Lagrangian Relaxation Neural Networks for Job Shop Scheduling

Peter B. Luh, *Fellow, IEEE*, Xing Zhao, Yajun Wang, and Lakshman S. Thakur

***Abstract*—Manufacturing scheduling is an important but difficult task. In order to effectively solve such combinatorial optimization problems, this paper presents a novel Lagrangian relaxation neural network (LRNN) for separable optimization problems by combining recurrent neural network optimization ideas with Lagrangian relaxation (LR) for constraint handling. The convergence of the network is proved, and a general framework for neural implementation is established, allowing creative variations. When applying the network for job shop scheduling, the separability of problem formulation is fully exploited, and a new neuron-based dynamic programming is developed making innovative use of the subproblem structure. Testing results obtained by software simulation demonstrate that the method is able to provide near-optimal solutions for practical job shop scheduling problems, and the results are superior to what have been reported in the neural network scheduling literature. In fact, the digital implementation of LRNN for job shop scheduling is similar to the traditional LR approaches. The method, however, has the potential to be implemented in hardware with much improved quality and speed.**

***Index Terms*—Integer optimization, Lagrangian relaxation, manufacturing scheduling, neural networks.**

I. 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 the inherent problem complexity and the sizes of real problems. This paper is to explore novel neural network optimization techniques to effectively solve job shop scheduling problems. Historically, neural networks for unconstrained optimization were developed 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 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 and can be obtained at a very fast speed [8].

For constrained optimization, the Hopfield-type recurrent networks have been based on the well-known "penalty methods," which convert a constrained problem to an unconstrained one by having penalty terms on constraint violations [8]. 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 through the fine tuning of algorithm parameters. The tradeoff, however, is generally difficult to make. For problems with integer variables, Hopfield networks approximate integer variables by continuous ones and induce integrality by using "high gain" functions or having additional constraints. These approaches, however, introduce convergence difficulties and impede solution quality. In addition, Hopfield-type networks may possess many local minima. Since escaping from local minima is not an easy task [19], the solution quality depends highly on initial conditions.

Hopfield-type networks and their variations have been developed for job shop scheduling [5], [6], [20]. Although these models demonstrate the possibility of using neural networks for solving small scheduling problems, they suffer from the above-mentioned difficulties. In addition, 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 within the job shop context [14], [15], [18]. The results, however, may be far from optimal.

The recent developments on neural networks for constrained optimization include combining Hopfield-type networks optimization ideas with *Lagrangian relaxation* (LR) or *augmented LR* for constraint handling, showing significant improvement on solution quality [10], [16]. The convergence of the resulting LRNN, however, has not been fully established, and integer variables are still approximated by continuous ones. Furthermore, with "traveling salesman problems" as the reference model by most researchers, the method development has been problem-specific, overlooking many important issues and missing great opportunities.

In this paper, the convergence of LRNN for constrained optimization problems is established in Section II. In the proof, there are no requirements on the differentiability of functions nor the continuity of decision variables, as long as the evolution

Manuscript received November 20, 1998; revised August 11, 1999. This paper was recommended for publication by Associate Editor Y. Narahari and Editor R. Volz upon evaluation of the reviewers' comments. This work was supported by the National Science Foundation under Grant DMI-9500037 and Grant DMI-9813176.

P. B. Luh is with the Department of Electrical and Systems Engineering, University of Connecticut, Storrs, CT 06269-2157 USA.

X. Zhao was with the Department of Electrical and Systems Engineering, University of Connecticut, Storrs, CT 06269-2157 USA. He is now with i2 Technologies, Cambridge, MA 02147 USA.

Y. Wang was with the Department of Electrical and Systems Engineering, University of Connecticut, Storrs, CT 06269-2157 USA. He is now at 2704 Birch Street #C, Alhambra, CA 91801 USA.

L. S. Thakur is with the School of Business Administration, Univerity of Connecticut, Storrs, CT 06269 USA.

Publisher Item Identifier S 1042-296X(00)03013-5.

of decision variables leads the so-called "surrogate dual" to decrease to a global minimum. Thus, a general framework for the implementation of "decision neurons" is provided allowing creative variations.

Building on this framework, LRNN is extended to solve separable integer optimization problems in Section III. It is well known that NP-hard integer optimization problems are difficult to solve. For separable problems where subproblems can be efficiently solved, however, LRNN can be a powerful approach. In LRNN, system-wide coupling constraints are relaxed and the problem is decomposed into many smaller and easier subproblems. Integer variables in these subproblems are represented directly by "discrete decision neurons" without approximation. Local constraints are then enforced by specifically designed subnetworks. These ideas enable LRNN to overcome the difficulties of traditional networks in handling constraints and integer variables, and obtain near-optimal solutions efficiently for complex separable integer programming problems.

As a specific example, LRNN is applied to separable job shop scheduling in Section IV. In this case, LRNN includes many subnetworks, one for each job (or part). To effectively handle integer variables and constraints for each job, a novel neuron-based dynamic programming (NBDP) is developed making innovative use of the dynamic programming (DP) structure with simple topology and elementary functional requirements. Testing results in Section V obtained by software simulation demonstrate that the method is able to provide near-optimal solutions for practical job shop scheduling problems, and the results are much better than what have been reported in the neural network scheduling literature. In fact, the digital implementation of LRNN for job shop scheduling is similar to the traditional LR approaches. The method, however, has the potential to be implemented in hardware with much improved quality and speed.

II. LRNN

A. Problem Formulation

Consider the following separable convex programming problem:

$$\min_x J \equiv \sum_{i=1}^{I} J_i(x_i) \tag{2.1}$$

$$\text{s.t.} \sum_{i=1}^{I} g_i(x_i) \leq 0, \qquad i = 1, \cdots, I \tag{2.2}$$

where $x_i \in R^{N_i}$ is an $N_i \times 1$ continuous decision variable with $\sum_{i=1}^{I} N_i = N$, $g_i(x_i)$ is an $M \times 1$ function, I is the number of subproblems, and$J_i(x_i)$ and $g_i(x_i)$ are convex and differentiable functions. For clarity of presentation, only inequality constraints are considered here (equality constraints can be handled similarly without any theoretical difficulties). Since both the objective function and constraints are additive, the problem is separable.

B. LR

Playing a fundamental role in constrained optimization over the decades, LR 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 relaxed problem 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

$$LD : \max_{\lambda \geq 0} L(\lambda). \tag{2.6}$$

Maximizing the dual without its explicit representation can be done by several methods, including the most widely used gradient method described by

$$\lambda^{k+1} = \max\{0, \lambda^k + \alpha^k \nabla L(\lambda^k)\} \tag{2.7}$$

with

$$\nabla L(\lambda^k) = \sum_{i=1}^{I} g_i(x_i(\lambda^k))$$

and

$$x_i(\lambda^k) = \arg\min_{x_i} \left[J_i(x_i) + (\lambda^k)^T g_i(x_i) \right].$$

Here, 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 a minimum solution (maybe nonunique) of the relaxed problem given the optimal multipliers λ^*. For convex programming problems, the optimal solution to the primal problem is included in x^* [2]. In this case, the λ^* and a primal optimal solution x^* form a Lagrange multiplier-optimal solution pair (λ^*, x^*) or a saddle point [2].

