Neural Network Processing Through Energy Minimization with Learning Ability to the Multiconstraint Zero-One Knapsack Problem

Hahn-Ming Lee and Ching-Chi Hsu

Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, R. O. C.

ABSTRACT

In this paper, we first define a neural network model NNCO for the multiconstraint zero-one knapsack problem and then present a methodology to approximately solve this problem in near real-time by use of the characteristic that many neural networks can be shown to minimize an energy function defined for the networks. In addition, to illustrate the process of the neural network model to approximately solve the multiconstraint zero-one knapsack problem, we overcome the difficulty that there is no guidance available as to what values the parameters ought to take by designing an algorithm ADJUSTPAR to automatically adjust the values of the parameters used in the "energy" function. Moreover, we compare the methodology we present with related works and demonstrate its significant improvements. We also simulate the neural network model with several appropriate activation functions to approximately solve a set of eleven relatively large and difficult multiconstraint zero-one knapsack optimization problems from literature with well-known optimum solutions. The result of the simulation demonstrates how the methodology presented here can work well to the multiconstraint zero-one knapsack problem and can be easily extended to solve other combinatorial optimization problems.

I. Introduction

It is believed that the multiconstraint zero-one knapsack problem is too complex to be solved in polynomial time[8]. This problem can be defined as:

$$\text{Maximize} \quad MP = \sum_{i=1}^{n} P_i * X_i$$

$$\text{Subject to} \quad \sum_{j=1}^{n} A_{ij} * X_j \le R_i \qquad (i = 1,..,m)$$

$$X_j \in \{0, 1\} \qquad (j = 1,..,n).$$

Without loss of generality, all R_i, A_{ij}, and P_i are assumed to be nonnegative.

Researches in this area aim at searching for the good solutions efficiently. In general, what is truly desired is to generate a feasible solution with values close to the values of an optimum solution in a reasonable amount of time. But for larger number of constraints there does not exist any reasonable fast exact algorithm and several proposed heuristics depend very much upon the statistical distribution of the coefficients[4]. In addition, these heuristics lack of the property of parallel processing. Neural network models typically consist of many simple neuron-like processing elements interacting via weighted connections. Each unit has a state value representing its state that is determined by the inputs received from the other units in the network and from externals. The nature of the neural network models is determined principally by their connectivity and by taking boolean or real valued variables[13,14,15,16,21,23]. Making use of these collective computational capabilities, neural network models hold promise for developing feasible solutions to those problems that are computationally intense. Besides, many neural network models are well suited for parallel implementation either in hardware [1,5,6,10,12,20] or in software[3,27,28]. So it goes without saying that we can use the properties of neural networks to propose a methodology which can do well even in conventional computers.

In this paper, we first define a neural network model called NNCO for the multiconstraint zero-one knapsack problem and then present a methodology based on NNCO to approximately solve this problem in near real-time by use of the characteristic that many neural networks can be shown to minimize an energy function defined for the networks[14,15]. The NNCO is modeled as a sparse network of neurons with two different types. One neuron type is called variable(master) nodes which activate autonomously and the other type is called auxiliary(slave) nodes which are activated by variable nodes. The variable nodes represent the desired result and the auxiliary nodes just do some computations for the variable nodes. These neurons of the same type can activate simultaneously and asynchronously. This model is inherently parallel in that many nodes of the same type can carry out their computations at the same time. Due to the property of asynchronous parallel processing, the model can easily map to parallel architectures.

The methodology presented here consists of four stages. First, formulate problem's constraints and desired optima by a specific mathematical function called "energy" function. This function must mathematically well characterize the desired problem and must have the feature that the stable states of the network model correspond to the "best" solutions of the problem. Second, map the problem to the neural network model, i.e. define each component of NNCO for the problem. Third, choose an appropriate activation function that can make the "energy" function be bounded and decreased in this neural network model. This shows the network will converge. Fourth, adjust parameters. Because there exists several parameters in the "energy" function, it is important to appropriately set the values of these parameters. It is assumed that there is no guidance available as to what values the parameters ought to take[30]. Here we design an algorithm ADJUSTPAR to automatically adjust the values of these parameters and find it is well suited for the multiconstraint zero-one knapsack problem. This operation of parameter adjusting can be regarded as the modification of connections in neural network models. It acts like unsupervised learning. Using the methodology presented above, the "programmer" can solve the multiconstraint zero-one knapsack problem by knowing what the problem is rather than how to solve it. It matches the claim that neural networks will eliminate or reduce the job of computer programmers[2].

In addition to the methodology presented above, we describe how to add heuristic rules to the neural network model NNCO in order to guide the network to find good solutions rapidly. Moreover, we compare the methodology we present with related works and demonstrate its significant improvements. We also simulate the neural network model with several appropriate activation functions to approximately solve the multiconstraint zero-one knapsack problem. Using this methodology, we solve a set of eleven relatively large and difficult test problems from literature with well-known optimum solutions. The result of the simulation demonstrates how the methodology based on NNCO presented here can work well to the multiconstraint zero-one knapsack problem and can be easily extended to solve other combinatorial optimization problems.

