Stochastic Multilevel Programming with a Hybrid Intelligent Algorithm

Jinwu Gao[a], Baoding Liu[b]
Uncertainty Theory Laboratory, Department of Mathematical Sciences
Tsinghua University, Beijing 100084, China
[a]jgao@orsc.edu.cn [b]liu@tsinghua.edu.cn

Abstract: A framework of stochastic multilevel programming is proposed for modelling decentralized decision-making problem in stochastic environment. According to different decision criteria, the stochastic decentralized decision-making problem is formulated as expected value multilevel programming, and chance-constrained multilevel programming. In order to solve the proposed stochastic multilevel programming models for the Stackelberg-Nash equilibriums, genetic algorithms, neural networks and stochastic simulation are integrated to produce a hybrid intelligent algorithm. Finally, two numerical examples are provided to illustrate the effectiveness of the hybrid intelligent algorithm.
Keywords: Multilevel programming; stochastic programming; neural network; genetic algorithm

1 Introduction

In many decision processes, there are multiple decision makers arranged within a hierarchical administrative structure. These decision makers have divergent even conflict objectives, which intervenes in the decisions to be made. This is the so called decentralized decision-making problems, which has long been recognized as an important and challenging topic in many research areas including economics [1][8][16][23], transportation [9][56][59], engineering [19][52], and so on.

Multilevel programming was first proposed by Bracken and McGill [14][15] for dealing with decentralized decision-making problems. After that, a special case called bilevel programming, which is equivalent to a two person Stackelberg game, has attracted much attention. In [10], Ben-Ayed and Blair proved that the bilevel programming problem is NP-hard. However, many numerical algorithms have been designed for solving this problem. For example, implicit enumeration scheme [17], the kth best algorithm [12][13], complementary pivot algorithm [13][28] , grid search algorithm [4][5], branch-and-bound algorithm [7][25], descent algorithm [53]. General cases with multiple followers [6][32] or multilevel levels [2][4][5][57] were also discussed.

Real-world situations are often not deterministic. For instance, in economic systems, costs and demands are often subject to fluctuations and difficult to measure. In engineering, external conditions and measurement or manufacturing errors introduce uncertainty into the problems. These situations emphasize the need for models which are able to tackle uncertainty inherent in decision systems. However, there is little research in literature about decentralized decision making problem in uncertain decision systems. In [50], Patriksson and Wynter discussed stochastic mathematical programs with equilibrium, which includes stochastic bilevel programming as a special case. In [51], Sakawa, *et al.* dealt with multilevel programming problems with fuzzy parameters by a fuzzy interactive approach [30][31][54]. The fuzzy interactive approach is formulated by combined use of the fuzzy tolerance membership functions and multi-objective decision-making, which makes the original problem much more simplified, and much easier to solve. However, it differs the traditional solution concept, Stackelberg-Nash equilibrium, because it is assumed that decision makers at all levels are essentially cooperative.

For the purpose of tackling uncertainty in decision systems, various stochastic programming models have been formulated in literature, among them we want to mention expected value model, and chance-constrained programming model [18]. Motivated by the above mentioned works, in this paper, we propose two classes of stochastic multilevel programming models including expected value multi-level programming model (EVMLP) and chance-constrained multilevel programming (CCMLP), and design an intelligent algorithm for Stackelberg-Nash equilibrium of the proposed stochastic models. Besides a new model CCMLP, the proposed EVMLP may have multiple followers, which distinguishes from stochastic mathematical programs with equilibrium [50]. Furthermore, we employ the Stackelberg-Nash equilibrium solution concept, which distinguishes our work from that of [51].

The purposes of this paper are twofolds, one is to establish an EVMLP and a CCMLP for a stochastic decentralized decision making problem with multiple followers. The other is to design efficient numerical algorithms for the Stackelberg solution. Toward that end, the following sections are organized as follows. Section 2 discusses the formulation of EVMLP and CCMLP, then the concepts of Nash equilibrium and Stackelberg-Nash

equilibrium are defined in this section. In Section 3.1, we first discuss how to approximate the uncertain functions involved in the proposed models. The motivation of the method is as follows, when searching for Nash equilibrium and Stackelberg-Nash equilibrium, we need many times of stochastic simulation to evaluate the uncertain functions. It is known that stochastic simulations are time-consuming processes. In order to speed up the solution process, we desire to use relatively simple functions, neural networks (NNs), to replace these uncertain functions after they are well-trained by input-output data produced by stochastic simulations. After that, Section 3.2 discusses approaches on computing Nash equilibrium with respect to a given control vector. Based on the above discussion, Section 3.3 discusses how to solve the EVMLP and CCMLP for the Stackelber-Nash equilibrium. To do this, we embed the above trained NNs, and algorithms for the Nash equilibrium into a GA to produce a hybrid intelligent algorithm (HIA). At the end of the paper, two numerical examples are provided to show the effectiveness of the proposed algorithm.

2 Stochastic Multilevel Programming

In order to formulating a stochastic version of multilevel programming, we start from a deterministic one.

Consider a decentralized two-level decision system in which there are one leader and m followers. Assume that the leader and followers may have their own decision variables and objective functions, and the leader can only influence the reactions of followers through his own decision variables, while the followers have full authority to decide how to optimize their own objective functions in view of the decisions of the leader and other followers. Let $\boldsymbol{x}$ and $\boldsymbol{y}_i$ be the control vectors of the leader and the ith followers, $i = 1, 2, \cdots, m$, respectively. We also assume that the objective functions of the leader and ith followers are $F(\boldsymbol{x}, \boldsymbol{y}_1, \cdots, \boldsymbol{y}_m, \boldsymbol{b})$ and $f_i(\boldsymbol{x}, \boldsymbol{y}_1, \cdots, \boldsymbol{y}_m, \boldsymbol{b})$, $i = 1, 2, \cdots, m$, respectively, where $\boldsymbol{b}$ is a vector represents the problem parameters.

In addition, let S be the feasible set of control vector $\boldsymbol{x}$ of the leader, defined by

$$G(\boldsymbol{x}, \boldsymbol{b}) \leq 0 \tag{1}$$

where G is a vector-valued function of decision vector $\boldsymbol{x}$ and 0 is a vector with zero components. Then for each decision $\boldsymbol{x}$ chosen by the leader, the feasible set of control vector $\boldsymbol{y}_i$ of the ith follower should be dependent on not only $\boldsymbol{x}$ but also $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{i-1}, \boldsymbol{y}_{i+1}, \cdots, \boldsymbol{y}_m$, and generally represented by the expected constraint

$$g_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{b}) \leq 0 \tag{2}$$

where g_i are vector valued functions, $i = 1, 2, \cdots, m$.

