

Chapter 2

Credibilistic Programming

The decision analysis with fuzzy objective or fuzzy constraints is natural in some real-world applications, and sometimes such analysis seems to be inevitable. Credibilistic programming is a type of mathematical programming for handling the fuzzy decision problems. In the past years, researchers have proposed various efficient modeling approaches based on different fuzzy ranking criteria. For example, Liu and Liu (2002) introduced a concept of expected value operator and then provided a spectrum of expected value model to maximize the average objective under certain expected constraints. Liu and Iwamura (1998a,b) introduced a max-max chance-constrained programming model, and Liu (1998) provided a max-min chance-constrained programming model, which respectively maximizes the optimistic objective and pessimistic objective under certain credibility constraints. Based on the concepts of fuzzy entropy, Li et al. (2011) formulated an entropy optimization model, which was extended by Qin et al. (2009) to the cross-entropy minimization model. Recently, Li et al. (2012) introduced a regret minimization model to minimize the distance between the fuzzy objective values and the best values.

This chapter mainly provides a general description on nonlinear programming, multi-objective programming, and credibilistic programming. In addition, a brief introduction on the solution methods will also be given, including the Kuhn-Tucker conditions and genetic algorithm.

2.1 Mathematical Programming

As one of the most widely used technique in operations research, mathematical programming is defined as a means of maximizing a quantity known as objective function, subject to a set of constraints. It is impossible that this section covers all concepts of mathematical programming. Therefore, this section only introduces some basic concepts and techniques such that readers can gain an understanding of them throughout the book.

2.1.1 Single-Objective Programming

In mathematical terms, the general form of a single-objective programming can be written as follows:

$$\begin{cases} \max & f(\mathbf{x}) \\ \text{s.t.} & g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \dots, n \end{cases} \quad (2.1)$$

where $\mathbf{x} = (x_1, x_2, \dots, x_m)$ is the decision vector, the first line defines the objective function to be maximized, and the second line defines the inequality constraints.

Definition 2.1 For the single-objective programming model (2.1), the set

$$S = \{\mathbf{x} \in \mathbb{R}^m \mid g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \dots, n\} \quad (2.2)$$

is called the feasible set. An element \mathbf{x} in S is called a feasible solution.

Definition 2.2 For the single-objective programming model (2.1), a feasible solution \mathbf{x}^* is called the local optimal solution if and only if there is a real number $\varepsilon > 0$ such that

$$f(\mathbf{x}^*) \geq f(\mathbf{x}) \quad (2.3)$$

for all feasible solution \mathbf{x} with $\|\mathbf{x} - \mathbf{x}^*\| < \varepsilon$.

Definition 2.3 For the single-objective programming model (2.1), a feasible solution \mathbf{x}^* is called the global optimal solution if and only if

$$f(\mathbf{x}^*) \geq f(\mathbf{x}) \quad (2.4)$$

for all feasible solution $\mathbf{x} \in S$.

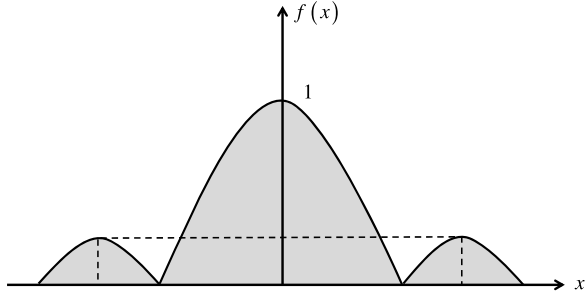
Remark 2.1 Note that a global optimal solution must be a local optimal solution, but a local optimal solution may be not a global optimal solution.

Example 2.1 In order to illustrate the concepts of feasible solution, local optimal solution, and global optimal solution, we consider the following single-objective programming problem

$$\begin{cases} \max & \max\{(x-1)(2-x), 1-x^2, -(x+1)(x+2)\} \\ \text{s.t.} & (x+2)(x-2) \leq 0. \end{cases}$$

It is easy to prove that the feasible set is a closed interval $S = [-2, 2]$. There are three local optimal solutions $x_1 = -1.5$, $x_2 = 0$, $x_3 = 1.5$, among which $x_2 = 0$ is the global optimal solution. See Fig. 2.1.

