A Mean Field Neural Network for Hierarchical Module Placement

M. Kemal Unaltuna and Vijay Pitchumani
Department of Electrical and Computer Engineering
Syracuse University, Syracuse, NY 13244

Abstract - **This paper proposes a mean field neural network for the two-dimensional module placement problem. An efficient coding scheme with only $O(N \log N)$ neurons is employed where N is the number of modules. The neurons are evolved in groups of N in $\log N$ iteration steps such that the circuit is recursively partitioned in alternating vertical and horizontal directions. In our simulations, the network was able to find optimal solutions to all test problems with up to 128 modules.**

1 Introduction

Since Hopfield and Tank published their seminal paper [6], Hopfield-type neural nets have been used for solving many combinatorial optimization problems. One major problem with such solutions is scaleability. With increasing problem size two things happen: first, the network becomes so big that simulation times are excessively long; and second, finding good parameters becomes increasingly hard that either the network converges to invalid solutions, or the quality of the solutions is poor [14], [2].

Two-dimensional module placement is an NP-hard combinatorial optimization problem which is very important in VLSI layout synthesis. In most of the previous attempts to solve this problem with Hopfield-type networks, researchers have concentrated on small-sized problems [8], [15], [7], [1], [4]. Unfortunately, most of the interesting placement problems in VLSI are very large, with more than 20,000 modules in some cases.

To overcome the difficulty with scaleability, we propose a neural network with $O(N \log N)$ neurons (instead of $O(N^2)$ neurons used in most of the previous applications) which solves the placement problem hierarchically, in a similar way to recursive min-cut bipartitioning methods. According to a recent survey [11], min-cut based algorithms are still very popular in solving large problems and the results obtained are second only to simulated annealing.

It has been independently shown [5], [9], [13] that the neuron update equations Hopfield used can be interpreted as one way of solving the mean field equations which arise from the *mean field approximation* to simulated annealing. We call our network a *mean field neural network*, because we use the straightforward update method of solving the mean field equations [9]. This kind of update is much faster than Hopfield's update, and generally produces better solutions.

2 Mean Field Neural Network for Hierarchical Placement

2.1 Problem Formulation and Mapping

We start with an $n \times m$ array of slots and $N = nm$ equal-sized modules. We also assume two-point connectivities between the modules, given by the $N \times N$ connectivity matrix $[c_{ij}]$. Thus, multi-pin nets have to be preprocessed and mapped to two-point connectivities. The objective is to assign the modules to slots such that the estimated wiring length is minimized and the resulting placement is routable.

In [5], Fox and Furmanski formulated the load balancing problem on hypercubes in terms of optimal partitioning of a computational graph into subgraphs, and solved it using a neural network. Here, we modify and expand their ideas to the problem at hand.

Let the two-tuple (r^p, s^p) represent the position of the slot to which module p is assigned. The row number r^p, and the column number s^p are $e = \log n$ and $d = \log m$ bit binary numbers, respectively, and are given by

$$r^p = \sum_{i=0}^{e-1} r_i^p 2^i$$

$$s^p = \sum_{i=0}^{d-1} s_i^p 2^i$$

Bits r_i^p and s_i^p are mapped to neural variables y_i^p and x_i^p such that $y_i^p = 2r_i^p - 1$ and $x_i^p = 2s_i^p - 1$. Thus, y_i^p (x_i^p) represents the i^{th} bit of the row number (column number) of p's slot. All neural variables can have continuous values between ± 1. The neural variables will be evolved in groups of N in $\log N$ steps. First, $x_{d-1}^0, x_{d-1}^1, \ldots, x_{d-1}^{N-1}$ representing the highest bits of the column numbers of the modules are evolved. After this step, the circuit is effectively bipartitioned in the vertical direction. Next, $y_{e-1}^0, y_{e-1}^1, \ldots, y_{e-1}^{N-1}$ representing the highest bits of the row numbers of the modules are evolved. This means bipartitioning the circuit in the horizontal direction. After this, we return to the vertical direction and evolve variables $x_{d-2}^0, x_{d-2}^1, \ldots, x_{d-2}^{N-1}$. This way, the recursive bipartitioning of the circuit continues in alternating vertical and horizontal directions until all neural variables are evolved.