Multilevel programming offers a means of studying decentralized decision systems. Assume that the leader first chooses his control vector $\boldsymbol{x} \in S$, and the followers determine their control array $(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m)$ after that. Then a general bilevel programming has the following form,

$$\begin{cases} \max\limits_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{b}) \\ \text{subject to:} \\ \quad G(\boldsymbol{x}, \boldsymbol{b}) \leq 0 \\ \quad \text{where each } \boldsymbol{y}_i (i = 1, 2, \cdots, m) \text{ solves} \\ \quad \begin{cases} \max\limits_{\boldsymbol{y_i}} f_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{b}) \\ \text{subject to:} \\ \quad g(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{b}) \leq 0. \end{cases} \end{cases} \tag{3}$$

Now, assuming that the problem parameters are random, and represented by a random vector $\boldsymbol{\xi}$. Then the objective function $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi})$ and $f_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi})$, $i = 1, 2, \cdots, m$ become stochastic. Moreover, the stochastic constraints $G(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0$ and $g_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi}) \leq 0$ do not define the feasible set of the optimization problem of the leader and the followers mathematically. According to different decision criteria, we propose two types of stochastic programming models in the following subsections.

2.1 Expected Value Multilevel Programming

With the idea of optimizing the expected value of objective functions subject to some expected constraints, we propose the first type of stochastic multilevel programming: expected value multilevel programming (EVMLP).

Let the feasible set of control vector $\boldsymbol{x}$ of the leader be defined by the expected constraint

$$\boldsymbol{E}[G(\boldsymbol{x}, \boldsymbol{\xi})] \leq 0 \tag{4}$$

where G is a vector valued function and 0 is a zero vector. Then for each decision $\boldsymbol{x}$ chosen by the leader, the feasible set of control vector $\boldsymbol{y}_i$ of the ith follower should be dependent on not only $\boldsymbol{x}$ but also $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{i-1}, \boldsymbol{y}_{i+1}, \cdots, \boldsymbol{y}_m$, and generally represented by the expected constraint

$$\boldsymbol{E}[g_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi})] \leq 0 \tag{5}$$

where g_i are vector valued functions, $i = 1, 2, \cdots, m$.

Assume that the leader first chooses his control vector $\boldsymbol{x}$, and the followers determine their control array $(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m)$ thereafter. In order to maximize the expected return of the leader, we have the following EVMLP,

$$\begin{cases} \max\limits_{\boldsymbol{x}} \boldsymbol{E}\left[F(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi})\right] \\ \text{subject to:} \\ \quad \boldsymbol{E}\left[G(\boldsymbol{x}, \boldsymbol{\xi})\right] \leq 0 \\ \quad \text{where each } \boldsymbol{y}_i \ (i = 1, 2, \cdots, m) \text{ solves} \\ \quad \begin{cases} \max\limits_{\boldsymbol{y_i}} \boldsymbol{E}\left[f_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi})\right] \\ \text{subject to:} \\ \quad \boldsymbol{E}\left[g_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi})\right] \leq 0. \end{cases} \end{cases} \tag{6}$$

A Nash equilibrium of followers is the feasible array $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*)$ with respect to x if

$$\begin{aligned} &\boldsymbol{E}\left[f_i(\boldsymbol{x}, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_{i-1}^*, \boldsymbol{y}_i, \boldsymbol{y}_{i+1}^*, \cdots, \boldsymbol{y}_m^*, \boldsymbol{\xi})\right] \\ &\leq \boldsymbol{E}\left[f_i(\boldsymbol{x}, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*, \boldsymbol{\xi})\right] \end{aligned} \tag{7}$$

for any feasible $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_{i-1}^*, \boldsymbol{y}_i, \boldsymbol{y}_{i+1}^*, \cdots, \boldsymbol{y}_m^*)$ and $i = 1, 2, \cdots, m$.

A Stackelberg-Nash equilibrium to the EVMLP is the array $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*)$ satisfying

$$\begin{aligned} &\boldsymbol{E}\left[F(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}_1, \overline{\boldsymbol{y}}_2, \cdots, \overline{\boldsymbol{y}}_m, \boldsymbol{\xi})\right] \\ &\leq \boldsymbol{E}\left[F(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*, \boldsymbol{\xi})\right] \end{aligned} \tag{8}$$

for any feasible $\overline{\boldsymbol{x}}$ and the Nash equilibrium $(\overline{\boldsymbol{y}}_1, \overline{\boldsymbol{y}}_2, \cdots, \overline{\boldsymbol{y}}_m)$ with respect to $\overline{\boldsymbol{x}}$.

2.2 Chance-Constrained Multilevel Programming

Chance-constrained programming, which was initialized by Charnes and Cooper [18], offers a powerful means for modelling stochastic decision systems. The essential idea of chance-constrained programming is to optimize the optimistic return with a given confidence level subject to some chance constraints. Inspired by this idea, we propose the second type of stochastic multilevel programming: chance-constrained multilevel programming (CCMLP).

By chance constraint we mean that the stochastic constraints will hold at a confidence level provided as an appropriate safety margin by the decision-maker. Let the feasible set of control vector $\boldsymbol{x}$ of the leader be defined by the chance constraint

$$\Pr\left\{G(\boldsymbol{x}, \boldsymbol{\xi})] \leq 0\right\} \geq \beta_0 \tag{9}$$

where G is a vector valued function, 0 is a zero vector, and β_0 is a confidence level at which it is desired that the stochastic constraints hold. Then for each decision $\boldsymbol{x}$ chosen by the leader, the feasible set of control vector $\boldsymbol{y}_i$ of the ith follower should be dependent on not only $\boldsymbol{x}$ but also $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{i-1}, \boldsymbol{y}_{i+1}, \cdots, \boldsymbol{y}_m$, and generally represented by the chance constraints

$$\Pr\left\{g_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi}) \leq 0\right\} \geq \beta_i \tag{10}$$

where g_i are vector valued functions, β_i is a confidence level at which it is desired that the stochastic constraints hold, and $i = 1, 2, \cdots, m$.