Fig. 2.1 Local optimal solution and global optimal solution



One of the most outstanding contributions to mathematical programming is known as the *Kuhn-Tucker conditions*. In order to introduce them, we first give some definitions. An inequality constraint $g_i \leq 0$ is said to be *active* at a point \mathbf{x} if $g_i(\mathbf{x}) = 0$. A feasible point \mathbf{x} is said to be *regular* if the gradient vector $\nabla g_i(\mathbf{x})$ of all active constraints are linearly independent.

Suppose that \mathbf{x}^* is a regular point of the single-objective programming model (2.1), and all the functions f and g_i , $i = 1, 2, \dots, n$ are differentiable. If \mathbf{x}^* is a local optimal solution, then there exist Lagrangian multipliers λ_i , $i = 1, 2, \dots, n$ such that the following Kuhn-Tucker conditions hold,

$$\begin{cases} \nabla f(\mathbf{x}^*) - \sum_{i=1}^n \lambda_i \nabla g_i(\mathbf{x}^*) = 0 \\ \lambda_i g_i(\mathbf{x}^*) = 0, \quad i = 1, 2, \dots, n \\ \lambda_i \geq 0, \quad i = 1, 2, \dots, n. \end{cases} \quad (2.5)$$

Furthermore, if functions g_i , $i = 1, 2, \dots, n$ are all convex and the objective function f is concave, then a regular point \mathbf{x}^* is the global optimal solution if and only if it satisfies the Kuhn-Tucker conditions.

Example 2.2 In this example, we apply the Kuhn-Tucker conditions to solve the following single-objective programming problem

$$\begin{cases} \max & -x_1^2 - x_2^2 - x_3^2 \\ \text{s.t.} & 1 - x_1 - x_2 - x_3 \leq 0 \\ & x_1, x_2, x_3 \geq 0. \end{cases}$$

It is clear that the objective function $f = -x_1^2 - x_2^2 - x_3^2$ is differentiable and concave, and the constraint function $g = 1 - x_1 - x_2 - x_3$ is differentiable and convex. Therefore, a point is the global optimal solution if and only if it satisfies the Kuhn-Tucker conditions

$$\begin{cases} -2x_i + \lambda = 0, \quad i = 1, 2, 3 \\ \lambda(1 - x_1 - x_2 - x_3) = 0 \\ \lambda \geq 0 \end{cases}$$

where λ is the Lagrangian multiplier. According to the inequality constraint, there is at least one index i such that $x_i > 0$, which implies that $\lambda > 0$ and

$$x_1 + x_2 + x_3 = 1.$$

Taking $x_i = \lambda/2$ into this equation, it is solved that the global optimal solution is $x_1 = x_2 = x_3 = 1/3$, and the Lagrangian multiplier is $\lambda = 2/3$.

2.1.2 Multi-Objective Programming

In multi-objective programming, also known as multi-attribute programming or multi-criteria programming, we attempt to simultaneously maximize two or more conflicting objectives subject to certain constraints, which is formulated as follows,

$$\begin{cases} \max & [f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_p(\mathbf{x})] \\ \text{s.t.} & g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \dots, n. \end{cases} \quad (2.6)$$

Similarly, the set $S = \{\mathbf{x} \in \mathbb{R}^m \mid g_i(\mathbf{x}) \leq 0, i = 1, 2, \dots, n\}$ is called the feasible set, and each element \mathbf{x} of S is called a feasible solution.

For a nontrivial multi-objective programming problem, one cannot identify a solution that simultaneously maximizes all objectives. If the decision-maker has a real preference function which aggregates all the objectives, then we may maximize the preference function under the same set of constraints. The obtained single-objective programming model is called a *compromise model* whose solution is called a *compromise solution*.

The first well-known compromise model is formulated to maximize the linearly weighted objective function

$$\begin{cases} \max & \sum_{i=1}^p \lambda_i f_i(\mathbf{x}) \\ \text{s.t.} & g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \dots, n \end{cases} \quad (2.7)$$

where $\lambda_1, \lambda_2, \dots, \lambda_p$ are nonnegative real numbers, which denote the preferences of the decision-maker on different objectives. Taking a two-objective programming model for example, if the first objective is more important than the second one, we set $\lambda_1 > \lambda_2$. Otherwise, we set $\lambda_1 \leq \lambda_2$.

