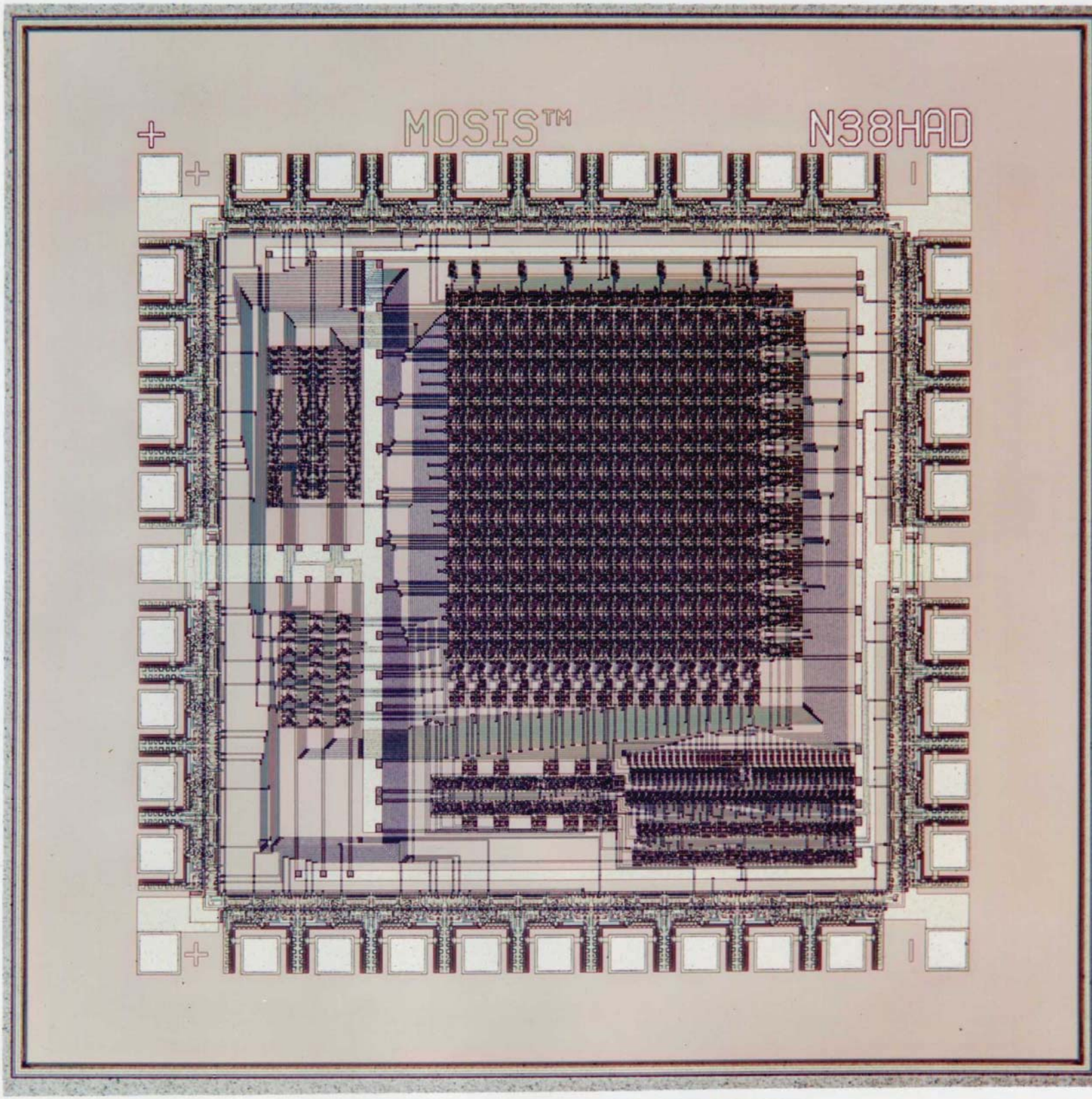
A DIGITAL NEURAL NETWORK CIRCUIT THAT AUTOMATICALLY REPAIRS FAULTY VLSI CHIPS TO IMPROVE CHIP MANUFACTURABILITY & SURVIVABILITY



Reseachers: M.D. Smith and P. Mazumder
The University of Michigan, Ann Arbor

Generation of Minimal Vertex Covers for Row/Column Allocation in Self-Repairable Arrays

Michael D. Smith, *Member, IEEE*,
and Pinaki Mazumder, *Senior Member, IEEE*

Abstract—This paper lays foundations for an approach to on-chip row/column allocation that exploits certain properties offered by laterally connected networks of simple threshold devices. As a sample application, it is demonstrated how electronic implementations of these networks can be used as the basis for effective memory array repair systems that require little hardware overhead.

Index Terms—Self-repair, redundant memory, embedded memory, neural network, vertex cover problem.

1 Introduction

THE decade of the 1980s saw phenomenal advances in VLSI circuit technology, as shrinking feature width and increasing die size allowed unprecedented levels of integration. These advances, unfortunately, also rendered fabrication processes more susceptible to impurity-related manufacturing defects, even as greater circuit complexity left many subsystems "embedded," where they cannot be observed externally or controlled directly. As prospects for further technological development reach previously unthinkable levels, it is becoming necessary to develop *built-in* systems which can automatically repair partially faulty integrated circuits.

This paper lays theoretical foundations for an approach to on-chip spare allocation in regular VLSI structures such as memories. In high-density DRAM chips, manufacturers commonly include spare rows and columns of memory cells so that rows and columns in which faulty cells are detected can be replaced. Repair by the replacement of rows and columns of memory cells, rather than by replacement of individual cells or entire memory sub-arrays, offers a good compromise between simplicity of reconfiguration hardware and efficient utilization of spare memory. This reconfiguration strategy, however, gives rise to a difficult optimization problem. Given a scattering of faulty cells in a two-dimensional rectangular array, determining whether the array can be repaired using a limited number of spare rows and columns is NP-complete [1].

The problem of efficient spare row and column allocation in redundant memories has been widely studied [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], and a variety of applicable algorithms are present in the literature. To be considered viable for the purpose of on-chip spare allocation, however, an algorithm must satisfy two challenging and mutually conflicting design requirements. First, it must lend itself to easy implementation using very little hardware, incurring a minimal penalty in terms of silicon area overhead, and second, the algorithm must be capable of rapidly generating high-quality solutions to spare allocation problem instances. Previously reported algorithms for spare row and column allocation, most of which were designed for incorporation into dedicated repair systems on integrated circuit production lines, generally fail to meet one or both of these criteria. In contrast to these existing methods of spare allocation, the approach presented in this paper offers the combined advantages of high execution speed, simplicity of implementation, and high rates of successful spare allocation in problem instances for which solutions exist.

