2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

A Genetic Algorithm Based Double Layer Neural Network for Solving Quadratic Bilevel Programming Problem

Jingru Li
Graduate School of Information, Production and System, Waseda University, Kitakyushu 808-0135 Japan
Email: jingruli@fuji.waseda.jp

Junzo Watada
Graduate School of Information, Production and System, Waseda University, Kitakyushu 808-0135 Japan
Email: junzow@osb.att.ne.jp

Shamshul Bahar Yaakob
Graduate School of Information, Production and System, Waseda University, Kitakyushu 808-0135 Japan
Email: shamshulbahar@gmail.com

***Abstract*—In this paper, an intelligent genetic algorithm (IGA) and a double layer neural network (NN) are integrated into a hybrid intelligent algorithm for solving the quadratic bilevel programming problem. The intelligent genetic algorithm is used to select a set of potential solution combinations from the entire generated combinations of the upper level. Then a meta-controlled Boltzmann machine, which is formulated by comprising the Hopfield model (HM) and the Boltzmann machine (BM), is used to effectively and efficiently determine the optimal solution of the lower level. Numerical experiments on examples show that the genetic algorithm based double layer neural network enables us to efficiently and effectively solve quadratic bilevel programming problems.**

I. Introduction

Bilevel programming problem (BLPP) has been increasingly addressed in the literature. The problem involves two optimization problems where the constraint region of the upper level problem is implicitly determined by another lower level optimization problem [1], [2]. This mathematical programming model arises when two independent decision makers, ordered within a hierarchical structure, have conflicting objectives [1], [3]. A decision maker at the lower level has to optimize its own objective function under the given parameters from an upper level decision maker, who in return, with complete information on the possible reactions of the lower level, selects the parameters so as to optimize its own objective function [1].

The bilevel programming model is extremely useful to the fields like government policies, economic systems, finance, power systems, transportation and network designs, and is particularly suitable for conflict resolution [4], [5], [6]. However, due to its inherent non-convexity and non-differentiated properties, bilevel programming problem is hard to solve, even the simplest case—Linear BLPP—has been proved to be a NP-hard problem [1], [2]. A conventional approach to solve the bilevel programming problem is to transform the original two level problems into a single level one by replacing the lower level optimization problem with its Karush-Kuhn-Tuchker (KKT) optimization conditions. Branch-and Bound method [7], [8], [9], decent algorithms [10], [11], and evolutionary method [12], [13] have been proposed for solving the bilevel programming problems based on this reformulation. Compared with classical optimization approaches, the prominent advantage of neural network is that it can converge to the equilibrium point (optimal solution) rapidly, and this advantage has been attracting researchers to solve bilevel programming problem using neural network approach. Shih [14] and Lan [15] recently proposed a neural network for solving the linear bilevel programming problem. But it deserves pointing out that there are still quite few reports on solving bilevel programming problem by using neural network, especially for the quadratic bilevel programming problem.

In this study, we formulate a double layer neural network comprising an intelligent genetic algorithm, a Hopfield network, and a Boltzmann machine in order to effectively and efficiently solve quadratic bilevel programming problem. The improved genetic algorithm is used to select a set of potential solution combinations from the entire generated combinations of the upper level problem. In the lower level problem, a Hopfield network and a Boltzmann machine are interconnected together as a double layer neural network. This network will first generate its structure by selecting a limited number of units, and then it will converge to the optimal solution/units from those units, thus constitutes an effective problem solving method [18]. Since the Hopfield neural network [16] itself will easily terminate at a local minimum of the describing energy function, the interconnected Boltzmann machine [17] which we also apply a structural learning method is possible to improve the performance by using probabilities to update both the state of a neuron and its energy function, such that the network will rarely trap in a local minimum.

The rest of this paper is organized as follows: Section II lists notations we will use in this paper. Section III derives an formulation of bilevel programming problem, which will be transformed into a form that can be solved by the neural network. Section IV introduces the Hopfield network and Boltzmann machine, then we will discuss about how to improve those networks. After that, we will present the hybrid intelligent double layer neural network for solving the quadratic bilevel programming problem. Simulation results are presented in Section V to show its performance. Finally, Section VI concludes the paper.

II. Notations

At first, we will list out the notations that we used in the following explanations.

$\mathbf{x}$: upper level variable vector $\in R^{n_1}$,
$\mathbf{y}$: lower level variable vector $\in R^{n_2}$,

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

F: upper level objective function $\in R^{n_1} \times R^{n_2} \to R^1$,
f: lower level objective function $\in R^{n_1} \times R^{n_2} \to R^1$,
$\mathbf{G}$: vector-valued function of upper level constraints $\in R^{n_1} \times R^{n_2} \to R^{m_1}$,
$\mathbf{g}$: vector-valued function of lower level constraints $\in R^{n_1} \times R^{n_2} \to R^{m_2}$,
$\mathbf{a}, \mathbf{c}$: vector $\in R^{n_1}$,
$\mathbf{b}, \mathbf{d}$: vector $\in R^{n_2}$,
$\mathbf{A}$: matrix $\in R^{m_1 \times n_1}$,
$\mathbf{B}$: matrix $\in R^{m_1 \times n_2}$,
$\mathbf{C}$: matrix $\in R^{m_2 \times n_1}$,
$\mathbf{D}$: matrix $\in R^{m_2 \times n_2}$,
$\mathbf{r}_1$: vector $\in R^{m_1}$,
$\mathbf{r}_2$: vector $\in R^{m_2}$,
$\mathbf{P}$: $n_1 + n_2$ dimension positive-definite symmetrical matrix,
$\mathbf{Q}$: $n_1 + n_2$ dimension positive-definite symmetrical matrix,
$\mathbf{Q}_0$: matrix $\in R^{n_2} \times R^{n_2}$,
$\mathbf{Q}_1$: matrix $\in R^{n_2} \times R^{n_1}$,
$\mathbf{Q}_2$: matrix $\in R^{n_1} \times R^{n_1}$,
E: energy function of the neural network,
w_{ij}: weight between neuron i and neuron j,
θ_i: the threshold of neuron i.

