Lagrangian Relaxation Neural Networks for Job Shop Scheduling

Peter B. Luh, Xing Zhao, and Yajun Wang
Department of Electrical & Systems Engineering
University of Connecticut, Storrs, CT 06269-2157, USA

Lakshman S. Thakur
School of Business Administration
University of Connecticut, Storrs, CT 06269,USA

Abstract. Manufacturing scheduling is an important but difficult task. Building on our previous success in developing optimization-based scheduling methods using Lagrangian relaxation for practical applications, this paper presents a novel Lagrangian relaxation neural network (LRNN) optimization techniques. The convergence of LRNN for separable convex programming problems is established. For separable integer programming problems, LRNN is constructed to obtain near optimal solution in an efficient manner. When applying LRNN to separable job shop scheduling, a new neural dynamic programming method is developed to solve subproblems making innovative use of the dynamic programming structure. The synergy of Lagrangian relaxation and neural dynamic programming leads to a powerful neural optimization method for job shop scheduling. Testing results obtained by software simulation demonstrate that the performance is superior to what has been reported in the neural network literature. Results are also very close to what were obtained by a state-of-the-art optimization algorithm, and should be much improved when the method is refined and implemented in hardware.

1. Introduction

Production scheduling is a major issue faced daily by almost all manufacturers. Deriving benefits from effective scheduling, however, has been recognized to be extremely difficult because of inherent problem complexity and the sizes of real problems. Recently, we have successfully developed a combined Lagrangian relaxation, dynamic programming, and heuristic approach to solve practical manufacturing scheduling problems with excellent results (Wang, Luh, Zhao, and Wang, 1997). Building on these results, this paper is to explore novel neural network optimization techniques to further improve solution quality and computational efficiency.

Neural networks for unconstrained optimization have been based on the "Lyapunov stability theory" of dynamic systems: if a network is "stable," its "energy" will decrease to a minimum as the system approaches and attains its "equilibrium state." If one can properly set up a network that maps the objective function of an optimization problem onto an "energy function," then the solution is a natural result of network convergence to its equilibrium, and can be obtained at a very fast speed (Hopfield and Tank, 1985).

For constrained optimization, the Hopfield-type networks have been based on the well known "penalty methods," which approximate a constrained problem as an unconstrained one by having penalty terms on constraint violations (Hopfield and Tank, 1985). The unconstrained problem is then solved by neural networks as mentioned above. Generally, a solution to the converted problem is the solution to the original one only when penalty coefficients approach infinity. As coefficients become large, however, the converted problem becomes ill conditioned. To obtain a solution without having coefficients tend to infinity, a tradeoff between solution optimality and constraint satisfaction has to be made by fine tuning penalty coefficients. The tradeoff, however, is generally difficult to make. In addition, Hopfield-type networks may possess many local minima, and escaping from local minima is not an easy task (Wilson and Pawley, 1988). The solution quality thus highly depends on initial conditions.

Hopfield-type networks and its extended versions were developed to solve job shop scheduling problems (Foo and Takefuji, 1988, Zhou et. al., 1991). Although these models demonstrate the possibility of using neural networks for scheduling problems, they suffer from the above-mentioned difficulties, and it is not easy to scale up these methods to solve practical problems. Heuristics have also been used to modify neuron dynamics to induce constraint satisfaction (Satake, Morikawa and Nakamura, 1994, Willems and Brandts, 1995, Sabuncuoglu and Gurgun, 1996). The results, however, may be far from optimal.

Recent developments on neural networks for constrained optimization include combining *Lagrangian relaxation* (LR) or *augmented Lagrangian relaxation* with Hopfield-type networks, showing significant improvement on solution quality (Wacholder, Han and Mann, 1989, Li, 1996). The convergence of LRNN, however, has not been fully established. Furthermore, with "traveling salesman problems" being the reference model used by most researchers, method development has been problem

specific with many important issues overlooked and great opportunity missed.

In this paper, the convergence of LRNN for separable convex programming is established in Section 2. For separable integer programming problems, LRNN is also constructed to obtain near optimal solution in an efficient manner. LRNN is then applied to separable job shop scheduling in Section 3. In this case, the LRNN consists of many sub-networks, one for each job (or part). In Section 4, a new neural dynamic programming is developed to solve the subproblems making innovative use of the dynamic programming structure to handle integer variables and subproblem constraints. Testing results presented in Section 5 demonstrate that the performance of the method is much better than what has been reported in the neural network scheduling literature. The results are also very close to what were obtained by our previous LR/DP approach results (Wang, Luh, Zhao, and Wang 1997), which is believed to be at the cutting edge of optimization-based scheduling methods. Yet, these results are based on software simulation, and should be greatly improved when the method is implemented in hardware.