Section 2 of this paper begins the theoretical discussion by describing a graph-theoretic model for the problem of array repair through row and column replacement, relating the problem to that of finding a constrained vertex cover in an undirected bipartite graph. Some relevant properties of laterally connected threshold devices are reviewed briefly in Section 3. Section 4 introduces a class of threshold device networks which is proven applicable to a generalized form of the vertex cover problem, and Section 5 concludes the theoretical discussion with an examination of certain convergence issues. Section 6 demonstrates how electronic implementations of the proposed networks can be applied to the problem of spare allocation in embedded memories. Simulation results indicate that these networks can be made to provide consistently feasible solutions to the computationally intractable spare allocation problem.

2 Graph-Theoretic Definitions and Model

We begin by defining some common terms from graph theory, regarding the vertex covers of undirected graphs.

DEFINITION 1. *Given an undirected graph G consisting of a set of vertices V and a set of edges E, i.e., G = (V, E), a subset V_c of V is called a* vertex cover *of G iff every edge in E has at least one endpoint in V_c.*

DEFINITION 2. *Given some subset R of V, a vertex cover V_c of G is said to be* minimal *with respect to R iff there exists no vertex i in $R \cap V_c$ such that $V_c - \{i\}$ is a vertex cover of G.*

The problem of finding a vertex cover whose cardinality is less than some specified constant is known as the vertex cover problem, and a vertex cover which contains the least number of vertices necessary to cover a given graph is considered optimal. For the purposes of this paper it will prove useful to define a new term, the generalized vertex cover (GVC) problem, as follows.

DEFINITION 3. *Let G = (V, E) be an undirected graph, and let $R_1, R_2, \ldots, R_D$ be disjoint subsets of its vertices. Given integer constants $C_1, C_2, \ldots, C_D$ we define the* generalized vertex cover problem *for G with respect to $R_1, R_2, \ldots, R_D$ to be that of finding a vertex cover V_c of G such that the cardinality of $R_i \cap V_c$ is less than C_i for $1 \le i \le D$.*

An optimal solution to the generalized vertex cover problem is a vertex cover with the property that

$$\sum_{i=1}^{D} |R_i \cap V_c|$$

is a minimum, where $|R_i \cap V_c|$ denotes the cardinality of $R_i \cap V_c$.

Finally, we define the connection matrix associated with an undirected graph, a construction which will prove useful in later analyses.

DEFINITION 4. *Given an undirected graph G = (V, E) in which V has cardinality N, let the* connection matrix *C associated with G be the N-by-N matrix whose element c_{ij} equals 1 iff an edge connects vertices i and j, i.e., iff $e(i, j) \in E$, or equals 0 otherwise.*

It is assumed that the N vertices of G are numbered 1 through N in some arbitrary manner. Note that, by definition, the connection matrix associated with an undirected graph must be symmetric. Furthermore, if no edge in G connects any vertex with itself, then the graph is said to possess no self-loops and the main diagonal of its associated connection matrix consists entirely of 0s.

• *The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122. E-mail: mazum@eecs.umich.edu.*

Manuscript received Aug. 23, 1993; revised Feb. 8, 1995.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number C95137.

0018-9340/96$05.00 © 1995 IEEE

An instance of the spare row/column allocation problem consists of a two-dimensional rectangular array of cells, some of which are designated faulty, and two integers providing, respectively, the number of spare rows and the number of spare columns available with which to make repairs. Any subset of the rows and columns in such an array constitutes a *spare allocation scheme*. In the context of memory stuck-at fault repair, we define a spare allocation scheme to be *valid* if and only if its constituent rows and columns, taken together, contain every faulty cell in the array at hand; a valid spare allocation scheme is said to be *minimal* if and only if there does not exist any proper subset of its constituent rows and columns which is itself valid. We define a valid spare allocation scheme to be *feasible* if and only if the number of designated rows and the number of designated columns each obey their respective upper bounds as set forth under the particular problem instance at hand. Finally, we consider *optimal* any feasible spare allocation scheme which designates for replacement the minimum total number of rows and columns.

Fig. 1 illustrates these concepts. Assuming there exist four spare rows and four spare columns with which to make repairs, the diagram in Fig. 1a represents an instance of the spare allocation problem. Fig. 1b indicates those rows and columns designated for replacement under one possible spare allocation scheme. This solution is valid, minimal, feasible, and optimal with respect to the simple problem instance at hand.

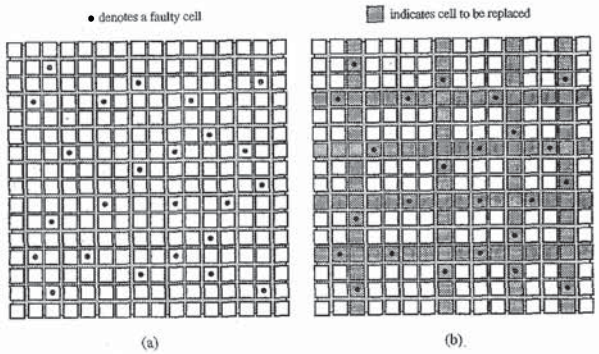


Fig. 1. A faulty array and a valid spare allocation scheme.

The problem of spare row/column allocation in a two-dimensional rectangular array can be modeled as that of computing a constrained vertex cover in an undirected bipartite graph [1]. Consider an array of M_1 by M_2 cells, containing a defect pattern in which faults occur in m_1 or fewer distinct rows and m_2 or fewer distinct columns, where $m_1 \leq M_1$ and $m_2 \leq M_2$. If a graph on $m_1 + m_2$ vertices is constructed such that each row and each column of the array corresponds to exactly one vertex, and such that an edge connects vertices i and j if and only if a faulty cell lies at the intersection of the corresponding row and column, then it is easily proven that valid spare allocation schemes for the faulty array exhibit a one-to-one correspondence with vertex covers of the associated graph. Applying the graph-theoretic definition provided earlier, furthermore, it is clear that an instance of the spare allocation problem is easily represented as a special case of the generalized vertex cover problem, in which $D = 2$, the subset R_1 consists of all vertices corresponding to array rows, R_2 consists of all vertices corresponding to array columns, and the constants C_1 and C_2 are determined by the number of available spare rows and spare columns, respectively. It is worth mentioning that this graph-theoretic model, and hence the results described in this paper, apply only to arrays of two dimensions.

As an example, consider the faulty 1,024-by-1,024 array illustrated in Fig. 2a. Fig. 2b depicts an associated bipartite graph in which vertices 1 through 4 correspond, respectively, to rows 42, 118, 629, and 823 of the array, and vertices 5 through 8 correspond, respectively, to columns 37, 125, 225, and 921. Every vertex cover of this graph can be mapped to a valid spare allocation scheme for the original array.

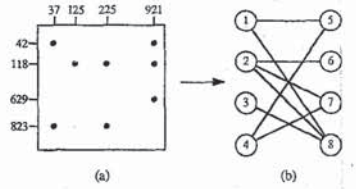


Fig. 2. A faulty array and an associated bipartite graph.

3 Relevant Properties of Laterally Connected Threshold Devices

A threshold device network of the form considered throughout this paper consists of highly interconnected simple processing elements, which are numbered 1 through N in some arbitrary manner. The current state s_i of threshold device i at any given time is either 0 or 1, and the current state S of the system is the binary vector determined collectively by the states of all N devices.