2.2 Bipartitioning Neural Network

For the first vertical bipartitioning step we will use the following energy function:

$$E_{x_{d-1}} = -\frac{A}{2} \sum_p \sum_{q \neq p} c_{pq} x_{d-1}^p x_{d-1}^q + \frac{B}{2} \sum_p \sum_{q \neq p} x_{d-1}^p x_{d-1}^q \qquad (1)$$

The first term has its minimum when the sum of the connections between modules in separate partitions is minimized, and the second term is minimized when the partitions are balanced. The mean field equations can be derived from (1) as

$$x_{d-1}^p = \tanh\left(\frac{u_{d-1}^p}{T}\right) \qquad (2)$$

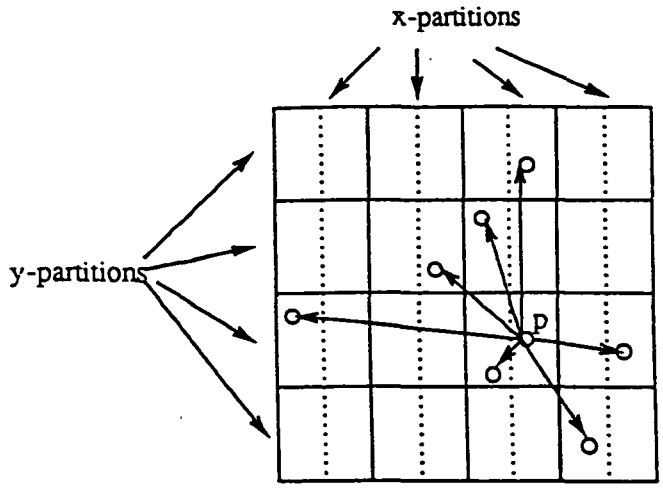

Figure 1: The situation after two vertical and two horizontal partitioning steps. The circuit is currently bipartitioned for the third time in the vertical direction. The arrows indicate attractive forces acting on module p.

$$\begin{aligned} u^p_{d-1} &= -\frac{\partial E_{x_{d-1}}}{\partial x^p_{d-1}} \\ &= A\sum_{q\neq p} c_{pq} x^q_{d-1} - B\sum_{q\neq p} x^q_{d-1} \end{aligned} \tag{3}$$

The first term in equation (3) is an attractive force where each module q connected to p tries to bring p to its side of the partition, and the second term tries to keep the partitions balanced.

2.3 Placement by Recursive Bipartitioning

In order to extend equations (2) and (3) to subsequent horizontal and vertical bipartitioning steps, it is not sufficient to consider only internal connections. We have to consider connections to modules in other partitions (at a higher level) as well.

Figure 1 shows the situation after two vertical and two horizontal bipartitioning steps when the resulting partitions are being partitioned for the third time in the vertical direction. At this stage, the modules can be grouped in the following way according to their locations relative to module p, *prior* to the current partitioning process:

1. Modules in the same x- and y-partition as p,
2. Modules in the same x-partition as p but not in the same y-partition,
3. Modules not in the same x-partition as p.

The balance force in the update equation for x^p_{d-3} should cover modules in the first group only. For the attractive forces, we need to consider two cases:

- Attractive forces from modules in the third group. Such a module q is in an x-partition either to the left or to the right of p and should bias p accordingly. However, the location of q with respect to the current bipartitioning process is irrelevant.
- Attractive forces from modules in the first or second group. Since such a module q is in the same x-partition as p, its location with respect to the current bipartitioning process should influence x^p_{d-3}.

To incorporate these different forces in the update equations, we need to make some definitions. We define X^i_{pq} to be one if modules p and q are in the same x-partition after $d-i-1$ vertical partitioning steps, and zero otherwise. In terms of neural variables this can be expressed as

$$X^i_{pq} = \frac{1}{2^{d-i-1}} \left(\prod_{k=i+1}^{d-1} (1 + x^p_k x^q_k) \right) \tag{4}$$

Similarly, Y^j_{pq} is defined to be one if modules p and q are in the same y-partition after $e-j-1$ horizontal partitioning steps, and zero otherwise:

$$Y^j_{pq} = \frac{1}{2^{e-j-1}} \left(\prod_{l=j+1}^{e-1} (1 + y^p_l y^q_l) \right) \tag{5}$$