Because of the stochastic parameters characterized by $\boldsymbol{\xi}$, the objective function is stochastic, which results that $\max_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi})$ is meaningless. A natural idea is to provide a confidence level α_0 at which it is desired that $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi}) \geq \overline{F}$, where the confidence level α_0 is provided as an appropriate safety margin by the leader. Then the objective of the leader is to maximize $\overline{F}$ with a chance constraint as follows,

$$\Pr\left\{F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi}) \geq \overline{F}\right\} \geq \alpha_0, \tag{11}$$

where $\overline{F}$ is referred to as the α_0-optimistic return of $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi})$. Then for each decision $\boldsymbol{x}$ chosen by the leader, the objective of the ith follower is to maximize the α_i-optimistic return of $f_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi})$, $\overline{f}_i$, with a chance constraint as follows,

$$\Pr\left\{f_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi}) \geq \overline{F}\right\} \geq \alpha_i, \tag{12}$$

where α_i is a confidence level provided as an appropriate safety margin by the ith follower.

Assume that the leader first chooses his control vector $\boldsymbol{x}$, and the followers determine their control array $(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m)$ thereafter. In order to maximize the α_0-optimistic return of the leader, we have the following CCMLP,

$$\left\{\begin{array}{l} \max\limits_{\boldsymbol{x}} \overline{F} \\ \text{subject to:} \\ \quad \Pr\left\{F(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi}) \geq \overline{F}\right\} \geq \alpha_0 \\ \quad \Pr\left\{G(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0\right\} \geq \beta_0 \\ \quad \text{where each } \boldsymbol{y}_i \ (i = 1, 2, \cdots, m) \text{ solves} \\ \quad \left\{\begin{array}{l} \max\limits_{\boldsymbol{y_i}} \\ \text{subject to:} \\ \quad \Pr\left\{f_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi}) \geq \overline{f}_i\right\} \geq \alpha_i \\ \quad \Pr\left\{g_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi}) \leq 0\right\} \geq \beta_i. \end{array}\right. \end{array}\right. \tag{13}$$

A Nash equilibrium of followers is the feasible array $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*)$ with respect to x if

$$\begin{aligned} &\max\left\{\overline{f}_i \,\middle|\, \Pr\left\{f_i(\boldsymbol{x}, \boldsymbol{y}_1^*, \cdots, \boldsymbol{y}_{i-1}^*, \boldsymbol{y}_i, \cdots, \boldsymbol{y}_m^*, \boldsymbol{\xi}) \geq \overline{f}_i\right\} \geq \alpha_i\right\} \\ &\leq \max\left\{\overline{f}_i \,\middle|\, \Pr\left\{f_i(\boldsymbol{x}, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*, \boldsymbol{\xi}) \geq \overline{f}_i\right\} \geq \alpha_i\right\} \end{aligned} \tag{14}$$

for any feasible $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_{i-1}^*, \boldsymbol{y}_i, \boldsymbol{y}_{i+1}^*, \cdots, \boldsymbol{y}_m^*)$ and $i = 1, 2, \cdots, m$.

A Stackelberg-Nash equilibrium to the CCMLP is the array $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*)$ satisfying

$$\begin{aligned} &\max\left\{\overline{F} \,\middle|\, \Pr\left\{F(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}_1, \overline{\boldsymbol{y}}_2, \cdots, \overline{\boldsymbol{y}}_m, \boldsymbol{\xi}) \geq \overline{F}\right\} \geq \alpha_0\right\} \\ &\leq \max\left\{\overline{F} \,\middle|\, \Pr\left\{F(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*, \boldsymbol{\xi}) \geq \overline{F}\right\} \geq \alpha_0\right\} \end{aligned} \tag{15}$$

for any feasible $\overline{\boldsymbol{x}}$ and the Nash equilibrium $(\overline{\boldsymbol{y}}_1, \overline{\boldsymbol{y}}_2, \cdots, \overline{\boldsymbol{y}}_m)$ with respect to $\overline{\boldsymbol{x}}$.

3 Hybrid Intelligent Algorithm for Stackelberg-Nash Equilibrium

It is known that multilevel programming, even in the simplest case, is NP-hard [10]. Thus, successful implementations of multilevel models rely largely on the development of efficient algorithms. In the past two decades, many intriguing numerical algorithms have been developed. These algorithms may fall into four categories:

- vertex enumeration approach, which aims to seek a comprising vertex by simplex algorithm based on adjusting the control variables [13][17];

- transformation approach, which transforms the lower level problems into constraints for the higher level by use of Karush-Kuhn-Tucker conditions or penalty functions [3][12][21][23].
- heuristic approaches like gradient techniques [20][22][29][55], and branch and bound method [6][7][21] [23][25].
- intelligent algorithms like genetic algorithm [32][52], and simulated annealing [60], which are especially suited for solving NP-hard problems like the multilevel programming problems.

As an extension of multilevel/stochastic programming, stochastic multilevel programming will further complicate ordinary multilevel programming. Correspondingly, resolution strategies will in many cases require some approximate methods for solving stochastic multilevel programming models. In the following subsections, we take the EVMLP as an example to discuss the numerical solution methods.

3.1 Uncertain function approximation

By uncertain functions we mean the functions with uncertain parameters. The uncertain functions in stochastic multilevel programming (6) and (13) may fall into three types. Due tho the complexity, we design stochastic simulations for computing the uncertain functions.

The first type of uncertain function is

$$U_1 : (\boldsymbol{x}, \boldsymbol{y}) \to \boldsymbol{E}\left[F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi})\right]. \tag{16}$$

In order to compute it, we design a stochastic simulation as follows,

Step 1. Set $U_1 = 0$

Step 2. Generate $\boldsymbol{\omega}$ from Ω according to the probability measure Pr.

Step 3. Compute the function value $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\omega})$ and denote it by c.

Step 4. $U_1 \leftarrow U_1 + c$.

Step 5. Repeat the second to fourth step for M times, where M is a sufficiently large number.

Step 6. return $U_1 \leftarrow U_1/M$

The second type of uncertain function is

$$\begin{aligned} U_2 : (\boldsymbol{x}, \boldsymbol{y}) \to \max\big\{\overline{F} \;\big|\; \\ \Pr\left\{F(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi}) \geq \overline{F}\right\} \geq \alpha\big\}. \end{aligned} \tag{17}$$

A procedure of stochastic simulation is given as follows.

Step 1. Generate $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \cdots, \boldsymbol{\omega}_M$ from Ω according to the probability measure Pr, where M is a sufficiently large number.

Step 2. For each $\boldsymbol{\omega}_k$, compute the function value $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\omega}_k)$ and denote it by c_k.

Step 3. Set M' as the integer part of αM

Step 4. Return the M'-th least element in $\{c_1, c_2, \cdots, c_M\}$.

