Proceedings of the
American Control Conference
San Francisco, California • June 1993

SOLVING A COMBINATORIAL OPTIMIZATION PROBLEM WITH FEEDFORWARD NEURAL NETWORKS

Xianzhong Cui and Kang G. Shin

Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109–2122

Abstract

A new approach is proposed to solve a typical combinatorial optimization problem using artificial neural networks. Unlike the popular idea of using a Hopfield network for optimization, we design a new network architecture which consists of two parts: a feedforward network for optimization, and a feedback network for meeting the constraints. Radial-based functions are adopted in the feedforward network in order to utilize its spatial locality and facilitate selection of the numbers of hidden layers and nodes. The convergence of the proposed scheme is proved and a vector-form training algorithm is developed. [1]

1 Introduction

Production scheduling in a computer–integrated manufacturing (CIM) system is a key combinatorial optimization problem for manufacturing productivity and equipment utilization. However, "optimal" production scheduling is a very difficult problem, especially for a CIM system with multiple machines and precedence constraints among machine operations. Conventionally, production schedules are generated by humans with the aid of software tools and computer simulations.

Neural networks (NNs) seem to be paving a new way to handle the difficulty of combinatorial optimization problems. There are two main issues to be addressed when applying NNs: (1). What is the NN architecture for problem characterization? (2). What is the training algorithm of the NN? For constrained optimization problems, a straightforward approach would be using a Hopfield network. The dynamics of a Hopfield network are represented by a set of first–order, nonlinear differential equations, called *motion functions*. The key issue in using such a network for optimization is to describe the problem in the form of motion functions. Generally, there are two ways to derive the motion functions. The first method is to define an energy function for a specified optimization problem, and the motion functions are then obtained by taking the derivative of the energy function. Lo and Bavarian used this method for production scheduling [4]. However, since the connections between neurons are usually not symmetric, convergence is not guaranteed [2]. Other potential problems are (1) the convergence may be very sensitive to the initial value of Lagrange multipliers in the energy function; (2) the NN architecture may not be feasible for large problems; (3) in practice, it may be difficult to find the energy function for certain problems in which the constraints are not readily expressible. The motion functions can also be derived heuristically. Fang and Li proposed a heuristic approach to derive the motion functions for a Hopfield network to solve combinatorial optimization problems [3]. The key point is to design the activation rules using heuristics such that neurons compete to become active under some constraints. Since the motion function is derived heuristically, the connections between the neurons are usually not symmetric. Therefore, the convergence of the network should be analyzed case by case. Moreover, some constraints may not be included in the motion functions. For combinatorial optimization problems, direct application of the original Hopfield network is usually inferior to some known conventional heuristic algorithms in terms of both the computation time and the quality of the solution [6].

As mentioned above, most of the related work relies on the structure of Hopfield network and usually has no learning ability. There are many issues to be resolve before applying NNs to scheduling problems, such as the representation of a scheduling problem with an NN structure, systematic training of NNs, convergence and learning ability, and so on. In this paper, a new NN structure is proposed to solve a typical combinatorial optimization problem. The proposed NN consists of two parts: a feedforward network for optimization, and a Hopfield network for meeting the constraints.

In Section 2, we formally state the combinatorial optimization problem under consideration. The architecture of the proposed NN for this constrained optimization problem is given in Section 3. To solve the combinatorial optimization problem, a special cost function is designed in Section 4. Section 5 presents the simulation results of the proposed architecture. Finally, the paper concludes with Section 6.

[1] The work reported in this paper was supported in part by the National Science Foundation under Grant No. DMC–8721492. Any opinions, findings, and recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the NSF.

2 Problem Statement

The combinatorial optimization problem addressed here is a general assignment problem:

- Given: a set of jobs: $\mathbf{J} = \{J_1, J_2, \cdots, J_n\}$ and a set of machines: $\mathbf{M} = \{M_1, M_2, \cdots, M_m\}$.
- Each $J_i \in \mathbf{J}$ can be processed by any machine $M_j \in \mathbf{M}$ with processing time p_{ij}.
- Constraints: no machine can process more than one job at a time, and each job must be assigned to one and only one machine.
- We want to find an assignment

$$\{J_1 \rightarrow M^1, J_2 \rightarrow M^2, \cdots, J_n \rightarrow M^n\},$$

which minimizes a cost function $E_c(J_1, J_2, \cdots, J_n)$ while meeting the above constraints, where $J_i \rightarrow M^i$ denotes that job i is assigned to machine $M^i \in \mathbf{M}$.