C. LRNN

LR has recently been combined with neural networks to solve constrained optimization problems. Since the dual function is always concave, the key idea of LRNN is to create a network to let the negative dual be the energy function shown in Fig. 1. If

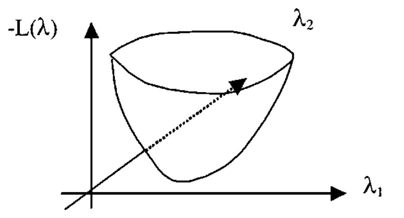


Fig. 1. Negative dual function.

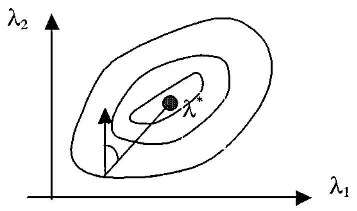


Fig. 2. Multiplier updating directions.

this can be done, then the negative dual will naturally approach its minimum (or the dual will approach its maximum) as the network converges, and then $x*$ can be found easily. However, since the negative dual is not explicitly available but must be obtained through the resolution of the relaxed problem (2.3) [or subproblems (2.4)] for various values of λ, the construction of the network is a bit complicated. Nevertheless, since there is no constraint in the relaxed problem, (2.3) can be solved by a recurrent neural network [or (2.4) can be solved by multiple subnetworks]. The crux of LRNN is to merge these two constructs, one for the negative dual and the other for the relaxed problem, and let them feed each other and converge simultaneously. In 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 then be described by the following differential equations:

$$\frac{d\lambda_m(t)}{dt} = \begin{cases} 0, & \text{for } \lambda_m(t) = 0 \text{ and } \dfrac{\partial \tilde{L}(\lambda(t), x(t))}{\partial \lambda_m(t)} < 0, \\ \alpha(t)\dfrac{\partial \tilde{L}(\lambda(t), x(t))}{\partial \lambda_m(t)}, & \text{otherwise,} \end{cases} \quad m = 1, \cdots, M \tag{2.8}$$

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

with

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

Here, $\alpha(t)$ and $\beta(t)$ are positive coefficients and can be time-varying. The energy function $\tilde{L}(t)$ used in LRNN is a continuous-time version of the "surrogate dual function" defined as

$$\tilde{L}(\lambda, x) \equiv \sum_{i=1}^{I} J_i(x_i) + \lambda^T \sum_{i}^{I} g_i(x_i). \tag{2.11}$$

In (2.11), minimization over x is not required as opposed to the definition of the dual function in (2.3). This surrogate dual is introduced because in a traditional LR method, the relaxed problem is optimally solved, and the dual value and gradient are obtained to update multipliers. Since LRNN does not wait for the convergence of the relaxed problem, the recurrent network obtains approximate dual values and gradients—surrogate dual values and surrogate gradients—according to (2.9). Based on the surrogate dual values and gradients, the multiplier neurons evolve according to (2.8) at the same time. The proof of convergence, the extension to integer variables, and handling problems with local constraints are the challenging issues.

D. Convergence of LRNN

The convergence of a specific implementation of (2.8) and (2.9) with $\alpha(t) = \beta(t) \equiv 1$ was established in 1958 for *strictly* convex problems [1]. This was done within the context of reaching the saddle point (λ^*, x^*). The approach, known as the "differential multiplier method," was developed years before the birth of neural networks. Since this method leads asymptotically to a periodic solution when both the objective function and constraints are linear [4], a modified version was developed for convex problems (not necessarily strictly convex) through a nonlinear transformation of constraints [1]. This nonlinear transformation, however, destroys the separability of the original problem. In addition, the differential multiplier method is based on the steepest descent idea to update continuous decision neurons (2.9). The dynamics of decision neurons, however, can be made more general to cover a broader range of applications, e.g., discrete decision neurons, as will be illustrated later.

Motivated by the ideas presented in [9] and [21], the following steps establish the convergence of LRNN for convex programming (not necessarily strictly convex) without destroying the separability of the original problem. What is more important is that it can be extended to nonconvex programming problems, and provides a general framework for the implementation of decision neurons allowing numerous creative variations.

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

$$\tilde{L}(t) < L^* \tag{2.12}$$

then the gradient of $\lambda(t)$ is always at an acute angle with the direction toward λ^* (shown in Fig. 2), i.e.,

$$0 < (\lambda^* - \lambda(t))^T \frac{d\lambda(t)}{dt}. \tag{2.13}$$

Proof: Since minimization is performed for $L(\lambda)$ according to (2.3), the surrogate dual always satisfies

$$L(\lambda) \leq \tilde{L}(\lambda, x). \tag{2.14}$$

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

$$L^* = L(\lambda^*) \leq \tilde{L}(\lambda^*, x(t)). \tag{2.15}$$

From (2.10) and (2.11), the right-hand side of (2.15) is

$$\begin{aligned}\tilde{L}(\lambda^*, x(t)) &= \sum_{i=1}^{I} J_i(x_i(t)) + \lambda^{*^T} \sum_{i}^{I} g_i(x_i(t)) \\ &= \tilde{L}(t) + (\lambda^* - \lambda(t))^T \sum_{i=1}^{I} g_i(x_i(t)) \\ &= \tilde{L}(t) + (\lambda^* - \lambda(t))^T \frac{\partial \tilde{L}(t)}{\partial \lambda(t)}. \end{aligned} \tag{2.16}$$

Thus (2.15) can be written as

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

Given (2.12), this yields

$$0 < L^* - \tilde{L}(t) \leq (\lambda^* - \lambda(t))^T \frac{\partial \tilde{L}(t)}{\partial \lambda(t)}. \tag{2.18}$$

From (2.8) and (2.18), we have

$$0 < \alpha(t)(\lambda^* - \lambda(t))^T \frac{\partial \tilde{L}(t)}{\partial \lambda(t)} \leq (\lambda^* - \lambda(t))^T \frac{d\lambda(t)}{dt} \tag{2.19}$$

and (2.13) is proved. Q.E.D.

Proposition 2: If the initial point satisfies

$$\tilde{L}(0) < L^* \tag{2.20}$$

and the coefficient in the dynamics of Lagrangian neurons satisfies

$$\alpha(t) \leq \frac{L^* - \tilde{L}(t)}{\left\| \partial \tilde{L}(t) / \partial \lambda(t) \right\|^2} \tag{2.21}$$

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

Proof: From (2.8)–(2.10)

$$\begin{aligned}\frac{d\tilde{L}(t)}{dt} &= \frac{\partial \tilde{L}(t)}{\partial \lambda(t)}^{\mathrm{T}} \cdot \frac{d\lambda(t)}{dt} + \frac{\partial \tilde{L}(t)}{\partial x(t)}^{\mathrm{T}} \cdot \frac{dx(t)}{dt} \\ &\leq \alpha(t) \left\| \frac{\partial \tilde{L}(t)}{\partial \lambda(t)} \right\|^2 - \beta(t) \left\| \frac{\partial \tilde{L}(t)}{\partial x(t)} \right\|^2. \end{aligned} \tag{2.22}$$

With (2.21), we have

$$\frac{d\tilde{L}(t)}{dt} \leq (L^* - \tilde{L}(t)) - \beta(t) \left\| \frac{\partial \tilde{L}(t)}{\partial x(t)} \right\|^2 \leq L^* - \tilde{L}(t). \tag{2.23}$$

Together with (2.20), it can be shown that

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

Q.E.D.