The third type of uncertain function is

$$U_3 : (\boldsymbol{x}, \boldsymbol{y}) \to \Pr\left\{F(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi}) \geq \overline{F}\right\}. \tag{18}$$

We also design a stochastic simulation as follows.

Step 1. Set $M' = 0$

Step 2. Generate $\boldsymbol{\omega}$ from Ω according to the probability measure Pr.

Step 3. Compute the function value $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\omega})$ and denote it by c.

Step 4. If $c < \overline{F}$, set $M' \to M' + 1$.

Step 5. Repeat the second to fourth step for M times, where M is a sufficiently large number.

Step 6. return M'/M

Although stochastic simulations are able to compute the uncertain functions, they are a time-consuming process. In order to speed up the solution process, we need relatively simple functions to approximate the uncertain functions. Multilayer feedforward NN, which was initially designed by Minsky and Papert [49], is essentially a nonlinear mapping from the input space to the output space. It has the high speed of operation after they are trained. It has been shown that the multilayer feedforward NNs with an arbitrary number of hidden neurons are universal approximators for continuous functions (Cybenko [11], Hornik et al [27]). For our purpose, multilayer feedforward NNs are used to approximate uncertain functions, and then are substituted for the work of simulation in the solution process. For detailed discussion on uncertain function approximation, the reader may consult Chapter 3 in the book [33] by Liu.

In order to train NNs to approximate uncertain functions, we should first determine a region Θ on which to approximate the uncertain function $U(\boldsymbol{x}, \boldsymbol{y})$. The region Θ may be a bit too large so that it has simple forms like multi-dimensional hypercube. Then the computer can easily sample points $(\boldsymbol{x}, \boldsymbol{y})$ from the hypercube, where the vector $\boldsymbol{y}$ is not necessarily the Nash equilibrium of the followers with respect to $\boldsymbol{x}$. In order to get a set of input-output data, we designed a computing procedure as follows.

Step 1. Set $k = 1$.

Step 1. Generate a random point $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ from the hypercube Θ.

Step 2. Compute the function value by stochastic simulation and denote it by z_k.

Step 7. Repeat Step the first to sixth step for N times.

Step 7. Return the set of input-output data $\left\{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, z^{(k)}) \mid k = 1, 2, \cdots, N\right\}$.

Now we hope to train a feedforward NN to approximate the function $U(\boldsymbol{x}, \boldsymbol{y})$. Given the number of neurons and architecture, then the network weights may be arranged into a vector $\boldsymbol{w}$. Thus the output of mapping implemented by the NN may be characterized by $U(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w})$. A training process is to minimize the error function

$$Err(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{N} |U(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{w}) - z^{(k)}|^2 \tag{19}$$

so as to find a weight vector $\boldsymbol{w}$ that provides the best possible approximation of $U(\boldsymbol{x}, \boldsymbol{y})$. Sometimes, we are interested in the average error defined as

$$Err(\boldsymbol{w}) = \frac{1}{N}\sum_{k=1}^{N} |U(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{w}) - z^{(k)}|. \tag{20}$$

Backpropagation algorithm is the original learning algorithm for multilayer feedforward NN. It is essentially a gradient descent minimization method. Here we use the backpropagation algorithm for the NN with one hidden layer. In order to speed up the learning process, we use an improved error function

$$\begin{aligned} E_k = \tfrac{1}{2}\big[&\lambda(z^{(k)} - U(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{w}))^2 \\ &+(1-\lambda)\Phi(z^{(k)} - U(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{w}))^2\big] \end{aligned} \tag{21}$$

where $\Phi(x) = \ln(\cos(\beta x))/\beta$, $\lambda = \exp(-\mu/E^2)$ is an adaptive parameter, μ and β are constant numbers between 0 and 1, for example, $\mu = 1$, $\beta = 4/3$.

Step 1. Initialize $\boldsymbol{w}$, and set $\mu = 1$, $\beta = 4/3$, $\alpha = 0.05$, $\eta = 0.01$, $E_0 = 0.05$, $\lambda = 1$, and $k = 0$.

Step 2. $k \leftarrow k + 1$.

Step 3. Adjust the weights $\boldsymbol{w}$.

Step 4. Calculate the error E_k

Step 5. If $k < N$, go to the second step.

Step 6. Set $E = \sum_{k=1}^{N} E_k$.

Step 7. If $E > E_0$, then set $k = 0$, $\lambda = \exp(-\mu/E^2)$ and go to the third step.

Step 8. End.

For detailed exposition to backpropagation algorithm, the reader may consult the books [58].

3.2 Computing Nash equilibrium

Define symbols

$$\boldsymbol{y}_{-i} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_{i-1}, \boldsymbol{y}_{i+1}, \cdots, \boldsymbol{y}_m),\ i = 1, 2, \cdots, m.$$

For any decision $\boldsymbol{x}$ revealed by the leader, if the ith follower knows the strategies $\boldsymbol{y}_{-i}$ of other followers, then the optimal reaction of the ith follower is represented by a mapping $\boldsymbol{y}_i = r_i(\boldsymbol{y}_{-i})$, which should solve the subproblem

$$\begin{cases} \max\limits_{\boldsymbol{y}_i} \boldsymbol{E}\left[f_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi})\right] \\ \text{subject to:} \\ \quad \boldsymbol{E}\left[g_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi})\right] \le 0. \end{cases} \tag{22}$$

In order to search for the Stackelberg-Nash equilibrium of an EVMLP, we should first compute the Nash equilibrium with respect to any decision revealed by the leader. It is clear that the Nash equilibrium of the m followers will be the solution $(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m)$ of the system of equations

$$\boldsymbol{y}_i = r_i(\boldsymbol{y}_{-i}), \quad i = 1, 2, \cdots, m. \tag{23}$$

In other words, we should find a fixed point of the vector-valued function $(r_1, r_2, \cdots, r_m)$. In order to solve the system of equations (23), we should design some efficient algorithms. The argument breaks down into three cases.

Case I. If we have explicit expressions of all functions r_i, $i = 1, 2, \cdots, m$, then we might get an analytic solution to the system (23). Unfortunately, it is almost impossible to do this in practice.