2. Lagrangian Relaxation with Neural Networks

Problem Formulation. Consider the following separable convex programming problem:

$$\min_{x} \quad J \equiv \sum_{i=1}^{I} J_i(x_i), \tag{2.1}$$

$$\text{st.} \quad \sum_{i=1}^{I} g_i(x_i) \le 0, \quad i = 1, \cdots, I, \tag{2.2}$$

were x_i is an $n_i \times 1$ continuous decision variable, $g_i(x_i)$ an $m \times 1$ function, and I is the number of subproblems. Assume that $\{J_i(x_i)\}$ and $\{g_i(x_i)\}$ are convex and differentiable functions. Since both the objective function and constraints are additive, the problem is separable.

Lagrangian Relaxation. Playing a fundamental role in constrained optimization over the decades, Lagrangian relaxation is powerful for the above separable problems. Since constraints (2.2) couple the decision variables x_i, they are "relaxed" by Lagrangian multipliers λ. The Lagrangian is thus given by

$$L(\lambda) \equiv \min_{x} \left[\sum_{i=1}^{I} J_i(x_i) + \lambda^T \sum_{i=1}^{I} g_i(x_i) \right]. \tag{2.3}$$

Here λ is an $m \times 1$ vector of Lagrangian multipliers, and the function $L(\lambda)$ is the "Lagrangian dual." Since the decision variables are decoupled through the introduction of Lagrangian multipliers λ, (2.3) can be written in terms of individual subproblems:

$$L_i(\lambda) \equiv \min_{x_i} \left[J_i(x_i) + \lambda^T g_i(x_i) \right], \tag{2.4}$$

and

$$L(\lambda) = \sum_{i=1}^{I} L_i(\lambda). \tag{2.5}$$

The dual problem is then given by

$$\text{LD:} \max_{\lambda \ge 0} L(\lambda), \tag{2.6}$$

where the optimal solution is denoted as $L^* = L(\lambda^*)$. Maximizing the dual without its explicit representation can be done by several methods, including the most widely used gradient method described by:

$$\lambda^{k+1} = \lambda^k + \alpha^k \nabla L(\lambda^k), \tag{2.7}$$

where k is the iteration index, λ^k the multipliers at iteration k, α^k the step size, and $\nabla L(\lambda^k)$ the gradient of the dual function $L(\lambda)$ evaluated at λ^k. The dual function is always concave, and provides a lower bound to the optimal primal cost. Let the optimal dual be denoted as $L^* = L(\lambda^*, x^*)$, where x^* is the minimum of the relaxed problem given the optimal multipliers λ^*.

Lagrangian Relaxation Neural Networks. LR has recently been combined with neural networks to solve constrained optimization problems. The key idea of LRNN is to view the multiplier updating formula (2.7) as a set of differential equations. The multipliers themselves can thus be viewed as a dynamic system. For a given set of multipliers, the relaxed subproblem (2.4) can be solved by using neural networks for unconstrained optimization. The multipliers together with subproblem networks then form the overall LRNN. The network elements that update multipliers will be referred to as the "*Lagrangian neurons.*" In contrast, neurons solving the subproblems will be called "*decision neurons.*" The dynamics of the LRNN can therefore be described by the following differential equations:

$$\frac{d\lambda(t)}{dt} = \alpha(t) \frac{\partial \tilde{L}(\lambda(t), x(t))}{\partial \lambda(t)}, \tag{2.8}$$

$$\frac{dx(t)}{dt} = -\beta(t) \frac{\partial \tilde{L}(\lambda(t), x(t))}{\partial x(t)}, \tag{2.9}$$

where

$$\tilde{L}(t) = \tilde{L}(\lambda(t), x(t)) = \sum_{i=1}^{I} J_i(x_i(t)) + \lambda(t)^T \sum_{i}^{I} g_i(x_i(t)) \tag{2.10}$$

Here $\alpha(t)$ and $\beta(t)$ are positive numbers and can be time varying.