III. Bilevel Programming Problem

Mathematically, the bilevel programming problem can be represented as

$$\begin{aligned} \min_{\mathbf{x}} \quad & F(\mathbf{x}, \mathbf{y}) \\ s.t. \quad & \mathbf{G}(\mathbf{x}, \mathbf{y}) \le \mathbf{0} \\ & \min_{\mathbf{y}} \quad f(\mathbf{x}, \mathbf{y}) \\ & s.t. \quad \mathbf{g}(\mathbf{x}, \mathbf{y}) \le \mathbf{0} \end{aligned} \tag{1}$$

where $\mathbf{x} \in R^{n_1}$ and $\mathbf{y} \in R^{n_2}$ are the upper level variables and lower level variables respectively. Similarly, the functions $F :\in R^{n_1} \times R^{n_2} \to R^1$ and $f :\in R^{n_1} \times R^{n_2} \to R^1$ are the upper level and lower level objective functions, while the vector-valued functions $\mathbf{G} :\in R^{n_1} \times R^{n_2} \to R^{m_1}$ and $\mathbf{g} :\in R^{n_1} \times R^{n_2} \to R^{m_2}$ are called the upper level and lower level constraints. More specifically, the quadratic bilevel programming can be formulated as a matrix form as

$$\begin{aligned} \min_{\mathbf{x}} \quad & F(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \mathbf{x}^T & \mathbf{y}^T \end{bmatrix} \mathbf{P} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} + \begin{bmatrix} \mathbf{a}^T & \mathbf{b}^T \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \\ s.t. \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \le \mathbf{r}_1 \\ & \min_{\mathbf{y}} \quad f(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \mathbf{x}^T & \mathbf{y}^T \end{bmatrix} \mathbf{Q} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \\ & \qquad\qquad + \begin{bmatrix} \mathbf{c}^T & \mathbf{d}^T \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \\ & s.t. \quad \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{y} \le \mathbf{r}_2 \end{aligned} \tag{2}$$

where $\mathbf{a}, \mathbf{c} \in R^{n_1}$ and $\mathbf{b}, \mathbf{d} \in R^{n_2}$, $\mathbf{A} \in R^{m_1 \times n_1}$, $\mathbf{B} \in R^{m_1 \times n_2}$, $\mathbf{C} \in R^{m_2 \times n_1}$, $\mathbf{D} \in R^{m_2 \times n_2}$, $\mathbf{r}_1 \in R^{m_1}$, $\mathbf{r}_2 \in R^{m_2}$. $\mathbf{P}$ and $\mathbf{Q}$ are n_1 + n_2 dimension symmetrical matrix.

Then, this quadratic bilevel programming problem follows definitions below [1], [2]

Definition 1: The constraint region of problem (2) is denoted as

$$\Omega = \{(\mathbf{x}, \mathbf{y}) : \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \le \mathbf{r}_1,\ \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{y} \le \mathbf{r}_2\}$$

Definition 2: The projection of Ω onto the upper level decision space is

$$I = \{\mathbf{x} : \exists \mathbf{y},\ \text{such that } (\mathbf{x}, \mathbf{y}) \in \Omega\}$$

Definition 3: The lower level rational reaction set for $\mathbf{x} \in I$ is defined as

$$R(\mathbf{x}) = \{\mathbf{y} : \arg\min \{f(\mathbf{x}, \mathbf{y}) : \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{y} \le \mathbf{r}_2\}\}$$

Definition 4: The induced region of quadratic bilevel programming problem is

$$IR = \{(\mathbf{x}, \mathbf{y}) : (\mathbf{x}, \mathbf{y}) \in \Omega, \mathbf{y} \in R(\mathbf{x})\}$$

A point $(\mathbf{x}, \mathbf{y})$ is called to be feasible if $(\mathbf{x}, \mathbf{y}) \in IR$. In order to ensure that problem (2) is well posed, we make assumptions that Ω is nonempty and compact, $\mathbf{P}$ is positive-definite.

Then, let $\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_2 & \mathbf{Q}_1^T \\ \mathbf{Q}_1 & \mathbf{Q}_0 \end{bmatrix}$, where $\mathbf{Q}_0 \in R^{n_2} \times R^{n_2}$, $\mathbf{Q}_1 \in R^{n_2} \times R^{n_1}$, $\mathbf{Q}_2 \in R^{n_1} \times R^{n_1}$, and $\mathbf{Q}$ is also positive-definite. The lower level programming problem of problem (2) will be strictly convex quadratic programming problem in $\mathbf{y}$ for each fixed $\mathbf{x}$, and then there exists a unique optimal solution for each $\mathbf{x}$. Implying that $R(\mathbf{x})$ is single-valued and the induced region could be replaced by a unique response function. In addition, $\mathbf{P}$ is a positive-definite matrix, namely, $F(\mathbf{x}, \mathbf{y})$ is strictly convex in $(\mathbf{x}, \mathbf{y})$, so the solution to problem (2) is guaranteed to exist [3].