Case II. In many cases, no analytic solution of (23) can be obtained. Thus the system of equations (23) might be solved by some iterative method that generates a sequence of points $\boldsymbol{y}^{(k)} = (\boldsymbol{y}_1^{(k)}, \boldsymbol{y}_2^{(k)}, \cdots, \boldsymbol{y}_m^{(k)})$, $k = 0, 1, 2, \cdots$ via the iteration formula

$$\boldsymbol{y}_i^{(k+1)} = r_i(\boldsymbol{y}_{-i}^{(k)}), \quad i = 1, 2, \cdots, m \tag{24}$$

where $\boldsymbol{y}_{-i}^{(k)} = (\boldsymbol{y}_1^{(k)}, \cdots, \boldsymbol{y}_{i-1}^{(k)}, \boldsymbol{y}_{i+1}^{(k)}, \cdots, \boldsymbol{y}_m^{(k)})$. When we solve some given problem on a computer, we might employ the iterative method. The procedure of the iterative method is given as follows.

Step 1. For a given control vector $\boldsymbol{x}$, initialize a feasible vector $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m)$.

Step 2. Calculate $\boldsymbol{y}_i' = r_i(\boldsymbol{y}_{-i})$.

Step 3. Calculate $\sum_{i=1}^{m} \|\boldsymbol{y}_i' - \boldsymbol{y}_i\|$.

Step 4. $\boldsymbol{y} \leftarrow \boldsymbol{y}'$.

Step 5. Repeat the second to fourth step until $\sum_{i=1}^{m} \|\boldsymbol{y}_i' - \boldsymbol{y}_i\| \le \varepsilon$ where ε is predetermined small number.

Step 6. Return the vector $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m)$ as the Nash equilibrium.

However, generally speaking, it is not easy to verify the conditions on the convergence of the iterative method for practical problems. If we indeed find a solution, then the problem is solved. Otherwise, we have to try other methods.

Case III. If the iterative method fails to find a fixed point, we may do it by solving the following minimization problem,

$$\begin{cases} \min R(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m) = \sum\limits_{i=1}^{m} \|\boldsymbol{y}_i - r_i(\boldsymbol{y}_{-i})\| \\ \text{subject to:} \\ \quad \boldsymbol{E}\left[g_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi})\right] \leq 0 \\ \quad i = 1, 2, \cdots, m. \end{cases} \tag{25}$$

If an array $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*)$ satisfies that

$$R(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*) = 0, \tag{26}$$

then $\boldsymbol{y}_i^* = r_i(\boldsymbol{y}_{-i}^*)$ for $i = 1, 2, \cdots, m$, and $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*)$ must be a solution of (23). In a numerical solution process, if an array $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*)$ satisfies that

$$R(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*) \leq \varepsilon, \tag{27}$$

where ε is a small positive number, we can also regard it as a solution of (23). That is, if the minimizing solution $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*)$ of the minimization problem (25) is such that equation (26) or inequality (27) hold, then $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \cdots, \boldsymbol{y}_m^*)$ is a solution of (23). Otherwise, the system of equations (23) might be considered inconsistent. In other words, there is no Nash equilibrium of followers in the given bilevel programming.

Now let us take attention to the optimization problem (25). On the one hand, the objective function $\sum\limits_{i=1}^{m} \|\boldsymbol{y}_i - r_i(\boldsymbol{y}_{-i})\|$ involves m mappings $r_i(\boldsymbol{y}_{-i})$, $i = 1, 2, \cdots, m$, which we cannot expect to have perfect properties such that the objective function have good mathematical properties. On the other hand, the feasible set defined by the system of constraints $\boldsymbol{E}\left[g_i(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_m, \boldsymbol{\xi})\right] \leq 0$, $i = 1, 2, \cdots, m$ may not be convex. For solve model (25), we have to employ some algorithms, which does not require specific mathematical analysis, but can provide good solutions of complex optimization problems.

GA is a stochastic search and optimization procedure motivated by natural principles and selection. GA starts from a population of candidate solutions rather than a single one, which increases the probability of finding a global optimal solution. Then GA improves the population step by step by biological evolutionary processes such as crossover and mutation, in which the search uses probabilistic transition rules, and can be guided towards regions of the search space with likely improvement, thus increasing the probability of finding a global optimal solution. In brief, the advantage of GAs is just able to obtain the global optimal solution fairly. In addition, GAs do not require the specific mathematical analysis of optimization problems, thus can be easily used to solve complex decision systems. Herein, we use genetic algorithms to search for the Nash equilibrium. For detailed exposition of GAs, interested readers may consult books [24][26][48].

For the optimization problem (23), each subproblem of the followers is a parametric optimization which may be solved by existing means of mathematical programming when the parameters are given. Since they are problem dependent, we do not discuss them here. Given a control vector $\boldsymbol{x}$, in order to solve the optimization problem (23) for the Nash equilibrium of the followers, we give the following GA procedure.

Step 0. Input a feasible control vector $\boldsymbol{x}$.

Step 1. generate *pop_size* chromosomes $\boldsymbol{y}^{(j)}$, $j = 1, 2, \cdots, pop\_size$ at random from the feasible set.

Step 2. Compute the system of equations $r_i(\boldsymbol{y}_{-i}), i = 1, 2, \cdots, m$ for each chromosome $\boldsymbol{y}^{(j)}$, and then the objective values $\sum\limits_{i=1}^{m} \|\boldsymbol{y}_i - r_i(\boldsymbol{y}_{-i})\|$, $j = 1, 2, \cdots, pop\_size$.

Step 3. Compute the fitness of each chromosome according to the objective values.

Step 4. Select the chromosomes by spinning the roulette wheel.

Step 5. Update the chromosomes by crossover and mutation operations.

Step 6. Repeat Steps 2–5 until the best chromosome satisfies inequality (27).

Step 7. Return the Nash equilibrium $\boldsymbol{y}(\boldsymbol{x}) = (\boldsymbol{y}_1(\boldsymbol{x}), \boldsymbol{y}_2(\boldsymbol{x}), \cdots, \boldsymbol{y}_m(\boldsymbol{x}))$.