Convergence of LRNN. The convergence of a specific LRNN implementation (2.8), (2.9) with $\alpha(t) = \beta(t) \equiv 1$ was established in 1958 for *strictly* convex problems (Arrow, Hurwicz, and Uzawa, 1958. p. 129). This method, however, leads asymptotically to a periodic solution when both the objective function and constraints are linear (Dorfman, Samuelson and Solow, 1958). To overcome this difficulty, a modified version was established for convex problems (not necessarily strictly convex) by introducing a nonlinear transformation of constraints (Arrow, Hurwicz, and Uzawa, 1958, p. 137). This nonlinear transformation, however, destroys the separability of the original problem.

The following steps establish the convergence of LRNN for convex programming (not necessarily strictly convex) without destroying the separability of the original problem.

Proposition 1. Given the current point $(\lambda(t), x(t))$, if

$$\tilde{L}(t) < L^*, \tag{2.11}$$

then the gradient direction of $\lambda(t)$ is always in an acute angle with the direction towards λ^*, e.g.,

$$0 < (\lambda^* - \lambda(t))^T \frac{d\lambda(t)}{dt}. \tag{2.12}$$

Proof. First define the surrogate dual as

$$\tilde{L}(\lambda, x) \equiv \sum_{i=1}^{I} J_i(x_i) + \lambda^T \sum_{i}^{I} g_i(x_i). \tag{2.13}$$

Then

$$\tilde{L}(\lambda, x(t)) = \sum_{i=1}^{I} J_i(x_i(t)) + \lambda^T \sum_{i}^{I} g_i(x_i(t))$$

$$= \tilde{L}(t) + (\lambda - \lambda(t))^T \sum_{i=1}^{I} g_i(x_i(t)) = \tilde{L}(t) + (\lambda - \lambda(t))^T \frac{\partial \tilde{L}(t)}{\partial \lambda(t)}. \tag{2.14}$$

Since minimization is performed in defining $L(\lambda)$ in (2.3), the surrogate dual always satisfies:

$$L(\lambda) \le \tilde{L}(\lambda, x). \tag{2.15}$$

The above is also true at $(\lambda^*, x(t))$, i.e.,

$$L^* = L(\lambda^*) \le \tilde{L}(\lambda^*, x(t)). \tag{2.16}$$

From (2.14), this can be written as

$$L^* \le \tilde{L}(t) + (\lambda^* - \lambda(t))^T \frac{\partial \tilde{L}(t)}{\partial \lambda(t)}. \tag{2.17}$$

Given (2.8) and (2.11), this yields

$$0 < L^* - \tilde{L}(t) \le \frac{(\lambda^* - \lambda(t))^T}{\alpha(t)} \frac{d\lambda(t)}{dt},$$

and (2.12) is proved. **Q.E.D.**

Proposition 2. If the initial point satisfies

$$\tilde{L}(0) < L^*, \tag{2.18}$$

and the dynamics of Lagrangian neurons follows

$$\alpha(t) \le \frac{L^* - \tilde{L}(t)}{\left\| \partial \tilde{L}(t) / \partial \lambda(t) \right\|^2}, \tag{2.19}$$

then

$$\tilde{L}(t) < L^*.$$

Proof. From (2.8), (2.9), and (2.10),

$$\frac{d\tilde{L}(t)}{dt} = \frac{\partial \tilde{L}(t)}{\partial \lambda(t)}^T \frac{d\lambda(t)}{dt} + \frac{\partial \tilde{L}(t)}{\partial x(t)}^T \frac{dx(t)}{dt}$$

$$= \alpha(t) \left\| \frac{\partial \tilde{L}(t)}{\partial \lambda(t)} \right\|^2 - \beta(t) \left\| \frac{\partial \tilde{L}(t)}{\partial x(t)} \right\|^2. \tag{2.20}$$

With (2.19), we have

$$\frac{d\tilde{L}(t)}{dt} \le (L^* - \tilde{L}(t)) - \beta(t) \left\| \frac{\partial \tilde{L}(t)}{\partial x(t)} \right\|^2 \le L^* - \tilde{L}(t). \tag{2.21}$$

Together with (2.18), it can be shown that

$$\tilde{L}(t) < L^*. \quad \textbf{Q.E.D.}$$

Thus we have the following Theorem:

Theorem 1. For a convex programming problem, $(\lambda(t), x(t))$ in the LRNN described by (2.8) and (2.9) will converge to an optimal point (λ^*, x^*) as long as

$$\tilde{L}(0) < L^*,$$

and

$$0 < \alpha(t) \le \frac{L^* - \tilde{L}(t)}{\left\| \partial \tilde{L}(t) / \partial \lambda(t) \right\|^2}. \tag{2.22}$$