II. The Neural Network Model NNCO

It is helpful to begin with an analysis of our neural network model, called NNCO, for the multiconstraint zero-one knapsack problem and then describe the various specific assumptions we make. The NNCO is modeled as a sparse network of neurons with two different types. One neuron type is called variable(master) nodes which activate autonomously and the other type is called auxiliary(slave) nodes which are activated bv variable nodes. The variable nodes represent the desired result and the auxiliary nodes just do some computations for the variable nodes. These neurons of the same type can activate simultaneously and asynchronously. Note that there is no difference between the terms neuron and node.

The model can be formally described as an eight-tuple$(G,W,S,I,B,\lambda,\delta,\sigma)$. Here G is a directed graph with nodes $V=\{1,2,..,n,n+1,..,n+m\}$ and edges $E \subseteq V \times V$, where n, m are the number of variable nodes and auxiliary nodes respectively. All of the processing of the model is carried out by these nodes. These nodes are only relatively simple units, each doing its own relatively simple job. A node's job is simply to receive inputs from its neighbors, external input and itself, and then to compute an output value. This model is inherently parallel in that many nodes of the same type can carry out their computations at the same time.

Each edge $(i,j) \in E$ has weight $W_{ji} \in W$ indicating the strengths of the connections, where W is a set of real numbers and edge (i,j) denotes the edge from node i to node j. If edge $(i,j) \notin E$, then $W_{ji} = 0$. In this model we assume that each node provides an additive contribution to the input of the nodes to which it is connected. The total input of each node from its neighbors is simply the weighted sum of the separate input from each of the individual nodes.

In addition, to the set of nodes and weights, we need a representation of the state for the system. This is primarily specified by a vector of real numbers S to represent the pattern of activations over the set of nodes. It is useful to see the processing of the model as the evolution of a pattern of activations over the set of nodes.

The initial condition of the network model is described by vector I. Each element of the vector has a list with two values which stand for the external input and the initial state value for one of the nodes. Vector B, showing the importance of these nodes, is the strengths of nodes. That is used to indicate the strength of each node. The type function λ denotes the node's type, that is

$$\lambda : V \rightarrow \{\text{'variable'}, \text{'auxiliary'}\}$$

We also need transition functions δ and σ to characterize how the states of variable nodes and auxiliary nodes are updated. These functions consist of two steps. To get the net input of the nodes first and then apply the appropriate activation function to get new states of the nodes. Formally, if S_i is the state of variable node i, then

$$S_i := \delta\,(\,S_i,\ \Sigma\{W_{ij}*S_j \mid \text{edge}(j,i)\in E,\ j \neq i\},\ B_i,\ I_i\,)$$

and if S_i is the state of auxiliary node i, then

$$S_i := \sigma\,(\,S_i,\ \Sigma\{W_{ij}*S_j \mid \text{edge}(j,i)\in E,\ j \neq i\},\ B_i,\ I_i\,)\,.$$

Where W_{ij} is the weight from node j to node i, B_i is strength of node i, and I_i is the initial element of node i.

From the above description, this model has the following characteristics which are proposed by Newell[11] for a computational model, and can be easily extended to other combinatorial optimization problems.

(1) All processing in the model is local.
(2) The processors(neurons) are limited in their information capacity.
(3) The bandwidth of communication between the processors is limited.
(4) The model is hierarchical.
(5) The model is massively parallel.

III. Methodology to the Multiconstraint Zero-One Knapsack Problem

The major concept used in the proposed methodology is that many neural networks can be shown to minimize an energy function. We suppose that if we can formulate an "energy" function for the multiconstraint zero-one knapsack problem in which the "lowest" "energy" states correspond to the "best" solutions, then we can make use of the properties of the speed and the computational power of neural networks to rapidly get good solutions. In addition, we propose an algorithm ADJUSTPAR to automatically adjust the strengths of the terms used in the "energy" function which describes the behavior of the desired problem, although it is assumed that there is no guidance available as to what values the parameters ought to take[30].

The neural network model used in the methodology is the NNCO. This model has been described in the previous section. At this point, we will describe the methodology to the multiconstraint zero-one knapsack problem in four stages as follows:

Stage 1. Define the "energy" function

It means we must formulate problem's constraints and desired optima by a specific mathematical function called "energy" function. This function must mathematically well characterize the behavior of the problem and must have the feature that the stable states of the network model correspond to the "best" solutions of the problem. This is what we said knowing what the problem is instead of knowing how to solve it.

We can derive the "energy" function of the multiconstraint zero-one knapsack problem in which the "lowest " "energy" states correspond to the "best" solutions. This can be separated into three terms. First, the "energy" function must favor strongly stable states to get the maximal profits and an appropriate form for this requirement can be chosen as :

$$\sum_{i=1}^{n} P_i - \sum_{i=1}^{n} P_i * X_i \qquad (1)$$

This term is equal to zero if all X_i equal one or otherwise it is greater than zero. From this, we can guarantee this term of the "energy" function is greater than or equal to zero and has "lowest" value while getting the maximal profits.

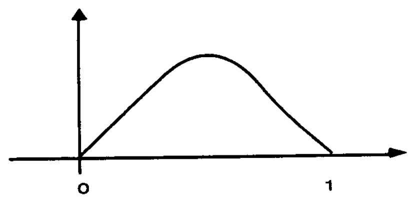
The second term contains information to match the constraints of the multiconstraint zero-one knapsack problem. We use a function F such that the value of F will be larger if the constraints are violated more. So this term can be written down as:

$$\sum_{i=1}^{m} F\left(\sum_{j=1}^{n} A_{ij} * X_j - R_i \right) \qquad (2)$$

If the constraints are not violated any more, then this term ought to equal zero. Thus we will define $F(z) = 0$ if $z \leq 0$ and $F(z) > 0$ if $z > 0$. In addition, we must make the value of this function increase when the value of z increases in order to meet the requirement that the value of function F will be larger if the constraints are violated more. The detail of this function will be described later. From above description, this term is also greater than or equal to zero.