3.3 HIA for Stackelberg-Nash Equilibrium

For any feasible control vector $\boldsymbol{x}$ revealed by the leader, it is reasonable for us to suppose that there always exists a Nash equilibrium with respect to it. Denote the Nash equilibrium with respect to $\boldsymbol{x}$ by $(\boldsymbol{y}_1(\boldsymbol{x}), \boldsymbol{y}_2(\boldsymbol{x}), \cdots, \boldsymbol{y}_m(\boldsymbol{x}))$, then the EVMLP (6) can be simplified as follows,

$$\begin{cases} \max\limits_{\boldsymbol{x}} \boldsymbol{E}\left[F(\boldsymbol{x}, \boldsymbol{y}_1(\boldsymbol{x}), \boldsymbol{y}_2(\boldsymbol{x}), \cdots, \boldsymbol{y}_m(\boldsymbol{x}), \boldsymbol{\xi})\right] \\ \text{subject to:} \\ \quad \boldsymbol{E}\left[G(\boldsymbol{x}, \boldsymbol{\xi})\right] \leq 0. \end{cases} \tag{28}$$

The uncertain objective function $\boldsymbol{E}\left[F(\boldsymbol{x}, \boldsymbol{y}_1(\boldsymbol{x}), \boldsymbol{y}_2(\boldsymbol{x}), \cdots, \boldsymbol{y}_m(\boldsymbol{x}), \boldsymbol{\xi})\right]$ involves not only uncertain parameters, but also a complex mapping $\boldsymbol{x} \to (\boldsymbol{y}_1(\boldsymbol{x}), \boldsymbol{y}_2(\boldsymbol{x}), \cdots, \boldsymbol{y}_m(\boldsymbol{x}))$, which makes the optimization problem difficult to solve. Therefore, GA is a good choice for solving such model for the Stackelberg-Nash equilibrium, although it is a relatively slow way.

Now we integrate stochastic simulations, NNs, and GAs to produce an HIA for solving general EVMLPs. In the HIA, input-output data of uncertain functions are first generated by stochastic simulation. Then NNs

are trained on these sample sets to approximate uncertain functions. After that, the trained NNs, and the iterative method (or GA) for Nash equilibrium are embedded into a GA so as to search for Stakelberg-Nash equilibrium efficiently. The procedure of the HIA is given as follows.

Step 1. Generate input-output data of uncertain functions.

Step 2. Train NNs by backpropagation algorithm.

Step 3. Initialize pop_size chromosomes $\boldsymbol{x}^{(i)}$, $i = 1, 2, \cdots, pop\_size$ randomly.

Step 4. Compute the Nash equilibrium with respect to each chromosome $\boldsymbol{x}^{(i)}$ by Algorithm 3 (or 4), and then the objective values $\boldsymbol{E}\left[F(\boldsymbol{x}^{(i)}, \boldsymbol{y}_1(\boldsymbol{x}^{(i)}), \boldsymbol{y}_2(\boldsymbol{x}^{(i)}), \cdots, \boldsymbol{y}_m(\boldsymbol{x}^{(i)}))\right]$ via the trained NNs, $i = 1, 2, \cdots, pop\_size$.

Step 5. Compute the fitness of each chromosome according to the objective values.

Step 6. Select the chromosomes by spinning the roulette wheel.

Step 7. Update the chromosomes by crossover and mutation operations.

Step 8. Repeat Steps 4–7 for a given number of cycles.

Step 9. Return the best chromosome as the optimal solution.

4 Numerical Examples

The computer code for the HIA to general EVMLPs with multiple followers has been written in C language. In order to illustrate its effectiveness, we provide two numerical examples performed on a personal computer.

Example. We consider a stochastic decentralized decision making problem in which there is one leader and three followers. Assume the control vector is $\boldsymbol{x} = (x_1, x_2)$, and the control vectors of the three followers are $\boldsymbol{y}_i = (y_{i1}, y_{i2})$, $i = 1, 2, 3$. Its EVMLP is given as follows.

$$
\begin{cases}
\max\limits_{x_1, x_2} \boldsymbol{E}\left[(x_1 + y_{11} + y_{21} + y_{31} + \xi_1)^2 \right. \\
\qquad \left. +(x_2 + y_{12} + y_{22} + y_{32} + \xi_2)^2\right] \\
\text{subject to:} \\
\quad x_1 + x_2 \le 10,\ x_1 \ge 0,\ x_2 \ge 0 \\
\quad \text{where } y_i = (y_{i1}, y_{i2})\ (i = 1, 2, 3) \text{ solve} \\
\quad \begin{cases}
\max\limits_{y_{11}, y_{12}} \boldsymbol{E}\left[\sqrt{x_1^2 + y_{11}^2 + 2y_{12}^2 + \eta_1^2}\right] \\
\text{subject to:} \\
\qquad y_{11} + 2y_{12} \le x_2,\ y_{11} \ge 0,\ y_{12} \ge 0
\end{cases} \\
\quad \begin{cases}
\max\limits_{y_{21}, y_{22}} \boldsymbol{E}\left[\sqrt{x_1^2 + y_{21}^2 + 2y_{22}^2 + \eta_2^2}\right] \\
\text{subject to:} \\
\qquad 2y_{21} + 3y_{22} \le x_2,\ y_{21} \ge 0,\ y_{22} \ge 0
\end{cases} \\
\quad \begin{cases}
\max\limits_{y_{31}, y_{32}} \boldsymbol{E}\left[\sqrt{x_1^2 + y_{31}^2 + 2y_{32}^2 + \eta_3^2}\right] \\
\text{subject to:} \\
\qquad 3y_{31} + 4y_{32} \le x_2,\ y_{31} \ge 0,\ y_{32} \ge 0,
\end{cases}
\end{cases}
$$

The stochastic parameters in the above model are given as follows:

$$
\begin{cases}
\xi_1 \sim \mathcal{N}(3, 1) \\
\xi_2 \sim \mathcal{N}(4, 1)
\end{cases}
\quad \text{and} \quad
\begin{cases}
\eta_1 \sim \mathcal{U}(0, 1) \\
\eta_2 \sim \mathcal{U}(1, 2) \\
\eta_3 \sim \mathcal{U}(2, 3).
\end{cases}
$$

In order to solve this problem, we generate input-output data for the uncertain functions

$$
\begin{aligned}
U_0 &: (\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3) \to \boldsymbol{E}\left[(x_1 + y_{11} + y_{21} + y_{31} + \xi_1)^2 \right. \\
&\qquad \left. +(x_2 + y_{12} + y_{22} + y_{32} + \xi_2)^2\right] \\
U_1 &: (\boldsymbol{x}, \boldsymbol{y}_1) \to \boldsymbol{E}\left[\sqrt{x_1^2 + y_{11}^2 + 2y_{12}^2 + \eta_1^2}\right] \\
U_2 &: (\boldsymbol{x}, \boldsymbol{y}_2) \to \boldsymbol{E}\left[\sqrt{x_1^2 + y_{21}^2 + 2y_{22}^2 + \eta_2^2}\right] \\
U_3 &: (\boldsymbol{x}, \boldsymbol{y}_3) \to \boldsymbol{E}\left[\sqrt{x_1^2 + y_{31}^2 + 2y_{32}^2 + \eta_3^2}\right].
\end{aligned}
$$

by stochastic simulation. Then we train four NNs to approximate the uncertain functions U_i, $i = 0, 1, 2, 3$, respectively. Table 1 gives the number of input-output data, input, hidden, and output neurons of the four NNs, and the error defined by (19). In order to further illustrate the accuracy of the trained NNs, three types of samples are used for testing. The first is to select 1000 input-output data from the train set randomly. The second is to sample 1000 new input-output data from the predetermined domain. Another 1000 input-output data is generated by sampling 1000 feasible control vector, computing out their corresponding Nash equilibrium, and then the function values by stochastic simulation. From the comparison results of the average errors of the three type of samples in Table 2, we can see that the maximum average error defined by (20) is less than 2.8%. So the NNs are good approximations of the four uncertain functions.