Obviously, this is a discrete optimization problem. There are $(n!)^m$ possible assignments for the above problem. Solving such a combinatorial optimization problem is *NP*-complete [4].

Let v_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$, be a variable representing a job assignment

$$v_{ij} = \begin{cases} 1, & \text{if } J_i \rightarrow M^j \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Then a simple cost function can be defined as the total makespan:

$$E_c = \sum_{j=1}^{m} \left(\sum_{i=1}^{n} p_{ij} v_{ij} \right), \tag{2}$$

and the constraints are represented by

$$\sum_{i=1}^{n} \left(\left(\sum_{j=1}^{m} v_{ij} \right) - 1 \right)^2 = 0. \tag{3}$$

3 A New NN Architecture for Combinatorial Optimization

The above combinatorial optimization problem can be viewed as a mapping from the input data (jobs, machines, processing times and constraints) to an optimal assignment. We want to obtain this optimal assignment by training an NN. For this purpose, we propose an approach using feedforward neural networks. When designing an NN architecture, we must consider the following requirements: (1) Completeness of the solution representation; (2) Satisfaction of constraints; (3) Deterministic selection of the number of hidden nodes (neurons) and layers; (4) Simplicity in designing a training algorithm with guaranteed convergence.

Let the input of this feedforward NN be the processing times of each job on all machines, i.e., the inputs to node n are the processing times of job n:

$$[p_{n1}, p_{n2}, \cdots\cdots, p_{nm}] \tag{4}$$

To represent all possible solutions, each node is designed to perform vector operations. The outputs of the NN represent all possible job assignments to machines. For example, the outputs of node n are the assignment of job n:

$$[v_{n1}, v_{n2}, \cdots\cdots, v_{nm}] \tag{5}$$

Clearly, to satisfy the constraints, one must guarantee that there is at most one "1" element in each output node. This is difficult, however, to implement in a conventional feedforward NN with continuous outputs. In our approach, a local feedback is introduced among the elements of each output node. More precisely, the proposed NN consists of two parts: a multilayer feedforward NN for optimization, and a Hopfield network for meeting the constraints. In this structure, the Hopfield network works as associative memory and meets the constraint of assigning each machine at most one job, that is, regardless of the solution given by the feedforward network, the constraints will be met.

A popular feedforward NN is the multilayer perceptron with sigmoid functions and the back propagation (BP) training algorithm. However, there is no systematic way of selecting a suitable number of hidden layers and nodes for such an NN. Moreover, if the sigmoid functions are used at the output layer, the NN's outputs are within (0, 1) which represents a pre-specified range of values. This requires that the NN's inputs must be scaled to (0, 1) in order to represent the same pre-specified range. For different input data of the combinatorial optimization problem, selection of a scaling algorithm (parameter) may affect the performance of the NN. Clearly, the above weaknesses may limit the direct application of a multilayer perceptron with the sigmoid function to the combinatorial optimization problem stated in Section 2.

Instead of using sigmoid functions, radial-based (Gaussian) functions (RBF) can be used as the activation functions in feedforward NNs. A feedforward NN with single hidden layer and RBF is capable of universal approximation [5]. Thus multiple hidden layers are not needed. In general, this NN has one hidden layer with RBF units and an output layer with summation units.[2] Let $X_i \in \mathbf{R}$, $i = 1, 2, \cdots, N$, be the inputs of the NN, and X_{1j}, $X_{2k} \in \mathbf{R}$, $j = 1, 2, \cdots, N_1$, $k = 1, 2, \cdots, N_2$, be the outputs of the HIDDEN and OUTPUT layer, respectively. Then the computation of this network is

$$X_{1j} = exp\left(-\sum_{i=1}^{N} \frac{(X_i - C_{ij})^2}{2\sigma_{ij}^2} \right), \quad j = 1, 2, \cdots, N_1, \tag{6}$$

where C_{ij} and σ_{ij} are the weight from node i at the INPUT layer to node j at the HIDDEN layer.

$$X_{2k} = \sum_{j=1}^{N_1} W_{jk} X_{1j}, \quad k = 1, 2, \cdots, N_2, \tag{7}$$

where W_{jk} is the weight from node j at the HIDDEN layer to node k at the OUTPUT layer. Let X_{2k}^d be the

[2] This neural network will henceforth be referred to as an RBF network.

desired value of the network output X_{2k} and define a cost function

$$E = \frac{1}{2}\sum_{k=1}^{N_2}\left(X_{2k}^d - X_{2k}\right)^2. \qquad (8)$$

Using the gradient decent algorithm, the weights are updated

By using this RBF network, we eliminate the problem associated with the selection of the number of HIDDEN layers. Moreover, it is not necessary to scale the inputs to any pre-specified range. It is also quite easy to determine the number of HIDDEN nodes in the NN that is designed to solve the combinatorial optimization problem.