The second way is formulated to minimize the distance between the objective vector and an ideal vector $(f_1^*, f_2^*, \dots, f_p^*)$, where f_i^* is the maximum value for the i th objective without considering other objectives. If the Euclidean distance is used, we have

$$\begin{cases} \min & \sqrt{\sum_{i=1}^p (f_i(\mathbf{x}) - f_i^*)^2} \\ \text{s.t.} & g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \dots, n. \end{cases} \quad (2.8)$$

Table 2.1 Database for notebook computer selection

Candidate	Price (RMB)	Weight (kg)	Size (inch)
x_1	6730	2.2	15.4
x_2	6980	2.2	15.4
x_3	9200	1.3	12.1
x_4	13000	2.2	12.1

The term of Pareto optimality and the related terms of Pareto dominance, Pareto solution, and Pareto set are the most basic concepts in multi-objective programming theory (Ehrgott 2000; Farina and Amato 2004; Li and Wong 2009; Pierro et al. 2007) and algorithms (Chou et al. 2008; Delgado et al. 2008; Ewald et al. 2008; Hung et al. 2008; Tan et al. 2005; Zou et al. 2008). Roughly speaking, Pareto optimality means that when we attempt to improve an objective further, other objectives suffer as a result.

Definition 2.4 For any feasible solutions $\mathbf{x}, \mathbf{y} \in S$, \mathbf{x} is said to Pareto dominate \mathbf{y} if and only if

- (a) $f_i(\mathbf{x}) \geq f_i(\mathbf{y})$ for all $i \in \{1, 2, \dots, p\}$;
- (b) $f_j(\mathbf{x}) > f_j(\mathbf{y})$ for at least one index $j \in \{1, 2, \dots, p\}$.

Definition 2.5 A feasible solution \mathbf{x} is said to be a Pareto solution if there is no feasible solution \mathbf{y} which Pareto dominates \mathbf{x} . The set of all Pareto solutions is called the Pareto set.

Example 2.3 Suppose that we would like to select a cheap, light and small notebook computer from the candidates $\{x_1, x_2, x_3, x_4\}$. The detailed data about each candidate is shown in Table 2.1. It is easy to prove that x_1 and x_3 are Pareto solutions. Note that x_2 is not a Pareto solution since it is dominated by x_1 , and x_4 is not a Pareto solution since it is dominated by x_3 .

Remark 2.2 The global optimal solution \mathbf{x}^* of compromise model (2.7) must be a Pareto solution. Otherwise, according to Definition 2.5, there is a feasible solution \mathbf{x} such that $f_i(\mathbf{x}) \geq f_i(\mathbf{x}^*)$ for all $i = 1, 2, \dots, p$, and the strict inequality holds for at least one index. Then it is easy to prove that

$$\sum_{i=1}^p \lambda_i f_i(\mathbf{x}) > \sum_{i=1}^p \lambda_i f_i(\mathbf{x}^*),$$

which is in contradiction with the fact that \mathbf{x}^* is the global optimal solution. Similarly, we can prove that the global optimal solution of compromise model (2.8) is also a Pareto solution.

2.2 Credibilistic Programming

Fuzzy programming is the mathematical programming in fuzzy environment, that is, the objective function f or constraint functions $g_i, i = 1, 2, \dots, n$ contain fuzzy parameters. Assume that \mathbf{x} is a decision vector, and ξ is a fuzzy vector, then the general fuzzy programming model can be written as

$$\begin{cases} \max & f(\mathbf{x}, \xi) \\ \text{s.t.} & g_i(\mathbf{x}, \xi) \leq 0, \quad i = 1, 2, \dots, n. \end{cases} \quad (2.9)$$

Example 2.4 In this example, we consider the portfolio selection problem. The term portfolio refers to any collection of financial assets such as stocks, bonds, and cash. Portfolio may be held by individual investors or managed by financial professionals, banks and other financial institutions.

