A Neural Network Design for Circuit Partitioning

JIH-SHYR YIH AND PINAKI MAZUMDER, MEMBER, IEEE

Abstract—**This paper proposes a neural network model for circuit bipartitioning. The massive parallelism of neural nets has been successfully exploited to balance the partitions of a circuit and to reduce the external wiring between the partitions. The experimental results obtained by neural nets are found to be comparable with those achieved by the Fiduccia and Mattheyses algorithm. The proposed approach can be implemented in hardware to accelerate the time consuming partitioning procedures.**

I. Introduction

GIVEN a set of circuit elements and the net-lists showing the connectivity between these elements, the objective of circuit partitioning is to allocate these elements into two or more blocks (partitions) such that the number of external wires interconnecting these blocks is minimized. The minimization of external wires reduces the wiring cost and also improves the reliability of the hardware. The problem of finding the minimum number of external wires is *NP*-complete. A number of heuristic algorithms have been proposed in the literature to obtain near-minimum solutions to the partitioning problem. An early approach is due to Kodres [9] who developed clusters of well-connected circuit elements around a set of predetermined elements (called seeds). The algorithm is fast, but often results in very poor partitioning. A better approach is to construct an initial partition and then successively reduce the number of external wires by iteratively moving the elements from one block to another block, while keeping the partitions somewhat balanced [1], [3], [8], [10], [13]. This study investigates the possibility of utilizing the collective computational properties of neural networks to obtain fast heuristic solutions. Current research work in artificial neural net implementations has demonstrated the feasibility of building analog electronic neural nets having hundreds or thousands of neurons, with state transition times comparable to the primitive binary device switching delay [4], [12], [15]. By associating iterative rearrangements of components' positions with neuron state transitions, a significant gain in speed can be achieved over the conventional software algorithms running on sequential computers.

In the following, some related prior works are briefly reviewed. We start with the seminal work by Kernighan and Lin [8], which was originally introduced for partitioning communication networks represented as graphs. The average complexity of the algorithm is known to be $O(n^2 \log n)$. Starting with an arbitrary partition, in each iteration the algorithm selects a vertex from each block and exchanges the pair so that a maximum reduction in the sum of the weighted external edges is achieved. Once a vertex changes its side in the partition, it is not considered as a candidate for later selections. If there are $2n$ vertices in the graph to be partitioned, the size of the search space for pairwise exchange decreases from n^2 to $(n-1)^2$, then to $(n-2)^2$, and so on. Let g_i be the edge-cut reduction achieved by the ith pairwise exchange. The value of g_i may be positive, zero or negative as long as it is the maximum reduction achievable at that time. In the end, the algorithm finds out the first k pairwise exchanges such that $\Sigma_i g_i$ is maximized.

Later, Schweikert and Kernighan illustrated the deficiencies of using graph models for partitioning circuits [13]. They proposed the *net-cut* circuit model by representing the true connection relationships among components joined by the same signal line. Consider a net of four components shown in Fig. 1(a). If modeled as a graph with an edge created for every connected pair of components, it would become a fully connected graph as shown in Fig. 1(b).

In the Schweikert and Kernighan algorithm, the external lines are reduced based on the following criteria.

1) When all components of the same net are in the same block, moving any one of the components to the other block will create an additional external line.

2) If a component is the only one in the net remaining in a block, moving it to the other block will remove the net from the cut.

3) When two or more, but not all, of the components in the same net are within a block, moving any one of the components will have no effect on the number of external connections.

In Fig. 2, given an example net of three components, the movements of component a in cases 1), 2), and 3) show the necessity of the above three criteria.

Fiduccia and Mattheyses [3] proposed a partitioning algorithm with linear runtime complexity, where the problem size is defined in the number of pins in the circuits. Rather than pairwise exchanges, each time only a single component from either block is chosen for a position change while keeping the partition roughly in balance. As in the previously described two algorithms, to prevent components from thrashing between the blocks, a component is not allowed to change its position twice. The

Manuscript received June 20, 1989. This work was supported in part by the National Science Foundation, the Digital Equipment Corporation and by the U.S. Army. This paper was recommended by Associate Editor A. E. Dunlop.

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 9037116.