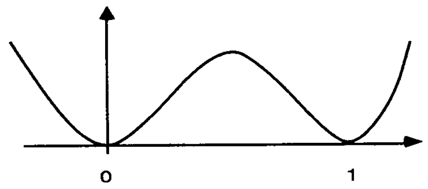
The last term is defined to emphasize the zero/one property of this problem. It has minimal value when, for each i, $X_i = 1$ or $X_i = 0$. Two appropriate forms and their corresponding graphs for this term are as follows:

$$\sum_{i=1}^{n} X_i * (1 - X_i) \qquad 0 \leq X_i \leq 1 \qquad (3)$$



or

$$\sum_{i=1}^{n} (X_i^4 - 2 * X_i^3 + X_i^2) \qquad X_i \in R \qquad (4)$$



The two equations can be easily derived from their corresponding graphs. For flexibility, i.e. $X_i \in R$, we use the equation (4) in our implementation.

Now we can write down the "energy" function of the multiconstraint zero-one knapsack problem as the sum of Eq. (1), Eq. (2), and Eq. (4) :

$$E = \alpha * \left(\sum_{i=1}^{n} P_i - \sum_{i=1}^{n} P_i * X_i \right)$$

$$+\beta * \sum_{i=1}^{m} \left(F\left(\sum_{j=1}^{n} A_{ij}*X_j - R_i \right)\right) + \gamma * \sum_{i=1}^{n} (X_i^4 - 2*X_i^3 + X_i^2)$$

Where α, β, γ are nonnegative constants and are used as the strengths of these terms. They are applied to denote how important each term is in the problem.

All the three terms in the "energy" function E can be easily verified to be greater than or equal to zero, thus E is bounded and greater than or equal to zero.

We can define the "energy" function in other forms, e.g. first use Lagrangean method (function)[25] to eliminate the constraints, then define the "energy" function of the transferred problem.

Stage 2. Map the problem to the neural network model

Now we can define each component of NNCO for the multiconstraint zero-one knapsack problem . We first define a variable node of the neural network model NNCO for each variable and an auxiliary node for each constraint. The other important thing that must be considered is to make the variable nodes converge to some stable states in which we can get the "lowest" "energy" and the good profits. Thus the operation can be defined as follows. The net input of each variable node is derived by the change of the "energy" function due to changing the state of this variable node. Then apply an appropriate activation function which meets the requirement that the "lowest" "energy" states correspond to the "best" solutions.

From above description, the net input of variable node i in the multiconstraint zero-one knapsack problem is derived by the change $\triangle E$ in E due to changing the state of node i by $\triangle X_i$, that is

$$\begin{aligned} net_i &= -\partial E / \partial X_i \\ &= \alpha * P_i - \beta * \sum_{j=1}^{m} [A_{ji} * f(\sum_{k=1}^{n} A_{jk} * X_k - R_j)] \\ &\quad - \gamma * (4 * X_i^3 - 6 * X_i^2 + 2 * X_i) \end{aligned}$$

where net_i is the net input of variable node i and $f(z) = \partial F(z) / \partial z$. Here we define function f as:

$$f(z) = 0 \qquad \text{if } z \le 0$$
$$f(z) = z \qquad \text{if } z > 0$$

The function f will be a larger positive value when the corresponding constraint equation it represents is being violated more. This definition makes function F meet the requirement that the value of function F is larger if the constraints are violated more and is zero if no constraint is violated. That is we can get $F(z) = 0$ if $z \le 0$ and $F(z) = z^2 / 2$ if $z > 0$.

The function f can be other forms also. The only requirement is to make the term of the "energy" function it represents be a larger positive value when the corresponding constraint is violated more and be a smaller value when the corresponding constraint is met. This operation is accomplished by the auxiliary nodes.

Before describe each component of the neural network model NNCO for the multiconstraint zero-one knapsack problem formally, let's sketch the model for this problem as the following figure.

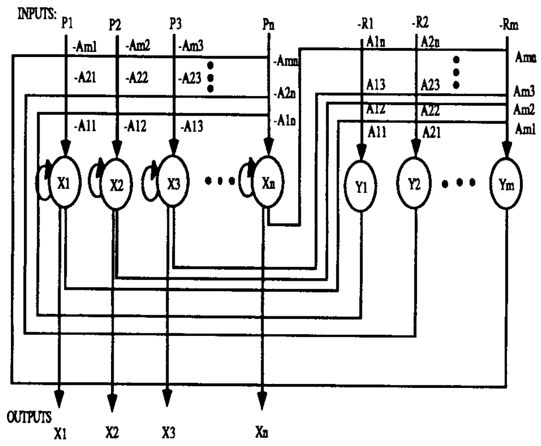


Figure : neural network model for multiconstraint zero-one knapsack problem

This figure is derived from the formula of the net input of the variable node i. A node's job is simply to receive inputs from its neighbors, external input, and itself, and then to compute an output value. Each node provides an additive contribution to the inputs of the node to which it is connected. The total input of each node from its neighbors is simply the weighted sum of the separate input from each of the individual nodes. In this figure, we define a variable node for each variable and an auxiliary node for each constraint. The external input to the variable node X_i is P_i and to the auxiliary node Y_i is $-R_i$. The weight from variable node X_j to auxiliary node Y_i is A_{ij} and from auxiliary node Y_i to variable node X_j is $-A_{ij}$. In addition, the variable nodes have self-connection. Note that P_i, R_i, and A_{ij} are the coefficients of the multiconstraint zero-one knapsack problem defined previously.

