Time-table scheduling using Neural Network Algorithms

T.L. Yu
Dept. of Computer Science
California State University at San Bernardino, CA 92407
ptongyu@calstate

ABSTRACT

This paper shows a useful application of neural networks -- we demonstrate how to use neural network algorithms to schedule classes in an education institute. Such a scheduling problem is basically a graph coloring or graph partitioning problem which belongs to the large class of NP (non-deterministic polynomial time) complete problems and are difficult to solve. Fox and Furmanski [1] proposed some neural network algorithms to decompose loosely synchronous problems onto parallel machines. We adopt these algorithms to schedule time tables. The algorithms can be implemented in a digital computer or in analog neural networks.

1. INTRODUCTION

Hopfield and Tank introduced the use of neural networks to find solutions to optimization problems such as A/D conversion and traveling salesman problem[2]. The method has been generalized by Fox and Furmanski to decompose loosely synchronous problems onto parallel machines; they formulate the decomposition with use of graph partition theory. Fox and Furmanski notice that the alogrithm has a spin interpretation of a magnetic system; the ferromagnetic interaction minimizes the communication and the long range paramagnetic force balances the load. The method can be implemented in various ways : sequential, concurrent on a parallel machine, analog on the neural network with adaptive wieghts.

Class scheduling is always an important problem for an education institute. The time scheduler has to consider the constraints of class rooms, teaching utilities, conflicts in courses and making both the faculties and students happy. The relation between the constraints and the time-slots can be represented by an edge-weighted graph which is very similar to the graph arised from the decomposition problem. We can therefore adopt the decompostion algorithms to schedule time-tables. Section 2 presents the relation between general graph-partitioning problems and neural network algorithms. Section 3 presents the complete algorithm to find optimal solutions. Section 4 discusses how this is applied to time-table scheduling.

2. Graph-Partitioning and Cost Function

We first consider the graph-partitioning formalism. Here, we follow the notations used in reference 1; we specify a graph G by : a) P vertices, labeled by p = 0, 1, .., P-1, each with a weigh w(p); b) P(P-1) communication links (edges) , each with weight C(p,p'). In reality, a

vertex may represent a job or a program to be processed with time-complexity w(p). The weighted-edge C(p,p') may represent the traffic load between vertices p' and p (e.g. the number of data records to be passed from p' to p prior to the processing of subproblem p.)

We want to partition G into N subgraphs G_n, i.e. $G = G_0 U G_1 U \cdots\cdots U G_{N-1}$, where $N = 2^d$ may be interpreted as the number of nodes (labeled by n = 0, 1,....,N-1) of a d-dimensional hypercube. (In this paper, to avoid confusion we use the nomenclature "vertices" when refer to a graph and "nodes" when refer to a hypercube.) The partition is optimal in the following sense :

i) load is balanced, i.e.,

$$W_n = \sum_{p \text{ in } G} w(p) = \text{constant (idependent of n)}; \qquad (1)$$

ii) communication time is minimized i.e.,

$$C = 1/2 \sum_{p \neq p'} C(p,p')t[n(p),n(p')] = \text{minimum} \qquad (2)$$

where n(p) is the node processing the job p and t[n,n'] is the time to travel from node n to node n'.

If the P-point graph G is to be decomposed onto an N-node hypercube H, we can use the spin variable $s_i(p)$ to specify a pair (p,i), where

$$(p,i) = (p\text{-th point in G, } i\text{-th bit in H}); \; p = 0, 1, \ldots, P-1; \; i = 0, 1, \ldots, d-1; \; d = \log_2 N \qquad (3)$$