Now consider the generalized situation where we have evolved neural variables x^p_k and y^p_l, for all $k > i$, $l > j$, and for all p. This means that the problem has been partitioned $d-i-1$ times in the vertical direction and $e-j-1$ times in the horizontal direction. Suppose we are evolving variables x^p_i, i.e. we are at the $(d-i)^{th}$ vertical partitioning step. With the help of the above definitions for X^i_{pq}, and Y^j_{pq}, we propose the following update equations for x^p_i:

$$x^p_i = \tanh\left(\frac{u^p_i}{T}\right) \tag{6}$$

$$u^p_i = A \sum_{q \neq p} X^i_{pq} c_{pq} x^q_i - B \sum_{q \neq p} X^i_{pq} Y^j_{pq} x^q_i + C \sum_{q \neq p} c_{pq} sign(Q^x_i - P^x_i) - D x^p_i \tag{7}$$

where

$$Q^x_i = \sum_{j=i+1}^{d-1} x^q_j 2^j$$

$$P^x_i = \sum_{j=i+1}^{d-1} x^p_j 2^j$$

$$sign(x) = \begin{cases} 1, & x > 0; \\ -1, & x < 0; \\ 0, & \text{otherwise} \end{cases}$$

and A, B, C, D are constants.

The first term in equation (7) represents the attractive force from modules q with $X^i_{pq} = 1$, i.e., from modules in the same x-partition as p. The second term is the balance force. This

term covers modules q for which $X^i_{pq}Y^j_{pq} = 1$, i.e., modules in the same x- and y-partition as p.

The third term represents the attractive force from modules outside the x-partition of p. Here, Q^x_i and P^x_i uniquely code the x-partitions of q and p, respectively. If module q is in an x-partition to the left (right) of p, then $sign(Q^x_i - P^x_i)$ will be -1 (+1) and the force exerted on p by q will be proportional to $-c_{pq}$ ($+c_{pq}$) which has the effect of bringing p closer to q. Note also that the definition of the *sign* function above ensures that the effect of modules in the same x-partition as p will be zero.

Finally, the fourth term in equation (7) is a computationally inexpensive way to add random noise to the neuron input u^p_i, suggested by Fox and Furmanski in [5]. Such a term with the *wrong* sign tries to flip the neuron currently being updated. The effect is negligible if the neuron output has already converged to its final value, whereas it helps the network to climb out of local minima in the early stages.

2.4 Corresponding Energy Functions

Since update equation (7) was not developed from an energy function, it is not clear if the network possesses the convergence and energy minimizing properties of Hopfield-type networks. However, a corresponding energy function can be derived from equations (6) and (7).

Let T^i_{pq} denote the connection weight between neurons (i,p) and (i,q), and I^i_p denote the bias term for neuron (i,p). The update equation for a neuron input in a mean field neural net in terms of T^i_{pq} and I^i_p is

$$u^p_i = \sum_{q \neq p} T^i_{pq} x^q_i + I^i_p, \tag{8}$$

and the corresponding energy function for a symmetric weight matrix is given by

$$E_{x_i} = -\frac{1}{2} \sum_p \sum_{q \neq p} T^i_{pq} x^p_i x^q_i - \sum_p I^i_p x^p_i, \tag{9}$$

By equating equations (7) and (8) (disregarding the noise term) we get for our network

$$T^i_{pq} = (1 - \delta_{pq}) X^i_{pq} (Ac_{pq} - BY^j_{pq}) \tag{10}$$

$$I^i_p = C \sum_{q \neq p} c_{pq} sign(Q^x_i - P^x_i) \tag{11}$$

Since $[T^i_{pq}]$ is symmetric, an energy function corresponding to equations (6) and (7) is

$$\begin{aligned} E_{x_i} = & -\frac{A}{2} \sum_p \sum_{q \neq p} X^i_{pq} c_{pq} x^p_i x^q_i + \frac{B}{2} \sum_p \sum_{q \neq p} X^i_{pq} Y^j_{pq} x^p_i x^q_i \\ & -C \sum_p \sum_{q \neq p} c_{pq} sign(Q^x_i - P^x_i) x^p_i \end{aligned} \tag{12}$$