With the definition of $\mathbf{Q}$, the lower level problem is transferred into

$$f(\mathbf{x}, \mathbf{y}) = \mathbf{c}^T\mathbf{x} + \mathbf{x}^T\mathbf{Q}_2\mathbf{x} + (\mathbf{d} + 2\mathbf{Q}_1\mathbf{x})^T\mathbf{y} + \mathbf{y}^T\mathbf{Q}_0\mathbf{y}$$

Note that $\mathbf{c}^T\mathbf{x} + \mathbf{x}^T\mathbf{Q}_2\mathbf{x}$ is a constant for each fixed $\mathbf{x}_0 \in \Omega(\mathbf{x}_0, \mathbf{y})$, we can assume $\mathbf{c} = \mathbf{0}$, $\mathbf{Q}_2 = \mathbf{0}$ to ignore those terms without loss of generality when solving the lower level programming problem. Thus the quadratic bilevel programming problem can be formulated finally as

$$\begin{aligned} \min_{\mathbf{x}} \quad & F(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \begin{bmatrix} \mathbf{x}^T & \mathbf{y}^T \end{bmatrix} 2\mathbf{P} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \\ & \qquad + \begin{bmatrix} \mathbf{a}^T & \mathbf{b}^T \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \\ s.t. \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \le \mathbf{r}_1 \\ & \min_{\mathbf{y}} \quad f(\mathbf{x}, \mathbf{y}) = \mathbf{y}^T\mathbf{Q}_0\mathbf{y} + (\mathbf{d} + 2\mathbf{Q}_1\mathbf{x})^T\mathbf{y} \\ & s.t. \quad \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{y} \le \mathbf{r}_2 \end{aligned} \tag{3}$$

In this study, a genetic algorithm based hybrid double layer neural network method is employed to solve the quadratic bilevel programming model of problem (3).

IV. An intelligent genetic algorithm and hybrid double layer neural networks

Conventionally, the number of units is decided on the basis of expert experience. In order to solve this problem, we formulate a hybrid intelligent algorithm consisting of both Hopfield and Boltzmann machine neural networks called meta-controlled Boltzmann machine and hybrid with an improved genetic algorithm. This hybrid intelligent algorithm can be employed to the quadratic bilevel programming problems. In the upper level of bilevel programming problem, we proposed an intelligent genetic algorithm that is effective in global searching to find a set of possible feasible $\mathbf{x}$ values. This set of feasible $\mathbf{x}$ will be substituted into lower level, which will generate a set of quadratic programming problems of variable $\mathbf{y}$. Then these problems will be solved by using meta-controlled Boltzmann machine to get optimal $(\mathbf{x}^*, \mathbf{y}^*)$.

A. Intelligent genetic algorithm for upper level

Genetic algorithm operates on a population of trial solutions that are initially generated at random. The GA seeks to maximize the fitness of the population by selecting the fittest individuals from the population and using their genetic information in mating operations to create a new population of solutions. Genetic algorithms have many advantages over the traditional optimization methods. In particular, GAs do not require function derivatives and work on function evaluations alone; they have a better possibility of locating the global optimum because they search a population of points rather than a single point and they allow for consideration of design spaces consisting of a mix of continuous and discrete variables. In addition, GAs provide the decision-makers with a set of acceptable optimal solutions (rather than a single solution) from which they can select the most appropriate one.

Many variations of the fundamental GA have been developed, but the algorithm implemented here is an intelligent GA with search directions by considering both the feasible region and the infeasible region of the bilevel programming problem. The probabilistic nature of GA helps to avoid convergence to false optimal. However, due to its randomness, genetic algorithm techniques alone may not be able to find the global optimal for the bilevel programming problem, and even the global optimal could be found, the convergence would be slow. To improve this portion, we proposed the intelligent genetic algorithm below.

Now we present the improved GA for problem (3) as follows.

Step 0 Parameter setting

Choose feasible population size M and infeasible population size M', probability of crossover p_c, probability of mutation p_m, the maximal number of generations $MaxGen$, Initialize the current generation: $k = 0$.

Step 1 Initialization

Define the feasible solution set $feaX$ and infeasible solution set $infeX$ as

$$\begin{aligned} feaX &= \{\mathbf{x} : \mathbf{x} \in \Omega(\mathbf{x})\} \\ infeX &= \{\mathbf{x} : \mathbf{x} \notin \Omega(\mathbf{x})\} \end{aligned} \tag{4}$$

where $\Omega(\mathbf{x})$ is the upper level constraint region of the bilevel programming problem. Randomly initialize feasible solution $\mathbf{fea\_x}_i(k)$ and infeasible solution $\mathbf{infe\_x}_j(k)$, where $i = 1, 2, \cdots, M$ and $j = 1, 2, \cdots, M'$. Then, put them into $feaX$ and $infeX$ respectively.

Step 2 Crossover operation

Step 2.1 During the i_{th} crossover, this algorithm generates a random number $\lambda_i \in (0, 1)$. For each chromosome $\mathbf{fea\_x}_i(k)$, generate a random number r. If $r < p_c$, then choose this chromosome as one of the parents for crossover operation and the corresponding infeasible chromosome as another parents.

Step 2.2 After getting a couple of chromosomes $(\mathbf{fea\_x}_i(k))$ and $(\mathbf{infe\_x}_i(k))$, we make two offsprings as $\mathbf{u}_i(k)$ and $\mathbf{v}_i(k)$ following the crossover rules:

$$\begin{aligned} \mathbf{u}_i(k) &= \lambda \cdot \mathbf{fea\_x}_i(k) + (1 - \lambda) \cdot \mathbf{infe\_x}_i(k) \\ \mathbf{v}_i(k) &= (1 - \lambda) \cdot \mathbf{fea\_x}_i(k) + \lambda \cdot \mathbf{infe\_x}_i(k) \end{aligned} \tag{5}$$

Step 2.3 Put two offsprings $\mathbf{u}_i(k)$ and $\mathbf{v}_i(k)$ into $feaX$ and $infeX$ by following rules:

$$\begin{cases} \text{put } \mathbf{u}_i(k) \text{ into } feaX, & \text{if } \mathbf{u}_i(k) \in \Omega(\mathbf{x}) \\ \text{put } \mathbf{u}_i(k) \text{ into } infeX, & \text{otherwise} \end{cases} \tag{6}$$

$$\begin{cases} \text{put } \mathbf{v}_i(k) \text{ into } feaX, & \text{if } \mathbf{v}_i(k) \in \Omega(\mathbf{x}) \\ \text{put } \mathbf{v}_i(k) \text{ into } infeX, & \text{otherwise} \end{cases} \tag{7}$$

After this crossover operation, we will get N_1 new feasible elements and N_1' new infeasible elements.