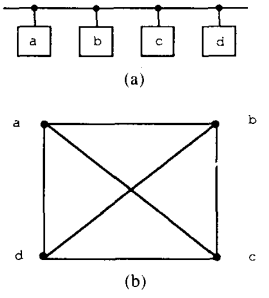


Fig. 1. Comparison between the net-cut and graph connectivity models. (a) Net-cut model. (b) Graph model.

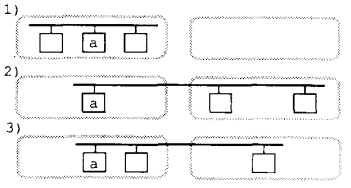


Fig. 2. Example cases of a net of three components in net-cut.

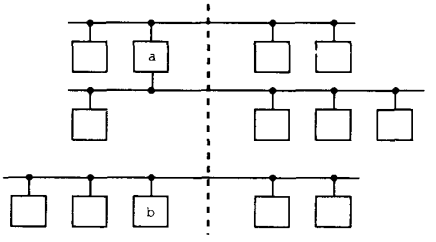


Fig. 3. Example for the demonstration of multilevel cell gain.

component selection criterion called *cell gain* is similar to the one identified by Schweikert and Kernighan. With each single component movement, component selection and cell gain information are updated by manipulating a smart bucket list data structure. Thus the runtime complexity is made proportional to the problem size.

A possible improvement on the linear algorithm was subsequently proposed by Krishnamurthy [10]. The generalization is based on the observation that a lot of ties are encountered when selecting a component with the largest cell gain. A multilevel lookahead for cell gain information is, therefore, suggested for tie-breaking. For example, consider the partition shown in Fig. 3. According to the single-level cell gain measure, it appears that moving either component a or b will have no effect on the external line reduction. However, if a is moved, two more nets will each have only one last component left in the block. On the contrary, moving b has none of this *second-level* cell gain.

The rest of the paper is organized as follows. Section II provides a brief overview of the neural network model and its computational properties. Section III illustrates the circuit partition representation in the form of neural network states. For both the graph and net-cut models, the criteria for selecting components for changing locations are described in terms of neuron connections and the corresponding strengths (weights) between neurons. Section IV described ways of balancing a partition for components of equal and unequal sizes. It also shows how the degree of evenness between partitions can be maintained by the neural networks.

II. Neural Net Model

The mathematical formalization of the human nervous system dates back to the work by McCulloch and Pitts in the early forties [11], where the nervous system is modeled as a set of *neurons* (computation elements) interconnected by *synapses* (weighted links). Each neuron in the model is potentially connected to other neurons, and is capable of receiving impulses from them, and can fire impulses as output to those neurons. This output function of a neuron depends on whether the total amount of input excitation received exceeds its predetermined threshold θ, or not. One convention is to describe the state of neuron i by a binary value s_i, with $s_i = 0$ being a nonfiring condition and $s_i = 1$ an impulse firing condition. Another way is to make s_i a bipolar state with $s_i = -1$ representing a negative-firing condition and $s_i = 1$ a positive-firing condition.

For neuron interactions, between two different neurons i and j, there is a synapse serving as a bidirectional communication channel. The signals passing through the synapse in both directions are considered independent. Moreover, the signal traveling from neuron i to neuron j is amplified by a weight factor w_{ji}, i.e., if the impulses fired by a neuron correspond to a unit influence, then the firing of neuron i produces w_{ji} amount of influence on neuron j.

Now, let s_i' denote neuron i's next state value. From the above description one can represent the neural net *state transition function* as follows:

$$s_i' = \begin{cases} 0(\text{or } -1), & \text{if } \sum_j w_{ij}s_j < \theta_i, \\ 1, & \text{if } \sum_j w_{ij}s_i > \theta_i, \\ s_i, & \text{otherwise.} \end{cases}$$

Actually in the human nervous system or any other artificial implementations, neural nets evolve in time and have smooth continuous transitions during state change. Neuron processors are expected to operate in an asynchronous and random fashion [5].