At this moment, we can formally define the eight-tuple$(G,W,S,I,B,\lambda,\delta,\sigma)$ of the NNCO for the multiconstraint zero-one knapsack problem. We define a variable node for each variable and an auxiliary node for each constraint. The graph G and the weight W are as the figure showed above. The weights of the edges are zero if the corresponding edges do not present in this figure. The state S is set binary value or bounded continuous value in our simulation. P_i is the input of node i (named X_i in the figure), i=1,..,n and the input of node n+i (named Y_i in the figure), i=1,..,m is $-R_i$. Initially we set the state of each node to be zero. The bias B is set depending on the problem instances and heuristics. It is used in transition function δ. The type function λ is defined as: λ(i)='variable' for i=1,..,n and λ(i)='auxiliary' for i=n+1,..,n+m. Formally, if S_i is the state of variable node X_i, the transition function δ is formulated as:

$$S_i = 0 \quad \text{if } B_i = 0$$
$$S_i = 1 \quad \text{if } B_i = 1$$
$$S_i = \text{ACT\_FUN}(\alpha * I_i + \beta * \sum_{j \ne i} (W_{ij} * S_j) + \gamma * G_i)$$
otherwise

Where $G_i = -4 * S_i^3 + 6 * S_i^2 - 2 * S_i$ and ACT_FUN is the appropriate activation function used.

Finally, if S_i is the state of auxiliary node Y_i, the transition function σ is defined as:

$$S_i = f(\sum (W_{ij} * S_j) + I_i),$$ where the function f is defined above.

For clarity, we describe the whole operation of the network. As the definition of NNCO, the variable nodes are autonomous and can activate simultaneously. When the variable nodes activate, they trigger all the auxiliary nodes and then update the states themselves. The operation continues until the network leads to one of the stable states that do not further change with time. At the moment, the states of the variable nodes are the result of the desired problem.

Stage 3. choose an appropriate activation function

Convergence to stable states is the essential feature of neural network models. In the previous stage we have formulated the net inputs of the variable nodes as the change of the "energy" function due to changing the states of the variable nodes. If we can choose the activation functions that can make the change of states of the variable nodes in the same direction as their net inputs, then any variable node can always behave so as to decrease the "energy" function. Moreover, since the "energy" function is bounded, the process of the network will lead to stable states that do not change any more.

In the multiconstraint zero-one knapsack problem, we formulate $net_i = - \partial E / \partial X_i$, i.e. $\triangle E = - net_i * \triangle X_i$. From this formula, we can choose an activation function that can make the change of the value of variable X_i in the same direction as the net input of X_i to make any change of "energy" function E be negative. There are many activation functions that match the requirement[23] and we will describe some of them in our simulation later. Since E is bounded, the process of the neural network model will lead to stable states that do not further change with time.

Stage 4. Adjust parameters.

In the proposed methodology we define an "energy" function for the multiconstraint zero-one knapsack problem and it exists several parameters that need to be selected to produce a sensible computation in the "energy" function. The parameters have significant influence on getting feasible solutions. For example, if we get an answer that violates the constraints of the problem, then we must increase the strengths of the constraints used in the "energy" function. This step must be repeated until we feel good for the answer responded. But if we make the values of the strengths of the constraints too large, the effect of the other terms in the "energy" function will disappear. In addition, the parameters suited for one problem instance may not suit for others.

In section four of Wilson and Pawley[30], it was stated that as there is no guidance available as to what values the parameters ought to take, several runs were made varying the values of the operating parameters. But here we will present an algorithm ADJUSTPAR which can automatically adjust the parameters to achieve valid and good solutions. The operation of parameter adjusting can be regarded as automatic modification of the connection weights in the neural network model. It acts like unsupervised learning.

Before describe the algorithm, we first separate the terms of the "energy" function into three categories: (1) Profit which is used to describe the objective of the problem, (2) Constraint which is used to describe the behavior of the conditions that can not be violated any more, (3) Restraint which is used to illustrate these conditions that can violate a few. From this definition, we can write down the "energy" function E in general as

$$E = \alpha' * \text{Profit} + \beta' * \text{Constraint} + \gamma' * \text{Restraint}.$$