Step 3 Mutation operation

Step 3.1 Choose a chromosome from feasible set $feaX$ and infeasible set $infeX$ randomly with probability p_m, mutation is operated to this chromosome.

Step 3.2 Randomly generating $\gamma \in (-1, 1)$, add this γ to the chosen chromosome, and we get a new chromosome.

Step 3.3 Put the new chromosome into $feaX$ or $infeX$ according to rule (11).

Step 3.4 When Step 3.1 to Step 3.3 go through $feaX$ and $infeX$, count the total number of the feasible mutation offsprings

and the infeasible mutation offsprings, denoted by N_2 and N_2' respectively.

Step 4 Calculation operation

For each element in $feaX$ and $infeX$, substitute it into the lower level of problem (3), then we can execute them with meta-controlled Boltzmann machine. We will obtain optimal $\bar{\mathbf{y}}$ of the lower level corresponding to each element, which can be denoted as pair $(\mathbf{x}_i(k), \bar{\mathbf{y}}_i(k))$ where $\mathbf{x}_i(k) \in feaX \cup infeX$. The upper level value $F_i(k)$ can be calculated for each $(\mathbf{x}_i(k), \bar{\mathbf{y}}_i(k))$. Such pairs and values will be processed during the selection operation.

Step 5 Selection operation

Finally, the size of $feaX$ and $infeX$ are $M + N_1 + N_2$ and $M' + N_1' + N_2'$, that means we have $M + N_1 + N_2$ feasible chromosomes and $M' + N_1' + N_2'$ infeasible chromosomes in total. In order to ensure convergence to the global optimal solution, sort the fitness values in the increasing order among parents and all offspring chromosomes and select M and M' chromosomes as the next generation. Elitist selection is also used to save the best chromosome.

Step 6 Stopping criterion

If the algorithm is executed to the maximal number of generations MaxGen, then the algorithm should stop executing, and output the best chromosome, the optimal solution, and the optimal value of problem (3). Otherwise, set $k = k + 1$, and return to Step 2.

B. Hopfield and Bolzmann Machine

Studies on mutually connected network behavior started around 1948. Simply speaking, it is difficult to select the required number of units that will minimize the corresponding energy function. This problem cannot be solved by either Hopfield or Boltzmann neural networks. In 1982, Hopfield [16] combined several previous ideas concerning recurrent neural networks, accompanied by a complete mathematical analysis. Currently, this type of network is generally referred to as Hopfield network. Although the Hopfield network is not appropriate for all applications, it nevertheless warrants a review in terms of its structure and internal mechanisms. This will lead to a modification, by incorporating mutual connections in the form of the Boltzmann machine, to overcome its drawbacks. The Hopfield network is a fully connected, recurrent neural network, which uses a form of the generalized Hebb's rule to store boolean vectors in its memory. Each unit neuron, which is represented by n, has a state value denoted by s_n. In any situation, combining the states of all units leads to a global state for the network. This global state is the input, which is stored, together with other prototypes, in the weighting matrix by Hebb's postulate, formulated as

$$w_{ij} = \frac{1}{2} \sum_{p} x_{i_p} x_{j_p} \tag{8}$$

where $p = 1 \ldots N$, w_{ij} is the weight of the connection from neuron i to neuron j, N is the dimension of the vector, p is the number of training patterns, and x_{i_p} is the pth input for neuron i. In other words, using Hebb's postulate, we create a weighting matrix that stores the entire prototype that we want the network to remember. Because of these features, it is sometimes referred to as an auto-associative memory. However, it is worth noting that the maximum number of prototypes that a Hopfield network can store is only 0.15 times more than the total number of units in the network [16].

One application of the Hopfield network is as an energy minimizer. This application is relevant because of the ability of Hopfield networks to minimize an energy function during its operation. The simplest form of energy function is given by the following:

$$E = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} s_i s_j \tag{9}$$

Here, w_{ij} denotes the strength of the influence of neuron i on neuron j. The w_{ij} values are created using Hebb's postulate as mentioned above, and they belong to a symmetric matrix with the main diagonal containing only zeroes (which means that there are no self-feedback connections). Because of this useful property, the Hopfield network can be used to solve combinatorial optimization problems. However, Hopfield networks suffer from a major disadvantage in that they sometimes converge to a local rather than a global minimum, which usually happens when dealing with noisy inputs. To overcome this problem, a modification was made to the Boltzmann machine [17].