Fig. 4 shows a primitive schematic design of an analog electronic neural net, in which a neuron processor (amplifier) may be realized simply as a series connection of two inverters [4], and a synapse is just a resistor with resistance value inversely proportional to its synaptic weight. In this case, a zero weight factor is indicated by a missing resistor. Then, the state of a neuron is represented by the amplifier's uninverted binary output value, and the influence is simulated by electrical current. As we

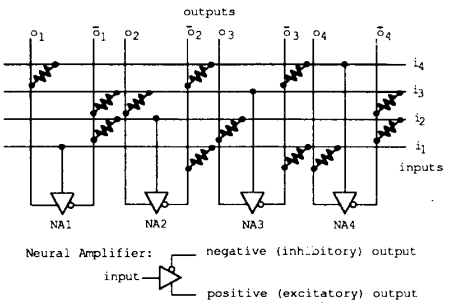


Fig. 4. Schematic of an analog electronic neural net.

can see, rather than actually having a dedicated link between two neurons, all input currents to a particular neuron are superimposed on a common bus connecting to the neuron's input.

Now, let us look at the overall network state transition behavior. Given a network state vector $X = (x_1, x_2, \cdots, x_N)$, where x_i is the state of neuron i, X is called a *fixed point* if and only if $x_i' = x_i$ for all i's. That is, of all the possible 2^N states of a neural net, only the fixed points are considered stable. If the current network state is not one of the fixed points, the neural net cannot maintain the present state. One immediate question about the network behavior could be on the state convergence: starting with an arbitrary state, will the network eventually reach a fixed point? To answer the question an energy function has been formulated [6]. To illustrate the energy convergence in a simple fashion, we will temporarily assume that neurons have binary behavior (nonfiring or firing), and that all neuron thresholds are zero in value. First, let us consider the case of a positive influence being fired to a neuron. According to the neuron's functional definition, this input will encourage the neuron to fire impulses. If the neuron is already in the firing state, this input can be looked upon as a negative energy, since by convention a system is more stable when the energy level is lower. Likewise, a negative influence sent to a neuron in the nonfiring state should also be considered as a negative energy. On the other hand, a positive energy is created if a firing neuron receives a negative influence or a nonfiring neuron receives a positive influence, since the network is potentially destabilized by the input.

Finally, the total energy E, also known as a Lyapunov function [5], is given by

$$E = -\frac{1}{2}\sum_i \sum_j s_i w_{ij} s_j.$$

Note that the change of state by neuron i will result in an absolute decrease of $\Sigma_j w_{ij} s_j$ in energy. This, taken together with the fact that the total energy is bounded, guarantees that the network will reach a local minimal energy fixed point [6].

III. External Wiring Minimization

In this section, we concentrate on partitioning circuits into two blocks. We will begin with circuits represented by the graph model, and then proceed to the net-cut model to demonstrate how neural networks can be applied to reduce the number of external lines. The issue of maintaining the balanced partition will be discussed in the next section.

3.1. Moving Cells Between Blocks

Let us consider the case in which every wire in a circuit is shared by exactly two different components, so that the connections can be modeled accurately by the graph model. The interconnection relation between n given components can then be represented by a matrix $T = \{t_{ij}\}$, where t_{ij} designates the number of wires between components i and j, $1 \le i, j \le n$.

Suppose that component i is in block X, $X \in \{A, B\}$. The number of associated internal lines, α_i, will be $\Sigma_{y \in X} t_{yi}$. Note that t_{ii} is zero for all i's to ignore connections between pins of the same component. Similarly, the number of associated external lines, β_i, will be $\Sigma_{y \in A} t_{yi}$ if $i \in B$, or $\Sigma_{y \in B} t_{yi}$ if $i \in A$.

It may be noted that if the component i is moved from block X to the other block, its associated external lines will turn to internal and its associated internal lines will become external. Thus the reduction of external lines due to movement of component i, denoted here as γ_i, will be the difference between the numbers of external and internal lines associated with component i, i.e., $\gamma_i = \beta_i - \alpha_i$.

