Journal of the Operations Research
Society of Japan
Vol. 42, No. 2, June 1999

DEPENDENT-CHANCE INTEGER PROGRAMMING APPLIED TO CAPITAL BUDGETING

Kakuzo Iwamura
Josai University

Baoding Liu
Tsinghua University

(Received April 21, 1997; Revised May 2, 1998)

Abstract This paper attempts to model capital budgeting problems by a new technique of dependent-chance integer programming as well as dependent-chance multiobjective programming and goal programming. Some examples are provided to illustrate the potential applications in the area of capital budgeting. A stochastic simulation based genetic algorithm is also designed to solve both chance constrained integer programming and dependent-chance integer programming models.

1. Introduction

The original capital budgeting is concerned with maximizing the total net profit subject to budget constraint by selecting appropriate combination of projects. With the requirement of considering uncertainty of future demand and multiple conflicting goals, chance constrained integer programming was employed to model capital budgeting by Keown and Martin[8] in the working capital management and by Keown and Taylor[9] in the production area. De *et al.*[2] extended chance constrained goal programming to the zero-one case and applied it to capital budgeting problems.

Chance constrained programming models can be converted into deterministic equivalents when the stochastic variables are normally distributed. However, it is usually difficult to transform them to deterministic forms if the distributions of stochastic variables belong to other classes or the constraints are irregular. In order to solve general chance constrained programming models (continuous case), Iwamura and Liu[5] proposed a stochastic simulation based genetic algorithm in which the stochastic simulation is employed to check the chance constraints. On the other hand, Liu and Iwamura[15] developed a technique of chance constrained programming with fuzzy parameters rather than stochastic parameters and designed a fuzzy simulation based genetic algorithm for solving such a kind of models. The capital budgeting problems in fuzzy environments have been discussed by the new approach in Iwamura and Liu[6]. In fact, a unifying treatment of uncertain parameters leads to that fuzzy capital budgeting models are identical with stochastic models except for the facts that stochastic parameters and Pr have been replaced by fuzzy parameters and Pos, respectively, where Pos represents the possibility of a fuzzy event.

Dependent-chance programming as well as dependent-chance multiobjective programming and dependent-chance goal programming[11, 12, 13, 14] are a new type of stochastic models. This paper will extend dependent-chance programming to integer case, and show some potential applications of this kind of models in the area of capital budgeting. A stochastic simulation based genetic algorithm is also designed to solve both chance constrained integer programmig and dependent-chance integer programming models. Finally,

we illustrate the effectiveness of stochastic simulation based genetic algorithm by some numerical examples.

2. Capital Budgeting

We assume that a company has the opportunity to install the machines in a plant. Suppose that there are n types of machines available. If we use x_i to denote the numbers of type i machines selected, $i = 1, 2, \cdots, n$, respectively, then x_i's are nonnegative integers, i.e., $x_i = 0, 1, 2, 3, \cdots$. Let a_i be the level of funds that needs to be allocated to type i machine and a be the total capital available for distribution, then we should have

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq a, \tag{1}$$

i.e., the total capital used for buying machines can not exceed the amount available.

The other constraint is the maximum space availability limitation for the machines. Suppose that b_i are the spaces used by type i machine, $i = 1, 2, \cdots, n$, respectively. If the total available space is b, then we have the following constraint,

$$b_1x_1 + b_2x_2 + \cdots + b_nx_n \leq b. \tag{2}$$

In addition, we suppose that different machines produce different products. Let c_i be the production capacity of the type i machine for product i, then the total amounts of products i are c_ix_i, $i = 1, 2, \cdots, n$, respectively. We also assume that the future demands for products i are d_i, $i = 1, 2, \cdots, n$. Since the production should satisfy the future demands, we have

$$c_ix_i \geq d_i, \ i = 1, 2, \cdots, n. \tag{3}$$

If p_i are the net profits per type i machines, $i = 1, 2, \cdots, n$, respectively, then the total net profit is $p_1x_1 + p_2x_2 + \cdots + p_nx_n$. Our objective is to maximize the total net profit, i.e.,

$$\max p_1x_1 + p_2x_2 + \cdots + p_nx_n. \tag{4}$$

Thus we have a deterministic model for capital budgeting based on integer programming,

$$\begin{cases} \max p_1x_1 + p_2x_2 + \cdots + p_nx_n \\ \text{subject to:} \\ \quad a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq a \\ \quad b_1x_1 + b_2x_2 + \cdots + b_nx_n \leq b \\ \quad c_ix_i \geq d_i, \quad i = 1, 2, \cdots, n \\ \quad x_i, \ i = 1, 2, \cdots, n, \quad \text{nonnegative integers.} \end{cases} \tag{5}$$

Certainly, real capital budgeting problems are much more complex than the above-mentioned model. However, this is enough for illustrating the new technique of dependent-chance integer programming.