The Boltzmann machine is an interconnected neural network and a modification of the Hopfield network, which helps it escape local minima. The main idea is to employ simulated annealing, a technique derived from the metallurgy industry. The term annealing comes from the technique of hardening a metal (i.e., finding a crystalline lattice state that is highly packed) by hammering it while it is initially very hot and then again at a succession of decreasing temperatures. Simulated annealing is an optimization technique. In Hopfield networks, local minima are used in a positive way, but in optimization problems, local minima are undesirable; one must have a way of escaping them. When optimizing a very large and complex system (i.e., a system with many degrees of freedom), instead of always going downhill, we try to go downhill most of the time. Initially, the probability of not going downhill should be relatively high (high temperature), but as time (the number of iterations) progresses, this probability should decrease (with the temperature decreasing according to an annealing schedule). The convergence time of a Boltzmann machine is usually extremely long, depending on the number of units employed. According to the annealing schedule, where T_0 is a critical system parameter that represents the initial value of the temperature. If T_0 is very large, then a strategy is pursued whereby neurons are flipped on and off at random, totally ignoring incoming information. If T_0 is close to zero,

the network behaves deterministically, i.e., like a network of McCulloch-Pitts neurons. Although the way in which a Boltzmann machine works is similar to a Hopfield network, we cannot use Hebb's postulate to create the weighting matrix representing the correlations between units. Instead, we use a training (learning) algorithm one based on the Metropolis algorithm. The Boltzmann machine can be viewed as a stochastic, generative counterpart of the Hopfield network. In the Boltzmann machine, probability rules are employed to update the states of the neurons and the energy function as follows: If $V_i(t+1)$ is the output of neuron i in the subsequent time iteration $t+1$, $V_i(t+1)$ is 1 with probability P and $V_i(t+1)$ is 0 with probability $1-P$, where

$$P[V_i(t+1)] = f(\frac{u_i(t)}{T}) \tag{10}$$

Here, $f(\cdot)$ is the sigmoid function, $u_i(t)$ is the total input to neuron i, as shown in equation (14), and T is the network temperature.

$$u_i(t) = \sum_{j=1} w_{ij} V_i(t) + \theta_i \tag{11}$$

Here, w_{ij} is a weight between neurons i and j, θ_i is the threshold of neuron i, and V_i is the state of unit i. The energy function, E, proposed by Hopfield, is written as

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} V_i V_j - \sum_{i=1}^{N} \theta_i V_i \tag{12}$$

Hopfield has shown that those energy functions decrease monotonically with learning [16]. Therefore, it is likely that this energy function converges to a local minimum. However, in the case of the Boltzmann machine, the energy function can increase with some minute probability. Therefore, the energy function will be unlikely to fall into a local minimum. Thus, the combination of a Hopfield network and a Boltzmann machine offers a solution to overcome the problem of finding the optimal number of units in the neural network. Accordingly, this study proposes a meta-controlled Boltzmann machine, which we will discuss in detail next.

C. Meta-Controlled Boltzmann Machine for lower level

The meta-controlled Boltzmann machine is a neural network model, as proposed by Watanabe and Watada [18]. This model deletes the units of the lower layer that are not selected in the meta-controlling layer in its execution. Then, the lower layer is restructured using the selected units. Because of this feature, the meta-controlled Boltzmann machine converges more efficiently than a conventional BM. This is an efficient method for solving selection problem with a large data set by transforming objective function into the energy function, because the Hopfield network and the Boltzmann machine converge at the minimum point of the energy function. The meta-controlled Boltzmann machine described above converts the objective functions into the energy functions of two components, namely, the meta-controlling layer (Hopfield network) E_u and the lower layer (Boltzmann machine) E_l, as described below.

Meta-controlling layer

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \sigma_{ij} x_i x_j + K_u \sum_{i=1}^{N} \mu_i x_i \tag{13}$$

Lower layer

$$\begin{aligned} E = -\frac{1}{2}(\sum_{i=1}^{N}\sum_{j=1}^{N} \sigma_{ij} y_i y_j + 2\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j) \\ + 2\sum_{i=1}^{N} y_i + K_l \sum_{i=1}^{N} \mu_i y_i \end{aligned} \tag{14}$$

Here, K_u and K_l are the meta controlling parameters of this double layer neural network, respectively, and x_i is the output of the i_{th} unit of the meta-controlling layer. The double-layered Boltzmann machine was tuned such that the upper layer influences the lower layer with an iteration ratio of $1-\varepsilon(t)$ and that the lower layer influences the upper layer with probability $\varepsilon(t)$. Here, $\varepsilon(t)$ is a monotonically decreasing function with time t. Thus, the double-layered Boltzmann machine was iterated with

$$Y_i(t) = (1-\varepsilon(t))y_i + \varepsilon(t)x_i \tag{15}$$

for the upper layer and

$$X_i(t) = x_i[(1-\varepsilon(t))y_i + \varepsilon(t)] \tag{16}$$

for the lower layer, where Y_i is a value transferred to the corresponding nodes in the upper layer, X_i is a value transferred to corresponding nodes in the lower layer, y_i is the value of the present state at node i in the upper layer, and x_i is the value of the present state at node i in the lower layer. The expression for X_i means that its value is influenced by of the value of node i in the upper layer. When y_i is 1, $X_i = x_i$; otherwise, when y_i is 0, only of x_i is transferred to the corresponding nodes. In turn, Y_i has an influence from the lower layer. Therefore, even if the upper layer converges to a local minimum, the perturbation from the lower layer allows the upper layer to escape from this local minimum. If the local minimum possesses a large barrier, dynamic optimization may be used, i.e., by dynamically changing and in a manner similar to simulated annealing.