It is interesting to note that the internal connections between two components in the same block can be mapped proportionally to a stabilizing negative energy in neural networks, and likewise the external connections between two components in different blocks can be mapped proportionally to a destabilizing positive energy in neural networks. This way, the neural network's convergence towards minimal network energy levels can be utilized for the purpose of circuit partitioning. To show how the mapping is done, let us first represent partitions as neural net states by mapping the position of each component to a neuron with bipolar states. We define

$$s_i = \begin{cases} -1, & \text{if the } i\text{th component is in block } A, \\ +1, & \text{if the } i\text{th component is in block } B. \end{cases}$$

If we choose to use the number of wires between two connected components as the degree of connectivity between them, then the weight of the corresponding synapse can be defined accordingly, i.e., $w_{ij} = G_1 \cdot t_{ij}$, where G_1 is some positive constant.

Now, if neuron i is in $+1$ state, then it will fire $G_1 \cdot t_{ji}$ amount of positive influence to neuron j, for all $j \neq i$, encouraging them to either stay in or change to $+1$ state. Meanwhile neuron i will be receiving some $\alpha_i G_1$ amount of positive influence to stay in $+1$ state, and some $\alpha_i G_1$ amount of negative influence to move to the other block. In other words, whether a component is to change side or not, is determined exactly by the neuron's next state transition function given in Section II. The previous as-

sumption that a neuron receiving a larger opposing influence in aggregation changes its state faster, will correspond naturally to the component selection for movement in each iteration. The ties are resolved in a random fashion.

3.2. The Net-Cut Circuit Model

Now let us consider the signal net shown in Fig. 1. Suppose that the signal net is now divided as shown in Fig. 5(a), with component a assigned to one block components b, c, and d to the other block. As we can see, moving component a will reduce the number of external lines by one. However, if the signal net is modeled by a graph as shown in Fig. 5(b), it appears that three external lines can be removed instead of one. Confusions of this sort do occur due to the inaccurate modeling of associated internal and external lines of component a.

Following Schweikert and Kernighan's net-cut model [13], γ_i will be calculated in the following fashion. Let $\mathfrak{N}$ be the set of nets that component i belongs to, and n is a net in $\mathfrak{N}$. We define, for the net n:

$$\alpha_i^n = \begin{cases} 0, & \text{if component } i \text{ of } n \text{ is alone in the block,} \\ 1, & \text{otherwise.} \end{cases}$$

$$\beta_i^n = \begin{cases} 0, & \text{if all components of } n \text{ are in the same block,} \\ 1, & \text{otherwise.} \end{cases}$$

Finally we have,

$$\gamma_i = \sum_{n \in \mathfrak{N}} (\beta_i^n - \alpha_i^n).$$

Since the underlying external line reduction calculation method has been drastically changed, we propose a new neural net design to represent the net-cut model. The new design must be able to avoid duplicated positive or negative influences being fired to a component neuron from all other component neurons in the net.

To achieve this goal, we propose the use of additional neurons for processing cell gain information. The idea is to prepare no synaptic interconnections between pairs of the component neurons of the same net unless there are only two components in the net. The influences that encourage a component neuron to keep or change its state will now be provided by some other additional noncomponent neurons.

Before we set out to describe the new neural net design, the functional behavior of a neuron is enhanced for the net-cut model. First, a reference value r_i is suggested as an input to neuron i, which is used to compare with the total received feedback influence to determine the next state. Basically, when $r_i = 0$, neuron i behaves just like a simple thresholding processor with threshold being zero. However, when a nonzero reference value is applied, the neuron must receive a larger total feedback in absolute value than the magnitude of the reference to be in a nonzero state. Moreover, a neuron is supposed to be in $+1$ (-1) state if a large enough positive (negative) total influence in magnitude is received. However, the correspondence is reversed when r_i is negative, i.e., a large enough positive (negative) total influence in magnitude will set the neuron in -1 $(+1)$ state. The above description can be summarized by diagrams shown in Fig. 6. The exact state transition function is formalized as follows:

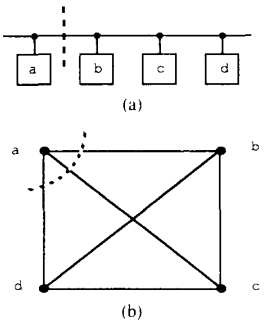


Fig. 5. Min-cut partitions of the two models shown in Fig. 1.