Assume that there are m stocks, and we use ξ_i to denote the return of the i th stock. In general, ξ_i is given as $(p'_i + d_i - p_i)/p_i$ where p_i is the closing price at present, p'_i is the closing price in the next year, and d_i is the dividend during the coming year. Note that the values of p'_i and d_i in a future time period are clearly unknown at present. If they are estimated as fuzzy quantities, then ξ_i is a fuzzy variable. Furthermore, for each portfolio (x_1, x_2, \dots, x_m) , where x_i denotes the proportion of the total capital invested in stock i , the total return

$$f(\mathbf{x}, \xi) = \xi_1 x_1 + \xi_2 x_2 + \dots + \xi_m x_m$$

is also a fuzzy variable. In this case, if the investor would like to maximize the total return, we get the following fuzzy programming model

$$\begin{cases} \max & \xi_1 x_1 + \xi_2 x_2 + \dots + \xi_m x_m \\ \text{s.t.} & x_1 + x_2 + \dots + x_m = 1 \\ & x_i \geq 0, \quad i = 1, 2, \dots, m \end{cases} \quad (2.10)$$

where the first constraint implies that all the capital will be invested to the m stocks, and the next set of constraints implies that short sale and borrowing are not allowed.

Generally speaking, it is meaningless to maximize a fuzzy objective since there is not a natural ordership in fuzzy world. Therefore, we need to define a *credibilistic mapping* from the collection of fuzzy variables to the set of real numbers, such that we can rank fuzzy variables according to the natural ordership of real numbers. For the fuzzy programming model (2.9), if the credibilistic mappings U, U_1, U_2, \dots, U_n are taken, we get the following model

$$\begin{cases} \max & U[f(\mathbf{x}, \xi)] \\ \text{s.t.} & U_i[g_i(\mathbf{x}, \xi)] \leq 0, \quad i = 1, 2, \dots, n. \end{cases} \quad (2.11)$$

Note that (2.11) is a crisp nonlinear programming model since the objective function and constraints are both well defined. In what follows, we will call it a *credibilistic programming* model. The following chapters will introduce some mainly used

credibilistic mappings including the expected value operator, optimistic value, pessimistic value, entropy, cross-entropy, and distance.

Definition 2.6 For the credibilistic programming model (2.11), the set

$$S = \{\mathbf{x} \in \mathbb{R}^m \mid U_i[g_i(\mathbf{x}, \boldsymbol{\xi})] \leq 0, i = 1, 2, \dots, n\} \quad (2.12)$$

is called the feasible set. An element \mathbf{x} in S is called a feasible solution.

Definition 2.7 For the credibilistic programming model (2.11), a feasible solution \mathbf{x}^* is called the local optimal solution if and only if there is a real number $\varepsilon > 0$ such that

$$U[f(\mathbf{x}^*, \boldsymbol{\xi})] \geq U[f(\mathbf{x}, \boldsymbol{\xi})] \quad (2.13)$$

for all feasible solution \mathbf{x} with $\|\mathbf{x} - \mathbf{x}^*\| < \varepsilon$.

Definition 2.8 For the credibilistic programming model (2.11), a feasible solution \mathbf{x}^* is called the global optimal solution if and only if

$$U[f(\mathbf{x}^*, \boldsymbol{\xi})] \geq U[f(\mathbf{x}, \boldsymbol{\xi})] \quad (2.14)$$

for all feasible solution $\mathbf{x} \in S$.

If there are multiple objective functions f_1, f_2, \dots, f_p , we can define the following multi-objective credibilistic programming model,

$$\begin{cases} \max & [U[f_1(\mathbf{x}, \boldsymbol{\xi})], U[f_2(\mathbf{x}, \boldsymbol{\xi})], \dots, U[f_p(\mathbf{x}, \boldsymbol{\xi})]] \\ \text{s.t.} & U_i[g_i(\mathbf{x}, \boldsymbol{\xi})] \leq 0, \quad i = 1, 2, \dots, n. \end{cases} \quad (2.15)$$

Definition 2.9 For any feasible solutions $\mathbf{x}, \mathbf{y} \in S$, \mathbf{x} is said to Pareto dominate \mathbf{y} if and only if

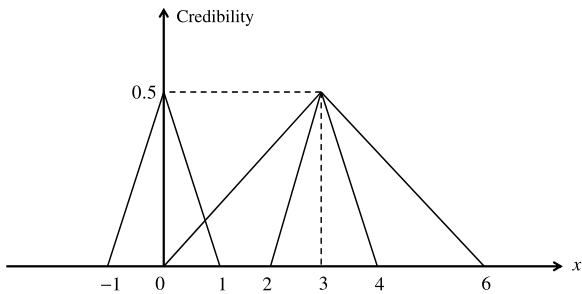
- (a) $U[f_i(\mathbf{x}, \boldsymbol{\xi})] \geq U[f_i(\mathbf{y}, \boldsymbol{\xi})]$ for all $i \in \{1, 2, \dots, m\}$;
- (b) $U[f_j(\mathbf{x}, \boldsymbol{\xi})] > U[f_j(\mathbf{y}, \boldsymbol{\xi})]$ for at least one index $j \in \{1, 2, \dots, m\}$.