3. Chance Constrained Programming Models

Chance constrained programming was pioneered by Charnes and Cooper [1] as a means of handling uncertainty by specifying a confidence level at which it is desired that the uncertain constraint holds. Here let us model the capital budgeting problems by chance constrained programming based on the works[2, 8, 9].

In practice, the production capacities c_i and future demands d_i are not necessarily deterministic. Here we suppose that they are stochastic variables. Let ϕ_i and ψ_i denote the

probability density functions of c_i and d_i, $i = 1, 2, \cdots, n$, respectively. Then the constraints $c_i x_i \geq d_i$ are uncertain. Suppose that the manager gives α_i as the probabilities of meeting the demands of products i, $i = 1, 2, \cdots, n$, respectively. Then we have the following chance constraints,

$$\Pr\{c_i x_i \geq d_i\} \geq \alpha_i, \quad i = 1, 2, \cdots, n \tag{6}$$

where $\Pr\{\cdot\}$ denotes the probability of the event $\{\cdot\}$. Thus, a chance constrained integer programming is immediately formulated as follows,

$$\begin{cases} \max p_1 x_1 + p_2 x_2 + \cdots + p_n x_n \\ \text{subject to:} \\ \quad a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq a \\ \quad b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \leq b \\ \quad \Pr\{c_i x_i \geq d_i\} \geq \alpha_i, \quad i = 1, 2, \cdots, n \\ \quad x_i, \ i = 1, 2, \cdots, n, \quad \text{nonnegative integers.} \end{cases} \tag{7}$$

Now we suppose that the following target levels and priority structure have been set by the manager:

Strict constraint: The maximum space availability limitation for machines, i.e., $b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \leq b$.

Priority 1: Budget goal (the total capital does not exceed the total available capital a as much as possible), i.e.,

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n + d_1^- - d_1^+ = a$$

where d_1^+ will be minimized.

Priority 2: Demand goal (the probability of meeting the demands achieves the given level α as much as possible), i.e.,

$$\Pr\{c_i x_i \geq d_i, \ i = 1, 2, \cdots, n\} + d_2^- - d_2^+ = \alpha$$

where d_2^- will be minimized.

Priority 3: Profit goal (the total profit is not less than the given level p as much as possible), i.e.,

$$p_1 x_1 + p_2 x_2 + \cdots + p_n x_n + d_3^- - d_3^+ = p$$

where d_3^- will be minimized.

Then we have a chance constrained integer goal programming as follows,

$$\begin{cases} \text{lexmin}\{d_1^+, d_2^-, d_3^-\} \\ \text{subject to:} \\ \quad a_1 x_1 + a_2 x_2 + \cdots + a_n x_n + d_1^- - d_1^+ = a \\ \quad \Pr\{c_i x_i \geq d_i, \ i = 1, 2, \cdots, n\} + d_2^- - d_2^+ = \alpha \\ \quad p_1 x_1 + p_2 x_2 + \cdots + p_n x_n + d_3^- - d_3^+ = p \\ \quad b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \leq b \\ \quad d_i^-, d_i^+ \geq 0, \quad i = 1, 2, 3 \\ \quad x_i, \ i = 1, 2, \cdots, n, \quad \text{nonnegative integers.} \end{cases} \tag{8}$$

where lexmin represents lexicographical minimization.

4. Dependent-Chance Programming Models

Roughly speaking, *dependent-chance programming* [11, 12, 13, 14] is related to optimizing some chance functions of events defined on a so-called stochastic set in a complex uncertain decision system. In deterministic models as well as expected value model and chance constrained programming, the feasible set is essentially assumed to be deterministic after the real problem is modeled, i.e., an optimal solution is always given regardless of whether or not it can be performed in practice. However, the given solution may not be performed if the realization of uncertain parameter goes to bad cases. So the dependent-chance programming model never assumes that the feasible set is deterministic. Although a deterministic solution is given by the dependent-chance programming model, this solution is only requested to be performed as much as possible. This special feature of dependent-chance programming is very different from the existing stochastic programming techniques else.