$$s_i' = \begin{cases} s_i, & \text{if } \Sigma w_{ij}s_j = 0 \text{ and } r_i = 0, \\ -1, & \text{if } (\Sigma w_{ij}s_j < 0 \text{ and } r_i = 0) \\ & \text{or } (-\Sigma w_{ij}s_j > |r_i| \text{ and } r_i > 0) \\ & \text{or } (\Sigma w_{ij}s_j > |r_i| \text{ and } r_i < 0), \\ +1, & \text{if } (\Sigma w_{ij}s_j > 0 \text{ and } r_i = 0) \\ & \text{or } (\Sigma w_{ij}s_j > |r_i| \text{ and } r_i > 0) \\ & \text{or } (-\Sigma w_{ij}s_j > |r_i| \text{ and } r_i < 0), \\ 0, & \text{if } |\Sigma w_{ij}s_j| < |r_i| \text{ and } r_i \neq 0. \end{cases}$$

Now, consider a signal net of k components, $k > 2$. To take care of the three observations made by Schweikert and Kernighan in the net-cut model, a neuron x is added so that every component neuron in the net is connected to x by synaptic weight G_1. Neuron x's reference input is set to the value $(k-3)G_1$. This way, if all components are in block X, each of the component neurons will receive $-G_1(G_1)$ amount of influence from neuron x to stay in the block if $X = A(B)$. Also, when both blocks have two or more components, neuron x will not fire any influence to the component neurons. As for the case where there is exactly one component in, say block A, and all others in block B, although the lone component neuron will receive G_1 amount of influence from neuron x to change state as expected, the rest of the components will each receive an extra G_1 amount of influence from x. Thus the second observation is invalidated. To remedy this problem, k additional neurons, $y_1, y_2, \cdots, y_k$, are added so that y_i

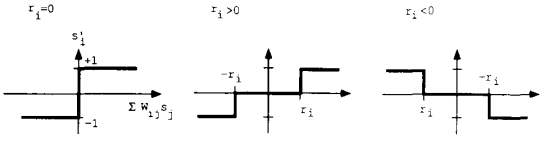


Fig. 6. The next state transition function for neurons.

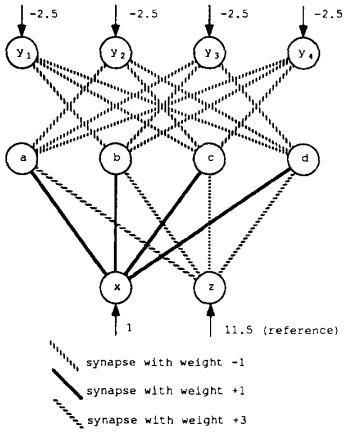


Fig. 7. A neural net constructed for a signal net of four components.

connects to all component neurons in the net except the ith one, with synaptic weights all being $-G_1$. Each y neuron has a reference $-(k-1.5)G_1$ so that, when detecting all $k-1$ connected component neurons in the same state, the extra influence fired by neuron x to these components is neutralized by the opposing influence fired by the y neuron. However, the addition of these y neurons has a side effect on the first observation, i.e., when all k components are in block X, each component neuron will also receive incorrectly an extra $(k-1)G_1$ (or $-(k-1)G_1$) amount of opposing influence from the y neurons if $X = A$ (or B). Finally, this can be compensated by introducing a neuron z which connects to all component neurons with synaptic weight $(k-1)G_1$ and has a reference being $(k(k-1) - 0.5)G_1$. The interconnections among the component neurons and the x, y, z neurons with $k = 4$; and $G_1 = 1$ is shown in Fig. 7. For an initial partitioning of having component a in block A and components b, c, and d in block B, the dynamic neural-net convergence operations for external wiring reduction are given in Table I. The neural subnetwork finally reaches a stable state by moving component a from block A to B, and the net-list is removed from the cut.

IV. Balancing the Partition

In this section, we address the issue of balancing the sizes of the two blocks that define a circuit partition. The association of block size evenness with the neural network system energy level is demonstrated. Here, to make things easier, we will start with the assumption that all components are of the same size, and later extend the solution for the general case.