uncertain functions	input-output data	input neurons	hidden neurons	output neurons	sum_squred error
U_0	3000	8	10	1	2100
U_1	3000	4	7	1	7.54
U_2	3000	4	7	1	2.16
U_3	3000	4	7	1	1.96

Table 1: Parameters in the four NNs of example 1

uncertain functions	number of samples	average error_1 (%)	average error_2 (%)	average error_3 (%)
U_0	1000	0.1855	0.3735	2.4455
U_1	1000	0.9681	1.8205	1.3860
U_2	1000	0.5692	1.1595	1.7615
U_3	1000	0.3858	0.7639	2.7921

Table 2: Comparison errors of three type of samples of example 1

In the GA procedure for the Stackelberg-Nash equilibrium, there are some parameters such as the population size (pop_size), the probability of crossover (P_c), the probability of mutation (P_m). In Table 3, we compare solutions when different parameters are taken with the same generations as a stopping rule. It appears that all the optimal solution and the optimum differ little from each other. In order to account for it, we present a parameter, called percent error, i.e. (actual value - optimal value)/ optimal value $\times$ 100%, where the optimal value is the minimal one of all the six maximum obtained above. The last column named by "error" in Table 3 is just this parameter. It follows from Table 2 that the percent error does not exceed 0.2% when different parameters are selected, which implies that the HIA is robust to the parameter settings and effective to solve model (6).

According to the computing result in Table 3, we can see that x_1 is very close to 0, and that x_2 is very close to 10. A direct computation of the control vector $(x_1, x_2) = (0, 10)$ shows that the optimal return of the leader is 652.77, the Nash equilibrium of the followers is

$$(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \boldsymbol{y}_3^*) = (9.9999, 0, 4.9999, 0, 3.3333, 0),$$

and the optimal return of the three followers are 9.9876, 5.5016, and 4.1140, respectively.

5 Concluding Remarks

In this paper, two classes of stochastic multilevel programming models were first proposed for dealing with decentralized decision making problem in stochastic environment. In order to solve general the proposed models for the Stackelberg-Nash equilibrium, an HIA was designed by integrating stochastic simulation, NNs, GA, and iterative method. One numerical example showed that the HIA is robust and effective.

Acknowledgments

This work was supported by National Natural Science Foundation of China Grant No.69804006, and Sino-French Joint Laboratory for Research in Computer Science, Control and Applied Mathematics (LIAMA).

References

[1] M.A. Amouzegar, K. Moshirvaziri, Determining optimal pollution control policies: An application of bilevel programming, *European Journal of Operational Research*, 119(1999), 100–120.

[2] G. Anandalingam, A mathematical programming model of decentralized multilevel system, *Journal of the Operational Research Society*, 39(1988), 1021–1033.

[3] J.F. Bard, and J.E. Falk, An explicit solution to the multi-level programming problem, *Computer & Operations Research*, 9(1982), 77–100.

[4] J.F. Bard, An algorithm for solving general bilevel programming program, *Mathematics of Operations research*, 8(1983), 260-272.

[5] J.F. Bard, An investigation of the linear three level programming problem, *IEEE Transactions on Systems, Man, Cybernetics*, SMC-14(1984), 711–717.

[6] J.F. Bard, Convex two-level optimization, *Mathematical Programming*, 40(1988), 15–27.

[7] J.F. Bard, and J.T. Moore, A branch and bound algorithm for the bilevel programming problem, *SIAM J. Sci. Statist. Coput.*, 11(1990), 281–292.

[8] J.F. Bard, J. Plummer, and J.C. Sourie, A bilevel programming approach to determining tax credits for biofuel production, *European Journal of Operational Research*, 120(2000), 30–46.

pop_size	P_c	P_m	gen	optimal solution	return	error(%)
30	0.3	0.1	400	$(0.0009, 9.9982)$	652.25	0.10
30	0.3	0.2	400	$(0.0003, 9.9981)$	652.65	0.18
30	0.2	0.1	400	$(0.0003, 9.9999)$	652.75	0.20
50	0.3	0.2	400	$(0.0002, 9.9997)$	652.75	0.20
50	0.3	0.1	400	$(0.0010, 9.9989)$	652.70	0.19
50	0.2	0.1	400	$(0.0212, 9.9787)$	651.42	0.00

Table 3: Comparison solutions of example 1

[9] O. Ben-Ayed, D.E. Boyce, and C.E. Blair, A general bilevel programming formulation of the network design problem, *Transportation Research*, B22(1988), 259–265.

[10] O. Ben-Ayed, and C.E. Blair, Computational difficulties of bilevel linear programming, *Operations Research*, 38(1990), 556–560.

[11] G. Cybenko, Approximations by superpositions of a sigmoidal function, *Mathematics of Control, Signals and Systems*, 2(1989), 183–192.

[12] W.F. Bialas, and M.H. Karwan, On two-level optimization, *IEEE Trans. Control*, AC-27(1982), 211–214.

[13] W.F. Bialas, and M.H. Karwan, Two-level linear programming, *Management Sci.*, 30(1984), 1004–1020.

[14] J. Bracken and J.M. McGill, Mathematical programs with optimization problems in the constraints, *Operations Research*, 21(1973), 37–44.

[15] J. Bracken and J.M. McGill, A method for solving Mathematical programs with nonlinear problems in the constraints, *Operations Research*, 22(1974), 1097–1101.

[16] W. Candler, W. Fortuny-Amat, and B. McCarl, The potential role of multi-level programming in agricultural economics, *American Journal of Agricultural Economics*, 63(1981), 521–531.

[17] W. Candler, and R. Townaley, A linear two-level programming problem, *Computer and Operations Research*, 9(1982), 59–76.