Given a network in the current state S, its next state S' is brought about by updating the state of exactly one of its constituent threshold devices. Thus, S and S' are separated by a maximum Hamming distance of one. The next state s_i of device i is determined by

$$s_i' = \begin{cases} 1 \text{ if } F_i(S) > 0 \\ 0 \text{ if } F_i(S) < 0 \\ s_i \text{ otherwise} \end{cases} \quad \text{where} \quad F_i(S) = \sum_{j=1}^{N} w_{ij} s_j + b_i$$

at the time the device is clocked. Each w_{ij} is a constant multiplicative weight between the output of threshold device j and the input of threshold device i, and b_i is a constant bias term. Fig. 3 presents a high-level schematic of a general threshold device network. Threshold devices are "laterally connected" in the sense that the input to each device is a function of the outputs from every other device in the network.

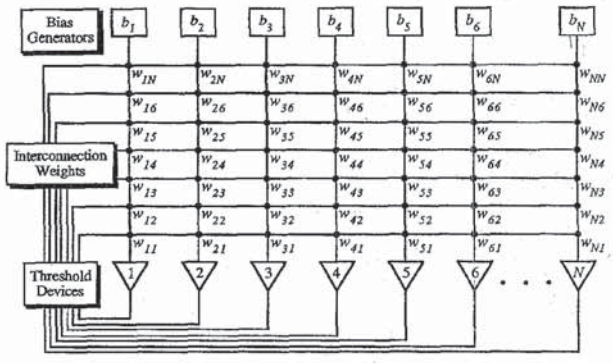


Fig. 3. Schematic of a general threshold device network.

If the matrix W of interconnection weights is symmetric and possesses no nonzero terms on its main diagonal, i.e., if $w_{ij} = w_{ji}$ and $w_{ii} = 0$ for $1 \leq i, j \leq N$, then the value of the function

$$E(S) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} s_i w_{ij} s_j - \sum_{i=1}^{N} b_i s_i$$

is guaranteed to decrease monotonically as the state of a threshold device network evolves. Stable states of a network, those states in which the next state s_i' of threshold device i equals s_i for $1 \le i \le N$, exhibit a one-to-one correspondence with the local minima of this "energy function." A network of laterally connected threshold devices whose stable states represent potential solutions to an optimization problem can be used as a simple means for computing these solutions [12].

4 Threshold Device Networks for Generalized Vertex Cover Problems

The primary theoretical contributions of this paper are expressed in three theorems, each of which contributes to the eventual development of methodologies for effective on-chip spare row/column allocation in VLSI arrays. The purpose of this section is to identify and analyze a class of threshold device networks which is proven applicable to the GVC problem.

Consider a network of threshold devices whose energy function is of the form

$$E_{VC}(S) = -A\sum_{i=1}^{N}\sum_{j=1}^{N} s_i c_{ij} s_j$$

where each c_{ij} is determined by the connection matrix of some undirected graph G with no self-loops, and where A is any positive constant. If a correspondence is established between device i of the network and vertex i of G for $1 \le i \le N$, and if a subset V_0 is defined of which vertex i is considered a member whenever $s_i = 0$, then we have the following lemma.

LEMMA 1. *Let H be a network of laterally connected threshold devices whose energy function is given by E_{VC}. A necessary and sufficient condition for any state to be a local minimum of E_{VC}, and hence a stable state of H, is that its associated vertex set V_0 be a vertex cover of G.*

PROOF OF NECESSITY. Suppose there exists some state S of H such that

1) the state is a local minimum of E_{VC}, and
2) its associated V_0 is not a vertex cover of G.

Recalling the correspondence established between threshold devices and vertices, and between connection matrix elements and graph edges, condition 2 implies that there must be some i and j such that $c_{ij} = s_i = s_j = 1$. Switching either s_i or s_j from 1 to 0, however, results in a decrease of system energy by $2A$, a strictly positive value. Thus S is not a local minimum of E_{VC}, and condition 1 is violated.

PROOF OF SUFFICIENCY. Inspection of E_{VC} reveals that its globally minimal value is 0. This minimal value is achieved iff $s_i = 0$, or $s_j = 0$, or both, whenever $c_{ij} = 1$. As every state S which represents a vertex cover must satisfy this condition, every such state is a local minimum of E_{VC}. □

Now consider a laterally connected network whose N threshold devices are conceptually divided into K arbitrary subsets, L_1 through L_K, such that each device is a member of exactly one subset and no subset is empty. Let us denote by z_i the number of threshold devices in set L_i which, at any given time, are in state 0. Suppose that the network energy function is given by

$$E_{MVC}(S) = E_{VC}(S) + \sum_{i=1}^{K} B_i z_i^2 .$$

Establishing the usual correspondence between vertices and threshold devices, and defining V_0 as before, we state the following lemma.

LEMMA 2. *Let H be a network of laterally connected threshold devices whose energy function is given by E_{MVC}, and whose parameters A and B_i are chosen so as to obey the inequalities*

$$A > \frac{B_i(2|L_i| - 1)}{2} \text{ and } B_i \ge 0 \text{ for } 1 \le i \le K.$$

A necessary condition for any state of H to be stable is that its associated V_0 be a vertex cover of G.

PROOF. Suppose there exists a stable state S of H for which V_0 is not a vertex cover of G. Then it must be true that

1) some edge in G has neither of its endpoints in V_0 and
2) system energy cannot be reduced by rectifying this situation.

We demonstrate that our choice of network parameters ensures conditions 1 and 2 cannot simultaneously be satisfied.

Condition 1 implies that $z_i < |L_i|$ for some i between 1 and K. Suppose that switching one of the remaining $|L_i| - z_i$ threshold devices in subset L_i from state 1 to state 0 will have the effect of including in V_0 an endpoint of n edges whose other endpoints did not previously lie in V_0. Doing so will prevent $2n$ nonzero connection matrix elements from contributing to the value of the energy function. Such a change in the state of H results in a net decrease in energy iff

$$-A(2n) + B_i\left[(z_i + 1)^2 - z_i^2\right] < 0$$

$$\text{or, in other words, } A > \frac{B_i(2z_i + 1)}{2n}$$

In the limiting case when only one member of L_i is in state 1, and switching the state of this device eliminates the contributions of two nonzero connection matrix elements, this inequality reduces to

$$A > \frac{B_i(2|L_i| - 1)}{2}.$$

Maintaining this inequality across every parameter of H guarantees that conditions 1 and 2 are incompatible. Hence, stable states of H are associated with vertex covers of G. □

Every stable state of a threshold device network satisfying the requirements of Lemma 2 represents some vertex cover of the network's associated graph G. It is not the case, however, that every vertex cover of G is represented by some stable state. We assert the following theorem.

THEOREM 1. *Let H be a network of laterally connected threshold devices which fulfills the requirements of Lemma 2, and let U denote the union of all sets L_i of threshold devices for which $B_i > 0$. A necessary and sufficient condition for any state of H to be stable is that its associated vertex set V_0 be a vertex cover of G which is minimal over the set of vertices corresponding to U.*

PROOF OF NECESSITY. Suppose there exists some state S of H such that

1) the state is stable, and
2) it does not represent a vertex cover of G which is minimal over the set of vertices corresponding to U.