In order to understand general dependent-chance programming correctly, we strongly reminds the potential readers that the feasible set of dependent-chance programming is stochastic. However, from the new applied area presented in this section, we can extend our object to the case in which the feasible set is indeed stochastic. This fact leads to that there is a common part between chance constrained programming and dependent-chance programming.

The simplest model based on dependent-chance integer programming for capital budgeting is

$$\begin{cases} \max \Pr\{c_i x_i \geq d_i,\ i = 1, 2, \cdots, n\} \\ \text{subject to:} \\ \quad a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq a \\ \quad b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \leq b \\ \quad x_i,\ i = 1, 2, \cdots, n, \quad \text{nonnegative integers} \end{cases} \tag{9}$$

which is concerned with finding the most safe solution, i.e., the one with the maximum probability that the productions meet the demands.

Sometimes, we may wish to maximize the reliability levels of all kinds of demands separately, then the problem may be formulated as the following dependent-chance multiobjective programming model,

$$\begin{cases} \max[\Pr\{c_1 x_1 \geq d_1\}, \Pr\{c_2 x_2 \geq d_2\}, \cdots, \Pr\{c_n x_n \geq d_n\}] \\ \text{subject to:} \\ \quad a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq a \\ \quad b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \leq b \\ \quad x_i,\ i = 1, 2, \cdots, n, \quad \text{nonnegative integers.} \end{cases} \tag{10}$$

In order to balance the multiple conflicting objectives, capital budgeting may be modeled by the following dependent-chance goal programming according to the target levels and priority structure set by the decision maker,

$$\begin{cases} \min \sum_{j=1}^{l} P_j \sum_{i=1}^{n} (u_{ij} d_i^+ + v_{ij} d_i^-) \\ \text{subject to:} \\ \quad \Pr\{c_i x_i \geq d_i\} + d_i^- - d_i^+ = \alpha_i,\ i = 1, 2, \cdots, n \\ \quad a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq a \\ \quad b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \leq b \\ \quad d_i^-, d_i^+ \geq 0, \quad i = 1, 2, \cdots, n \\ \quad x_i,\ i = 1, 2, \cdots, n, \quad \text{nonnegative integers.} \end{cases} \tag{11}$$

where P_j = the preemptive priority factor which expresses the relative importance of various goals, $P_j \gg P_{j+1}$, for all j, u_{ij} = weighting factor corresponding to positive deviation for goal i with priority j assigned, v_{ij} = weighting factor corresponding to negative deviation for goal i with priority j assigned, d_i^+ = positive deviation from the target of goal i, d_i^- = negative deviation from the target of goal i, α_i = the target value according to goal i, l = number of priorities.

We should mention that the dependent-chance integer goal programming (11) is essentially identical with chance constrained integer goal programming under our assumptions. This means that there is a common part between dependent-chance programming and chance constrained programming, i.e., a dependent-chance goal programming degenerates into a chance constrained goal programming when the stochastic feasible set degenerates into a deterministic feasible set. However, if some of the parameters a, b, a_i and b_i, $i = 1, 2, \cdots, n$ in (11) are assumed stochastic, then the model is indeed different from the chance constrained programming because the feasible set determined by

$$\begin{cases} a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq a \\ b_1x_1 + b_2x_2 + \cdots + b_nx_n \leq b \\ x_i,\ i = 1, 2, \cdots, n, \quad \text{nonnegative integers} \end{cases}$$

is a stochastic set.

5. Stochastic Simulation Based Genetic Algorithm

Genetic algorithms are a stochastic search method for optimization problems based on the mechanics of natural selection and natural genetics, i.e., the principle of evolution-survival of the fittest. Genetic algorithms have demonstrated considerable success in providing good solutions to many complex optimization problems and received more and more attentions during the past three decades. When the objective functions to be optimized in the optimization problems are multimodal or the search spaces are particularly irregular, algorithms need to be highly robust in order to avoid getting stuck at local optimal solution. The advantage of genetic algorithms is just to obtain the global optimal solution fairly. Genetic algorithms (including evolution programs and evolution strategies) have been well discussed and summarized by numerous literature, such as Goldberg[4], Michalewicz[16] and Fogel[3], and applied to a wide variety of problems, such as optimal control problems, transportation problems, traveling salesman problems, drawing graphs, scheduling, group technology, facility layout and location, as well as pattern recognition.