[18] Charnes A. and Cooper W.W., Chance-constrained programming, Management Science, Vol.6 (1959), 73–79.

[19] P.A. Clark and A. Westerberg, Bilevel programming for chemical process design—I. Fundamentals and algorithms, *Computer and Chemical Engineering*, 14(1990), 87–97.

[20] A.H. DeSilva, and G.P. McCormick, Implicitly defined optimization problems, *Annals Operations Research*, 34(1992), 107–124.

[21] T.A. Edmunds, and J.F. Bard, Algorithms for nonlinear bilevel mathematical programming programs, *IEEE Transactions on Systems, Man, and Cebernetics*, 21(1991), 97–113.

[22] J.E. Falk, and J. Liu, On bilevel programming, Part I: general nonlinear cases, *Mathematic Programming*, 70(1995) 47–72.

[23] J. Fortuny-Amat, and B. McCarl, A representation and economic interpretation of a two-level programming problem, *Journal of the Operational Research Society*, 32(1981), 783–792.

[24] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley, MA, 1989.

[25] P. Hansen, B. Jaumard, and G. Savard, New branch and bound rules for the linear bilevel programming problem, *SIAM J. Sci. Statist. Coput.*, 13(1992), 1194–1217.

[26] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[27] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, 2(1989), 359–366.

[28] J.J. Júdice, and A.M. Faustino, A sequential LCP method for bilevel linear programming, *Annals Operations Research*, 34(1992), 89–106.

[29] C.K. Kolstad, and L.S. Lasdon, Derivative evaluation and computational experience with large bilevel mathematical programs, *Journal of Optimization Theory and Applications*, 65(1990), 485–499.

[30] Y.J. Lai, Hierachical optimization: A satisfactory solution, *Fuzzy Sets and Systems*, 77(1996), 321–335.

[31] E.S. Lee, and H.S. Shih, *Fuzzy and Multi-level Decision Making*, Springer-Verlag, London, 2001.

[32] B. Liu, Stackelberg-Nash equilibrium for multilevel programming with multiple follower using genetic algorithm, *Comput. Math. Appl.*, 36(1998), 79–89.

[33] B. Liu, *Theory and Practice of Uncertain Programming*, Physica-Verlag, Heidelberg, 2002.

[34] B. Liu, Dependent-chance goal programming and its genetic algorithm based approach, *Mathematical and Computer Modelling*, 24(1996), 43–52.

[35] B. Liu, Dependent-chance programming: A class of stochastic programming, *Computers & Mathematics with Applications*, 34(1997), 89–104.

[36] B. Liu, and K. Iwamura, Modelling stochastic decision systems using dependent-chance programming, *European Journal of Operational Research*, 101(1997), 193–203.

[37] B. Liu, and K. Iwamura, Chance constrained programming with fuzzy parameters, *Fuzzy Sets and Systems*, 94(1998), 227–237.

[38] B. Liu, Outline of uncertain programming, *Proceedings of the Third International Symposium on Operations Research and Applications*, p. 480–489, Kunming, China, August 19–22, 1998.

[39] B. Liu, Uncertain programming: Modelling, evolutionary computation and applications, *Proceedings of Fourth Joint Conference on Information Sciences*, Vol. 3, p. 433–437, North Carolina, October 23–28, 1998,

[40] B. Liu, Minimax chance constrained programming models for fuzzy decision systems, *Information Sciences*, 112(1998), 25–38.

[41] B. Liu, and R. Zhao, *Stochastic Programming and Fuzzy Programming*, Tsinghua University Press, Beijing, 1998.

[42] B. Liu, Dependent-chance programming with fuzzy decisions, *IEEE Transactions on Fuzzy Systems*, 7(1999), 354–360.

[43] B. Liu, and A.O. Esogbue, *Decision Criteria and Optimal Inventory Processes*, Kluwer Academic Publishers, Boston, 1999.

[44] B. Liu, *Uncertain Programming*, Wiley, New York, 1999.

[45] B. Liu, Uncertain programming: Optimization theory in uncertain environments, in *Proceedings of the Ninth IEEE International Conference on Fuzzy Systems*, p. 941–944, San Antonio, Texas, USA, May 7–10, 2000.

[46] B. Liu, Dependent-chance programming in fuzzy environments, *Fuzzy Sets and Systems*, 109(2000), 97–106.

[47] B. Liu, Uncertain programming: A unifying optimization theory in various uncertain environments, *Applied Mathematics and Computation*, 120(2001), 227–234.

[48] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs,* 3rd ed., Springer-Verlag, Berlin, 1996.

[49] M. Minsky, and S. Papert, *Perceptrons*, Mit Press, Cambridge, MA, 1969.

[50] M. Patriksson, L. Wynter, Stochastic mathematicl programs with equilibrium constraints, *Operations research letters*, 25(1999), 159–167.

[51] M. Sakawa, I. Nishizaki, and Y. Umura, Interactive fuzzy programming for multi-level linear programming problems with fuzzy parameters, *Fuzzy Sets and Systems*, 109(2000), 3–19.

[52] K.H. Sahin, and A.R. Ciric, A dual temperature simulated annealing approach for solving bilevel programming problems, *Computers and Chemical Engineering*, 23(1998) 11–25.

[53] G. Savard, and J. Gauvin, The steepest descent direction for nonlinear bilevel programming problem, *Operations Research Letters*, 15(1994), 265–272.

[54] H.S. Shih, Y.J. Lai, and E.S. Lee, Fuzzy approach for multi-level programming problems, *Computer & Operations Research*, 23(1996), 73–91.

[55] K. Shimizu, and E. Aiyoshi, A new computational method for Stackelberg and min-max problems by use a penalty method, *IEEE Transactions on Automatic Control*, AC-26(1981), 460–466.

[56] S. Suh, and T. Kim, Solving nonlinear bilevel programming models of the equilibrium network desing problem: A comparative review, *Annals Operations Research*, 34(1992), 203–218.

[57] U.P. Wen, and W. Bialas, The hybrid algorithm for solving three-level linear programming problem, *Computer and Operations Research*, 13(1986), 367–377.

[58] H. White, et al., *Neural Networks Approximation and Learning Theory*, Blackwell Publishers, USA, 1992.

[59] H. Yang, and M.G.H. Bell, Transport bilevel programming problems: recent methodological advances, *Transportation Research*, Part B: 35(2001), 1-4.

[60] Y. Yin, Genetic-algorithms-based approach for bilevel programming models, *Journal of Transportation Engineering*, 126(2000), 115–120.