The idea of the algorithm ADJUSTPAR is to assume that the larger α' is the better solutions we will get. But we can not violate any term in Constraint. The operation of this algorithm is to make β' and γ' be constant and to modify the value of α' continuously until the network converges with maximum α' to valid solutions which mean no term in Constraint is violated. If the network converges to a state with an invalid solution then decrease α' in proportion to the magnitude of the Constraint violated. The algorithm is described as follows:

```
Algorithm ADJUSTPAR
  Step0. Set decreasing rate EPSILON and tolerant error
         ERROR.
  Step1. Calculate the maximum value of the terms of
         Constraint that can be violated in the net input and
         then let it be M.
  Step2. Set FIRST = true and PREV_ $\alpha'$ = 2 * $\alpha'$ .
         /* if $\alpha'$ is initially set too small then double it */
  Step3. Activate the network until it converges.
  Step4. Calculate the value of terms of Constraint that are
         violated in the net input and let it be M'.
  Step5. Set $\theta$ = EPSILON * M' / M.   /* the proportion of
         Constraint violated */
  Step6. If $\theta$ = 0.0 then {      /* get a valid solution */
             EPSILON = EPSILON / 2;
             /* make $\alpha'$ be decreased less */
         $\alpha'$ = PREV_ $\alpha'$ ;
         if (FIRST)
         /* if the network never converges to an invalid */
             PREV_ $\alpha'$ = 2 * $\alpha'$ ;
  /* solution then double $\alpha'$, otherwise set $\alpha'$ to be */
  /* its previous value that gets an invalid solution */
         }
  Step7. If $\theta$ > 0.0 then {      /* get an invalid solution */
             FIRST = false;
             PREV_ $\alpha'$ = $\alpha'$ ;
     /* keep the value of $\alpha'$ that gets an invalid solution */
             $\alpha'$ = $\alpha'$ * ( 1.0 - $\theta$ );
     /* decrease $\alpha'$ in proportion to the magnitude of */
         /* the Constraint violated */
         }
  Step8. If EPSILON > ERROR then goto step3.
  Step9. Stop.
```

The variable PREV_ α' is used to make α' larger if it is set too small initially and to make it back to the previous value that gets an invalid solution. The latter means we may decrease α' more so as to make it get a valid solution, thus we must go back and decrease α' less. The decrease of α' is in proportion to the ratio of magnitude of Constraint violated to the maximum magnitude of Constraint can be violated controlled by decreasing rate EPSILON.

The algorithm must be executed several times, with different random number sets which make the network activate in different ways, to get the better parameter α' which always makes us get better and valid solutions. In our simulation, we find this algorithm is very suited for the multiconstraint zero-one knapsack problem.

IV. Simulation Results

The multiconstraint zero-one knapsack problem is known to be NP-hard[8], thus a good algorithm for its optimum solution is very unlikely to exist. In this paper, we use the concept of the neural network models to efficiently derive good solutions. In order to demonstrate how well the proposed methodology is, a simulation is conducted on PC/AT using Microsoft C. We solve a set of eleven relatively large and difficult test problems from the literature[22,24,29] with well-known optimum solutions. Each problem has been solved ten times with different random number sets.

In this experiment of the operation of the network, we first select variable nodes at random to make it similar to asynchronous parallel processing and then compute their net inputs. After computing the net inputs of variable nodes, the state values of these nodes are then determined according to an appropriate activation function which meets the requirement that it can make the change of the value of variable X_i in the same direction as the net input of X_i. Several variants are available as follows:

(1) Hard limit :

$X_i = 1$ if $net_i \geq 0$
$X_i = 0$ if $net_i < 0$

(2) Threshold logic :

$X_i = X_i + net_i$
if $(X_i > 1)$ then $X_i = 1$
if $(X_i < 0)$ then $X_i = 0$

(3) Schema model

$X_i = X_i + net_i * (1 - X_i)$ if $net_i \geq 0$
$X_i = X_i + net_i * X_i$ if $net_i < 0$
if $(X_i > 1)$ then $X_i = 1$
if $(X_i < 0)$ then $X_i = 0$

(4) Unbounded threshold logic

$X_i = X_i + net_i$

(5) Unbounded schema model

$X_i = X_i + net_i * (1 - X_i)$ if $net_i \geq 0$
$X_i = X_i + net_i * X_i$ if $net_i < 0$

(6) Sigmoid :

$X_i = 1 / (1 + \exp(-net_i/tp))$
where tp is a nonnegative constant

(7) Stochastic :

$P (X_i = 1) = 1 / (1 + \exp(-net_i/T))$
where T is a nonnegative constant and $P (X_i = 1)$ means the probability that the variable X_i equals one.

The activation functions used in our implementation are chosen to make the change of the state values of variable nodes be in the same direction as the net inputs of these nodes. In activation functions (1) to (6), the value of the "energy" function continues to decrease until all variable nodes converge to some fixed values. This is a hill-climbing procedure that simply ensures the system will find the local optimal solutions. The activation function (7) is used to reduce the deficiency. The parameter T is slowly decreased. This is the same activation function used in Boltzmann machine[13,21,23]. It overcomes the local optimal solutions and gets a near optimum solution, but it spends more time. In addition, if the parameter T is decreased quickly, then it may get a worse local optimal solution.

From our experiment, we find the activation functions (1) to (5) will always get the same local optimal solutions in an almost the same short time, and the activation functions (6) and (7) will always get the better local optimal solutions in a longer time. In activation function (6), when the network converges, we choose the variable nodes one by one according to the state values until any one of the constraints is violated. These chosen variable nodes are the result of the desired problem. We also find the values of parameters α, β, and γ are more important when we use the activation functions (4) and (5). Their values should be smaller in the two activation functions, otherwise the system will overflow. Due to this observation, we think it is better to add a constraint into the requirement of the choice of activation functions. That is, in addition to the requirement described above, we had better select these activation functions that are bounded.