TABLE I
An Example Neural Net Convergence for External Wiring Reduction

	component neurons				non-component neurons					
	a	b	c	d	x	y_1	y_2	y_3	y_4	z
step 0: state	1	-1	-1	-1	0	0	0	0	0	0
influence	0	0	0	0	-2	3	1	1	1	-6
step 1: state	1	-1	-1	-1	-1	-1	0	0	0	0
influence	-1	0	0	0	-2	3	1	1	1	-6
step 2: state	-1	-1	-1	-1	-1	-1	0	0	0	0
influence	-1	0	0	0	-4	3	3	3	3	-12
step 3: state	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
influence	-1	-1	-1	-1	-4	3	3	3	3	-12

As before we choose to represent the n components by n neurons with bipolar states, i.e., $s_i \in \{-1, +1\}$, $1 \le i \le n$. Moreover, if component i is currently in block $A(B)$ then neuron i is said to be in $-1(+1)$ state. Let E^{CP} denote the *degree of unevenness* of a circuit partition. We can define

$$E^{CP} = G_2/2 \cdot \left(\sum s_i\right)^2$$

where G_2 is some positive constant. E^{CP} is minimized to zero when there are equal numbers of $+1$'s and -1's as the states of the component neurons.

Note that we can rewrite the unevenness measure alternatively as

$$E^{CP} = G_2/2 \cdot \sum_i \sum_j s_i s_j.$$

Compared with the form of the neural net energy function

$$E^{NN} = -\frac{1}{2} \cdot \sum_i \sum_j w_{ij} s_i s_j$$

we find that E^{CP} can be made equivalent to E^{NN} by assigning all synaptic weights between component neurons of the neural net to $-G_2$. Note that the inclusion of self-feedback loops is indicated by the transformation. To avoid possible oscillation in neural net convergence, we need to ignore the creation of synapses as self-feedback loops. Thus a neuron's own state (position) is not taken into consideration by the very neuron processor when trying to balance the partition. The effect is insignificant when partitioning large circuits. Otherwise, the states of a neural net must be monitored closely to force the network to stop if an oscillation is detected. An empirical maximum convergence time limit may also be needed to prevent the problem.

Now, let us consider the case of having k components in block A. This way every component neuron will receive $-(n-2k)G_2$ amount of influence. If there are fewer components in A, i.e., $k < n/2$, the influence will be negative. Consequently, neurons which correspond to components in B will be encouraged to change state from

$+1$ to -1, while other neurons in A will be encouraged to stay in -1 state.

To combine the considerations of external line reduction and partition balancing, the weight values w_{ij}'s obtained here are added, respectively, with the ones obtained in Section III to define the final synaptic strengths for neuron interactions. In other words, this kind of influence created due to block size balancing will be combined with the influence created due to cell gain consideration to determine the component movements between blocks.

In the following, three typical circuits are divided evenly into two blocks to provide some neural net circuit partitioning performance measurements. For each experiment, we adopt the following technique to obtain the final partition.

1) Starting with two randomly formed circuit halves A and B, apply neural net partitioning to obtain an improved minimal partition as blocks A' and B'.

2) Partition A' into two circuit halves A'_1 and A'_2, and B' into B'_1 and B'_2.

3) Form new initial partitions as $A'_1 \cup B'_1$ and $A'_2 \cup B'_2$, and perform neural net partitioning.

4) Form second new initial partition as $A'_1 \cup B'_2$ and $A'_2 \cup B'_1$, and perform neural net partitioning.

5) Record the best partitioning obtained in steps 1), 3), and 4) as the result.

Of the three sample circuits, circuit 1 has 62 components and 33 signal nets, circuit 2 has 100 components and 53 signal nets, and circuit 3 has 58 components and 69 signal nets. For each circuit, 300 random initial partitions are tried for each experiment to obtain the performance results. Histograms of the number of external lines (cut size) achieved are shown in Fig. 8, with the overall performance summarized in Table II.

On the average, for all three example circuits used here, the Fiduccia and Mattheyses algorithm is able to find a partition with smaller cut. However, the average differences are within one or two lines, or less than 10% of the average cut size obtained by the Fiduccia and Mattheyses algorithm. As for the smallest cut size achieved, in one example the neural net approach is more effective by one external line, and in the other two examples the Fiduccia and Mattheyses method is better by two external lines. For both methods, the choice of initial partition for an iterative improvement is expected to have significant impact on the final solution's quality, which explains the large variances in cut sizes, σ^2, observed in the simulation results. In this respect, the substantial speed gain achievable by neural computing will be more attractive in attempting a large amount of random trials.