D. Hybrid intelligent algorithm

In order to solve the quadratic bilevel programming problem of problem (3), we need to transform the objective function of the lower level into the energy function form of meta-controlled Boltzmann machine:

$$\begin{aligned} (\mathbf{d}+2\mathbf{Q}_2\mathbf{x})^T\mathbf{y} &= 2\mathbf{y} + (\mathbf{d}+2\mathbf{Q}_2\mathbf{x}-2\mathbf{h})^T\mathbf{y} \\ &= 2\mathbf{h}^T\mathbf{y} + K_l(\frac{\mathbf{d}+2\mathbf{Q}_2 x - 2\mathbf{h}}{K_l})^T\mathbf{y} \\ &= 2\mathbf{h}^T\mathbf{y} + K_l\boldsymbol{\mu}^T\mathbf{y} \end{aligned} \tag{17}$$

where $\boldsymbol{\mu}$ is a n_2 dimension vector of $[\mu_1, \mu_2, ..., \mu_{n2}]^T$ and $\mathbf{h}$ is a n_2 dimension vector of all 1s and:

$$\begin{aligned}\mathbf{y}^T\mathbf{Q}_1\mathbf{y} &= \mathbf{y}^T(\mathbf{Q}_1+\mathbf{H})\mathbf{y} - \mathbf{y}^T\mathbf{H}\mathbf{y} \\ &= -\frac{1}{2}(\mathbf{y}^T(-2\mathbf{Q}_1-2\mathbf{H})\mathbf{y} + 2\mathbf{y}^T\mathbf{H}\mathbf{y}) \\ &= -\frac{1}{2}(\mathbf{y}^T\boldsymbol{\sigma}\boldsymbol{y} + 2\mathbf{y}^T\mathbf{H}\mathbf{y})\end{aligned} \tag{18}$$

where $\mathbf{H}$ is a $n_2 \times n_2$ dimension matrix of all 1s and $\boldsymbol{\sigma}$ is a symmetric matrix of $n_2 \times n_2$ dimension that $\sigma_{ij} = (-2\mathbf{Q}_1 - 2\mathbf{H})_{ij}$. Thus, we got the parameters for the energy function of meta-controlled Boltzmann machine as equation (13)(14) for the lower level of the bilevel programming problem. Then we can get the solution by using our hybrid intelligent algorithm.

The proposed algorithm is summarized as follows and is shown in Fig. 1:

Step 0. Start with a series randomly generated combinations. Then using the proposed IGA to select a set of potential solution combinations for the upper level of the quadratic bilevel programming problem.

Step 1. Set each parameter to its initial value for meta-controlled Boltzmann machine.

Step 2. Input K_u and K_l.

Step 3. Execute the meta-controlling layer of the network.

Step 4. If the output value of a unit in the meta-controlling layer is 1, add some amount of this value to the corresponding unit in the lower layer. Execute the lower layer.

Step 5. After executing the lower layer at a constant frequency, decrease the temperature.

Step 6. If the output value of a unit in the lower layer is large enough, add some amount of values to the corresponding unit in the meta-controlling layer.

Step 7. Iterate Step 3 to Step 6 until the temperature reaches the restructuring temperature.

Step 8. Restructure the lower layer using the units selected in the meta-controlling layer.

Step 9. Execute the lower layer until the termination condition is reached.

Step 10. After finding the optimal f^* for this iteration, store $(\mathbf{x}^*, \mathbf{y}^*)$ and get the best F^* of this iteration for upper level problem. Then using the IGA to select another set of potential solution combinations for next interation.

By above steps, the meta-controlled Boltzmann machine can effectively and efficiently select units from those available units to find the optimal solution. The Hopfield network works in the meta-controlling layer to delete the lower layer Boltzmann machine units, which are not selected in the upper layer during execution. This is achieved in Steps 3 through 6 of this algorithm. Then, the lower layer is restructured with those selected units. The Boltzmann machine is employed as the lower layer to find the optimal units from the units selected by the meta-controlling layer.

V. Numerical Example

In this section we will present two examples provided in [19] to make comparison. The examples are as follows:

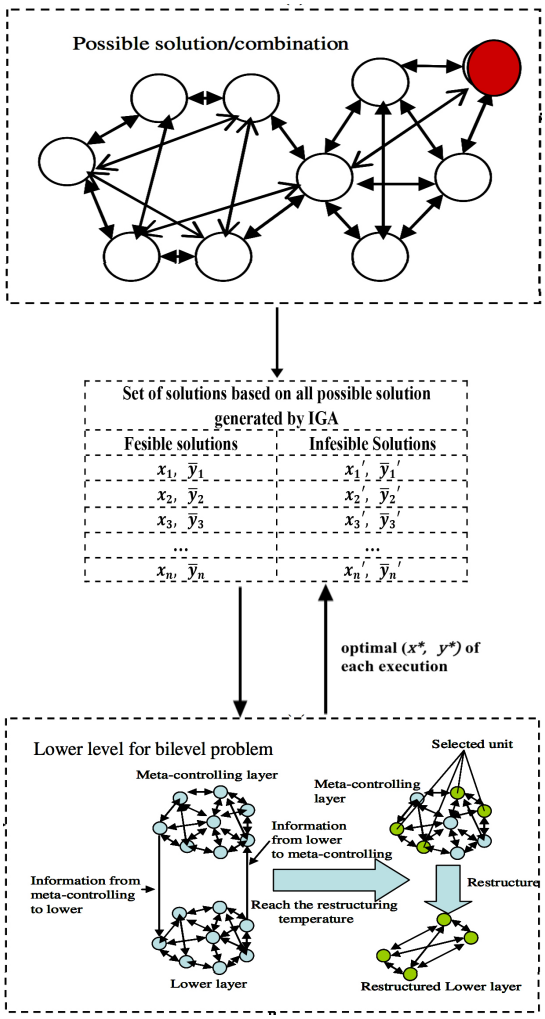

Fig. 1. Hybrid double layer neural network for solving the quadratic bilevel programming problem