In this section, we design a stochastic simulation based genetic algorithm for solving both chance constrained integer programming and dependent-chance integer programming models (including multiobjective programming and goal programming). We will discuss representation structure, handling constraints, initialization process, evaluation function, selection process, crossover operation and mutation operation in turn.

5.1 Representation structure

There are two ways to represent a solution of an optimization problem, binary vector and floating vector. We can use a binary vector as a chromosome to represent real value of decision variable, where the length of the vector depends on the required precision. The necessity for binary codings has received considerable criticism.

An alternative approach to represent a solution is the floating point implementation in which each chromosome vector is coded as a vector of floating numbers, of the same length as the solution vector. Here we use a vector $V = (x_1, x_2, \cdots, x_n)$ as a chromosome

to represent a solution to the optimization problem, where n is the dimension. Certainly, in capital budgeting problems all variables x_i's will be confined to integer values. In fact, if we code the algorithm by C language, then we can ensure that the vector V is integer by defining it as an integer array.

5.2 Initialization process

We define an integer *pop_size* as the number of chromosomes and initialize *pop_size* chromosomes randomly. Usually, it is difficult for complex optimization problems to produce feasible chromosome explicitly. So we employ one of the following two ways as the initialization process, depending on what kind of information the decision maker can give.

First case is that the decision maker can determine an interior point, denoted by V_0, in the constraint set. This is very possible for real decision problem. We also need to define a large positive number M which ensures that all the genetic operators are probabilistically complete for the feasible solutions. This number M is used for not only initialization process but also mutation operation. The *pop_size* chromosomes will be produced as follows. We randomly select a direction d in $\Re^n$ and define a chromosome V as $V_0 + M \cdot d$ if it is feasible for the inequality constraints, otherwise, we set M by a random number between 0 and M until $V_0 + M \cdot d$ is feasible. If a new feasible chromosome is not obtained in a given number of times, then we take V_0 as the chromosome. Repeat this process *pop_size* times and produce *pop_size* initial feasible solutions $V_1, V_2, \cdots, V_{pop\_size}$.

If the decision maker fails to give such an interior point, then he can predetermine a region which contains the feasible set. Usually, this region will be designed to have nice sharp, for example, an n-dimensional hypercube, because the computer can easily generate points from a hypercube. We generate a random point from the hypercube and check the feasibility of this point. If it is feasible, then it will be accepted as a chromosome. If not, then re-generate a point from the hypercube randomly until a feasible one is obtained. Repeat the above process *pop_size* times, we can make *pop_size* initial feasible chromosomes $V_1, V_2, \cdots, V_{pop\_size}$.

5.3 Evaluation function

Evaluation function, denoted by $eval(V)$, is to assign a probability of reproduction to each chromosome V 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 by using *roulette wheel selection.*

Let $V_1, V_2, \cdots, V_{pop\_size}$ be the *pop_size* chromosomes at the current generation. One well-known method is based on allocation of reproductive trials according to rank rather than actual objective values. No matter what kind of mathematical programming (single-objective or multiobjective), it is reasonable to assume that the user can give an order relationship among the *pop_size* chromosomes $V_1, V_2, \cdots, V_{pop\_size}$ such that the *pop_size* chromosomes can be rearranged from good to bad, i.e., the better the chromosome is, the smaller ordinal number it has. For example, we can compute all objectives at every priority level for a given goal programming by employing the technique of stochastic simulation [5, 11]. Then, we have the following order relationship for the chromosomes: for any two chromosomes, if the higher-priority objectives are equal, then, in the current priority level, the one with minimal objective value is better. This relationship is an order on the feasible set and can rearrange these chromosomos from good to bad. If two different chromosomes have the same objective values, then we rearrange these chromosomes randomly.

Now let a parameter $a \in (0, 1)$ in the genetic system be given, then we can define the

so-called *rank-based evaluation function* as follows,

$$eval(V_i) = a(1-a)^{i-1}, \quad i = 1, 2, \cdots, pop\_size. \tag{12}$$

We mention that $i = 1$ means the best individual, $i = pop\_size$ the worst individual.