Based on Propositions 1 and 2, 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 solution pair (λ^*, x^*) of the dual problem as long as

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

and

$$0 < \alpha(t) \leq \frac{L^* - \tilde{L}(t)}{\left\| \partial \tilde{L}(t) / \partial \lambda(t) \right\|^2}. \tag{2.24}$$

Proof: From Propositions 1 and 2, the gradient of $\lambda(t)$ is always at an acute angle with the direction pointing to λ^*. From (2.13), we have

$$\frac{d\|\lambda^* - \lambda(t)\|^2}{dt} = -2(\lambda^* - \lambda(t))^T \frac{d\lambda(t)}{dt} < 0. \tag{2.25}$$

This means that $\lambda(t)$ gets closer and closer to λ^*. Now suppose that $(\lambda(t), x(t))$ converges to (λ^+, x^+). Then, with the convexity of the problem, (2.9) implies that x^+ is a global minimal solution for the given λ^+, and (2.8) implies that $L(\lambda^+) = L^*$. Thus, (λ^+, x^+) is an optimal solution pair of the dual problem. It can also be shown by contradiction that $(\lambda(t), x(t))$ always converges to a certain point. Thus, the theorem is proved. Q.E.D.

Convexity is required in Theorem 1 so that the decision neuron dynamics (2.9) converges to a global minimum x^*. For a nonconvex problem, x^+ may be a local minimum but may not be a global minimum given λ^+; therefore, Theorem 1 may not hold. However, if a global minimum can be guaranteed by the dynamics of decision neurons, LRNN will still converge to (λ^*, x^*) for the nonconvex case.

The above proof is for a particular implementation of LRNN based on the gradient approach (2.9). As can be seen, there are two conditions on decision neuron dynamics for the convergence of LRNN. The first is that $(\partial \tilde{L}(t)/\partial x(t))^T (dx(t)/dt) < 0$ as required by (2.22) and (2.23). The other is that global minimum should be guaranteed as required by Theorem 1. This implies that as long as the dynamics of the decision neurons cause the surrogate dual $\tilde{L}(t)$ to decrease to a global minimum for the given multipliers, i.e.,

$$\tilde{L}(\lambda(t), x(t) + \Delta x(t)) < \tilde{L}(\lambda(t), x(t)) \tag{2.26}$$

and

$$\Delta x(t) = 0, \text{ if and only if } x(t) = \arg\min_{x(t)} \tilde{L}(\lambda(t), x(t)) \tag{2.27}$$

then LRNN will converge. Note that this is true in general regardless whether x is continuous or discrete. In fact, requirements such as convexity of the problem, differentiability of functions or the continuity of the decision variables are not necessary. Creative variations beyond (2.9) can therefore be developed to fit the specific needs of individual problems [9], [21]. For example, it will be shown in later sections that decision neurons can also be discrete variables following various dynamics.

Example 1: This example is to show the structure of LRNN and its convergence. Consider the following quadratic programming problem:

$$\begin{aligned} \min \quad & \frac{1}{2} x^T Q x + c^T x, \\ \text{s.t.} \quad & Ax = b \end{aligned}$$

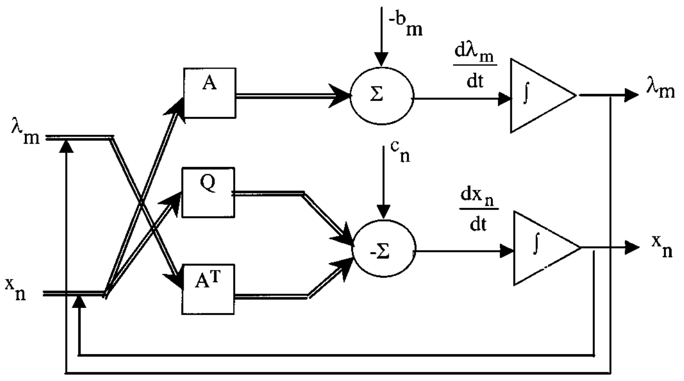


Fig. 3. The structure of LRNN for quadratic programming.

where $x \in R^N$ are continuous decision variables, Q a positive definite $N \times N$ matrix, and A is an $M \times N$ matrix. The LRNN dynamics can be described as

$$L(\lambda, x) = \frac{1}{2}x^TQx + c^Tx + \lambda^T(Ax - b)$$

$$\frac{dx_n}{dt} = -\left(\sum_{j=1}^{N} Q_{nj}x_j + c_n + \sum_{m=1}^{M} A_{mn}\lambda_m\right), \quad n = 1, \cdots, N$$

and

$$\frac{d\lambda_m}{dt} = \sum_{n=1}^{N} A_{mn}x_n - b_m, \qquad m = 1, \cdots, M.$$

The structure of LRNN is shown in Fig. 3. To show the convergence of the above network, the following problem is tested:

$$\min_{x_1, x_2} \quad 0.5x_1^2 + 0.1x_2^2$$

$$\text{s.t.} \quad x_1 - 0.2x_2 \geq 48$$
$$5x_1 + x_2 \geq 250$$

and

$$x_1, x_2 \in R.$$

The two constraints are relaxed by introducing multipliers λ_1 and λ_2, and the problem is decomposed into two subproblems, each with one decision variable. This problem can then be mapped onto an LRNN where the surrogate dual is

$$\begin{aligned}\tilde{L}(\lambda_1, \lambda_2, x_1, x_2) =& 0.5x_1^2 + 0.1x_2^2 + \lambda_1(48 - x_1 + 0.2x_2) \\ &+ \lambda_2(250 - 5x_1 - x_2) \\ =& (0.5x_1^2 - \lambda_1x_1 - 5\lambda_2x_1) + (0.1x_2^2 \\ &+ 0.2\lambda_1x_2 - \lambda_2x_2) + 48\lambda_1 + 250\lambda_2.\end{aligned}$$

The dynamics of the multiplier neurons are

$$\frac{d\lambda_1}{dt} = \alpha(t)\frac{\partial\tilde{L}}{\partial\lambda_1} = 48 - x_1 + 0.2x_2$$

and

$$\frac{d\lambda_2}{dt} = \alpha(t)\frac{\partial\tilde{L}}{\partial\lambda_2} = 250 - 5x_1 - x_2.$$

The dynamics of the decision neurons are

$$\frac{dx_1}{dt} = -\beta\frac{\partial\tilde{L}}{\partial x_1} = -\beta(x_1 - \lambda_1 - 5\lambda_2)$$

and

$$\frac{dx_2}{dt} = -\beta\frac{\partial\tilde{L}}{\partial x_2} = -\beta(0.2x_2 + 0.2\lambda_1 - \lambda_2).$$

The above LRNN is simulated by using a set of difference equations with the initial point $[\lambda_1(0), \lambda_2(0), x_1(0), x_2(0)] = [0, 0, 0, 0]$. The coefficient $\alpha(t)$ is calculated as $(L^* - \tilde{L}(t))/ \|\partial\tilde{L}(t)/\partial\lambda(t)\|^2$based on (2.21), where $L^* = 1203$ is assumed to be known for simplicity. The coefficient β can be any positive value, and $\beta = 1$ is used in the simulation. The trajectories of multipliers, decision variables, and $\tilde{L}$ are shown in Fig. 4. It can be seen that LRNN converges to the known optimal solution $[\lambda_1^* \ \lambda_2^* \ x_1^* \ x_2^*] = [22 \ 5.4 \ 49 \ 5]$.