Example 1

$$\min_{\mathbf{x}} \; F = x^2 - 4x + y_1^2 + y_2^2$$

$$s.t. \quad 0 \le x \le 2$$

$$\min_{\mathbf{y}} \; f = y_1^2 + 0.5y_2^2 + y_1y_2 + (1-3x)y_1 + (1+x)y_2$$

$$s.t. \quad y_1, y_2 \ge 0$$

$$2y_1 + y_2 - 2x \le 1$$

Example 2

$$\min_{\mathbf{x}} \; F = -7x_1 + 4x_2 + y_1^2 + y_3^2 - y_1y_3 - 4y_2$$

$$s.t. \quad x_1, x_2 \ge 0$$

$$x_1 + x_2 \le 1$$

$$\min_{\mathbf{y}} \; f = (1-3x_1)y_1 + (1+x_2)y_2 + y_1^2 + 0.5y_2^2 + 0.5y_3^2 + y_1y_2$$

$$s.t. \quad y_1, y_2, y_3 \ge 0$$

$$x_1 - 2x_2 + 2y_1 + y_2 - y_3 + 2 \le 0$$

The important parameters for Example 1. are as follows:

$$\mathbf{P}=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix},\ \mathbf{A}=\begin{bmatrix}1\\-1\end{bmatrix},\ \mathbf{B}=\mathbf{0}$$

$$\mathbf{a}=[-4],\ \mathbf{b}=\mathbf{0},\ \mathbf{c}=\mathbf{0},\ \mathbf{d}=\begin{bmatrix}1\\1\end{bmatrix},\ \mathbf{r}_1=\begin{bmatrix}2\\0\end{bmatrix}$$

$$\mathbf{Q}_0=\begin{bmatrix}1&0.5\\0.5&0.5\end{bmatrix},\ \mathbf{Q}_1=\begin{bmatrix}-1.5\\0.5\end{bmatrix},\ \mathbf{Q}_2=\mathbf{0}$$

$$\mathbf{C}=\begin{bmatrix}0\\0\\-2\end{bmatrix},\ \mathbf{D}=\begin{bmatrix}-1&0\\0&-1\\2&1\end{bmatrix},\ \mathbf{r}_2=\begin{bmatrix}0\\0\\1\end{bmatrix}$$

and the important parameters for Example 2. are as follows:

$$\mathbf{P}=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&1&0&-0.5\\0&0&0&0&0\\0&0&-0.5&0&1\end{bmatrix},\ \mathbf{A}=\begin{bmatrix}-1&0\\0&-1\\1&1\end{bmatrix},\ \mathbf{B}=\mathbf{0}$$

$$\mathbf{a}=\begin{bmatrix}-7\\4\end{bmatrix},\ \mathbf{b}=\begin{bmatrix}0\\-4\\0\end{bmatrix},\ \mathbf{c}=\mathbf{0},\ \mathbf{d}=\begin{bmatrix}1\\1\\0\end{bmatrix},\ \mathbf{r}_1=\begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\mathbf{Q}_0=\begin{bmatrix}1&0.5&0\\0.5&0.5&0\\0&0&0.5\end{bmatrix},\ \mathbf{Q}_1=\begin{bmatrix}-1.5&0\\0&0.5\\0&0\end{bmatrix},\ \mathbf{Q}_2=\mathbf{0}$$

$$\mathbf{C}=\begin{bmatrix}0&0\\0&0\\0&0\\1&-2\end{bmatrix},\ \mathbf{D}=\begin{bmatrix}-1&0&0\\0&-1&0\\0&0&-1\\2&1&-1\end{bmatrix},\ \mathbf{r}_2=\begin{bmatrix}0\\0\\0\\-2\end{bmatrix}$$

We execute the proposed algorithm on each of the above 2 problems using the MATLAB 7.14. During the simulations, we adopted values for the following parameters as: feasible population size: 20, infeasible population size: 20, crossover probability: $p_c = 0.7$, mutation probability: $p_m = 0.2$, maximal number of generations: $MaxGen = 5$, and the K for the meta-controlled Boltzmann machine is 0.3.

Table I shows that the results found by the proposed algorithm are better than or equal to those compared algorithms in the literature. For Examples 1, the proposed algorithm can find much better solution. For Example 2, the results are the same. Fig 2. and Fig. 3 show the transient behavior of upper level value of Example 1 and Example 2 respectively. We can see from those figures, by using the IGA, the upper level value will be searched in a very wide range, which will effectively avoid in sticking into local optimal. On the other hand, by using the meta-controlled Boltzmann machine, the proposed method performs pretty well in terms of the resulting solution accuracy, as well as efficiency of the global convergence.

TABLE I. COMPARISONS OF THE RESULTS BY THE PROPOSED ALGORITHM AND THE RESULTS IN THE REFERENCES

Example literature	Result by the proposed algorithm		
	$(\mathbf{x}^*, \mathbf{y}^*)$	$F(\mathbf{x}^*, \mathbf{y}^*)$	$f(\mathbf{x}^*, \mathbf{y}^*)$
Example 1	$(0.8402, 0.7603, 0)$	-2.0768	-0.5781
Example 2	$(0.6100, 0.3900,$ $0, 0, 1.83)$	0.6389	1.6744

Example literature	Result in the reference [19]		
	$(\mathbf{x}^*, \mathbf{y}^*)$	$F(\mathbf{x}^*, \mathbf{y}^*)$	$f(\mathbf{x}^*, \mathbf{y}^*)$
Example 1	$(1.0, 0, 1.0)$	-2	2.5
Example 2	$(0.6100, 0.3900,$ $0, 0, 1.83)$	0.6389	1.6744

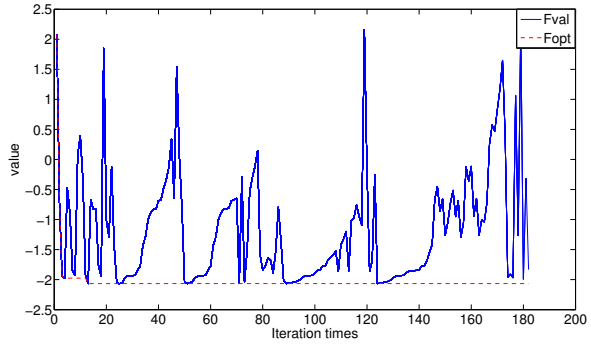

Fig. 2. The transient behavior of upper level value in Exampe 1.