5.4 Selection process

The selection process is based on spinning the roulette wheel pop_size times, each time we select a single chromosome for a new population in the following way:

Step 1. Calculate the cumulative probability q_i for each chromosome V_i,

$$\begin{aligned} q_0 &= 0 \\ q_i &= \sum_{j=1}^{i} eval(V_j), \quad i = 1, 2, \cdots, pop\_size. \end{aligned} \tag{13}$$

Step 2. Generate a random real number r in $[0, q_{pop\_size}]$.

Step 3. Select the i-th chromosome V_i $(1 \leq i \leq pop\_size)$ such that $q_{i-1} < r \leq q_i$.

Step 4. Repeat steps 2 and 3 pop_size times and obtain pop_size copies of chromosomes.

5.5 Crossover operation

We define a parameter P_c of a genetic system as the probability of crossover . This probability gives us the expected number $P_c \cdot pop\_size$ of chromosomes which undergo the crossover operation.

In order to determine the parents for crossover operation, let us do the following process repeatedly from $i = 1$ to pop_size: generating a random real number r from the interval $[0, 1]$, the chromosome V_i is selected as a parent if $r < P_c$.

We denote the selected parents as $V_1', V_2', V_3', \cdots$ and divide them to the following pairs:

$$(V_1', V_2'),\ (V_3', V_4'),\ (V_5', V_6'), \cdots$$

Let us illustrate the crossover operator on each pair by (V_1', V_2'). At first, generate a random number c from the open interval $(0, 1)$, then the crossover operator on V_1' and V_2' will produce two children X and Y as follows:

$$X = c \cdot V_1' + (1-c) \cdot V_2' \quad \& \quad Y = (1-c) \cdot V_1' + c \cdot V_2'. \tag{14}$$

Generally speaking, this arithmetical crossover does not ensure that both children are feasible because the feasible set consists of discrete points. So we must check the feasibility of each child. If both children are feasible, then we replace the parents by them. If not, we keep the feasible one if exists, and then re-do the crossover operator by regenerating the random number c until two feasible children are obtained or a given number of cycles is finished. In this case, we only replace the parents by the feasible children.

5.6 Mutation operation

We define a parameter P_m of a genetic system as the probability of mutation. This probability gives us the expected number of $P_m \cdot pop\_size$ of chromosomes which undergo the mutation operations.

Similar to the process of selecting parents for crossover operation, we repeat the following steps from $i = 1$ to pop_size: generating a random real number r from the interval $[0, 1]$, the chromosome V_i is selected as a parent for mutation if $r < P_m$.

For each selected parent, denoted by $V = (x_1, x_2, \cdots, x_n)$, we mutate it by the following way. We choose a mutation direction d in $\Re^n$ randomly, if $V + M \cdot d$ is not feasible for the

constraints, then we set M as a random number between 0 and M until it is feasible, where M is a large positive number defined in the section of Initialization process. If the above process can not find a feasible solution in a predetermined number of iterations, then sets $M = 0$. We replace the parent V by its child

$$X = V + M \cdot d. \tag{15}$$

5.7 Stochastic simulation based genetic algorithm

Following selection, crossover and mutation, the new population is ready for its next evaluation. The genetic algorithm will terminate after a given number of cyclic repetitions of the above steps. We can summarize the stochastic simulation based genetic algorithm for solving both chance constrained integer programming and dependent-chance integer programming models as follows.

Procedure Stochastic Simulation based Genetic Algorithm

Input parameters: pop_size, P_c, P_m;
Initialize the chromosomes by Initialization Process;
REPEAT
Update chromosomes by crossover and mutation operators;
Compute the evaluation function for all chromosomes by stochastic simulation;
Select chromosomes by sampling mechanism;
UNTIL(*termination_condition*)

It is known that the best chromosome does not necessarily appear in the last generation. So we have to keep the best one from the beginning. If we find a better one in the new population, then replace the old one by it. This chromosome will be reported as the solution after finishing the evolution.

6. Numerical Examples

The computer code of the stochastic simulation based genetic algorithm for solving both chance constrained integer programming and dependent-chance integer programming models has been written in C language. To illustrate the effectiveness of the proposed genetic algorithm, a set of numerical examples has been performed, and the results are successful. Here we give some numerical examples which are all performed on a personal computer with the following parameters:
the population size is 30, the probability of crossover P_c is 0.3, the probability of mutation P_m is 0.2, the parameter a in the rank-based evaluation function is 0.05.