Condition 1 and Lemma 2 guarantee that S does in fact represent a vertex cover of G. Condition 2 implies that there exists

some vertex i in V_0 whose elimination from the set would yield a new set which is also a vertex cover, and Lemma 1 dictates that the value of E_{VC} cannot change as a result. The elimination of vertex i from V_0 is effected by switching s_i from 0 to 1, a process which results in a reduction of system energy given by

$$B_j[z_j^2 - (z_j - 1)^2]$$

assuming that device i is a member of set L_j, and that z_j members of this set are in state 0 when H is in state S. As this reduction is a strictly positive value whenever device i is a member of U, it must be true that S is not a minimum of E_{MVC}. Hence, S is not a stable state of H.

PROOF OF SUFFICIENCY. Suppose H is currently in some state S which represents a vertex cover that is minimal over the vertices corresponding to U. The next state of H can differ from S in one of three ways. Either

1) the state of some device is switched from 1 to 0, or
2) the state of some device which does not belong to U is switched from 0 to 1, or
3) the state of some device which belongs to U is switched from 0 to 1.

Lemma 1 guarantees that S is a local minimum of E_{VC}. Since the sum of the remaining terms in E_{MVC} never decreases as the number of 0s in S increases, option 1 cannot result in a reduction of system energy. Nor can option 2, since any device which is not a member of U must belong to some set L_i for which $B_i = 0$. Because option 3 inevitably yields a new state which does not represent a vertex cover of G, it must lead to an increase in system energy, as made evident in the proof of Lemma 2. We conclude that S is a local minimum of E_{MVC} and hence a stable state of H. □

Clearly, a vertex cover which is nonminimal over some vertex set R can never be an optimal solution to the GVC problem with respect to sets $R_1, R_2, \ldots, R_D$ if R is a subset of $R_1 \cup R_2 \cup \ldots \cup R_D$. The threshold device network of Theorem 1 is naturally applicable to the GVC problem, and a simple method for mapping any specific problem instance onto such a network is apparent. For a graph G with any given choice of disjoint vertex sets, one simply constructs a network whose energy function is of the form of E_{MVC}, where each c_{ij} is defined by the connection matrix associated with G, and where some set of threshold devices corresponds to each vertex set over which vertex covers are to be minimized. A positive B_i value is chosen for each such set of threshold devices, while B_i values for any remaining sets are left at 0, and the constant A is chosen as per the specification of Theorem 1.

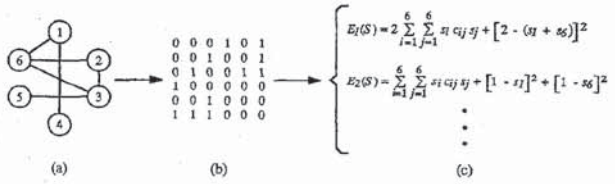


Fig. 4. An undirected graph, its connection matrix, and energy functions of networks that find minimal covers with respect to vertices 1 and 6.

Aspects of the mapping process are illustrated in Fig. 4. Given the graph depicted in Fig. 4a, suppose it is desired to generate vertex covers which are minimal with respect to the set of vertices consisting of vertex 1 and vertex 6. The graph's connection matrix, shown in Fig. 4b, is used to determine each c_{ij} in the network energy functions of Fig. 4c. Function E_1 results from the mapping technique described above. Function E_2, the product of an alternative mapping process, gives rise to identical stable states and serves to demonstrate that the mapping from a given instance of the GVC problem to a network of threshold devices is not generally unique.

5 Algorithms for Network Convergence

Given any instance of the GVC problem, it is now possible to construct a network of laterally connected threshold devices whose stable states represent potentially optimal solutions. Nevertheless, the applicability of such a network to spare allocation is limited without some means to ensure that stable states are encountered within a reasonable length of time. This section describes threshold device clocking procedures which, for networks that fulfill the requirements of Theorem 1, guarantee convergence to a stable state in a maximum number of steps.

Theorem 2 describes a procedure which can be preset, in the sense that an inflexible sequence can be established in which devices are to be clocked. It is presented below without proof.

THEOREM 2. *Let H be a network of laterally connected threshold devices which fulfills the requirements of Theorem 1, let U denote the union of all sets L_i of threshold devices for which $B_i > 0$, and let S be any initial state. If each threshold device in H is clocked exactly once in any arbitrary order, after which each threshold device in the set U is again clocked exactly once in any arbitrary order, then the resulting state S' of H will be stable.*

It should be noted that a similar scheme was derived in [13] for a less general class of networks.

One drawback to the procedure of Theorem 2 is that it disregards information present in the current state of the network which can be used to inform the choice of which network state to select next. Theorem 3 describes an alternative procedure which makes use of this information.

THEOREM 3. *Let H be a network of size N which fulfills the requirements of Theorem 1, and whose constants obey the inequality*

$$A > B_i (2 \mid L_i \mid - 1) \text{ for } 1 \le i \le K.$$

Define U in the usual manner, and for each threshold device i in H, define a function G_i such that

$$G_i(S) = \begin{cases} F_i(S) \text{ if } F_i(S) > 0 \text{ and } s_i = 0 \\ -F_i(S) \text{ if } F_i(S) < 0 \text{ and } s_i = 1 \\ 0 \text{ otherwise.} \end{cases}$$

Suppose a clocking procedure is established under which threshold device i in H is clocked iff G_i is greater than or equal to G_j for all other $j \neq i$, with ties broken in some arbitrary manner. Then, convergence to a stable state is guaranteed in no more than $N + \mid U \mid - 1$ steps.

Intuitively, the function G_i quantifies the encouragement received by a threshold device i to switch its state. It is easily shown that the procedure of Theorem 3 effectively leads a threshold device network along the path of gradient descent in E_{MVC}.

6 On-Chip Spare Allocation in Embedded Memories: A Sample Application

Section 2 established that the problem of spare row/column allocation can be modeled as a special case of the generalized vertex cover problem. Theorem 1 went on to demonstrate that, given any instance of the GVC problem, a network of laterally connected threshold devices exists which possesses a unique stable state representing each minimal solution to the instance at hand. Further-

more, given a specified network architecture, it is clear that different problem instances can be accommodated by simply modifying the connection matrix upon which the network's energy function is based. It follows from these results that a programmable network of an appropriate size could provide the basis for an optimization system capable of devising spare allocation schemes for any pattern of faults likely to occur in a given array. This section demonstrates how Theorems 2 and 3, when applied to appropriate network architectures, can offer a high probability of encountering stable states which represent *feasible* spare allocation schemes. Electronic implementations of the resulting systems, though extremely simple, nevertheless offer a high rate of success in solving the computationally intractable spare allocation problem.