In practice, L^* is not known and an estimation has to be used. There are two cases when the estimation of L^* is off. For an underestimation, the system will converge to the estimated value, and the resulting solution is not optimal. For an overestimation, there is no stable point in the system, and the surrogate dual will oscillate. How to estimate L^* properly, however, is problem dependent. A feasible solution obtained by heuristics can be used as an upper bound for L^*, and a dual solution is always a lower bound for L^*. Based on this information, the estimation can be adjusted dynamically. Several techniques to estimate L^* using the lower bound and the upper bound have been introduced in [2].

III. LRNN for Separable Integer Programming Problems

A. LR for Integer Programming

Integer programming problems are generally difficult to solve because of their inherent combinatorial complexity. For separable integer programming problems, however, LR has proven to be particularly powerful. Consider now the problem described by (2.1) and (2.2) with variables x restricted to be integers, i.e., $x_i \in Z^{N_i}$ where Z is the set of integers. For such a problem, the hard coupling constraints (2.2) are first relaxed through the introduction of Lagrangian multipliers. The relaxed problem can then be decoupled into individual subproblems. If these subproblems belong to class-P problems, they can be efficiently solved for a given set of multipliers. Multipliers are then iteratively adjusted based on the level of constraint violation. For integer programming problems, however, the x^* from the relaxed problem may not be feasible [12]. Simple heuristics are thus applied to adjust subproblem solutions to form a feasible solution satisfying all constraints at the termination of such updating iterations. Since subproblem solutions will tend to satisfy the relaxed coupling constraints and approach an optimal feasible solution over the iterations, LR provides a powerful approach to obtain near-optimal solutions for NP-hard integer programming problems. 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.

B. LRNN with Discrete Decision Neurons

LRNN can be constructed for separable integer programming problems based on the above idea. How to handle integer variables, however, has been a challenging issue. The traditional neural optimization for 0–1 integer programming problems is

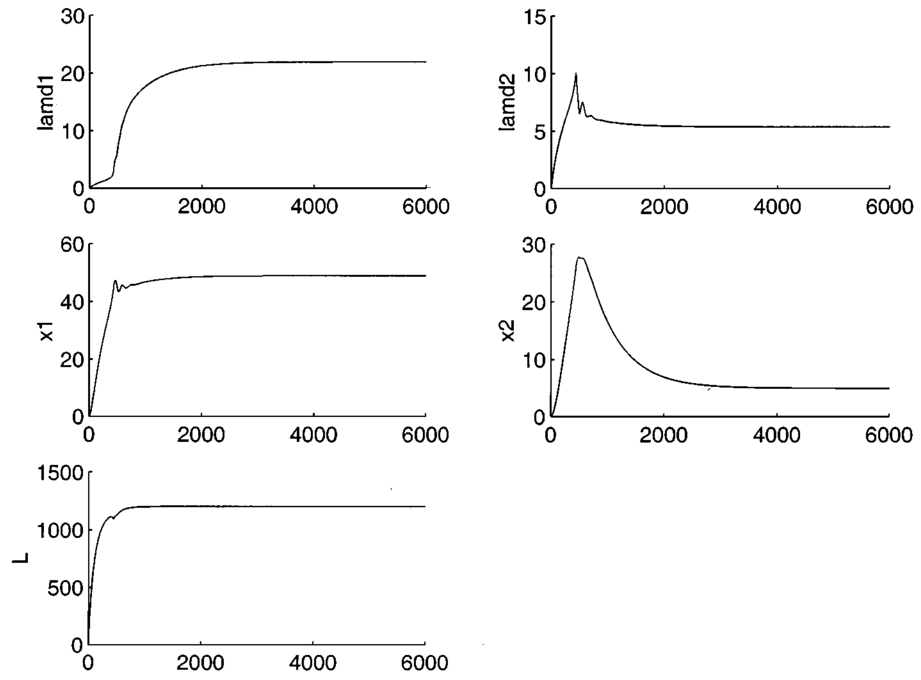


Fig. 4. Neuron trajectories in LRNN.

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 [8]. If the gain is too high, however, the problem will be ill conditioned. The penalty term $[x \cdot (1-x)] = 0$ can also be used to induce x to either 0 or 1. These penalty terms, however, may impede solution quality. Approximating integer variables by continuous ones is therefore not satisfactory.

Based on the general framework provided in Section II, decision neurons can in fact be discrete, and approximating them by continuous ones is not necessary. This leads to LRNN with discrete decision neurons introduced. According to (2.26), the dynamics of these discrete decision neurons in LRNN should cause the surrogate dual to decrease to a global minimum, i.e.,

$$\tilde{L}(\lambda(t), x(t) + \Delta x(t)) < \tilde{L}(\lambda(t), x(t)),\ x(t) \in Z^N,\quad \Delta x(t) \in Z^N \tag{3.1}$$

and

$$\Delta x(t) = 0, \qquad \text{if and only if } x(t) = \arg\min_{x(t)} \tilde{L}(\lambda(t), x(t)). \tag{3.2}$$

For separable problems, a sufficient condition for (3.1) with respect to decomposed subproblems is

$$\tilde{L}_i(\lambda(t), x_i(t) + \Delta x_i(t)) < \tilde{L}_i(\lambda(t), x_i(t)),\quad x_i(t) \in Z^{N_i},\ \Delta x_i(t) \in Z^{N_i} \tag{3.3}$$

and

$$\Delta x_i(t) = 0, \qquad \text{if and only if } x_i(t) = \arg\min_{x_i(t)} \tilde{L}(\lambda(t), x_i(t)). \tag{3.4}$$

LRNN is thus composed of multiple subnetworks satisfying (3.3) and the multiplier neurons as described by (2.8), and they feed each other and converge to (λ^*, x^*).