In this simulation we calculate the following statistics for each problem selected from literature and each activation function used.

(1) Average deviation from true optimum, i.e. $e^* = (\Sigma e_i) / 10$ with : $e_i = (MP_i^* - MP_i) / MP_i^*$, $i=1,..,10$ where MP_i is the profit obtained at run i by the method considered and MP_i^* is the true optimum profit for the same problem instance. Note that $e_i \geq 0$.
(2) Average standard deviation of deviation s defined by : $s = [\Sigma (e_i - e_i^*)^2 / 9]^{1/2}$, $i = 1,..,10$.
(3) Maximum deviation from true optimum, i.e. $e_{max} = Max_{i=1,..,10}\, e_i$.
(4) Average CPU times required to solve one problem by each method in seconds.

Table I shows the first three statistics for each problem and activation function used. In the first column the problems are identified with their authors' name. Column two and three give the dimensions of each problem. Column four to six show the average deviation from true optimum for the activation functions used. The activation functions (1) to (3) always get the same result, so we show them in the same column. The system easily overflows while using activation functions (4) and (5), so the two activation functions are not used in this experiment. Column seven to nine show the average standard deviation of deviation and column ten to twelve indicate the maximum deviation from optimum solution.

The last column gives the parameters used in each problem. The values of parameter α used in the problems of Petersen1 and Weingartner1 are obtained by the algorithm ADJUSTPAR which is described in stage 4 of the methodology presented above. Initially we give the value of α to be 200 for the Petersen1 problem and to be 10 for the Weingartner1 problem. After the termination of the algorithm we get the values of α to be 2.98 and 1.71 respectively. The function of parameter T used for stochastic activation function in Petersen's problems is $T_t = T_0 / (1+t)$ where T_0 is the initial value of parameter T. When T is used in the other problems, it is linearly interpolation from T_0. The time t used in stochastic function is measured in the number of cycles. Note that in this column we just show these parameters that are changed.

The result of table I suggests that deterministic activation functions in general do not produce as good solutions as stochastic activation function. But if the parameter T is decreased too fast, the stochastic activation function may get a worse result.

For comparative purpose we design a branch-and-bound algorithm which is similar to Gavish and Pirkul[9] to get the time for obtaining the optimum solution. Table II shows the result. This result demonstrates our methodology works rapidly and well on problems with arbitrary problem dimensions, i.e. the time needed is not exponential as the branch-and-bound method. In addition, we can find our methodology needs short execution time even in the conventional personal computer.

In short, we can confirm from the results of the simulation, the methodology presented here can be used to get near optimum solutions rapidly for the multiconstraint zero-one knapsack problem. Moreover, because the execution time is short, we can start the desired problem several times with different random number sets to get almost all the optimal solutions.

Table I: Results of selected literature test problems[22,24,29,31]

Problem	variable number	constraint number	average deviation from true optimum: activation function (1)-(3)	(6)	(7)	average standard deviation of deviation: activation function (1)-(3)	(6)	(7)	maximum deviation from optimum: activation function (1)-(3)	(6)	(7)	comment
petersen 1	6	10	0.166	0	0.005	0.172	0	0.010	0.368	0	0.026	$\alpha=2.98$ $\beta=30$
2	10	10	0.190	0.166	0.080	0.116	0.100	0.085	0.330	0.251	0.240	$\gamma=1$
3	15	10	0.122	0.070	0.111	0.041	0.008	0.096	0.177	0.188	0.248	tp=1000
4	20	10	0.228	0.190	0.212	0.101	0.088	0.111	0.433	0.288	0.333	To=1000
5	28	10	0.230	0.172	0.162	0.085	0.070	0.091	0.360	0.295	0.306	
6	39	10	0.087	0.080	0.170	0.090	0.090	0.098	0.240	0.222	0.270	$\alpha=50$ tp=4000
7	50	5	0.130	0.130	0.091	0.060	0.050	0.046	0.240	0.240	0.200	To=4000
Weingartner 1	28	2	0.166	0.107	0.050	0.117	0.030	0.040	0.340	0.148	0.119	$\alpha=1.71$ $\beta=6$
2	105	2	0.050	0.033	0.018	0.025	0.025	0.007	0.100	0.070	0.030	tp=1000 To=5000
Senju, Toyoda 1*	60	30	0.307	0.150	0.070	0.140	0.048	0.040	0.480	0.200	0.148	T30=150
2*	60	30	0.227	0.111	0.057	0.108	0.033	0.033	0.340	0.151	0.111	$\alpha=500$ $\beta=1$ tp=5000

* The optimum solutions, which do not report in the paper of Senju and Toyoda, are obtained from[31].

Table II: Average CPU times required to solve one problem by each method(in seconds)*

Problem	variable number	constraint number	branch and bound	methodology presented here: activation function(1)-(3)	activation function (6)	activation function (7)
Petersen 1	6	10	0.12	0.17	0.24	0.38
2	10	10	0.78	1.01	1.32	1.80
3	15	10	21.87	1.89	2.97	8.01
4	20	10	482.79	5.45	11.70	9.90
5	28	10	7571.91	11.88	22.20	12.47
6	39	5	90730.51	17.57	23.94	20.30
7	50	5	?**	30.29	42.64	46.00
Weingartner 1	28	2	1660.16	5.14	16.16	26.49
2	105	2	?**	54.14	84.60	199.66
Senju, 1	60	30	?**	199.94	307.60	429.36
Toyoda 2	60	30	?**	207.63	253.77	446.02

* The computer in simulation is IBM PC/AT whose computing index relative to IBM/XT is 11.7.