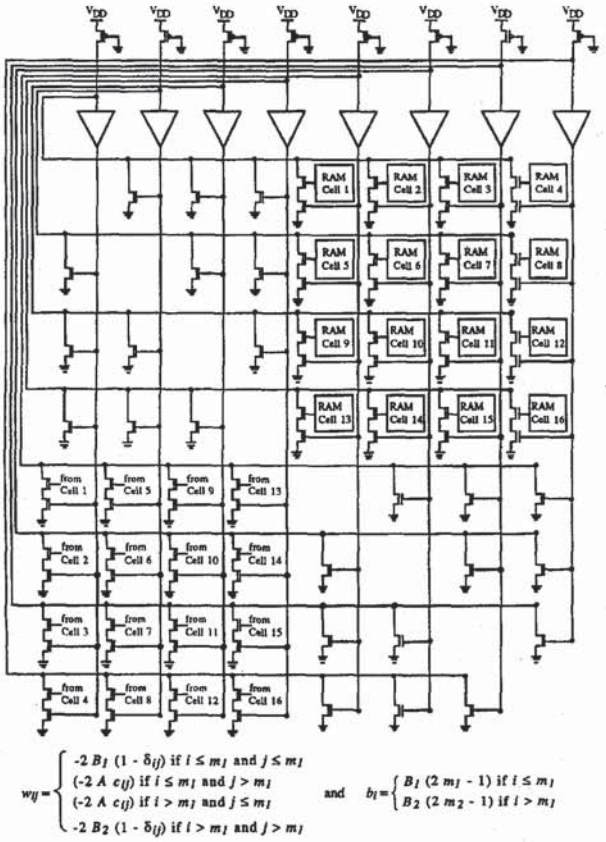


Fig. 5. An implementation of network H_1 for $m_1 = m_2 = 4$.

Employing the technique described in Section 4 to map an instance of the GVC problem onto a network of threshold devices, the spare allocation problem can be represented as follows. With faults expected to occur in m_1 or fewer distinct rows and m_2 or fewer distinct columns, construct a network H_1 of size N, where $N = m_1 + m_2$, and choose subsets L_1 and L_2 of cardinality m_1 and m_2, respectively. Let the threshold devices of set L_1 be numbered from 1 to m_1, let those of L_2 be numbered from $m_1 + 1$ to N, and establish a correspondence between device i and row i of the fault pattern for $1 \le i \le m_1$, and between device i and column $i - m_1$ for $(m_1 + 1) \le i \le N$. Finally, let connection matrix elements be determined by a graph associated with the specific pattern of faults to be repaired. The energy function thus created is

$$E_1(S) = A\sum_{i=1}^{N}\sum_{j=1}^{N} s_i c_{ij} s_j + B_1\left[M_1 - \sum_{i=1}^{M_1} s_i\right]^2 + B_2\left[M_2 - \sum_{i=m_1+1}^{N} s_i\right]^2 .$$

By expanding squared terms and relating the result to the general energy function of Section 3, values can be derived for network interconnection weights and bias terms. These values, along with an electronic implementation of the resulting network, are illustrated in Fig. 5.

If the initial state of H_1 is fixed to the N-vector consisting entirely of 1s, then the gradient descent procedure of Theorem 3 allocates spare rows and columns according to what, in effect, is a simple heuristic algorithm. Gradient descent allocates the first spare to cover some row or column containing the maximum number of faults. Spares are assigned thereafter in the same manner, with ties between any row and column broken in favor of that set from which fewer spares have already been allocated. The tendency to eliminate the maximum number of faults with each allocation may even be overridden if the number of spares allocated from one set greatly exceeds that of the other. Intuitively, the system will tend to preserve a balance between the number of spare rows and spare columns assigned, thereby preferring stable states which represent feasible spare allocation schemes.

Theorem 2 can be used as the basis for an even less complex alternative system. Since the preset updating procedure is unable to exploit any heuristic qualities present in a given energy function, however, it is worthwhile to construct a new energy function whose implied network H_2 is as simple as possible. Let us modify network H_1 by establishing N threshold device subsets, L_1 through L_N, rather than the original L_1 and L_2. The resulting energy function

$$E_2(S) = A\sum_{i=1}^{N}\sum_{j=1}^{N} s_i c_{ij} s_j + B_1\left[1 - s_1\right]^2 + B_2\left[1 - s_2\right]^2 + \cdots + B_N\left[1 - s_N\right]^2$$

gives rise to the interconnection weights and bias terms detailed in Fig. 6. An electronic implementation of network H_2, also shown in Fig. 6, is easily constructed using strictly digital components. Approximating the silicon layout area of an integrated circuit by the number of transistors it comprises, and given an expected maximum fault pattern size of 32 by 32 elements, a network of this kind would require only 0.29 percent of the total area of a 1 MBit DRAM.

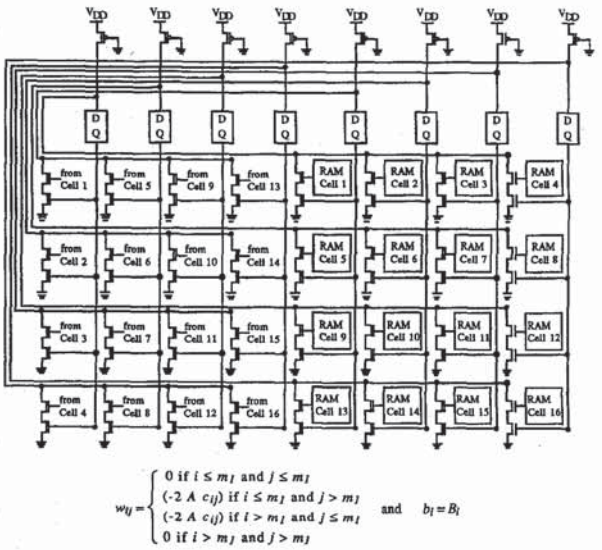


Fig. 6. An implementation of network H_2 for $m_1 = m_2 = 4$.

Consideration must now be given to a means of generating initial states for network H_2, and to a strategy for clocking its threshold devices. In order to ensure that neither row nor column re-

placements are preferred, and to increase the likelihood of devising feasible spare allocation schemes, it is reasonable to establish an updating protocol which alternates between threshold devices corresponding to rows and devices corresponding to columns. Initial states may be chosen in a random fashion to allow for thorough exploration of the search space, and repeated iterations performed until the network converges to a state representing a feasible spare allocation scheme.

Fig. 7 illustrates a paradigm for using a threshold device network to provide for on-chip spare allocation in embedded memory arrays. Faulty cells are located by BIST hardware, and the resulting pattern of faults is used to configure the interconnection weight matrix of an appropriate network. (It is important to note that, typically, fault distributions within large memories can be represented by compacted patterns which are many orders of magnitude smaller.) With programming completed, control hardware resets the threshold devices to some initial state, and begins an updating procedure which brings the network to convergence. The resulting stable state is evaluated to determine whether it represents a feasible solution to the problem instance at hand. If so, a signal is generated which directs reconfiguration hardware to repair the memory as per the suggested spare allocation scheme.