Proof. From Propositions 1 and 2, the gradient direction of $\lambda(t)$ is always in an acute angle with the direction pointing to λ^*. From (2.12), we have

$$\frac{d\left\| \lambda^* - \lambda(t) \right\|^2}{dt} = -2(\lambda^* - \lambda(t))^T \frac{d\lambda(t)}{dt} < 0. \tag{2.23}$$

This means that $\lambda(t)$ gets closer and closer to λ^*. Now assume that $(\lambda(t), x(t))$ converges to (λ^+, x^+). Then (2.8) and (2.9) imply that x^+ is the optimal solution for the given λ^+, and $L(\lambda^+) = L^*$. Thus (λ^+, x^+) is an optimal solution. It can also be shown by contradiction that $(\lambda(t), x(t))$ always converge to a certain point. Thus the theorem is proved. **Q.E.D.**

The above proof is for a particular implementation of LRNN based on the gradient approach (2.9). In fact, the key condition is $\frac{\partial \tilde{L}(t)}{\partial x(t)} \frac{dx(t)}{dt} < 0$ as required by (2.20) and (2.21). This implies that as long as the dynamics of the decision neurons causes $\tilde{L}(t)$ to decrease for the given multipliers, LRNN will converge. This is general regardless if x is continuous or discrete. Creative variations beyond (2.9) can therefore be developed to fit the specific needs of individual problems (Zhao, Luh, and Wang, 1999).

LRNN for Separable Integer Programming Problems. Integer programming problems are generally difficult to solve because of their inherent combinatorial complexity. Lagrangian relaxation, however, has been proven to be particular powerful for separable integer programming problems. Given such a problem, the hard coupling constraints are first relaxed through the introduction of Lagrangian multipliers. The relaxed problem can be decoupled into individual subproblems. If these subproblems are not NP hard, they can be solved for a given set of multipliers. Multipliers are then iteratively adjusted based on the levels of constraint violation. At termination of such updating iterations, simple heuristics are applied to adjust subproblem solutions to form a feasible result satisfying all constraints.

Since subproblem solutions will tend to satisfy the constraints and approach the optimal feasible solution over the iterations, Lagrangian relaxation provides a powerful approach to obtain near optimal solutions for NP hard integer programming problem. Furthermore, since dual costs are lower bounds to the optimal cost, quality of the solution can be quantitatively evaluated by comparing the feasible cost with the highest dual cost obtained.

Based on the above idea, LRNN can be constructed for separable integer programming problems. Constructing subnetworks to effectively solve subproblems, however, is a challenging task and may be case dependent. The traditional neural optimization approach for 0-1 integer programming problems is to *approximate discrete variables* by *continuous ones.* For example, it is known that "high gain" of "sigmoid" activation functions can be used to induce integer solutions (Hopfield and Tank, 1985). If the gain is too high, however, the problem will be ill conditioned. The penalty term $[x \cdot (1-x)]$ can also be used to induce x to either 0 or 1. These penalty terms, however, may impede solution quality. The handling of constraints within a subproblem poses another difficulty. If these constraints are handled by using the penalty method, solutions may not be feasible, and local minima cannot be avoided. If relaxation is used, additional multipliers have to be introduced, and this will lead to slow convergence.

In spite of the above difficulties, it is possible to design specific subnetworks to optimally solve subproblems while handling integrality and subproblem constraints if the subproblems are not NP hard. The synergy of Lagrangian relaxation and these specific subnetworks enables LRNN to obtain near optimal solution with quantifiable quality in an efficient manner for complex integer programming problems. In the following, we will apply LRNN to a separable job shop scheduling problems.

3. Job Shop Scheduling via LRNN

In the basic job shop formulation, each job (or part) has its due date, priority, and requires a series of operations for completion. Each operation is to be performed on a machine of a specified machine type for a given period of time. The processing may start only after its preceding operations have been completed, satisfying the *operation precedence constraints.* Furthermore, the number of operations scheduled on a machine type at any time may not exceed the number of machines available, satisfying the *machine capacity constraints*. Through appropriate selection of decision variables, these constraints are formulated in additive forms (Hoitomt, Luh and Pattipati, 1993). The time-based competition goals of on-time deliveries and low inventory are modeled as penalties on delivery tardiness and on releasing raw materials too early. The problem is to determine operation beginning times so that the objective function is minimized. Unlike other prevalent formulations in the lierature, the key feature of our formulation is its *separability.*

Within the LR framework, machine capacity constraints are relaxed by using Lagrange multipliers. Since the formulation is *separable,* the relaxed problem can be decomposed into decoupled *part subproblems* for a given set of multipliers. Each subproblem represents the scheduling of a part to minimize its tardiness and earliness penalties and the costs for using machines (as reflected by the values of Lagrangian multipliers for different machine types at various time slots).