Note that the inputs of the network (the processing times of a set of jobs on different machines) are fixed for the entire optimization process. For those networks in which adjustable parameters (weights) have global effects on the learned input/output (I/O) map, fixed inputs may lead to divergence or a large error. This input fixation can be accommodated by the *spatial localization* property [1] of RBF networks. Note that the multilayer perceptron with sigmoid functions does not have this property. This is another advantage of the RBF network when it is used to solve combinatorial optimization problems.

In the proposed structure, a local feedback is introduced among the elements of each OUTPUT node to meet the constraint: "each job can only be assigned to one machine." The basic structure is a Hopfield network [2] and, in our application, the Hopfield network is cascaded to the RBF network. The external input to the Hopfield network can be used to represent other constraints, for example, job processing priorities. The outputs of the RBF network are the initial internal states of the Hopfield network. Therefore, the computation of the entire network consists of two stages: (1) Compute the outputs of the RBF network; (2) The outputs of the RBF network are sent to the Hopfield network as the initial internal states. When the Hopfield network reached a stable state, its output represents the machine to which a job is assigned. More formally, let X_{2k} be the k-th ($1 \le k \le n$) output of RBF network for job k,

$$X_{2k}(t_0) = [x_{2k1}(t_0),\ x_{2k2}(t_0),\ \cdots,\ x_{2km}(t_0)]^T,$$

where m is the total number of machines and t_0 is a time index. Let v_{ij} denote the job assignment, where

$$v_{ij} = \begin{cases} 1, & \text{if } J_i \rightarrow M^j \\ 0, & \text{otherwise.} \end{cases}$$

Then the motion function of the Hopfield network for job k is

$$\frac{dx_{2ki}(t)}{dt} = -x_{2ki}(t) + \sum_{j=1}^{m} T_{ij}v_{kj} + I_i, \qquad (9)$$
$$i = 1,\ 2,\ \cdots,\ m, \quad t > t_0.$$

When the transient period of Eq. (9) is over, the output of the Hopfield network, v_{ij}, indicates the job assignment. That is, the Hopfield network is used as an associative memory. Whenever the RBF network presented an output of optimization, the Hopfield network will output a machine assignment satisfying the constraints specified by the weights and external inputs.

For the simplest case, suppose there is only one job with different processing times on m machines. Considering the two parts separately, we know that the outputs of the Hopfield network converge because the network is designed with symmetric weights. The outputs of the RBF network also converge if its weights are updated by using the gradient decent algorithm to minimize its output error Eq. (8).[3] However, for the entire network, we have (1) the output of the RBF network is used as the initial value of the internal state of the Hopfield network, and (2) the weights of the RBF network is updated by minimizing the output error of the Hopfield network. An immediate question is then "does the output of the entire network converge?"

Let v_{kj}^d be the desired value of v_{kj}. A cost function in terms of the output of the Hopfield network is defined by

$$E = \frac{1}{2}\sum_{p=1}^{m}\left(v_{2kp}^d - v_{2kp}\right)^2. \qquad (10)$$

Since the Hopfield network always converges to a stable state irrespective of initial internal states, the convergence problem of the entire network is equivalent to: by minimizing the cost function Eq. (10) with the gradient decent algorithm, does the RBF network converge? This question is answered by the following theorem (stated without proof due to the space limit).

Theorem 1: Suppose (1) the weights of the Hopfield network are symmetric and selected according to a set of stored patterns; (2) the outputs of the RBF network are the initial internal states of the Hopfield network; (3) after the transient period is over, the outputs of the Hopfield network has reached a stable state which is one of the stored patterns, and is used as the outputs of the entire network; (4) a cost function is defined using this stable state as Eq. (10). If the weights of the RBF network are updated by minimizing the cost function Eq. (10) with the gradient decent algorithm, then the outputs of the RBF network will converge to a stable state. □

Though the proposed neural network is stable, a cost function in the form of Eq. (10) cannot be used for the combinatorial optimization problem, v_{kp}^d (the desired job assignment) is unknown. Thus, a new cost function must be defined, which is the topic of next section.