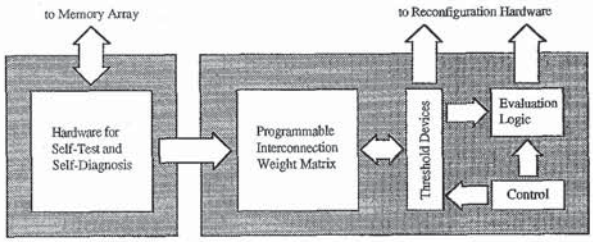


Fig. 7. Application of a threshold device network to spare allocation in embedded memories.

Tables 1 and 2 summarize the performance of the heuristic and iterative approaches, respectively, under variations of fault pattern size, number of available spare rows and columns, and average number of faulty elements per pattern. Fault patterns were generated randomly through a process which guarantees repairability. Table 3 summarizes the performance of Repair Most [10], the most viable alternative algorithm for on-chip spare allocation, when applied to the same sets of test instances. Tables 4 and 5 detail performance of the iterative approach as a function of the number of iterations completed. Here, "rate of success" refers to the percentage of problem instances for which the algorithm under test was capable of devising a feasible solution. It is assumed that the spare allocation systems under test are themselves fault free.

TABLE 1
PERFORMANCE OF THE HEURISTIC APPROACH, SHOWING THE RATE OF SUCCESS FOR EACH CASE

	16-by-16 Fault Pattern			32-by-32 Fault Pattern		
	4 Spare Rows and 4 Spare Columns	6 Spare Rows and 6 Spare Columns	8 Spare Rows and 8 Spare Columns	8 Spare Rows and 8 Spare Columns	12 Spare Rows and 12 Spare Columns	16 Spare Rows and 16 Spare Columns
10% Faulty	98.6	100	100	99.8	99.8	99.8
20% Faulty	100	99.4	99.0	100	100	99.8
30% Faulty	100	100	98.0	100	100	99.8

TABLE 2
PERFORMANCE OF THE ITERATIVE APPROACH, SHOWING THE RATE OF SUCCESS FOR EACH CASE

	16-by-16 Fault Pattern			32-by-32 Fault Pattern		
	4 Spare Rows and 4 Spare Columns	6 Spare Rows and 6 Spare Columns	8 Spare Rows and 8 Spare Columns	8 Spare Rows and 8 Spare Columns	12 Spare Rows and 12 Spare Columns	16 Spare Rows and 16 Spare Columns
10% Faulty	100	100	100	100	100	100
20% Faulty	100	100	100	100	100	100
30% Faulty	100	100	100	100	100	100

TABLE 3
PERFORMANCE OF REPAIR MOST, SHOWING THE RATE OF SUCCESS FOR EACH CASE

	16-by-16 Fault Pattern			32-by-32 Fault Pattern		
	4 Spare Rows and 4 Spare Columns	6 Spare Rows and 6 Spare Columns	8 Spare Rows and 8 Spare Columns	8 Spare Rows and 8 Spare Columns	12 Spare Rows and 12 Spare Columns	16 Spare Rows and 16 Spare Columns
10% Faulty	71.0	83.4	94.4	94.8	85.6	91.2
20% Faulty	99.2	80.0	78.4	100	99.2	92.8
30% Faulty	100	97.2	80.6	100	100	98.9

TABLE 4
PERFORMANCE OF THE ITERATIVE APPROACH WHEN APPLIED TO 16-BY-16 FAULT PATTERNS, SHOWING RATE OF SUCCESS AS A FUNCTION OF NUMBER OF ITERATIONS

		Iterations									
		10	20	30	40	50	60	70	80	90	100
4 Spare Rows and 4 Spare Columns	10% Faulty	77.6	89.4	93.0	94.8	96.2	97.2	97.2	98.0	98.8	99.0
	20% Faulty	90.8	96.0	98.0	98.8	98.8	99.0	99.4	99.4	99.6	99.6
	30% Faulty	97.4	99.2	99.8	100	100	100	100	100	100	100
6 Spare Rows and 6 Spare Columns	10% Faulty	96.6	99.2	99.8	99.8	100	100	100	100	100	100
	20% Faulty	73.2	84.8	90.0	91.6	93.2	94.6	95.0	96.0	96.6	97.6
	30% Faulty	83.0	91.6	94.0	95.0	95.8	96.6	97.6	98.4	98.6	98.6
8 Spare Rows and 8 Spare Columns	10% Faulty	99.8	100	100	100	100	100	100	100	100	100
	20% Faulty	92.0	97.0	98.4	99.2	99.8	99.8	99.8	99.8	99.8	99.8
	30% Faulty	78.2	88.0	92.0	95.2	96.4	96.8	97.4	98.0	98.4	98.6

TABLE 5
PERFORMANCE OF THE ITERATIVE APPROACH WHEN APPLIED TO 32-BY-32 FAULT PATTERNS, SHOWING RATE OF SUCCESS AS A FUNCTION OF NUMBER OF ITERATIONS

		Iterations									
		20	40	60	80	100	120	140	160	180	200
8 Spare Rows and 8 Spare Columns	10% Faulty	78.6	87.8	91.4	94.8	96.2	97.0	97.8	98.0	98.6	98.8
	20% Faulty	97.0	98.8	99.8	99.8	99.8	99.8	100	100	100	100
	30% Faulty	98.9	99.8	100	100	100	100	100	100	100	100
12 Spare Rows and 12 Spare Columns	10% Faulty	82.8	93.0	96.4	96.6	97.8	98.0	98.4	98.6	98.8	99.0
	20% Faulty	75.4	84.0	90.0	93.0	94.8	95.6	96.4	97.2	97.4	97.8
	30% Faulty	93.4	96.4	97.8	98.8	98.8	98.8	98.8	98.8	99.2	99.2
16 Spare Rows and 16 Spare Columns	10% Faulty	99.4	100	100	100	100	100	100	100	100	100
	20% Faulty	83.0	91.2	94.8	96.4	97.0	98.0	98.6	99.0	99.0	99.2
	30% Faulty	72.2	82.0	87.2	89.2	91.6	92.8	93.0	94.0	94.4	95.0

7 CONCLUSION

This paper lays foundations for an approach to on-chip spare allocation in rectangular VLSI arrays, demonstrating how properly designed networks of simple threshold devices can be used as the basis for optimization systems which are simple and effective. It should be emphasized that the proposed hardware systems are *not* as powerful as some existing software-based spare allocation techniques. Such existing techniques, however, were developed for utilization by external repair equipment, and the algorithms employed typically require the full resources of a general-purpose digital computer. Hardware implementations would incur far too much area overhead to be considered viable for the purpose of built-in self-repair. In contrast, the systems developed in this paper achieve near perfect success in devising spare allocation schemes, using hardware whose complexity is negligible when compared with a VLSI array of any substantial size.

Earlier research along these lines [14] succeeded in developing threshold device networks which, intuitively, seemed applicable to the spare allocation problem, but which occasionally failed to yield valid spare allocation schemes in practice. While they do build upon these earlier results, the systems described in this pa-

per include a number of major refinements which make their actual implementation for the purpose of built-in self-repair an immediate practical possibility. First and most importantly, the general network architectures developed are proven to possess stable states which represent only valid spare allocation schemes. Second, certain special instances of these networks can be implemented in a straightforward manner using strictly digital components. It deserves mention also that Theorem 1, as derived herein, is easily extended to the case of threshold device networks which operate in the analog domain [15], raising the possibility of alternative optimization system designs that offer much greater speed of operation than those described here.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation under Grant Number MIP-9013092.