Definition 2.10 A feasible solution \mathbf{x} is said to be a Pareto solution for the multi-objective credibilistic programming model (2.15) if there is no feasible solution \mathbf{y} which Pareto dominates \mathbf{x} . The set of all Pareto solutions is called the Pareto set.

Example 2.5 Let us reconsider the portfolio selection problem. Suppose that there are three stocks and the returns are independent triangular fuzzy variables $\xi_1 = (0, 3, 6)$, $\xi_2 = (2, 3, 4)$, and $\xi_3 = (-1, 0, 1)$. See Fig. 2.2.

It is clear that the second stock has a better return than the third stock since it follows from the credibility inversion theorem that $\text{Cr}\{\xi_2 \geq \xi_3\} = 1$. However, it is

Fig. 2.2 Credibility functions for ξ_1 , ξ_2 and ξ_3



difficult to compare the returns arising from the second stock and the first stock. As a result, if we take portfolios $\mathbf{x} = (1, 0, 0)$ and $\mathbf{y} = (0, 1, 0)$, it is difficult to decide which one is better. In fact, it follows from the independence that fuzzy vector (ξ_1, ξ_2) has a joint credibility function $v = v_1 \wedge v_2$. Then according to the credibility inversion theorem, event $\{\xi_1 \geq \xi_2\}$ has a credibility

$$\begin{aligned} \text{Cr}\{\xi_1 \geq \xi_2\} &= 1 - \sup_{x_1 < x_2} v(x_1, x_2) \\ &= 1 - \sup_{x_1 < 3, x_2 = 3} (v_1(x_1) \wedge v_2(x_2)) \\ &= 0.5, \end{aligned}$$

and event $\{\xi_2 \geq \xi_1\}$ has a credibility

$$\text{Cr}\{\xi_2 \geq \xi_1\} = 1 - \sup_{x_2 < 3, x_1 = 3} (v_1(x_1) \wedge v_2(x_2)) = 0.5.$$

If there is a credibilistic mapping such that $U(\xi) = a$ for each triangular fuzzy variable $\xi = (a, b, c)$, then the credibilistic programming model is

$$\begin{cases} \max & 2x_2 - x_3 \\ \text{s.t.} & x_1 + x_2 + x_3 = 1 \\ & x_1, x_2, x_3 \geq 0. \end{cases}$$

It is easy to calculate that the optimal portfolio is $\mathbf{x}^* = (0, 1, 0)$. If there is another credibilistic mapping which takes value c for each triangular fuzzy variable $\xi = (a, b, c)$, we have the following credibilistic programming model

$$\begin{cases} \max & 6x_1 + 4x_2 + x_3 \\ \text{s.t.} & x_1 + x_2 + x_3 = 1 \\ & x_1, x_2, x_3 \geq 0. \end{cases}$$

In this case, the optimal portfolio is $\mathbf{x}^* = (1, 0, 0)$.

2.3 Genetic Algorithm

For a general credibilistic programming model, if the credibilistic mappings have analytical expressions and the objective and constraint functions have good mathematical properties, such as differentiability and convexity, we can design efficient solution algorithms by using the Kuhn-Tucker conditions. However, the credibilistic mappings generally have no analytical expressions or the expressions have bad properties. In this case, we can try to obtain suboptimal solution by using the genetic algorithm.

Genetic algorithm is a stochastic search method for optimization problems based on the mechanics of natural selection and natural genetics, i.e., survival of the fittest, which has been well-documented in the literatures, such as in Holland (1975), Goldberg (1989), Michalewicz (1996), Koza (1992, 1994), and so on. In the past decades, genetic algorithm has obtained considerable success in providing satisfactory solutions to many complex optimization problems and received more and more attentions. This section introduces the basic steps for genetic algorithm including representation structure, initialization, evaluation function, selection process, crossover operation, and mutation operation. At the end of this section, a general procedure of the genetic algorithm is also given.