4 Solving the Problem with the Proposed NN

Training a multilayer perceptron is an optimization process that minimizes a certain cost function. Under the BP algorithm, optimization problems are solved by the gradient decent algorithm. This optimization process is driven by modifying the weights of the NN

[3] In Eqs. (8) and (10), the subscript k denotes the job number. Though it is meaningless for the case of only one job, it keeps the notation consistent with the case of multiple jobs.

according to the cost function Eq. (10). To select a cost function, we must consider not only the nature of a specified problem, but also the possibility to derive a training algorithm and its convergence property. Therefore, if a cost function is not in the form of Eq. (10), a new training algorithm should be developed and its convergence property should be investigated, though the basic idea of back propagation can still be used.

For the combinatorial optimization problem, we want to define a cost function directly from the cost of job assignment instead of the assignment error as in Eq. (10). For example, a cost function can be defined as the total makespan (Eq. (2)). However, in Eq. (2), the cost function reaches a minimum by assigning all jobs to the machines on which they will have the minimum processing time. Clearly, this is not what we want; we want not only to minimize the makespan but also to balance the workload on the machines.

Let $\bar{P}$ denote the minimum average makespan:

$$\bar{P} = \frac{1}{m}\sum_{i=1}^{n}\left(\min_{j}(p_{ij})\right). \tag{11}$$

$\bar{P}$ gives the lower bound of the makespan on each machine with a balanced workload, and is the value we really want to achieve. Then we define a cost function in terms of the minimum average makespan

$$E_c = \frac{1}{2}\sum_{j=1}^{m}\left(\bar{P} - \sum_{i=1}^{n} p_{ij}v_{ij}\right)^2. \tag{12}$$

The lower bound of E_c is zero, but this lower bound is generally not achievable, since each job must be processed by one machine without interruption. In other words, minimizing E_c is a discrete optimization problem, and some values of E_c may not be reachable for a certain set of p_{ij}'s.

When training a neural network to minimize a general cost function rather than a cost function expressed in terms of network output errors as in Eq. (10), we must first ensure convergence to a (not necessarily global) minimum. (Note that for most combinatorial optimization problems, some local minimum is usually acceptable as a suboptimal solution.) The cost function defined in Eq. (10) becomes zero when the NN outputs reach their desired values. In such case we no longer need to modify the weights of the NN. However, for the cost function defined in Eq. (12), the lower bound (zero) of E_c is usually not reachable. That is, even when the cost function reaches its reachable minimum, the weights of the NN are still modified, which may drive the cost function away from the minimum. This may in turn result in an oscillation. Therefore, what we want to ensure is not "the NN will converge to a stable state", but "minimizing the cost function is a convergent process." If the convergence is ensured, then we can define a criterion to terminate the training of the NN. For example, the maximum number of training steps, T_{max}, can be used as a criterion. After training the NN for T_{max} steps, we choose the minimum value of the cost function during this training period, and the corresponding job assignment is the (suboptimal) solution.

Theorem 2: If the gradient decent algorithm is used to train the proposed NN to minimize the cost function Eq. (12), the process to minimize the cost function will converge. □

Proof of this theorem is similar to that of Theorem 1. The basic idea is to show that for the cost function Eq. (12), the search direction of the gradient decent algorithm will not be affected and the weight update is finite. This theorem shows that the cost function Eq. (12) leads to a convergence process, but not that the NN converges to a stable state.

Since both the inputs and outputs of the proposed NN are vectors as shown in Eqs. (4) and (5), we need to design a training algorithm in vector form. Using the basic idea of back propagation, such a training algorithm has been derived. In developing the training algorithm, the proposed NN is treated as an RBF network with outputs 1.0 and 0.0. That is, the portion of Hopfield network that enforces the constraints is viewed as a unit of static transformation, since the weights of the RBF network are updated only after the transient period of the Hopfield network is over.

Let X_i denote the inputs of the RBF network, and X_{1j}, X_{2k} denote the outputs of its HIDDEN and OUTPUT layers, respectively, for $i, j, k = 1, 2, \cdots, n$. The inputs of the RBF network are the processing times $X_i = [p_{i1}, p_{i2}, \cdots, p_{im}]^T$, where p_{iq} is the processing time of job i on machine q. The outputs are the job assignments

$$X_{2k} = [v_{k1}, v_{k2}, \cdots, v_{km}]^T, \tag{13}$$

where

$$v_{kq} = \begin{cases} 1.0, & \text{if } J_k \rightarrow M^q \\ 0.0, & \text{otherwise.} \end{cases}$$

The details of the training algorithm are omitted due to the limited space (it is based on the gradient decent algorithm).