Note that energy function (12) is very similar to the bipartitioning energy function (1), except for the last term which is the sum of the individual bias terms (11). Actually, E_{x_i}

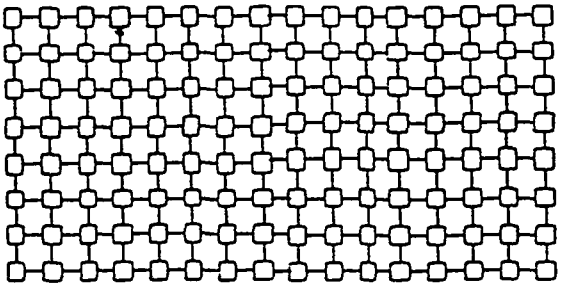

Figure 2: The 128-module example solved

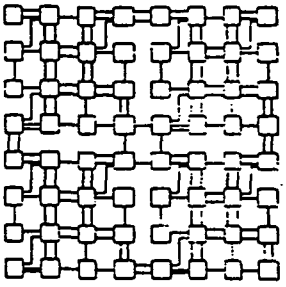

Figure 3: This 64-module test problem is the largest one in the literature, to our knowledge, solved by a Hopfield type neural network. The neural network of Sriram and Kang was able to find a solution with a cost of 182, whereas the optimal solution shown has a cost of 168. This solution was easily found by our network.

consists of 2^{d-i-1} individual (and independent) energy functions, one for each x-partition. If we limit p and q to a specific x-partition, say XP_0, then the energy function for XP_0 is given by

$$E_{x_i}^{XP_0} = -\frac{A}{2}\sum_{p\in XP_0}\sum_{\substack{q\neq p\\ q\in XP_0}} c_{pq}x_i^p x_i^q + \frac{B}{2}\sum_{p\in XP_0}\sum_{\substack{q\neq p\\ q\in XP_0}} Y_{pq}^j x_i^p x_i^q$$
$$-C\sum_{p\in XP_0}\sum_{q\neq p} c_{pq}sign(Q_i^x - P_i^x)x_i^p \qquad (13)$$

Our simulations show that the network indeed converges to local minima of the energy functions given in (13).

3 Simulation Results

The above described neural network algorithm was implemented in C on SUN4 SPARCstations and was tested on several hand-constructed examples with known optimal solutions

with up to 128 modules. In all cases, optimal solutions were found. Figure 2 shows the 128-module test problem.

We also tested the algorithm on the biggest example in the literature to our knowledge, solved with a Hopfield type neural network. This 64-module example, shown in Figure 3, is taken from [12], where the optimal solution couldn't be found. Our hierarchical algorithm needed $\log 64 = 6$ recursive partitioning steps to solve this problem. The number of iterations it took to converge at each step are: 12, 12, 24, 11, 40, and 30, respectively, where in one iteration we update all participating neurons exactly once (asynchronously). The optimal solution shown in the figure was easily obtained. The whole process took less than 10 CPU seconds.

In the simulations, we used a straightforward implementation of equation (7). This is not a very efficient way of implementing the algorithm on serial computers. The simulation algorithm can be speeded up considerably if we consider the following:

- X_{pq}^{i}, Y_{pq}^{j}, and Q_{i}^{x} are independent of the states of the neurons at the current level. They can be calculated *before* the $(d-i)^{th}$ vertical partitioning step. Thus, they can be treated as constants in equation (7).

- The connection matrix $[c_{ij}]$ can be stored as an array of linked lists where the i^{th} list consists of modules connected to module i. Thus, the first and third sums in equation (7) cover only modules q connected to p, instead of all q. This will achieve substantial speedup since the connection matrix is usually very sparse.

- The sum for the balance force stays constant from one update to the next, except for the term involving the last updated neuron. Thus, only the *change* in the balance force due to the change in the output of this neuron has to be calculated from one update to the next, which can be done in constant time.

These speedup techniques were incorporated in the first vertical bipartitioning step and a more than 10-fold speedup was observed. They will be implemented fully in the future versions of the algorithm.