2.3.1 Representation Structure

The first problem for genetic algorithm is how to construct a one to one mapping between the solution space and the chromosome space such that the following operations such as crossover and mutation, can be simplified. The mapping from the solution space to the chromosome space is called *encoding*, and the mapping from the chromosome space to the solution space is called *decoding*.

The representation of solution is generally problem dependent, while binary encoding and floating encoding are two mainly used representation structures. Taking the floating encoding for example, let $\mathbf{x} = (x_1, x_2, \dots, x_m)$ be a solution vector in the solution space satisfying

$$\begin{cases} x_1 + x_2 + \dots + x_m = 1 \\ x_i \geq 0, \quad i = 1, 2, \dots, m. \end{cases} \quad (2.16)$$

We may encode the solution by a chromosome $\mathbf{v} = (v_1, v_2, \dots, v_m)$ satisfying

$$v_i \geq 0, \quad i = 1, 2, \dots, m. \quad (2.17)$$

Then the encoding and decoding processes are determined by the equations

$$x_i = \frac{v_i}{v_1 + v_2 + \dots + v_m}, \quad i = 1, 2, \dots, m. \quad (2.18)$$

2.3.2 Initialization

Define an integer *pop-size* as the size of population, which generally depends on the nature of the problem. Randomly generate *pop-size* chromosomes for the initialized population. Usually, it is difficult to produce feasible chromosomes explicitly for complex optimization problems with irregular feasible set. However, if the decision-maker can predetermine a region with regular sharp which contains the optimal solution, we may initialize the population in the regular field.

We generate a random point from the region and check its feasibility. If it is feasible, then it will be accepted as a chromosome. If not, we regenerate a point randomly until a feasible one is obtained. We repeat this procedure *pop-size* times, and generate the first population. The initialization process is summarized as follows.

Algorithm 2.1 (Initialization Process)

Step 1. Set $i = 1$.

Step 2. Randomly generate a chromosome from the predetermined region.

Step 3. If it is feasible, set $i = i + 1$. Otherwise, go to step 2.

Step 4. If $i \leq \text{pop-size}$, go to step 2.

Step 5. Return the initialized population $\mathbf{v}_i, i = 1, 2, \dots, \text{pop-size}$.

2.3.3 Evaluation Function

Evaluation function assigns each chromosome a probability of reproduction so that its likelihood of being selected is proportional to its fitness relative to the other chromosomes in the population. That is, the chromosomes with higher fitness will have more chance to produce offspring.

Assume that the decision-maker can give an order relationship among these *pop-size* chromosomes such that they are rearranged from good to bad. For example, for a single-objective programming problem, a chromosome with a larger objective value is better, while for a multi-objective programming problem, a chromosome with a larger aggregating preference function is better. One well-known evaluation function is based on allocation of reproductive trials according to rank rather than actual objective values. For each $\alpha \in (0, 1)$, we define the rank-based evaluation function as follows,

$$\text{Eval}(\mathbf{v}_i) = \alpha(1 - \alpha)^{i-1}, \quad i = 1, 2, \dots, \text{pop-size}. \quad (2.19)$$

Note that \mathbf{v}_1 is the best chromosome, and $\mathbf{v}_{\text{pop-size}}$ is the worst one.

Algorithm 2.2 (Evaluation Process)

Step 1. Initialize a real number $\alpha \in (0, 1)$.

Step 2. Calculate the objective values f_i for all chromosomes.

Step 3. Reorder these chromosomes according to their objective values.

Step 4. Set $i = 1$.

Step 5. Calculate the evaluation value for the i th chromosome

$$Eval(\mathbf{v}_i) = \alpha(1 - \alpha)^{i-1}.$$

Step 6. If $i < pop-size$, set $i = i + 1$, and goto step 5.

2.3.4 Selection Process

During each successive generation, a proportion of the existing population is selected to breed a new generation. The selection process is based on spinning the roulette wheel $pop-size$ times, and selecting a single chromosome at each time. The roulette wheel is a fitness-proportional selection, where fitter chromosomes (as measured by the objective value) are typically more likely to be selected. The selection process is summarized as follows.

Algorithm 2.3 (Selection Process)

Step 1. Calculate the reproduction probability q_i for each chromosome \mathbf{v}_i ,

$$q_0 = 0, \quad q_i = \sum_{j=1}^i Eval(\mathbf{v}_j), \quad i = 1, 2, \dots, pop-size.$$

Step 2. Generate a random number r in $(0, q_{pop-size}]$.