$S_i(p)$ takes on values -1 and +1 corresponding to the 0 and 1 values of the i-th bit of the node in H. $S_i(p,t)$ represents the value of $S_i(p)$ at time step t. The time t[n,n'] to communicate between two hypercube nodes n, n' can be represented by the Hamming distance between the two nodes which in trun can be expressed in terms of S_i . One can then construct a cost function E(S) associated with the problem. Optimally partitioning the graph G is equivalent to optimizing the following cost function E(S) given by[1],

$$E(S) = (\beta/2) \sum_{p,p'} C(p,p') \sum_{i=0}^{d-1} \{1 - S_i(p)S_i(p')\} + (b/2N^2) \sum_{p,p'} w(p)w(p') \prod_{i=0}^{d-1} \{1 + S_i(p)S_i(p')\} \qquad (4)$$

where ß and b are numerical network parameters to be tuned. We assume that E(t) is the value of E(S) at time step t. The terms on the right side of (4) has a spin interpretation of a magnetic system; the first term can be interpreted as the ferromagnetic interaction minimizing the communication cost; the second term may be interpreted as a long range paramagnetic force balancing the load.

Several algorithms have been proposed in reference 1 to minimize the E(s) shown in (4). A quick simple way to do this is to update the system in a series of "sweeps" : flipping a spin S_i -> $-S_i$ will cause a change in energy,

$$\Delta_i E = E(.....-S_i.....) - E(.....S_i.....) \tag{5}$$

The spin-flip is accepted if $\Delta_i E < 0$ otherwise the it is rejected. That is, $S_i(t+1) = \text{sign}(\Delta_i E(t) S_i(t))$ and we have $E(t+1) \leq E(t)$. Thus E(S) will evolve to a minimum as we continue to make the sweeping,

A simple way to obtain an expression for $\Delta_i E$ is to temporarily treat S_i as a continuous variable. Then $\Delta_i E$ can be evaluated by

$$\Delta_i E = \left(\frac{\partial E}{\partial S_i}\right) \Delta S_i \tag{6}$$

where $\Delta S_i = -S_i - S_i = -2S_i$. By differentiating (4) with respect to S_i one can obtain

$$\Delta_i E = \beta S_i(p) \sum_{p' \neq p} (C(p',p) + C(p,p')) S_i(p') - (2b/N^2) S_i(p) \sum_{p' \neq p} w(p) w(p') S_i(p') \prod_{j \neq i} \{1 + S_j(p) S_j(p')\} \tag{7}$$

If we assume C(p',p) = C(p,p') and absorb the 2 in the constants, we can rewrite (7) as

$$\Delta_i E = \beta S_i(p) \sum_{p' \neq p} C(p',p) S_i(p') - (b/N) S_i(p) \sum_{p' \neq p} L_i(p',p) S_i(p') \tag{8}$$

where

$$L_i(p',p) = w(p) w(p') (1/N) \prod_{j \neq i}^{d-1} \{1 + S_j(p) S_j(p')\} \tag{9}$$

In (8), the communication part C is pre-determined by the graph G and is time independent; the load balancing part L may be regarded as a learning term; it "learns during the evolution"; it is responsible to distribute the jobs uniformly among the nodes. In the class-scheduling problem, this term spreads the classess uniformly in the time-slots.

3. Finding the Optimal solution

As mentioned in the last section, we can find an optimal solution by flipping spins S_i -> $-S_i$ and accept the new configuration if the flipping leads to a decrease in energy E(s); eventually, E(s) will evolve to a minimum. However, in most cases, this method will make the system trapped in a local minimum. In order to escape from the local minima, one has to introduce a "noise term" when updating the spins. The noise term can be analog or digital. If the problem is to be simulated with a digital

computer, it is better to consider a digital noise term and we will concentrate our discussion on this method. A popular method to do this is to employ the simulated annealing[3]. In this process, a Boltzmann distribution at "temperature" T is obtained by asynchronously updating the system in a series of "sweeps": flipping a spin S_i -> $-S_i$ will cause a change in energy,

$$\Delta_i E = E(.....-S_i.....) - E(.....S_i.....) \qquad (10)$$

Instead of rejecting the new configuration whenever $\Delta_i E > 0$, we accept it with a certain probability which gives rise to the Boltzmann distribution : the spin-flip is accepted with probability 1 if $\Delta_i E < 0$ and with probability $e^{-\Delta_i E/T}$ if $\Delta_i E > 0$. By slowly decreasing T, one can obtain the optimal spin configuration. When $\Delta_i E/T >> 1$, the probability of accepting a new configuration with $\Delta_i E > 0$ tends to zero; the system is "frozen" into an optimal configuration.