References

[1] S.-Y. Kuo and W.K. Fuchs, "Efficient spare allocation for reconfigurable arrays," *IEEE Design & Test*, vol. 4, pp. 24-31, Feb. 1987.
[2] D. Blough and A. Pelc, "A clustered failure model for the memory array reconfiguration problem," *IEEE Trans. Computers*, vol. 42, pp. 518-528, May 1993.
[3] M.-F. Chang, W.K. Fuchs, and J.H. Patel, "Diagnosis and repair of memory with coupling faults," *Proc. 1988 Int'l Conf. Computer-Aided Design*, pp. 524-527, Nov. 1988.
[4] J.R. Day, "A fault-driven, comprehensive redundancy algorithm," *IEEE Design & Test*, vol. 2, pp. 35-44, June 1985.
[5] R.C. Evans, "Testing repairable RAMs and mostly good memories," *Proc. 1981 Int'l Test Conf.*, pp. 49-55, Oct. 1981.
[6] N. Funabiki and Y. Takefuji, "A parallel algorithm for allocation of spare cells on memory chips," *IEEE Trans. Reliability*, vol. 40, pp. 338-346, Aug. 1991.
[7] R.W. Haddad and A.T. Dahbura, "Increased throughput for the testing and repair of RAMs with redundancy," *Proc. 1987 Int'l Conf. Computer-Aided Design*, pp. 230-233, Nov. 1987.
[8] N. Hasan and C.L. Liu, "Minimum fault coverage in reconfigurable arrays," *Proc. 1988 Int'l Symp. Fault-Tolerant Computing*, pp. 348-353, June 1988.
[9] F. Lombardi and W.K. Huang, "Approaches for the repair of VLSI/WSI DRAMs by row/column deletion," *Proc. 1988 Int'l Symp. Fault-Tolerant Computing*, pp. 342-347, June 1988.
[10] M. Tarr, D. Boudreau, and R. Murphy, "Defect analysis system speeds test and repair of redundant memories," *Electronics*, vol. 57, pp. 175-179, Jan. 12, 1984.
[11] C.-L. Wey and F. Lombardi, "On the repair of redundant RAMs," *IEEE Trans. Computer-Aided Design*, vol. 6, pp. 222-231, Mar. 1987.
[12] J.J. Hopfield and D.W. Tank, " 'Neural' computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52, pp. 141-152, July 1985.
[13] Y. Shrivastava, S. Dasgupta, and S.M. Reddy, "Guaranteed convergence in a class of Hopfield networks," *IEEE Trans. Neural Networks*, vol. 3, pp. 951-961, Nov. 1992.
[14] P. Mazumder and J.-S. Yih, "A new built-in self-repair approach to memory yield enhancement by using neural-type circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 124-136, Jan. 1993.
[15] J.J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Nat'l Academy of Science USA*, vol. 81, pp. 3,088-3,092, May 1984.

Hyperneural Network—An Efficient Model for Test Generation in Digital Circuits

Suresh Rai and Weian Deng

Abstract—This paper considers the problem of applying neural network for logic circuit testing and proposes an efficient method based on *hyperneural network* (HNN). The HNN uses an energy function that not only considers binary relations but also captures all higher order relations among N neurons. We illustrate the hyperneural concept using two formulations. First, a constraint engery function is defined and the gate model is obtained. Second, the Hopfield network is reformulated to generate the gate level hyperneural model. The gate level HNNs are used to give a mathematical form to the digital circuit that, in turn, requires optimization techniques to solve the test generation problem. We have used ISCAS'85 benchmark circuits to illustrate the method. Results are compared with those obtained from PODEM, MODEM, and FAN.

Index Terms—Hyperneural model, logic circuit testing. neural network, NP-completeness, optimization, pseudo-Boolean programming, satisfiability problem.

1 Introduction

A Boolean difference (BD) approach defines a complete set of tests for a given fault and, thus, offers a unique advantage over path sensitization techniques [4], [9], [10]. It was disfavored because of the difficulty in the manipulation of algebraic forms using computers. Currently, reformulating BD between the unfaulted and faulted circuits using pseudo-Boolean programming [1], [5], Boolean satisfiability and implication graph [11], and neural network [3], [8] has renewed interest in it. They also help make the BD problem machine computable.

This paper extends the concept of neural networks (NN) modeling of digital circuits for test generation. The NN are interconnections of *neurons* that are simple computing elements and are characterized by a pseudo-Boolean quadratic function, called the *energy function* [3], [6]. Chakradhar et al.[3] and Fujiwara [8] use Hopfield network and the *basis* NN set for two input gates. Hopfield network of N neurons describes only binary relations between neurons. Gates with more than two inputs need *hidden* neurons. Even two inputs XOR and XNOR gates require four neurons. Inclusion of an additional neuron doubles the search space. Finding a valid test set is, therefore, either increasingly hard or the network converges to an invalid solution. We propose here a new model, henceforth called *hyperneural network* (HNN), to overcome the difficulties with existing NN modeling techniques of logic gates. The proposed technique captures all (including binary) relations among N neurons. Recently, Ortega et al. [13] have independently defined a similar function, known as generalized Hopfield network (GHN). They use GHN and spectral coefficients in the design of the extra modules of a concurrent testable circuit.

The lay out of the paper is as follows: Section 2 provides the background needed in later sections. We discuss two formulations to present the concept of HNN in Section 3. Section 4 describes the technique. Section 5 provides experimental results for some typi-

• *The authors are with the Electrical and Computer Engineering Department, Louisiana State University, Baton Rouge, LA 70803. E-mail: suresh@gate.ee.lsu.edu.*

Manuscript received September 1993; revised February 1995.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number C95139.

0018-9340/96$05.00 ©1996 IEEE

A New Built-In Self-Repair Approach to VLSI Memory Yield Enhancement by Using Neural-Type Circuits

Pinaki Mazumder, *Member, IEEE*, and Yih-Shyr Jih, *Member, IEEE*

***Abstract*—As VLSI chip size is rapidly increasing, more and more circuit components are becoming inaccessible for external testing, diagnosis, and repair. Memory arrays are widely used in VLSI chips, and restructuring of partially faulty arrays by the available spare rows and columns is a computationally intractable problem. Conventional software memory-repair algorithms cannot be readily implemented within a VLSI chip to diagnose and repair these faulty memory arrays. Intelligent hardware based on a neural-network model provides an effective solution for such built-in self-repair (BISR) applications. This paper clearly demonstrates how to represent the objective function of the memory repair problem as a neural-network energy function, and how to exploit the neural network's convergence property for deriving optimal repair solutions. Two algorithms have been developed using a neural network, and their performances are compared with the repair most (RM) algorithm that is commonly used by memory chip manufacturers. For randomly generated defect patterns, the proposed algorithm with a hill-climbing (HC) capability has been found to be successful in repairing memory arrays in 98% cases, as opposed to RM's 20% cases. The paper also demonstrates how, by using very small silicon overhead, one can implement this algorithm in hardware within a VLSI chip for BISR of memory arrays. The proposed auto-repair approach is shown to improve the VLSI chip yield by a significant factor, and it can also improve the life span of the chip by automatically restructuring its memory arrays in the event of sporadic cell failures during the field use.**