To illustrate the above idea, reconsider Example 1 of Section II again except that the decision variables are now assumed to be integers. The dynamics of decision neurons of LRNN can be described as

$$x_1(t+\Delta t) = \begin{cases} x_1(t)+1, & \text{if } x_1(t) - \lambda_1(t) - 5\lambda_2(t) + 0.5 < 0 \\ x_1(t)-1, & \text{if } -(x_1(t) - \lambda_1(t) - 5\lambda_2(t)) + 0.5 < 0 \\ x_1(t), & \text{otherwise} \end{cases}$$

and

$$x_2(t+\Delta t) = \begin{cases} x_2(t)+1, & \text{if } 0.2x_2(t) + 0.2\lambda_1(t) - \lambda_2(t) + 0.2 < 0 \\ x_2(t)-1, & \text{if } -(0.2x_2(t) + 0.2\lambda_1(t) - \lambda_2(t)) + 0.2 < 0 \\ x_2(t), & \text{otherwise.} \end{cases}$$

The above dynamics are designed to satisfy (3.3), and simulation shows that the network converges to the optimal solution $[\lambda_1^*\ \lambda_2^*\ x_1^*\ x_2^*] = [22\ 5.4\ 49\ 5]$.

Since integer variables are represented directly by discrete neurons, high gain function, or additional penalty terms to enforce integrality is no longer needed in LRNN. The major concern for LRNN, however, is to design subnetwork satisfying (3.3).

C. Subnetworks

Thus far, only system-wide constraints are considered in the formulation, and once they are relaxed, there is no constraint within a subproblem. In most applications, however, this is not the case, and subproblem i may have to satisfy many local constraints involving variable x_i. These local constraints add another level of complexity to the design of subnetworks. If these constraints are handled by the penalty method, solutions may not be feasible, and local minima cannot be avoided. If relaxation is used, additional multipliers have to be introduced, leading to slow convergence.

In spite of the above difficulties, it is possible to design specific subnetworks to satisfy (3.3) while handling local constraints if the subproblems are not NP-hard. The synergy of LR with these specific subnetworks enables LRNN to obtain near-optimal solution with quantifiable quality in an efficient manner for complex integer programming problems. Constructing subnetworks to effectively solve subproblems, however, is a challenging task and may be problem dependent. In the following, we will apply LRNN to job shop scheduling problems and design specific subnetworks to handle the subproblems.

IV. Job Shop Scheduling via LRNN

A. Problem Formulation

In applying LRNN to job shop scheduling, the separable structure of our previous job shop formulation is exploited. In the formulation, each "part" has its due date, weight (or priority), and requires a series of operations for its completion. Each operation is to be performed on a machine of a specified type for a given period of time. The processing may start only after its preceding operations have been completed, satisfying the *operation precedence constraint*. Furthermore, the number of operations assigned to a machine type may not exceed the number of machines available at any time, satisfying the *machine capacity constraints*. The problem is to determine operation beginning times so that the total weighted part earliness and tardiness is minimized. Through appropriate selection of decision variables, these constraints are formulated in additive forms [7], [17]. Unlike other prevalent formulations [13], the key feature of our formulation is its *separability*. The variables used in the problem formulation are listed below as follows.

B_i	Beginning time of the first operation of part i.
b_{ij}	Beginning time of operation (i, j)—the jth operation on part i.
C_i	Completion time of the last operation of part i.
c_{ij}	Completion time of operation (i, j).
D_i	Due date of part i.
E_i	Earliness of part i, defined as $E_i = \max[0, S_i - B_i]$.
H	Set of all machine types.
h	Machine type index, $h \epsilon H$.
i	Part index ($i = 1, \cdots, I$).
j	Operation index ($j = 1, \cdots, J$).
k	Time index ($k = 1, \cdots, K$).
M_{hk}	Capacity of machine type h at time k.
P_{ijh}	Processing time of operation (i, j) on machine type $h \epsilon H_{ij}$.
S_i	Desired raw material release time for part i.
T_i	Tardiness of part i, defined as $T_i = \max[0, C_i - D_i]$.
W_i	Tardiness weight for part i.
β_i	Earliness weight for part i.
δ_{ijhk}	0-1 operation variable which is one if operation (i, j) is performed on machine type h at time k, and zero otherwise.

The constraints and objective function are briefly presented below.

1) Machine Capacity Constraints: The number of operations assigned to machine type h at time k should be less than or equal to M_{kh}, the number of machines available at that time, i.e.,

$$\sum_{ij} \delta_{ijkh} \le M_{kh}, \qquad k = 1, \cdots, K, \quad h \in H. \tag{4.1}$$

2) Operation Precedence Constraints: An operation cannot be started until its preceding operation has been completed, i.e.,

$$c_{i,j-1} + 1 \le b_{ij}, \qquad i = 1, 2, \cdots, I, \quad j = 2, \cdots, J_i. \tag{4.2}$$

3) Processing Time Requirements: Each operation must be assigned the required amount of time for processing on the specified machine type, i.e.,

$$c_{ij} = b_{ij} + P_{ijh} - 1, \qquad i = 1, \cdots, I, \quad j = 1, 2, \cdots, J_i. \tag{4.3}$$

4) Objective Function: The time-based competition goals of on-time delivery and low inventory are modeled as penalties on delivery tardiness and on releasing raw materials too early, and the objective function is

$$\sum_{i=1}^{I} (W_i T_i^2 + \beta_i E_i^2). \tag{4.4}$$

The problem is to determine operation beginning times b_{ij} for individual operations to minimize (4.4) subject to machine capacity constraints (4.1), operation precedence constraints (4.2), and processing time requirements (4.3). The key feature of this formulation is its *separability*.

B. Solution Methodology

Within the LR framework, machine capacity constraints are relaxed by using Lagrange multipliers π_{kh}, and the "relaxed problem" is given by

$$\min_{\{b_{ij}\}} L \text{ with } L \equiv \sum_i (W_i T_i^2 + \beta_i E_i^2) + \sum_{ij} \sum_{k=b_{ij}}^{c_{ij}} \pi_{kh} - \sum_{kh} \pi_{kh} M_{kh} \tag{4.5}$$

s.t. (4.2) and (4.3). Since the formulation is *separable*, the relaxed problem can be decomposed into the following decoupled part *subproblems* for a given set of multipliers:

$$\min_{\{b_{ij}\}} L_i, \text{ with } L_i \equiv W_i T_i^2 + \beta_i E_i^2 + \sum_{j=1}^{J_i} \sum_{k=b_{ij}}^{c_{ij}} \pi_{kh}, \qquad i = 1, \cdots, I \tag{4.6}$$

s.t. (4.2) and (4.3). Each subproblem represents the scheduling of a single part to minimize its tardiness and earliness penalties and the costs for using machines (as reflected by the values of Lagrangian multipliers for various machine types at different time periods).

Each part subproblem is a multistage optimization problem and can be efficiently solved by using DP with polynomial complexity. A typical DP structure is shown in Fig. 5. 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 backward, cumulative costs of individual states belonging to a particular

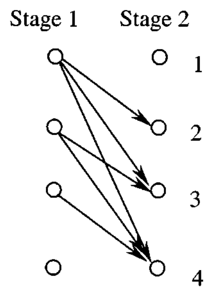


Fig. 5. DP structure.

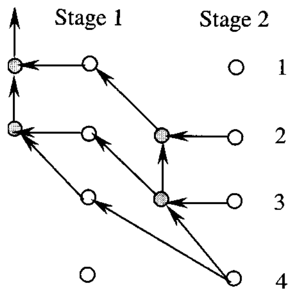


Fig. 6. NBDP structure.

stage are computed based on the stagewise 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 pairwise comparisons, starting from the last state (largest possible operation beginning time) of the succeeding stages [17]. 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 forward.