A relatively efficient way of updating the spin configurations is to update the spins "bit-by-bit". A hypercube of dimension 1 is first considered and the 0-th bit is "balanced" to obtain the optimized $\{S_0(p)\}$; a hypercube of dimension 2 is then considered and the already-optimized $\{S_0(p)\}$ is used to balance the first bit to obtain the optimized $\{S_1(p)\}$ and so on; when we are balancing the i-th bit, we always treat the hypercube as if it were of dimension i+1, that is, we ignore all the $S_k(p)$ with $k > i$. The complete optimization algorithm with use of simulated annealing may be expressed as follow :

1) generate random spin configurations $\{S_i(p)\}$ for i = 0, 1, ... d-1,
2) set i = 0, N = 2,
3) set "temperature" T to a large value,
4) start with p = 0,
5) generate a random number r between 0 and 1,if r is larger than a predefined constant, say, 0.8, proceed to step 7),
6) evalutate $\Delta_i E = ßS_i(p)\sum_{p'\neq p} C(p',p)S_i(p') - (b/N)S_i(p)\sum_{p'\neq p} L_i(p',p)S_i(p')$

   where

   $L_i(p',p) = w(p)w(p')(1/N)\prod_{j<i}\{1+S_j(p)S_j(p')\}$, with $L_0(p'p) = 1$,

   if $\Delta_i E < 0$, update $S_i(p)$ to $-S_i(p)$

   if $\Delta_i E > 0$, generate a real random number μ between 0 and 1;
   if $\mu < \exp(-\Delta_i E/T)$ update $S_i(p)$ to $-S_i(p)$
   otherwise proceed to step 7
7) increment p and repeat steps 5) to 6) until p = P - 1,
8) iterate steps 4) to 7) for a few times,
9) decrease T and repeat steps 4) to 8) until T -> 0,
10) increment i; multiply N by 2; repeat steps 3) to 9) until i = d -1.

To obtain the optimal solution in a relatively short time, one has to

decrease T as fast as possible. However, if T is decreased too rapidly, the process may be trapped in a local minimum and the solution obtained may be poor. This is like the case that quenching will occur when a crystal is formed by cooling the liquid too rapidly; the cystal thus formed is imperfect. In order to avoid quenching in our optimization process but maintain a relatively efficient cooling schedule we can first rapidly decrease T until it is very near the "freezing point", and then decrease T slowly until the "thermal fluctuation" almost disappears; we can then decrease T very fastly again to obtain the final optimal solution[4]. The freezing point can be estimated from the specific heat C(T); when a crystal freezes, large amount of energy is released with very little change in temperature. Thus C(T) is exceptionally large near the freezing point. The specific heat C(T) can be estimated by standard iterative formulas[5]. From the C(T) curves, one can deduce the freezing points for small size problems and obtain the freezing temperatures for other sizes by finite size scaling.

4. Time Table Scheduleing and Results

Let us first consider a very simple example to illustrate the use of the neural algorithms described above to schedule time tables. Consider the case that we want to schedule 64 classes into two time-slots; a class has conflicts with about four other classes and the conflicts are quite unifromly distributed among all the classes. Then we may approximate the constraints by a graph which is in the form of a square lattice as shown in Fig. 1. In Fig. 1, a vertex represents a class and the links represent the constraints. In this representation, if there is a link between two vertices, there is conflict between the vertices and they should not be assigned to the same time-slot; if there is no link, there is no conflict and the two vertices can be assigned to the same time-slot. Thus theoretically, we can set $C(p,p') = \infty$ if p and p' are nearest neighbors, and set $C(p,p') = 0$ otherwise. Of course, in practice, we set C(p,p') to finite values. Fig. 2 shows the solution found using the neural algorithm presented in section 3; the paremeters that we have used in the simulation are as follow : $\beta = 0.6$, $b = 2.5$, $w(p) = 1$, $C(p,p') = 1$ if p and p' are nearest neighbors and = 0 otherwise. In Fig. 2, the 0 and 1 denotes the two nodes (which represents the time-slots) of the hypercube. It is quite obvious that the solution found (Fig. 2) is an exact optimal solution. We tried simulations with other graph configurations and in general good solutions can be found with a PC/AT- compatible in a few minutes for a graph size of the order 100 vertices.