4 Discussion and Conclusion

In this paper, we proposed a $O(N \log N)$ mean field neural network for the module placement problem. The problem is solved in a divide-and-conquer fashion by recursive bipartitioning where at each bipartitioning step exactly N neurons are evolved. The neural algorithm is similar to min-cut methods, yet maintains a level of globality, since all participating neurons evolve *simultaneously.*

The performance of the algorithm in our simulations has been very encouraging and merits further investigation. With the speedup techniques discussed in the previous section implemented, we can expect that the algorithm will take less than one CPU second to solve the 64-module problem shown in Figure 3 on a SUN4 SPARCstation. Since the speeded up version of the algorithm will be very fast, we will tackle larger and more realistic placement problems, including benchmark circuits.

The algorithm can be extended to popular layout styles like standard cell and sea-of-gates by integrating the area constraints. For example, the balance term in the update equation (7) can be modified to

$$- B \sum_{q \neq p} X^i_{pq} Y^j_{pq} x^q_i area_q \tag{14}$$

such that the bisection process produces partitions of approximately equal area.

One very important aspect of the algorithm is finding good parameters, and a good simulation temperature for each bipartitioning step. It is well known that mean field nets posses a *critical temperature*, T_c, [10], [5], [13], [4], where the bulk of the optimization occurs. Estimating T_c, and annealing the network around it not only shortens the running times, but also improves the solutions. In our simulations so far, we used a single temperature and empirically determined parameters. Future research will include the estimation of the parameters, the critical temperature, and the use of annealing.

Acknowledgement

We would like to thank Dr. G. C. Fox and Dr. W. Furmanski for useful discussions.

References

[1] H. Date, M. Seki, and T. Hayashi, "LSI module placement methods using neural computation networks", *Proceedings of the IEEE International Joint Conference on Neural Networks*, Vol. III, pp. 831-836, Jun. 1990.

[2] Gerald W. Davis, Jr., "Sensitivity analysis in neural net solutions", *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. 19, No. 5, pp. 1078-1082, 1989.

[3] A. E. Dunlop and B. W. Kernighan, "A procedure for placement for standard cell VLSI circuits", *IEEE Transactions on Computer Aided Design CAD-4*, Vol. 1, pp. 92-98, 1985.

[4] L. Fang, W. H. Wilson, and T. Li, "Mean field annealing neural net for quadratic assignment", *International Neural Networks Conference, INNC 90 Paris,* pp. 282-286, 1990.

[5] G. C. Fox and W. Furmanski, "Load balancing loosely synchronous problems with a neural network", *Caltech Concurrent Computation Project,* Report No. C3P-363b, 1988.

[6] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems", *Biological Cybernetics,* Vol. 52, pp. 141-152, 1985.

[7] J. Naft, "NEUROPT: Neurocomputing for multiobjective design optimization for printed circuit board component placement", *Proceedings of the IEEE International Joint Conference on Neural Networks*, Vol. I, pp. 503-506, 1989.

[8] G. Persky, "Experiments in cell placement with a simulated network", *Proceedings of International Workshop on Placement and Routing, Research Triangle Park,* North Carolina, May 10-13, 1987.

[9] C. Peterson, J. Anderson, "Neural networks and NP-complete optimization problems", *Complex Systems*, Vol. 2, pp. 59-89, 1988.

[10] C. Peterson, B. Soederberg, "A new method for mapping optimization problems onto neural networks", *International Journal of Neural Systems,* Vol. 1, No. 1, pp. 3-22, 1989.

[11] K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques", *ACM Computing Surveys*, Vol. 23, No.2, 1991.

[12] M. Sriram and S. M. Kang, "A modified Hopfield network for the two-dimensional module placement", *IEEE International Symposium on Circuits and Systems*, pp. 1664-1667, 1990.

[13] D. E. Van den Bout and T. K. Miller, "Graph partitioning using annealed neural networks", *IEEE Transactions on Neural Networks*, Vol. 1, No. 2, 1990.

[14] G. V. Wilson and G. S. Pawley, "On the stability of the traveling salesman problem of Hopfield and Tank", *Biological Cybernetics,* Vol. 58, pp. 63-70, 1988.

[15] M. L. Yu, "A study of the applicability of Hopfield decision neural nets to VLSI CAD", *26th ACM/IEEE Design Automation Conference,* pp. 412-417, 1989.