $F(\bar{x})$. The second stage, defined by (2.21), stretches the neighborhood of $\bar{x}$ upwards, since it assigns higher function values to those points. Neither stage changes the local optimal solutions which can produce more optimal results than $\bar{x}$. Thus, the location of the global solution can be left unchanged.

Because PSO requires only primitive mathematical operators and is computationally inexpensive in terms of both memory requirements and speed (Parsopoulos and Vrahatis 2002), it has good convergence performance and has been successfully applied in many fields such as neural network training (Zhang et al. 2007), integral programming (Kitayama and Yasuda 2006), multi-objective optimization (Ho et al. 2006), and decision making (Nenortaitė 2007).

2.8 Summary

This chapter addresses the basic concepts and models of optimization theory: linear programming, non-linear programming, goal programming, multi-objective programming, Stackelberg games, and particle swarm optimization are introduced. These concepts, models and solution techniques will be used in the rest of this book.