In the LR approach, multipliers are iteratively adjusted, and the subproblems are repeatedly solved. Such iterative process in practice is terminated before algorithm convergence. The solutions to part subproblems, when put together, are generally associated with an infeasible schedule, i.e., capacity constraints might be violated at some time periods. Heuristic is thus required to adjust subproblem solutions to a feasible solution of the original problem [17]. In the heuristics, a list of immediately performable operations is created in the ascending order of their beginning times from part subproblem solutions. Operations are then scheduled on the required machine types according to this list as machines become available. If the capacity constraint for a particular machine type is violated at time k, a greedy heuristic determines which operations should begin at that time and which ones are to be delayed by one time unit. The subsequent operations of those delayed ones are then delayed by one time unit if precedence constraints are violated. The process repeats until the last operation in the list.

C. NBDP

The above LR approach can be naturally mapped onto an LRNN presented in Section III. The key challenge is to develop efficient subnetworks to solve part subproblems. One approach is to map the above DP states onto neurons and mathematically delineate state transitions by having constraints among neurons. These constraints can then be handled by penalty or relaxation methods. As mentioned in Section III, however, penalty method may lead to infeasibility and local minima, while the relaxation method will cause slow convergence.

To overcome the above difficulties, NBDP is developed.[1] The key idea is to make the best use of the DP structure that already exists and implement the DP functions by neurons. In doing this, the DP structure illustrated in Fig. 5 is utilized where each state is represented by a neuron to obtain the cumulative cost. The neuron will add up two values, the stagewise cost derived from multipliers and the minimum of the cumulative costs of the succeeding stage. The *pairwise 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 Fig. 6, 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 required operations for part i, and K the time horizon.

Since the DP structure is fully exploited in NBDP, the solution satisfies all subproblem constraints that are enforced by DP. Difficulties such as infeasibility, local minima, and slow convergence of subproblem solutions encountered by using the penalty or relaxation methods do not exist any more.

[1]This is fundamentally different from the "neuro-dynamic programming" which was developed for the better training of feedforward neural networks [3].

TABLE I
DATA AND FEASIBLE SCHEDULE FOR CASE 1

Part I	Oper. J	H	p_{ij}	b_{ij}
0	0	1	3	0
	1	2	3	3
	2	0	1	8
1	0	0	4	0
	1	1	3	4
	2	2	2	7
2	0	1	1	3
	1	0	4	4
	2	2	4	9
3	0	2	3	0
	1	1	2	7
	2	0	3	9

TABLE II
SIMULATION RESULTS FOR CASE 2

Primal dimensions MT/M/P/O(a)	Optimization Method	Dual Cost	Primal Cost(b)	Duality Gap(c)	CPU Time(d)
8/8/20/90	SSG	7310	8081	10.55%	0:22
	SG	7346	8081	10.00%	0:46
7/13/40/231	SSG	88758	97680	10.05%	1:49
	SG	89020	97680	9.73%	2:31
8/14/80/447	SSG	590856	648784	9.80%	4:46
	SG	550651	649410	17.94%	5:00

(a)The notation MT/M/P/O provides number of machine types (MT), number of machines (M), number of parts (P) and number of operations (O).
(b)Heuristics are applied after all the subproblems are solved once to obtain a feasible schedule.
(c)Pseudo duality gap is actually used here, since the gap is calculated based on the primal cost obtained from a feasible schedule instead of the true primal optimal.
(d)CPU time is in minute:second.

TABLE III
DATA AND FEASIBLE SCHEDULES FOR THE FIRST PROBLEM IN CASE 2

Notation: Due date (D), Operation (O), Machine_type (M), Beginning_time (BT), Processing_time (PT)

Part	D	O	M	PT	BT
P_1	10	OP_10	M01_01	5	4
P_1		OP_20	M02_01	7	9
P_1		OP_30	M03_01	5	16
P_1		OP_40	M04_01	3	21
P_2	15	OP_10	M01_01	5	10
P_2		OP_20	M02_01	7	20
P_2		OP_30	M03_01	5	27
P_2		OP_40	M04_01	3	32
P_3	20	OP_10	M01_01	5	15
P_3		OP_20	M02_01	7	35
P_3		OP_30	M03_01	5	42
P_3		OP_40	M04_01	3	47
P_4	25	OP_10	M01_01	5	32
P_4		OP_20	M02_01	7	42
P_4		OP_30	M03_01	5	49
P_4		OP_40	M04_01	3	54
P_5	30	OP_10	M01_01	5	22
P_5		OP_20	M02_01	7	49
P_5		OP_30	M03_01	5	56
P_5		OP_40	M04_01	3	61
P_6	10	OP_10	M05_01	4	0
P_6		OP_20	M06_01	8	4
P_6		OP_30	M07_01	4	12
P_7	20	OP_10	M05_01	4	4
P_7		OP_20	M06_01	8	12
P_7		OP_30	M07_01	4	22
P_8	30	OP_10	M05_01	4	23
P_8		OP_20	M06_01	8	36
P_8		OP_30	M07_01	4	46
P_9	40	OP_10	M05_01	4	44
P_9		OP_20	M06_01	8	52
P_9		OP_30	M07_01	4	62
P_10	50	OP_10	M05_01	4	56
P_10		OP_20	M06_01	8	68
P_10		OP_30	M07_01	4	76
P_11	10	OP_10	M01_01	1	0
P_11		OP_20	M02_01	4	1
P_11		OP_30	M03_01	4	5
P_11		OP_40	M04_01	2	9
P_11		OP_50	M05_01	2	11
P_12	15	OP_10	M01_01	1	2
P_12		OP_20	M02_01	4	5
P_12		OP_30	M03_01	4	9
P_12		OP_40	M04_01	2	13
P_12		OP_50	M05_01	2	17
P_13	20	OP_10	M01_01	1	9
P_13		OP_20	M02_01	4	16
P_13		OP_30	M03_01	4	21
P_13		OP_40	M04_01	2	25
P_13		OP_50	M05_01	2	27
P_14	25	OP_10	M01_01	1	21
P_14		OP_20	M02_01	4	27
P_14		OP_30	M03_01	4	32
P_14		OP_40	M04_01	2	36
P_14		OP_50	M05_01	2	38
P_15	30	OP_10	M01_01	1	20
P_15		OP_20	M02_01	4	31
P_15		OP_30	M03_01	4	36
P_15		OP_40	M04_01	2	40
P_15		OP_50	M05_01	2	42
P_16	20	OP_10	M07_01	6	0
P_16		OP_20	M08_01	6	6
P_16		OP_50	M05_01	4	13
P_16		OP_60	M06_01	8	20
P_16		OP_70	M07_01	4	28
P_16		OP_80	M08_01	4	32
P_17	30	OP_10	M07_01	6	6
P_17		OP_20	M08_01	6	12
P_17		OP_50	M05_01	4	19
P_17		OP_60	M06_01	8	28
P_17		OP_70	M07_01	4	36
P_17		OP_80	M08_01	4	40
P_18	40	OP_10	M07_01	6	16
P_18		OP_20	M08_01	6	22
P_18		OP_50	M05_01	4	29
P_18		OP_60	M06_01	8	44
P_18		OP_70	M07_01	4	52
P_18		OP_80	M08_01	4	56
P_19	50	OP_10	M07_01	6	40
P_19		OP_20	M08_01	6	46
P_19		OP_50	M05_01	4	52
P_19		OP_60	M06_01	8	60
P_19		OP_70	M07_01	4	68
P_19		OP_80	M08_01	4	72
P_20	60	OP_10	M07_01	6	56
P_20		OP_20	M08_01	6	62
P_20		OP_50	M05_01	4	68
P_20		OP_60	M06_01	8	76
P_20		OP_70	M07_01	4	84
P_20		OP_80	M08_01	4	88