Each subproblem can be viewed as a multistage optimization problem, and can be solved by using dynamic programming (DP). A typical DP structure is

shown in Figure 1. With stages corresponding to operations and states corresponding to operation beginning times, the backward DP algorithm starts with the last stage, and computes the tardiness penalties and machine utilization costs. As the algorithm moves backwards, cumulative costs of individual states belonging to a particular stage are computed based on the stage-wise costs and the minimum of the cumulative costs for the succeeding stage, subject to allowable state transitions as delineated by operation precedence constraints. This minimization can be efficiently implemented by pair-wise comparison, starting from the last state (largest possible operation beginning time) of the succeeding stage (Wang, Luh, Zhao and Wang 1997). The optimal subproblem cost is then obtained as the minimum of the cumulative costs at the first stage, and the optimal beginning times for individual operations can be obtained by tracing the stages forwards.

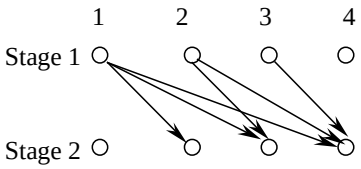

Figure 1. DP Structure

Iterative adjustment of multipliers, repeated resolutions of subproblems, and the final heuristic adjustment of subproblem solutions lead to near-optimal solutions of the original problem.

4. Neural Dynamic Programming

The above LR approach can be naturally mapped onto an LRNN presented in Section 2. The key challenge is to develop efficient sub-networks to solve part subproblems. To effectively handle integer variables and operation precedence with each part, neural dynamic programming (NDP) is developed[1]. The key idea is to make the best use of the DP structure already existed, and implement the DP functions by neurons. In doing this, the DP structure illustrated in Figure 2 is maintained, and each state is represented by a neuron to obtain the cumulative cost by adding up two values, the stage-wise cost derived from multipliers and the minimum of the cumulative costs of the succeeding stage. The *pair-wise comparison* to obtain the minimum cumulative costs of the succeeding stage is carried out through the introduction of another layer of "*comparison neurons*." The connections of comparison neurons and "state neurons" are subject to state transitions as shown in Figure 2, where comparison neurons are represented by gray circles. The traditional backward DP algorithm is thus mapped onto a neural network with simple topology and elementary functional requirements that can be implemented in hardware. The number of neurons required for subproblem i is roughly twice the number of states in its DP structure, i.e., $2\times J_i\times K$, where J_i is the number of operations for part i, and K the time horizon.

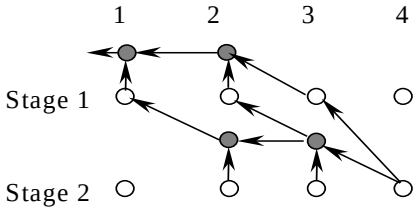

Figure 2. NDP Structure

Since the DP structure is fully exploited in NDP, integrality is satisfied and local minima of subproblem solutions are avoided. In addition, the solution satisfies all subproblem constraints that are handled by the standard DP. Difficulties such as infeasibility, local minima, and slow convergence of subproblem solutions encountered by using the penalty or relaxation method do not exist any more.

The stage-wise costs in NDP depend on multipliers, and change continuously as multipliers are updated. Therefore at any instant of time, the NDP results may not be optimal for the multipliers at that snap shot. However, the NDP can obtain the optimal solution after the signal propagated from the last stage to the first stage, and this takes almost *zero time* in hardware implementation. Thus the overall convergence of LRNN is similar to the LRNN presented in Section 2.

5. Testing Results

Testing of LRNN for basic job shop scheduling problems described in Sections 3 and 4 has been conducted by simulating the LRNN and then feeding the results to our previous heuristics. At the final iteration, multipliers are fixed and optimal subproblem solutions are obtained by NDP. In this way, the dual cost is calculated and the duality gap provides a measure of solution quality. Exciting results for problems of various sizes have been obtained on a Pentium Pro 200 MHz PC as summarized in the following Table.

[1] This is fundamentally different from the "Neuro-Dynamic Programming" of Bertsekas and Tsitsiklis (1996) which was developed for the better training of feedforward neural networks.