Step 3. Select the chromosome \mathbf{v}_i such that $q_{i-1} < r \leq q_i$.

Step 4. Repeat the second and third steps $pop-size$ times and obtain $pop-size$ chromosomes.

2.3.5 Crossover Operation

Crossover is one of the mainly used operations for generating a second population. First, we define a parameter P_c to denote the probability of crossover. Then we randomly select some chromosomes as parents from the pool selected previously. Repeat the following process $pop-size$ times: generate a random number r from $[0, 1]$, and select the chromosome \mathbf{v}_i if $r < p_c$. Note that not all chromosomes can be selected, and different chromosomes have the equal chance to be selected. Denote the selected parents by $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \dots, \mathbf{u}_c$ and divide them into the following pairs:

$$(\mathbf{u}_1, \mathbf{u}_2), \quad (\mathbf{u}_3, \mathbf{u}_4), \quad (\mathbf{u}_5, \mathbf{u}_6), \quad \dots$$

Let us illustrate the crossover operation by the first pair $(\mathbf{u}_1, \mathbf{u}_2)$. Generate a random number λ from the open interval $(0, 1)$, then the crossover operator on \mathbf{u}_1 and \mathbf{u}_2 will produce two children \mathbf{x} and \mathbf{y} as follows:

$$\mathbf{x} = \lambda \mathbf{u}_1 + (1 - \lambda) \mathbf{u}_2, \quad \mathbf{y} = (1 - \lambda) \mathbf{u}_1 + \lambda \mathbf{u}_2.$$

We check the feasibility for each child before accepting it. If both children are feasible, then we replace the parents with them. If not, we keep the feasible one if it exists, and then redo the crossover operator until two feasible children are obtained or a number of cycles is finished.

Algorithm 2.4 (Crossover Operation)

- Step 1.** Initialize a crossover probability P_c , and set $i = 1$.
- Step 2.** Generate a random number r from $[0, 1]$.
- Step 3.** If $r \leq P_c$, select chromosome \mathbf{v}_i as the parent, and set $i = i + 1$.
- Step 4.** If $i \leq \text{pop-size}$, go to step 2.
- Step 5.** Denote the selected parents by $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \dots, \mathbf{u}_c$, and set $j = 1$.
- Step 6.** Generate a random number λ from $(0, 1)$, and produce two children

$$\mathbf{x} = \lambda \mathbf{u}_j + (1 - \lambda) \mathbf{u}_{j+1}, \quad \mathbf{y} = (1 - \lambda) \mathbf{u}_j + \lambda \mathbf{u}_{j+1}.$$

Redo this operation until two feasible children are obtained or a number of cycles is finished.

- Step 7.** Replace the parents with the feasible children, and set $j = j + 2$.
- Step 8.** If $j \leq c$, go to step 6.

2.3.6 Mutation Operation

Mutation is another operation for updating the chromosomes. We define a parameter P_m to denote the probability of mutation, and randomly select some chromosomes as parents in a similar way to the process of selecting parents for crossover.

For each selected parent \mathbf{v} , we mutate it in the following way. Let λ be an approximate large positive number, and let \mathbf{d} be a mutation direction. If $\mathbf{v} + \lambda \mathbf{d}$ is not feasible, then we randomly decrease the value of λ until it is feasible. If the above process cannot find a feasible solution in a predetermined number of iterations, we set $\lambda = 0$. Anyway, we replace the parent chromosome \mathbf{v} with its child $\mathbf{v} + \lambda \mathbf{d}$. The mutation operation is summarized as follows.

Algorithm 2.5 (Mutation Operation)

- Step 1.** Initialize a mutation probability P_m , and set $i = 1$.
- Step 2.** Generate a random number r from $[0, 1]$.
- Step 3.** If $r \leq P_m$, generate a mutation direction \mathbf{d} and a parameter λ such that $\mathbf{v}_i + \lambda \mathbf{d}$ is a feasible chromosome. Replace \mathbf{v}_i with $\mathbf{v}_i + \lambda \mathbf{d}$.
- Step 4.** Set $i = i + 1$.
- Step 5.** If $i \leq \text{pop-size}$, go to step 2.