** The execution is terminated after running a week, i.e. its execution time is longer than a week.

V. Heuristics

In addition, to the advantages we have described, we can easily add heuristics to the methodology based on NNCO. It can carry out by changing the "energy" function, weight matrix, initial state, biases, etc. For example, in the multiconstraint zero-one knapsack problem, it is intuitionally assumed that some variables have known values. This can be done by setting biases. Table III compares the results of some problems with and without using this heuristic. The heuristic used here is to set two variables with the largest profits to have the state value one. Using this simple heuristic we can get better solutions more rapidly.

Table III: Results of selected problems while using heuristic

problem	no heuristic: average CPU time	average deviation	average standard deviation of deviation	maximum deviation	with heuristic: average CPU time	average deviation	average standard deviation of deviation	maximum deviation
Petersen 2	1.01	0.190	0.166	0.080	0.97	0.009	0.006	0.015
Petersen 3	1.87	0.122	0.070	0.111	1.53	0.077	0.044	0.147
Petersen 4	5.45	0.228	0.190	0.212	3.79	0.083	0.043	0.150

VI. Compare with Related Works

There are some known combinatorial optimization problems solved by neural networks. They are done by using well-known neural network models to compute solutions via first choosing connectivities and external inputs which appropriately represent the "energy" function to be minimized[16,17,26]. That is we must first formulate desired problem in an "energy" function that well characterizes the behavior of this problem and then compare it with the "energy" function of the well-known neural network model to get the connectivities and external inputs. If the "energy" function of the problem contains terms which can not transform to the form of the model, then use Taylor expansion to retain these terms that match the form of the model[18]. But it is not always possible to express the problems of interest in the known neural network models. They also can be solved by analog circuits[26]. But it is not practical to implement a circuit for each problem instance and it is difficult to adjust the parameters used in the "energy" function of the problem. Simulated annealing is another way to solve combinatorial optimization problems[4,19]. It uses the Metropolis algorithm for an appropriate numerical simulation of the behavior of a collection of nodes system at a finite temperature to provide a natural tool for bringing the techniques of statistical mechanics to solve combinatorial optimization problems. But it is radically slow.

Recently, Fort uses Kohonen algorithm to solve the traveling salesman problem[7]. The important defects of this method are the large time needed to achieve good solutions and there is no clear methodology to apply this method to other combinatorial optimization problems. In addition, its mathematical analysis is difficult and needs sophisticated probabilistic concepts. Based on these, we intend to formally propose a methodology to approximately solve the multiconstraint zero-one knapsack problem in a reasonable time and to be easily implemented on parallel computers. Besides, the methodology we present has the following significant improvements: we can define the "energy" function of the desired problem in any continuously differentiable form, of the desired problem in any continuously differentiable form, the parameters used in the "energy" function of the desired problem can self-adjust, and the methodology can be easily extended to other combinatorial optimization problems.

VII. Conclusion

We have presented a neural network model NNCO for the multiconstraint zero-one knapsack problem and a methodology based on this model to approximately solve this problem in near real-time by the use of this simple concept that an energy function generates a good solution without detailed algorithms. This methodology consists of four stages. First, define the "energy" function of the problem. Second, put the problem to the neural network model. Third, choose an appropriate activation function to make the "energy" function be bounded and decreased. Fourth, adjust parameters automatically. This stage can be regarded as automatic modification of the connection weights in the neural network model. It acts like unsupervised learning. In addition, we describe how to add heuristic rules to the neural network model to guide it to find good solutions rapidly. Moreover, we compare the methodology with related works.

We have also demonstrated how the methodology presented here can work well for the multiconstraint zero-one knapsack problem. There appears to be a large class of computation problems for which the methodology can generate a near optimum solution in near real-time without the need of detailed algorithms. The major advantages of this methodology are summarized as follows :

(1) Can rapidly provide a good solution.
(2) Due to the short execution time, we can start the network several times with different random number sets to get almost all the optimal solutions.
(3) Can automatically adjust the values of the parameters used in the "energy" function and this can be regarded as automatic modification of the connection weights in the neural network model. It acts like unsupervised learning.
(4) The "energy" function can be defined in any continuously differentiable form.
(5) Since the major differences of optimization problems in neural network models are the "energy" functions and the parameters, we can extend the methodology to solve a lot of combinatorial optimization problems with different "energy" functions and the parameters as inputs.

References

[1] Y.S.Abu-Mostafa and D.Psaltis," Optical Neural Computers," *Scientific American*, Mar. 1987, pp 88-95.

[2] J.A.Anderson, E.J.Wisniewski and S.R.Viscuso," Software for Neural Networks," *Computer Architecture News*, ACM Press, Vol. 16, No. 1, Mar. 1988, pp 26-36.

[3] P.Delbar," A Parallel Approach to Rule Based Systems," *Microprocessing and Microprogramming* 21(1987), pp 507-514.

[4] A.Drexl," A Simulated Annealing Approach to the Multiconstraint Zero-One Knapsack Problem," *Computing* 40, 1988, pp 1-8.