Now we suppose that there are 5 types of machines which produce 5 different products. We assume that the production capacities of type i machines are lognormally distributed. Here the density functions $\phi_i(c_i)$ of production capacities of type i machines are

$$\phi_i(c_i) = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma_i c_i} \exp\left[-\frac{(\ln c_i - \mu_i)^2}{2\sigma_i^2}\right], & 0 \le c_i < \infty \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

where (μ_i, σ_i) are (3.0, 1.0), (4.0, 1.6), (5.0, 1.6), (4.0, 1.2) and (3.0, 0.8), $i = 1, 2, \cdots, 5$, respectively. We also assume that the demands d_i of products i have exponential distributions, i.e., their densities are

$$\psi_i(d_i) = \begin{cases} \frac{1}{\beta_i} \exp\left(-\frac{d_i}{\beta_i}\right), & 0 \le d_i < \infty \\ 0, & \text{otherwise} \end{cases} \tag{17}$$

where β_i are 10, 15, 20, 18 and 16, $i = 1, 2, \cdots, 5$, respectively.

The levels of funds a_i that need to be allocated to type i machines are 300, 800, 700, 900 and 1000, $i = 1, 2, \cdots, 5$, respectively, and the total capital available for distribution a is 12500. In addition, the spaces b_i used by per type i machines are 20, 30, 50, 30 and 10, $i = 1, 2, \cdots, 5$, respectively, and the maximum available space b is 500.

If we want to find a capital budgeting plan with the maximum probability that the productions meet the demands, then we can model it by the following dependent-chance integer programming,

$$\begin{cases} \max \Pr\{c_i x_i \geq d_i,\ i = 1, 2, \cdots, 5\} \\ \text{subject to:} \\ \quad 300x_1 + 800x_2 + 700x_3 + 900x_4 + 1000x_5 \leq 12500 \\ \quad 20x_1 + 30x_2 + 50x_3 + 30x_4 + 10x_5 \leq 500 \\ \quad x_i,\ i = 1, 2, \cdots, 5, \quad \text{nonnegative integers} \end{cases}$$

where c_i and d_i are stochastic parameters with probability density functions (16) and (17), $i = 1, 2, \cdots, 5$, respectively, and x_i, $i = 1, 2, \cdots, 5$ are decision variables representing the numbers of type i machines selected. A run of the stochastic simulation based genetic algorithm with 300 generations shows that the optimal capital budgeting is

$$(x_1^*, x_2^*, x_3^*, x_4^*, x_5^*) = (6, 3, 2, 3, 4)$$

whose reliability level is 85.06%.

If we set the following target levels and priority structure:

Strict constraint: Space availability limitation,

$$20x_1 + 30x_2 + 50x_3 + 30x_4 + 10x_5 \leq 500.$$

Priority 1: The probability level of meeting the demand d_1 must achieve 97% as much as possible, i.e.,

$$\Pr\{c_1 x_1 \geq d_1\} + d_1^- - d_1^+ = 0.97$$

where d_1^- is to be minimized.

Priority 2: The probability level of meeting the demand d_2 must achieve 95% as much as possible, i.e.,

$$\Pr\{c_2 x_2 \geq d_2\} + d_2^- - d_2^+ = 0.95$$

where d_2^- is to be minimized.

Priority 3: The probability level of meeting the demands d_3, d_4 and d_5 must achieve 90% as much as possible, i.e.,

$$\Pr\{c_i x_i \geq d_i,\ i = 3, 4, 5\} + d_3^- - d_3^+ = 0.90$$

where d_3^- is to be minimized.

Priority 4: The total capital used for buying machines does not exceed 12500 as much as possible, i.e.,

$$300x_1 + 800x_2 + 700x_3 + 900x_4 + 1000x_5 + d_4^- - d_4^+ = 12500$$

where d_4^+ is to be minimized.