Now let us return to the issue of adjusting the degree of evenness of a partition by neural network convergence. Since the partitioning done by a neural net used two combined criteria, the importance of block size balancing relative to external line minimization can thus be adjusted by the G_2/G_1 ratio. G_2 can be set to a relatively larger number to obtain a high degree of partition evenness. On the other hand, if strict balancing of the partition is not required, G_2 can be set to a relatively smaller number so that the number of external lines may be further reduced by keeping the circuit roughly divided during the partitioning process. A simulation study is performed on circuit 4 which has 23 components and 23 signal nets, with results provided in Table III.

It may be noted that as the G_2/G_1 ratio increases, the variance in the block sizes improves but the average cut size tends to be larger, which may be explained by the lessened emphasis on selecting components with larger cell gains.

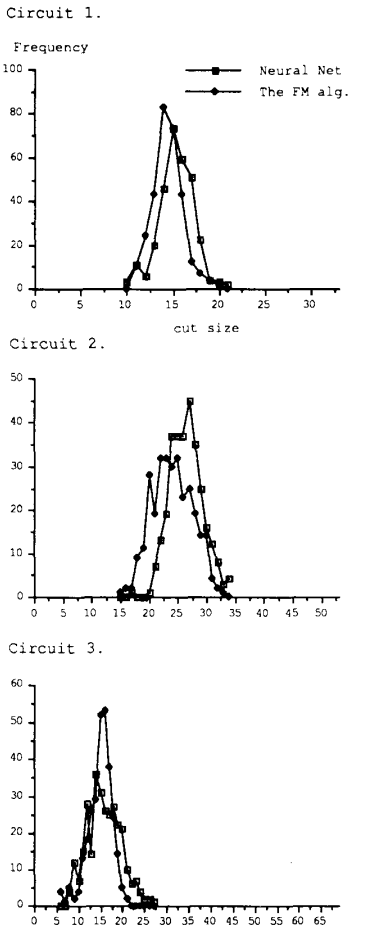


Fig. 8. Performance comparison of the Neural and the Fiduccia–Mattheyses algorithms.

TABLE II
CUT SIZES OBTAINED BY NEURAL NET AND FIDUCCIA–MATTHEYSES ALGORITHMS (σ = STANDARD DEVIATION)

	neural net				F&M's alg.			
	min	avg	max	σ	min	avg	max	σ
ckt 1	10	15.4	21	1.9	11	14.4	20	1.6
ckt 2	17	26.5	34	2.8	15	24.1	33	3.5
ckt 3	8	15.8	27	3.8	6	14.7	21	3.1

TABLE III
CUT SIZE VERSUS G_2/G_1 WITH EQUAL SIZE COMPONENTS (σ = STANDARD DEVIATION)

G_2/G_1	avg cut size	σ	avg size of A	σ
0.0	6.5	3.3	11.9	5.5
0.1	7.2	2.7	11.6	3.6
0.2	7.8	2.5	11.5	2.4
0.3	8.1	2.3	11.7	1.9

TABLE IV
CUT SIZE VERSUS G_2/G_1 WITH UNEQUAL SIZE COMPONENTS (α = STANDARD DEVIATION)

G_2/G_1	avg cut size	σ	avg size of A	σ
0.05	7.8	2.2	32.9	9.9
0.10	8.4	2.3	33.5	6.6
0.15	8.5	2.3	32.2	4.9
0.20	8.6	2.4	32.8	4.0

Finally, we will modify the neural net connection programming scheme to partition circuits with unequal component sizes. Recall that a component in A is represented by a neuron in a -1 state, and a component in B is represented by a neuron in a $+1$ state. Our intention is to make the positive (negative) influences received by a neuron proportional to the total size of block $A(B)$, so that a larger A will cause all neurons to receive a net positive influence, and a larger B will cause all neurons to receive a net negative influence.