LRNN can be implemented by analog circuits or by digital circuits. It is clear that given a set of multipliers, NBDP obtains a global optimal solution for the subproblem after the signals propagate from the last stage to the first stage. If NBDP is implemented by an analog circuit and assume that signals can instantly propagate through the stages, then the convergence of LRNN is similar to that of the traditional LR method however with almost zero time to solve subproblems. If signals cannot instantly propagate through the stages, the convergence proof of LRNN is quite complicated since it is difficult to model the propagation and interaction of signals in DP. If NBDP is implemented by a digital circuit, signal propagation has to be explicitly considered in the overall network design. One way is to update multipliers after NBDP solves all the subproblems. This will be referred to as the "SG approach" since it follows the traditional LR framework using the subgradient method to update the multipliers. Another way is to update multipliers after NBDP solves only one subproblem. This will be denoted as the "SSG approach" because it follows a particular implementation of the surrogate subgradient method, where previous results for other subproblems are used to obtain the surrogate dual and the surrogate subgradient [21]. Since one subproblem is minimized while solutions for others are kept the same, the surrogate dual is decreased satisfying (3.1). In fact, the digital version of LRNN is a discrete-time system, and its convergence can be guaranteed if the proof is modified appropriately as what was presented in [21]. Preliminary architectural design of LRNN hardware for job shop scheduling can be found in [11].

Both the SG approach and the SSG approach for the digital version of LRNN will converge. The convergence, however, may be slow for such a combinatorial optimization problem. The algorithm is therefore stopped when certain criteria are satisfied based on CPU time or the number of iterations. Heuristics are then applied to obtain a feasible schedule, and the duality gap will provide a measure of solution quality. For SSG, only the surrogate dual is available. To obtain the true duality gap, multipliers are fixed at the final iteration of SSG, and all subproblems are resolved by using NBDP's. In this way, the dual cost is obtained and the duality gap is calculated.

V. Simulation Results

Testing of LRNN for basic job shop scheduling problems described in Section IV has been conducted by simulating the digital version of LRNN on a PC and then feeding the results to heuristics [17]. Recall that Hopfield-type networks and its variations have been developed for job shop scheduling. Because these methods have difficulties in dealing with integer variables and system-wide as well as local constraints, results reported are mostly for small problems (the 14 parts and 7 machines problem in [15] and the 20 parts and 5 machines problem in [14]). In contrast, LRNN aims at solving practical problems. In our testing, both the SG approach and the SSG approach are simulated, and results for problems of various sizes have been obtained on a Pentium II 400-MHz PC as summarized in the following.

Case 1: This case demonstrates that LRNN generates the optimal schedule for a small problem. There are 3 machine types with 1 machine each and 4 parts with a total of 12 operations. For every part, the due date is –1 and the weight is 5, and there is no part earliness penalty. The operation beginning times obtained by SG and SSG within one second are identical as presented in Table I. in conjunction with the operation processing times p_{ij} and the required machine types h. Since the duality gap obtained is 0, the solution is optimal.

Case 2: This case is to demonstrate that LRNN generates near-optimal schedules for practical problems with data sets based on a factory of Sikorsky Aircraft at Stratford, CT. The algorithm is stopped if the CPU time is more than 5 min or the duality gap is less than 10%. Summary of testing results are presented in Table II. Since the amount of data is substantial, detailed information is only provided for the first problem in the Appendix. Other information is available at our web site.[2]

The small duality gap implies that near-optimal solutions are obtained. Compared with other results in the neural network scheduling literature, LRNN is able to solve much larger problems with satisfactory quality. In fact, the digital version of LRNN can also be considered as the hardware implementation of the traditional LR and DP method (LRDP). Therefore, simulation results should be the same as those obtained by LRDP [17], [21]. Our ultimate goal, however, is to implement LRNN in hardware. In view of NBDP's simple topology and the elementary function requirements as illustrated in Section IV, hardware implementation is feasible. Such an implementation can drastically reduce the computation time, thus allowing more iterations to be performed within a fixed amount of time to improve solution quality. It also allows larger and more complicated problems to be solved within reasonable amount of computation time.

[2][Online]. Available: http://www.engr.uconn.edu/msl/test_data/ra_data

VI. Conclusions

In this paper, a novel LRNN is developed for separable optimization problems by combining recurrent neural network optimization ideas with LR for constraint handling. Its general convergence is established, allowing flexibility for the decision neuron dynamics. A framework is thus established for LRNN to solve this class of problems, where the key is the design of efficient subnetworks for subproblems. As a specific application to job shop scheduling, the novel NBDP is developed to effectively solve part subproblems. With NBDP, difficulties of integer variables and subproblem local minima and solution infeasibility are avoided. Numerical testing demonstrated that the method is able to provide near-optimal solutions for practical job shop scheduling problems and is much better than what have been reported in the neural network scheduling literature. In addition, the method has the potential to be implemented in hardware with much improved quality and speed, where applications are not limited to job shop scheduling problems only. Rather, it can be applied to separable integer or mixed-integer optimization problems with stagewise additive subproblem cost functions, a generic class of problems with many important applications.

Appendix

This appendix presents the input data and the resulting schedules for the first problem of Case 2, as shown in Table III. For this problem, each machine type has one machine, and for every part, the tardiness weight is 1 and there is no part earliness penalty.

Acknowledgment

The authors would like to thank Prof. S.-C. Chang, Prof. T.-D. Chieu, and K.-H. Chen of the National Taiwan University for collaborating on hardware design, and Prof. Y. C. Ho of Harvard University, and Prof. M. Caramanis, Prof. C. Cassandras, and Prof. D. Castanon of Boston University for valuable comments and suggestions.

References

[1] K. J. Arrow, L. Hurwitz, and H. Uzawa, *Studies in Linear and Nonlinear Programming*. Stanford, CA: Stanford Univ. Press, 1958, pp. 133–145.
[2] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA: Athena Scientific, 1995, pp. 418–449, pp. 507–508.
[3] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific, 1996.
[4] R. Dorfman, P. A. Samuelson, and R. Solow, *Linear Programming and Economic Analysis*. New York: McGraw-Hill, 1958, p. 63.
[5] Y. P. S. Foo and Y. Takefuji, “Stochastic neural networks for solving job-shop scheduling,” in *Proc. IEEE 2nd Int. Conf. Neural Networks*, 1988, pp. 275–290.