Table 4.1. Testing results for LRNN

Size	P	O	M	D	CPU
Small	6	12	5	0.0%	0:02
Middle	20	200	10	0.4%	1:00
Large	82	752	14	6.9%	9:34

Part, Operation, Machine, Duality-Gap are respectively represented by their first letters in the table. CPU-Time is in minute:second format.

It is amazing to see that the performance of the LRNN, in terms of sizes of problems solved, solution quality, and CPU times, is much better than what has been reported in the neural network scheduling literature. Examples include the 14 part problem by Satake, 1994, and the 20 part problem by Sabuncuoglu 1996. The results are also very close to what were obtained by our previous LR/DP approach, which is believed to be at the cutting edge of optimization-based scheduling methods.

6. Conclusion

The convergence proof of LRNN presented in this paper provides a framework allowing creative implementation variations. The specific LRNN for job shop scheduling contains the novel neural dynamic programming to overcome the difficulties associated with local minima and solution infeasibility encountered by conventional Hopfield type networks. Numerical testing demonstrated that the method is much better than what has been reported in the neural network literature, and results should be much improved when the method is implemented in hardware.

Acknowledgment

This work was supported in part by the National Science Foundation grant DMI-9500037.

References

1. R. Dorfman, P. A. Samuelson, and R. Solow, *Linear Programming and Economic Analysis*, McGraw-Hill, 1958, pp. 63, fn. 1.
2. K. J. Arrow, L . Hurwitz and H. Uzawa, *Studies in Linear and Nonlinear Programming*, Stanford Univ. Press, 1958, pp. 133-145.
3. D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, Massachusetts, 1996.
4. Y. P. S. Foo and Y. Takefuji, "Stochastic Neural Networks for Solving Job-shop Scheduling," Proc. *IEEE IJCNN*, 88, 1988(a), pp. 275-290.
5. Y. P. S. Foo and Y. Takefuji, "Integer-linear Programming Neural Networks for Job-shop Scheduling," *Proc. IEEE IJCNN*, 88, 1988(b), pp. 341-348.
6. D. J. Hoitomt, P. B. Luh and K. R. Pattipati, "A Practical Approach to Job Shop Scheduling Problems," *IEEE Transactions on Robotics and Automation*, Vol. 9, No. 1, February 1993, pp. 1-13.
7. J. J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problems," *Biol. Cybern.*, Vol. 52, 1985, pp. 141-152.
8. Z. Li, "Improving Convergence and Solution Quality of Hopfield-Type Neural Networks with Augmented Lagrange Multipliers," *IEEE Transactions on Neural Networks*, Vol. 7, No. 6, November 1996, pp. 1507-1516.
9. I. Sabuncuoglu and B. Gurgun, "A Neural Network Model for Scheduling Problems," *European Journal of Operational Research*, Vol. 9, No. 2, Sep. 1996, pp. 288-299.
10. K. Satake, K. Morikawa and N. Nakamura, "Neural Network Approach for Minimizing the Makespan of the General Job-shop," *International Journal of Production Economics*, Vol. 33, No. 1, Jan. 1994, pp. 67-74.
11. E. Wacholder, J. Han and R. C. Mann, "A Neural Network Algorithm for the Multiple Traveling Salesmen Problem," *Biol. Cybern.*, Vol. 61, 1989, pp. 11-19.
12. J. Wang, P. B. Luh, X. Zhao and J. Wang, "An Optimization-Based Algorithm for Job Shop Scheduling," *SADHANA*, Vol. 22, Part 2, April 1997, pp. 241-256.
13. T. M. Willems and L. E. M. W. Brandts, "Implementing Heuristics as an Optimization Criterion in Neural Networks for Job-shop Scheduling," *Journal of Intelligent Manufacturing*, Vol. 6, No. 6, Dec. 1995, pp. 377-387.
14. V. Wilson and G. S. Pawley, "On the Stability of the Traveling Salesman Problem Algorithm of Hopfield and Tank," *Biological Cybernetics*, Vol. 58, 1988, pp. 63-70.
15. D. N. Zhou, V. Cherkassky, T. R. Baldwin and D. E. Olson, "A Neural Network Approach to Job-shop Scheduling," *IEEE Trans. Neural Networks*, Vol. 2, No. 1, Jan. 1991, pp. 175-179.
16. X. Zhao, P.B. Luh, and J. Wang, "The Surrogate Gradient Algorithm for Lagrangian Relaxation," *Journal of Optimization Theory and Applications*, 1999 (to appear).