2.3.7 General Procedure

Following selection, crossover and mutation operations, a new population is generated. Genetic algorithm will terminate after a given number of cyclic iterations of the above steps. We now summarize the general procedure for genetic algorithm as follows.

Algorithm 2.6 (Genetic Algorithm)

- Step 1.** Randomly Initialize *pop-size* chromosomes.
- Step 2.** Calculate the objective values for all chromosomes.
- Step 3.** Evaluate the fitness of each chromosome via the objective values.
- Step 4.** Select the chromosomes by spinning the roulette wheel.
- Step 5.** Update the chromosomes by using crossover and mutation.
- Step 6.** Repeat the second to fifth steps for a given number of cycles.
- Step 7.** Report the best found chromosome as the suboptimal solution.

References

- Chou TY, Liu TK, Lee CN, Jeng CR (2008) Method of inequality-based multiobjective genetic algorithm for domestic daily aircraft routing. *IEEE Trans Syst Man Cybern, Part A* 38(2):299–308
- Delgado M, Cuellar MP, Pegalajar MC (2008) Multiobjective hybrid optimization and training of recurrent neural networks. *IEEE Trans Syst Man Cybern, Part B* 38(2):381–403
- Ehrgott M (2000) *Multicriteria optimization*. Springer, Berlin
- Ewald G, Kurek W, Brdys MA (2008) Grid implementation of a parallel multiobjective genetic algorithm for optimized allocation of chlorination stations in drinking water distribution systems: Chojnice case study. *IEEE Trans Syst Man Cybern, Part C* 38(4):497–509
- Farina M, Amato P (2004) A fuzzy definition of “optimality” for many-criteria optimization problems. *IEEE Trans Syst Man Cybern, Part A* 34(2):315–326
- Goldberg DE (1989) *Genetic algorithms & engineering optimization*. Wiley, New York
- Holland JH (1975) *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor
- Hung MH, Shu LS, Ho SJ, Hwang SF, Ho SY (2008) A novel intelligent multiobjective simulated annealing algorithm for designing robust PID controllers. *IEEE Trans Syst Man Cybern, Part A* 38(2):319–330
- Koza JR (1992) *Genetic programming*. MIT Press, Cambridge
- Koza JR (1994) *Genetic programming, II*. MIT Press, Cambridge
- Li X, Ralescu D, Tang T (2011) Fuzzy train energy consumption minimization model and algorithm. *Iran J Fuzzy Syst* 8(4):77–91
- Li X, Shou BY, Qin ZF (2012) An expected regret minimization portfolio selection model. *Eur J Oper Res* 218:484–492
- Li X, Wong HS (2009) Logic optimality for multi-objective optimization. *Appl Math Comput* 215:3045–3056
- Liu B (1998) Minimax chance constrained programming models for fuzzy decision systems. *Inf Sci* 112(1–4):25–38
- Liu B, Iwamura K (1998a) Chance constrained programming with fuzzy parameters. *Fuzzy Sets Syst* 94(2):227–237

- Liu B, Iwamura K (1998b) A note on chance constrained programming with fuzzy coefficients. *Fuzzy Sets Syst* 100(1–3):229–233
- Liu B, Liu YK (2002) Expected value of fuzzy variable and fuzzy expected value models. *IEEE Trans Fuzzy Syst* 10(4):445–450
- Michalewicz Z (1996) Genetic algorithms + data structures = evolution programs, 3rd edn. Springer, Berlin
- Pierro F, Khu ST, Savic DA (2007) An investigation on preference order ranking scheme for multiobjective evolutionary optimization. *IEEE Trans Evol Comput* 11(1):17–45
- Qin ZF, Li X, Ji XY (2009) Portfolio selection based on fuzzy cross-entropy. *J Comput Appl Math* 228:139–149
- Tan KC, Khor EF, Lee TH (2005) Multiobjective evolutionary algorithms and applications. Springer, London
- Zou XF, Chen Y, Liu MZ, Kang JS (2008) A new evolutionary algorithm for solving many-objective optimization problems. *IEEE Trans Syst Man Cybern, Part B* 38(5):1402–1412

<http://www.springer.com/978-3-642-36375-7>

Credibilistic Programming

An Introduction to Models and Applications

Li, X.

2013, IX, 144 p., Hardcover

ISBN: 978-3-642-36375-7