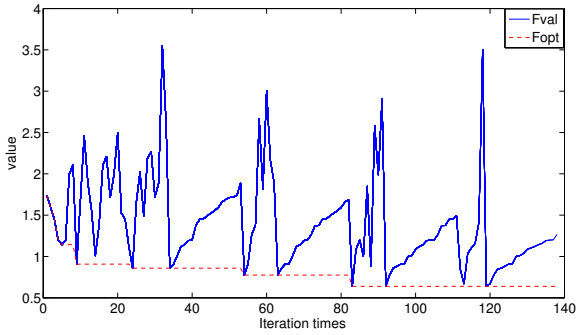

Fig. 3. The transient behavior of upper level value in Exampe 2.

VI. CONCLUSIONS

This paper presents an improved genetic algorithm based double layer neural network for solving quadratic bilevel programming problem. In the upper level of bilevel programming problem, we proposed an intelligent genetic algorithm that is effective in global searching, and we transform the lower level into the energy function of meta-controlled Boltzmann machine which converges very efficiently. The proposed hybrid intelligent algorithm is showed to be very efficient from computational point of view and quality of solutions. By using the token, we could conclude that proposed method is capable of solving the quadratic bilevel programming problem. Further more, the approach is easy to use and has universal applicability to solve such bilevel programming problem. Simulation results also show that the proposed method is very efficient and robust in terms of computational cost, quality of solutions

and success rate.

ACKNOWLEDGMENT

This work was supported partly by Grants-in-Aid for Science Research MEXT (No23500289) and also in part under a Waseda University Grant for Special Research Projects (Project number 2014A-040).

REFERENCES

[1] Z. H. Gümüs, C. A. Floudas, "Global optimization of nonlinear bilevel programming problems." *Journal of Global Optimization,* 2001, 20: 1-31.

[2] L. D. Muu, N. V. Quy, "A global optimization method for solving convex quadratic bilevel programming problems." *Journal of Global Optimization,* 2003, 26: 199-219.

[3] T. Edmunds, J. F. Bard. "Algorithms for nonlinear bilevel mathematical programming." *IEEE Trans. on Systems, Man, and Cybernetics,* 1991, 21: 83-89.

[4] M. Carrion, J. M. Arroyo, A. J. Conejo, "A bilevel stochastic programming approach for retailer futures market trading." *IEEE Transactions on Power Systems* , 24(3), 1446-1456 (2009)

[5] K. Shimizu, Y. Ishizuka, J. F. Bard, "Nondifferentiable and two-level mathematical programming." Kluwer Academic, Boston (1997).

[6] B. Colson, P. Marcotte, G. Savard, "Bilevel programming : A survey." *International Journal of Operations Research* 3(2), 87-107 (2005)

[7] J. F. Bard, J. T. Moore, "A branch-and-bound algorithm for the bilevel programming problem." *SIAM Journal on Scientific and Statistical Computing* 11(2), 281-292 (1990)

[8] F. Al-Khayyal, R. Horst, P. Paradalos, "Global optimization on concave functions subject to quadratic constraints: an application in nonlinear bilevel programming." *Annals of Operations Research* 34(1), 125-147 (1992)

[9] T. Edmunds, J. F. Bard, "An algorithm for the mixed-integer nonlinear bilevel programming problem." *Annals of Operations Research* 34(1), 149-162 (1992)

[10] G. Savard, J. Gauvin, "The steepest descent direction for the nonlinear bilevel programming problem." *Operations Research Letters* 15(5), 265-272 (1994)

[11] L. Vicente, G. Savarg, J. Judice, "Descent approaches for quadratic bilevel programming." *Journal of Optimization Theory and Applications* 81(2), 379-399 (1994)

[12] S. R. Hejazi, A. Memariani, G. Jahanshanloo, M. M. Sepehri, "Bilevel programming solution by genetic algorithms." *Proceeding of the First National Industrial Engineering Conference* (2001)

[13] C.P. Wu, "Hybrid technique for global optimization of hierarchical systems." *Proceedings of the IEEE International Conference on Systems, Man and Cybernatics,* vol. 3, pp. 1706-1711 (1996)

[14] H. S. Shih, U. P. Wen, "A neural network approach for multi-objective and multi-level programming problems." *Journal of Computer and Mathematics with Applications* 48(1-2), 95-108 (2004)

[15] K. M. Lan, U. P. Wen, "A hybrid neural network approach to bilevel programming problems." *Applied Mathematics Letters* 20(8), 880-884 (2007)

[16] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." *PNAS*, vol. 79, no. 8, pp. 2554-2558, 1982.

[17] D.H. Ackley, G.E. Hinton, T.J. Sejnowski, "A Learning Algorithm for Boltzmann machine." *Cognitive Science*, vol. 9, pp. 147-169, 1985.

[18] T. Watanabe and J. Watada, "A Meta-Controlled Boltzmann machine for Rebalancing Portfolio Selection." *Central Eur. J. Oper. Res.* vol. 12, no.1, pp. 93-103, 2004.

[19] Y. Lv, Z. Chen, Z. Wan, "A neural network for solving a convex quadratic bilevel programming problem." *Journal of Computational and Applied Mathematics* 234 (2010) 505-511, Elsevier.

[20] J. F. Bard, *Practical bilevel optimization: Algorithm and applications.* Kluwer Academic Publishers Dordrecht (1998).