[5] S.E.Fahlman and G.E.Hinton," Connectionist Architectures for Artificial Intelligence," *IEEE Computer*, Jan. 1987, pp 100-108.

[6] B.M.Forrest, D.Roweth, N.Stroud, D.J.Wallace and G.V.Wilson," Implementing Neural Network Models on Parallel Computers," *The Computer Journal*, Vol. 30, No. 5, 1987, pp 413-419.

[7] J.C.Fort," Solving a Combinatorial Problem via Self-Organizing Process: An Application of the Kohonen Algorithm to the Traveling Salesman Problem," *Biol. Cybern.*, Vol. 59, 1988, pp 33-40.

[8] M.Garey and D.Johnson, *Computers and Intractability : a Guide to the Theory of NP-Completeness.* San Francisco, Freeman 1979.

[9] B.Gavish and H.Pirkul," Efficient Algorithms for Solving Multiconstraint Zero-One Knapsack Problems to Optimality," *Mathematical Programming*, Vol. 31, 1985, pp 78-105.

[10] J.Ghosh and K.Hwang," Critical Issues in Mapping Neural Networks on Message-Passing Multicomputers," *The 15th Annual International Symposium on Computer Architecture, IEEE*, May 1988, pp 3-11.

[11] A.Goel," A Report on the AAAI Symposium on Parallel Models of Intelligence: How Can Slow Components Think So Fast?," *SIGART Newsletter*, No. 106, Oct. 1988, p29.

[12] H.P.Graf, L.D.Jackel, and W.E.Hubbard,"VLSI Implementation of a Neural Network Model," *IEEE Computer*, Mar. 1988, pp 41-49.

[13] G.E.Hinton," Connectionist Learning Procedures," technical report CMU-CS-87-115.

[14] J.J.Hopfield," Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA*, Vol. 79, Apr. 1982, pp 2554-2558.

[15] J.J.Hopfield," Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Natl. Acad. Sci. USA*, Vol. 81, May 1984, pp 3088-3092.

[16] J.J.Hopfield and D.W.Tank," Computing with Neural Circuits : A Model," *Science*, Vol. 233, 8 Aug. 1986, pp 625-633.

[17] J.J.Hopfield and D.W.Tank," 'Neural' Computation of Decisions in Optimization Problems," *Biological Cybern.* , Vol. 52, 1985, pp 141-152.

[18] W.Jeffrey and R.Rosner," Neural Network Processing as a Tool for Function Optimization," in *Neural Networks for Computing - AIP Conference Proceedings 151*, edited by John S. Denker, American Institute of Physics, New York, 1986.

[19] S.Kirkpatrick, C.D.Gelatt and M.P.Vecchi," Optimization by Simulated Annealing," *Science*, Vol. 220, No.4598, 13 May 1983, pp 671-680.

[20] S.Y.Kung," Parallel Architectures for Artificial Neural Nets," *International Conference on Systolic Arrays, IEEE*, 1988, pp 163-174.

[21] R.P.Lippmann," An Introduction to Computing with Neural Nets," *IEEE Acoustics, Speech, and Signal Processing*, Vol. 4, 1987, pp 4-22.

[22] C.C.Petersen," Computation Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Project," *Management Science*, Vol. 13, No. 9, May 1967, pp 736-750.

[23] D.E.Rumelhart, G.E.Hinton, and J.L.McClelland," A General Framework for Parallel Distributed Processing," in D.E.Rumelhart, J.L.McClelland, and the PDP research group (Eds.), *Parallel Distributed Processing : Explorations in the Microstructure of Cognition.Volume 1: Foundations*, The MIT Press, Cambridge, Massachusetts, 1986.

[24] S.Senju and Y.Toyoda," An Approach to Linear Programming with 0/1 Variables," *Management Science*, Vol. 15, No. 4, Dec. 1968, pp B196 - 207.

[25] H.A.Taha, *Operations Research - An Introduction*, Third Edition.

[26] D.W.Tank and J.J.Hopfield," Simple 'Neural' Optimization Networks : An A/D Converter, Signal Decision Circuits and a Linear Programming Circuit," *IEEE Trans. on Circuits and Systems*, Vol. CAS-33, No. 5, May 1986, pp 533-541.

[27] D.S.Tourelzky and G.E.Hinton," Symbols Among the Neurons : Details of a Connectionist Inference Architecture," *IJCAI* 85, pp 238-243.

[28] D.S.Tourelzky and G.E.Hinton," Pattern Matching and Variable Binding in a Stochastic Neural Network," in L.Davice, *Genetic Algorithms and Simulated Annealing*, BBN Lab. , Cambridge, Massachusetts, 1987.

[29] H.M.Weingartner and D.N.Ness," Methods for the Solution of the Multi-dimensional 0/1 Knapsack Problem," *Operations Research*, Vol. 15, 1967, pp 83-103.

[30] G.V.Wilson and G.S.Pawley," On the Stability of the Travelling Salesman Problem Algorithm of Hopfield and Tank," *Biol. Cybern.* , Vol. 58, 1988, pp 63-70.

[31] S.H.Zanakis," Heuristic 0-1 Linear Programming: An Experimental Comparison of three Methods," *Management Science*, Vol. 24, No. 1, Sept. 1977, pp 91-104.