According to the above mentioned priority structure and target levels, the following dependent-chance goal programming is formulated,

$$\begin{cases} \text{lexmin}\{d_1^-, d_2^-, d_3^-, d_4^+\} \\ \text{subject to:} \\ \quad \Pr\{c_1 x_1 \geq d_1\} + d_1^- - d_1^+ = 0.97 \\ \quad \Pr\{c_2 x_2 \geq d_2\} + d_2^- - d_2^+ = 0.95 \\ \quad \Pr\{c_i x_i \geq d_i,\ i = 3,4,5\} + d_3^- - d_3^+ = 0.90 \\ \quad 300x_1 + 800x_2 + 700x_3 + 900x_4 + 1000x_5 + d_4^- - d_4^+ = 12500 \\ \quad 20x_1 + 30x_2 + 50x_3 + 30x_4 + 10x_5 \leq 500 \\ \quad d_i^-, d_i^+ \geq 0, \quad i = 1,2,3,4 \\ \quad x_i,\ i = 1,2,\cdots,5, \quad \text{nonnegative integers.} \end{cases}$$

A run of the stochastic simulation based genetic algorithm with 400 generations shows that the optimal capital budgeting is

$$(x_1^*, x_2^*, x_3^*, x_4^*, x_5^*) = (5,4,2,4,4)$$

which satisfies the first three objectives, but the fourth objective is 1200. In fact, we have

$$\begin{aligned} &\Pr\{c_1 x_1^* \geq d_1\} = 97.65\% \\ &\Pr\{c_2 x_2^* \geq d_2\} = 95.34\% \\ &\Pr\{c_i x_i^* \geq d_i,\ i = 3,4,5\} = 91.89\% \end{aligned}$$

and the total capital that needs to be distributed is 13700.

7. Conclusion

In this paper we extended dependent-chance programming to integer case and modeled capital budgeting problems by dependent-chance integer programming. It was also shown that there is a common part between chance constrained programming and dependent-chance programming via capital budgeting problems. In addition, we designed a stochastic simulation based genetic algorithm for solving both chance constrained integer programming and dependent-chance integer programming. Finally, the effectiveness of the stochastic simulation based genetic algorithm was illustrated by some numerical examples.

References

[1] A. Charnes and W.W. Cooper: Chance-constrained programming. *Management Science*, **6**-1 (1959) 73–79.

[2] P.K. De, D. Acharya and K.C. Sahu: A chance-constrained goal programming model for capital budgeting. *Journal of the Operational Research Society*, **33** (1982) 635–638.

[3] D.B. Fogel: An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, **5** (1994) 3–14.

[4] D.E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, 1989).

[5] K. Iwamura and B. Liu: A genetic algorithm for chance constrained programming. *Journal of Information & Optimization Sciences*, **17**-2 (1996) 409–422.

[6] K. Iwamura and B. Liu: Chance constrained integer programming models for capital budgeting in fuzzy environments. to appear in *Journal of the Operational Research Society*.

[7] P. Kall and S.W. Wallace: *Stochastic Programming* (John Wiley & Sons, 1994).
[8] A.J. Keown and J.D. Martin: A chance constrained goal programming model for working capital management. *Engng. Econ.*, **22** (1977) 153–174.
[9] A.J. Keown and B.W. Taylor: A chance-constrained integer goal programming model for capital budgeting in the production area. *Journal of the Operational Research Society*, **31**-7 (1980) 579–589.
[10] V.V. Kolbin: *Stochastic Programming* (D.Reidel Dordrecht, 1977).
[11] B. Liu: Dependent-chance programming: a class of stochastic programming. *Computers & Mathematics with Applications*, **34**-12 (1997) 89–104.
[12] B. Liu and C. Ku: Dependent-chance goal programming and an application. *J. of Systems Engineering & Electronics*, **4**-2 (1993) 40–47.
[13] B. Liu: Dependent-chance goal programming and its genetic algorithm based approach. *Mathematical and Computer Modelling*, **24**-7 (1996) 43–52.
[14] B. Liu and K. Iwamura: Modelling stochastic decision systems using dependent-chance programming. *European Journal of Operational Research*, **101**-1 (1997) 193–203.
[15] B. Liu and K. Iwamura: Chance constrained programming with fuzzy parameters. *Fuzzy Sets and Systems*, **94**-2 (1998) 227–237.
[16] Z. Michalewicz: *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd ed.(Springer-Verlag, New York, 1994).
[17] J.K. Sengupta: *Stochastic Programming: Methods and Applications* (North-Holland, Amsterdam, 1972).
[18] S. Vajda: *Probabilistic Programming* (Academic Press, New York, 1972).

Kakuzo Iwamura
Josai University
Sakado, Saitama 350-02
E-mail: kiwamura@math.josai.ac.jp