In general, the values of C(p,p') s are predefined and depend on the problem that we consider. The more serious the conflict between p and p', the larger the value of C(p,p') is; if p and p' do not have any conflict, we set C(p,p') to zero. The classes are represnted by the vertices of the graph and the time-slots are reprented by the nodes of the hypercube. For example, if we want to schedule the classes into 16 time-slots, we have to consider a 4-dimensional hypercube. If the number of time-slots is not an exact power of 2, the case becomes more complicated and we have to consider more than one hypercube. To illustrate how we handle this, let us consider the case that the number of time-slots is 10. We first express 10 as the

sum of powers of 2, i.e. $10 = 2^3 + 2^1$. We have to decompose the graph into two hypercubes, one of which is of 3-dimension (say H_3) and the other is of 1-dimension (say H_1). We may assume that the Hamming distance between a node of H_1 and a node of H_3 to be 4. The spin variable $S_i(p)$ discussed in section 2 has to be modified so that it also specifies which hypercube it is in; we have to use the spin variable $S_{ij}(p)$ so that it specifies

$$(p,i,j) = (p\text{-th point in G, i-th bit in } H_j);\ p = 0, 1, \ldots, P-1;$$
$$i = 0, 1, \ldots, d_j-1;\quad d_j = \log_2 N_j;$$
$$j \text{ runs through the hypercubes used (} H_j \text{)} \qquad (11)$$

The generalization of other parts of the algorithm is straight forward. Finally, we speculate that under certain special conditions, one may incorporate the technique of renormalization group to speed up the simulated process[6].

REFERENCES

[1] G.C. Fox, and W. Furmanski, "Load Balancing Loosely Synchronous Problems with a Neural Network", Proceed of 3rd Conference on Hypercube Concurrent Comput and Appl, Vol. 1, Edited by G. Fox, p. 241-278 (1988).

[2] J.J. Hopfield and D.W. Tank, "Computing with Neural Circuits: A Model", Science **233**, 625-639 (1986).

[3] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing", Science **220**, p. 671-680 (1983).

[4] T.L. Yu and K.W. Yu, "Annealing Schedule of Monte Carlo Network for Load Balancing of Loosely Synchronous Problems", Proceedings of the Fourth Conf on Hypercube Con. Comput. and Appl. at Monterey, CA. (1989).

[5] K.W. Yu and T.L. Yu, "Solving the Traveling Salesman Problem in PC", 1989 Beijing International Symposium for Young Professionals (BISYCP' 89)

[6] B. Gidas, "A Renormalization Group Approach to Image Processing Problems", IEEE Trans. Pattern Anal. Machine Intel., **11**, 164-180(1989)

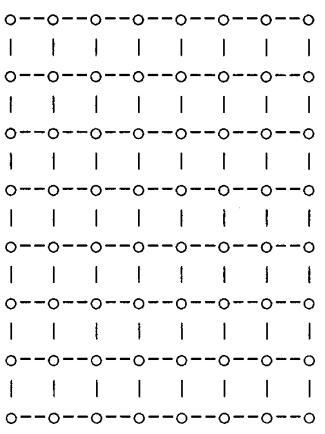

Fig. 1

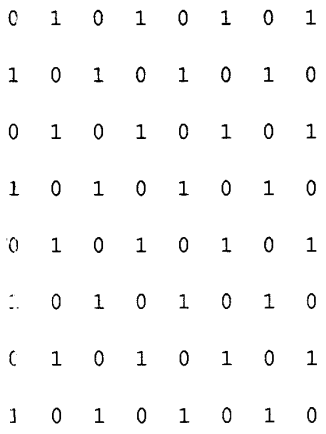

Fig. 2