5 Simulation Results

We have performed extensive simulations to test the capability of the proposed scheme, and one of the simulated examples is summarized below. In this example, there are 15 jobs with different processing times on three machines as listed in Table 1. The processing times are generated randomly. Then, using Eq. (11), we get the minimum average makespan: $\bar{P} = 59.333$. Obviously, this minimum average makespan is not reachable. The cost function is defined as Eq. (12). The proposed NN is used to minimize this cost function. The initial weights of the NN are set to small random numbers. Since $\bar{P}$ is an unreachable lower bound and this is a discrete minimization problem, after 500 learning iterations, the cost function finally oscillates among three values: $E_c = 58.333$, $E_c = 71.167$, and $E_c = 235.833$. The assignment corresponding to $E_c = 58.333$ is shown in Table 2. In this example, the external inputs of the Hopfield networks are $I_i = 0$ in Eq. (9), since the only constraint is "each job can

Job #	Processing time on each machine		
	Machine 1	Machine 2	Machine 3
1	21	17	14
2	10	18	26
3	99	83	16
4	51	80	12
5	19	10	55
6	18	20	19
7	19	14	19
8	19	12	28
9	17	10	18
10	57	15	11
11	17	15	78
12	10	44	53
13	13	14	13
14	85	19	1
15	21	17	12

Table 1: Processing times on different machines.

Job #	Machine assigned and processing time		
	Machine 1	Machine 2	Machine 3
1			14
2	10		
3			16
4			12
5		10	
6	18		
7		14	
8		12	
9		10	
10			11
11		15	
12	10		
13	13		
14			1
15			12
Makespan	51	61	66

Table 2: Job assignment and makespan.

only be assigned to one machine". For such a case, the Hopfield network may not be really necessary, which performed a "winner takes all" process. Our immediate work is adding more constraints, represented by $I_i \neq 0$, to the problem.

The simulation results indicate that the proposed NN structure is suitable for a special combinatorial optimization problem. The convergence is ensured and 500 — 700 learning iterations are long enough to determine a suboptimal assignment.

6 Conclusion

A new NN structure is proposed to solve combinatorial optimization problems. The NN consists of two parts: a feedforward network with radial-based functions for optimization, and a Hopfield network for constraints. The convergence of the proposed scheme is proved and a vector-form training algorithm is developed.

To solve a combinatorial optimization problem, a new cost function is defined because of an unreachable lower bound of makespan when computing the minimum cost. The NN drives the cost function to approach its minimum and a suboptimal solution is determined by the optimization process. Using the RBF feedforward network for optimization, the proposed scheme can learn to adapt itself to different set of input data. Moreover, unlike using sigmoid functions, the numbers of HIDDEN layers and nodes can be easily selected. Systematic design of NN structure and cost function, guaranteed convergence and promising simulation results have indicated that the proposed approach is a good alternative to heuristics commonly used to solve combinatorial optimization problems.

References

[1] W. L. Baker and J. Farrell, "Connectionist Learning Systems for Control", *Proc. of the 1990 SPIE Conference*, Boston, MA.

[2] F. A. Camargo, "Learning Algorithms in Neural Networks", *Technique Report of the DCC Laboratory, Computer Science Department, Columbia University*, CUCS-062-90.

[3] L. Fang and T. Li, "Design of Competition-Based Neural Networks for Combinational Optimization", *Int'l J. on Neural Systems*, Vol. 1, No. 3, pp. 221-235, 1990.

[4] Z.-P. Lo and B. Bavarian, "Scheduling with Neural Networks for Flexible Manufacturing Systems", *Proc. of the 1991 IEEE Int'l Conf. on Robotics and Automation*, pp. 818-823.

[5] T. Poggio and F. Girosi, "Networks for Approximation and Learning", *Proc. of the IEEE*, vol. 78, No. 9, pp. 1481-1497, September 1990.

[6] D. N. Zhou, V. Cherkassky, T. R. Baldwin and D. E. Olsen, "A Neural Network Approach to Job-Shop Scheduling", *IEEE Trans. on Neural Networks*, vol. 2, No. 2, pp. 175-179, Jan. 1991.