[6] Y. P. S. Foo and Y. Takefuji, "Integer-linear programming neural networks for job-shop scheduling," in *Proc. IEEE 2nd Int. Conf. Neural Networks*, 1988, pp. 341–348.

[7] D. J. Hoitomt, P. B. Luh, and K. R. Pattipati, "A practical approach to job shop scheduling problems," *IEEE Trans. Robot. Automat.*, vol. 9, pp. 1–13, Feb. 1993.

[8] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, pp. 141–152, 1985.

[9] C. A. Kaskavelis and M. C. Caramanis, "Efficient Lagrangian relaxation algorithms for industry size job-shop scheduling problems," *IIE Trans.*, vol. 30, no. 11, pp. 1085–1097, Nov. 1998.

[10] Z. Li, "Improving convergence and solution quality of hopfield-type neural networks with augmented Lagrange multipliers," *IEEE Trans. Neural Networks*, vol. 7, pp. 1507–1516, Nov. 1996.

[11] P. B. Luh, X. Zhao, L. S. Thakur, K. H. Chen, T. D. Chieu, and S. C. Chang, "Architectural design of neural network hardware for job shop scheduling," *CIRP Annals*, vol. 48, no. 1, pp. 373–376, Aug. 1999.

[12] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*. New York: Wiley, 1988, p. 325.

[13] M. Pinedo, *Scheduling—Theory, Algorithm and Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1995, p. 128.

[14] Sabuncuoglu and B. Gurgun, "A neural network model for scheduling problems," *Eur. J. Operational Res.*, vol. 9, no. 2, pp. 288–299, Sept. 1996.

[15] K. Satake, K. Morikawa, and N. Nakamura, "Neural network approach for minimizing the makespan of the general job-shop," *Int. J. Production Economics*, vol. 33, no. 1, pp. 67–74, Jan. 1994.

[16] E. Wacholder, J. Han, and R. C. Mann, "A neural network algorithm for the multiple traveling salesmen problem," *Biol. Cybern.*, vol. 61, pp. 11–19, 1989.

[17] J. Wang, P. B. Luh, X. Zhao, and J. Wang, "An optimization-based algorithm for job shop scheduling," *SADHANA*, pt. 2, vol. 22, pp. 241–256, Apr. 1997.

[18] T. M. Willems and L. E. M. W. Brandts, "Implementing heuristics as an optimization criterion in neural networks for job-shop scheduling," *J. Intell. Manufacturing*, vol. 6, no. 6, pp. 377–387, Dec. 1995.

[19] V. Wilson and G. S. Pawley, "On the stability of the traveling salesman problem algorithm of hopfield and tank," *Biol. Cybern.*, vol. 58, pp. 63–70, 1988.

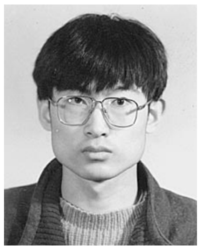
[20] D. N. Zhou, V. Cherkassky, T. R. Baldwin, and D. E. Olson, "A neural network approach to job-shop scheduling," *IEEE Trans. Neural Networks*, vol. 2, pp. 175–179, Jan. 1991.

[21] X. Zhao, P. B. Luh, and J. Wang, "Surrogate gradient algorithm for Lagrangian relaxation," *J. Optimization Theory and Applications*, vol. 100, no. 3, pp. 699–712, Mar. 1999.

Peter B. Luh (S'77–M'80–SM'91–F'95) received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, R.O.C., in 1973, the M.S. degree in aeronautics and astronautics engineering from the Massachusetts Institute of Technology, Cambridge, in 1977, and the Ph.D. degree in applied mathematics from Harvard University, Cambridge, MA, in 1980.

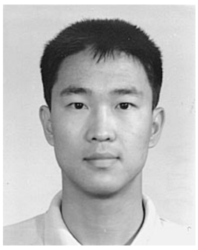
Since 1980, he has been with the University of Connecticut, Storrs. Currently, he is the Director of the Booth Center for Computer Applications and Research and also a Distinguished Engineering Professor in the Department of Electrical and Systems Engineering. He is interested in planning, scheduling, and coordination of design and manufacturing activities, and has developed near-optimal and computationally efficient methods to improve on-time delivery of products and reduce work-in-progress inventory. He is also interested in planning, scheduling, and coordination of design and manufacturing activities, and has developed near-optimal and computationally efficient unit commitment, hydro-thermal coordination, transaction/bidding, and load/price forecasting methods to minimize total cost or maximize the profit. He owns one U.S. patent.

Dr. Luh received three Best Paper Awards. He is a member of the Connecticut Academy of Science and Engineering. He is currently the Editor-in-Chief of IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION. He was also an editor of *IEEE Robotics and Automation Magazine* (1996–1998), and an Associate Editor for IEEE TRANSACTIONS OF AUTOMATION CONTROL (1989–1991), as well as an Associate Editor of *IIE Transactions on Design and Manufacturing, Discrete Event Dynamic Systems,* and *International Journal of Intelligent Control and Systems*.

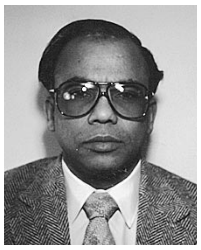


Xing Zhao received the B.S. degree in electrical engineering from Harbin Insitute of Technology, Harbin, China, in 1993, the M.S. degree in electrical engineering from Tsinghua University, Beijing, China, in 1996, and the Ph.D. degree in electrical engineering from the University of Connecticut, Storrs, in 1999.

Currently, he works at i2 Technologies, Cambridge, MA.



Yajun Wang received the M.S. degree in electrical engineering from the University of Connecticut, Storrs, in 1998. Currently, he is working toward the Ph.D. degree in the Department of Electrical and Systems Engineering, University of Connecticut, Storrs.



Lakshman S. Thakur received the B.Sc. degree in mathematics and physics from Bombay University, Bombay, India, in 1963, and the Ph.D. degree in operations research and industrial engineering from Columbia University, New York, in 1971.

He is an Associate Professor of Operations and Information Management at the University of Connecticut, Storrs. He has over 25 years of experience in teaching, consulting and research on optimization under resource constraints with numerous refereed publications. He has been Consultant to IBM on manpower planning with risk assessment and product warranty systems, as well as a Senior Consultant and Director of Management Science in a consulting organization. He was a Visiting Professor of Operations Research at the Yale School of Management, Yale University, New Haven, CT, from 1985 to 1987. He has been published in *Management Science, Mathematics of Operations Research, SIAM Journal on Applied Mathematics, Siam Journal on Optimization and Control, Journal of Mathematical Analysis and Applications, Naval Research Logistics*, IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, and other journals. His primary research interests are in the development and applications of linear, nonlinear, and integer optimization methods in management science and spline function approximation in mathematics. His current research focuses on production scheduling, product design, and facility location problems. His research on scheduling (with Dr. Peter B. Luh) is supported by National Science Foundation Grants.

Dr. Thakur is an Associate Editor of the *Naval Research Logistics Journal* and a full member of Informs (Operations Research Society of America and The Institute of Management Sciences) and the Mathematical Programming Society.