One way to achieve this is to make the neural net interconnection weight matrix asymmetric, i.e., let w_{ij}, the connection strength from neuron j to neuron i, be negative and proportional to the size of component i in magnitude. Thus component j with size z_j in block $A(B)$ will send out through amplification of synapses, say $G_2 \cdot z_j(-G_2 \cdot z_j)$ amount of influence to push all neurons to block $B(A)$. Conversely, each neuron receives a total influence of $\Sigma_j(-G_2 \cdot z_j)$, which reflects the evenness of the partition with actual sizes of the components considered. Note that due to the asymmetric weight matrix, the neural net convergence is not guaranteed. Special care may be needed to terminate the operation. By setting z_j to the number of pins of component j, the neural net partitioning is carried out on circuit 4. In Table IV, the simulation results show that the quality of partitioning with unequal size components is similar to the one with equal size components.

V. CONCLUSIONS

In this paper, the circuit partitioning problem is solved using neural networks. We have demonstrated how the neural net state convergence properties can be utilized to balance the partition and also to select circuit elements for movement. According to the simulation results, both the average and the best-case solutions are comparable to those obtained by the Fiduccia and Mattheyses algorithm. The design is readily implementable in hardware on existing programmable electronic neural network chips [12], with some functional modifications to the neuron processors. Since neuron processors collectively perform binary thresholding operations, fast convergence to minimal solutions can be guaranteed. Thus the proposed approach can be used as a hardware acceleration of partitioning problems that normally require very high computation times. The technique discussed in this study can also be applied to the gate matrix placement problem [7].

REFERENCES

[1] L. I. Corrigan, "A placement capability based on partitioning," in *Proc. Design Automation Conf.*, pp. June 1979, pp. 406-413.

[2] A. E. Dunlop and B. W. Kernighan, "A procedure for placement of standard-cell VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 92-98, Jan. 85.

[3] C. M. Fiduccia and R. M. Mattheyses, "A linear heuristic for improving network partitions," in *Proc. Design Automation Conf.*, June 1982, pp. 175-181.

[4] H. P. Graf *et al.*, "VLSI implementation of a neural network memory with several hundreds of neurons," in *Proc. Conf. on Neural Networks for Computing*, Amer. Inst. of Phys., 1986, pp. 182-187.

[5] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," in *Proc. Nat. Acad. Sci. USA*, vol. 79, Apr. 1982, pp. 2554-2558.

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

[7] D. K. Hwang, W. K. Fuchs, and S. M. Kang, "An efficient approach to gate matrix layout," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 802-809, Sept. 87.

[8] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291-307, Feb. 1970.

[9] U. R. Kodres, "Partitioning and card selection," in *Design Automation of Digital Systems*, M. A. Breuer, ed. Englewood Cliffs, NJ: Prentice-Hall, 1972, pp. 173-212.

[10] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438-446, May 1984.

[11] W. S. McCulloch and W. H. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bull. Math. Biol.*, vol. 5, pp. 115-133, 1943.

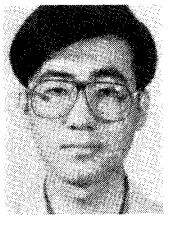
[12] A. F. Murray and A. V. W. Smith, "Asynchronous VLSI neural networks using pulse-stream arithmetic," *IEEE J. Solid-State Circuits*, vol. 23, pp. 688-697, June 1988.

[13] D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," in *Proc. 9th Design Automation Workshop*, Dallas, 1979, pp. 57-62.

[14] C. Sechen and D. Chen, "An improved objective function for mincut circuit partitioning," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1988, pp. 502-505.

[15] M. A. Sivilotti, M. R. Emerling, and C. A. Mead, "VLSI architectures for implementation of neural networks," in *Proc. Conf. on Neural Networks for Computing*, Amer. Inst. of Phys., 1986, pp. 408-413.

*



Jih-Shyr Yih received the B.S. degree in computer science and information engineering from National Taiwan University, Taipei, Republic of China, in 1981, and the M.S. degree in computer science from Michigan State University, East Lansing, in 1985. He is currently working towards the Ph.D. degree in the Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor.

His research interests include digital system testing, VLSI computer-aided design, fault-tolerant computing, computer architecture, and neural networks.

*

Pinaki Mazumder (S'84-M'87) for a photograph and biography please see page 511 of the May 1990 issue of this TRANSACTIONS.