I. Introduction

As THE VLSI device technological feature width is rapidly decreasing to the range of hundreds of nanometers, the chip yield tends to reduce progressively due to increased chip area, complex fabrication processes, and shrinking device geometries. In order to salvage partially faulty chips, redundant circuit elements are incorporated in the chips, and an appropriate reconfiguration scheme is employed to bypass and replace the faulty elements. In high-density dynamic random-access memory (DRAM) chips, redundant rows and columns are added to reconfigure the memory subarrays, where the rows or columns in which defective cells appear, are eliminated by using techniques such as electrically programmable latches or laser personalization. The problem of optimal reconfiguration and spare allocation has been widely studied by many researchers [1], [9], [10], [14], [15], [29]. A review of these algorithms for memory diagnosis and repair can be seen in [4]. But all of these algorithms cannot be readily applied to embedded arrays, where they are neither controllable by external testers, nor are their responses readily observable. Built-in self-test (BIST) circuits are commonly used to comprehensively test such embedded arrays and bad chips are discarded when they fail to pass the relevant test procedures. In order to salvage the partially faulty chips, new built-in self-repair (BISR) circuits must be developed for optimal repair of the faulty memory arrays.

In this paper, a novel self-repair scheme is proposed that uses a built-in neural network to determine how to automatically repair a faulty memory array by utilizing the spare rows and columns. Two algorithms have been proposed to illustrate how a neural network can solve random defects, and their performances are compared with the conventional software repair algorithms, such as *Repair Most* [28]. The Repair Most (RM) algorithm is a greedy algorithm that iteratively assigns a spare row (column) to replace the row (column) which currently has the maximum uncovered cells until all the cells are covered or bypassed (providing a successful solution), or all the spare rows and columns are exhausted (failing to give a repairable solution, if all faulty cells are not bypassed), whichever occurs first. This algorithm is selected for comparison because it is relatively simple and can be implemented by digital hardware, unlike other software algorithms, such as simulated annealing or approximation algorithms. A hardware version of simulated annealing will require a Gaussian white noise generator to generate the random moves and a logarithmic circuit to schedule the temperature. The other heuristic-based algorithms (like FLCA, LECA, and BECA [15], and branch-and-bound [14]) will require microcode-driven complex digital circuits for solving the problem by hardware. The performance of neural algorithms has been compared with

Manuscript received May 1, 1990; revised January 16, 1992. This work was supported in part by the National Science Foundation under Grant MIPS 9013092, by the Office of Naval Research under Grant 00014-85-K-0122, and by the U.S. Army URI Program under Grant DAAL 03-87-K-0007. This paper was recommended by Associate Editor F. Brglez.

P. Mazumder is with the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

Y.-S. Jih was with the University of Michigan, Ann Arbor, MI. He is now with IBM T.J. Watson Research Center, Yorktown Heights, NY.

IEEE Log Number 9200909.

0278-0070/93$03.00 © 1993 IEEE

RM by running these algorithms with repairable fault patterns and examining what percentages of these fault patterns can be repaired by them. The simulation studies made in this paper demonstrate that a neural network using a hill-climbing (HC) capability provides a fast and good solution for a repairable array. The technique is implemented within the chip by an electronic neural network, and it promises to be a potential application area for neural networks.

Historically, the memory repair problem dates back to the evolution of 64 Kbit DRAM's in the late 1970's [25], when, to improve the chip yield, two or three extra rows and extra columns were added to each quadrant consisting of 64 Kbit subarray of memory cells. Simple greedy [2], [28] and exhaustive search [1] algorithms were developed to repair the faulty memory subarrays. Since the search space was very small for such problem sizes, all these algorithms provided high throughput in memory diagnosis and repair. But as the memory size has increased to several megabits over the last few years, the defect patterns have become sufficiently complex and the search space has grown extensively. The problem of memory repair has been shown to be NP-complete [9], [14] by demonstrating that the repair problem is transformable in polynomial time to the Constrained Clique problem in a bipartite graph. A number of heuristic algorithms, such as branch-and-bound [14], approximation [14], best-first search [10], and others [15], [29], recently have been proposed to solve the memory array repair problem. The two key limitations of these algorithms are a) that their worst-case complexities are nearly exponential and b) they are not readily implementable within the chip for BISR. These algorithms are generally written in a high-level programming language and are executable on general-purpose computers. This paper addresses two fundamental aspects of the memory repair problem: how to devise efficient algorithms so that the overall production throughput improves along with the chip yield, and how to generate such repair algorithms in hardware so that they can be applied to repairing memories, embedded within the VLSI chips. The contribution of this paper is to demonstrate how to solve the array repair problem by using neural networks, and how to implement a BISR scheme in hardware. A neural network can produce solutions by the collective computing of its neuron processors with far faster speed than the abovementioned sequential algorithms running on conventional computers. Since these neural processors are simple threshold devices, the basic computing step of neural nets is comparable to the on–off switching of a transistor [7], [8], [22], [24]. Another potential advantage of using neural networks is that they are robust and fault-tolerant in the sense that they may compute correctly even if some components fail and/or the exciting signals are partially corrupted by noise. Thus the reliability of the self-repair circuit using electronic neural networks is very high. This has been experimentally verified in Section VI.

This paper has been organized as follows: Section II provides a brief overview of the neural network model and its dynamic behavior. Section III provides a formal framework for the memory repair problem using the concepts of neural net computation. Two algorithms are developed by programming the synaptic weight matrix of the neural net. The first algorithm is greedy, and starting from any random configuration (i.e., arbitrary neuron states), it can solve the problem by monotonically reducing the energy function of the neural net. The second algorithm uses the HC technique to yield a near-optimum solution. Section IV gives the simulation results of the neural net algorithms. From the simulation experiments, it is seen that the neural net algorithms are superior to the RM algorithm. An electronic implementation of neural networks is demonstrated in Section V. The intrinsic fault-tolerant ability of a neural net is studied in Section VI. The yield improvement and hardware overhead caused by the neural-net self-repair circuitry are examined in Section VII.

II. Neural Net Computation Model

The earliest mathematical formalization of the human nervous system can be found in the work by McCulloch and Pitts [19], where it was modeled as a set of *neurons* (computation elements) interconnected by *synapses* (weighted links). Each neuron in the model is capable of receiving impulses as input from and firing impulses as output to potentially all neurons in the network. The output function of a neuron depends on whether the total amount of input excitation received exceeds its predetermined threshold value θ, or not. The state of neuron i is represented by an all-or-none binary value s_i, with $s_i = 0$ being a nonfiring condition and $s_i = 1$ denoting an impulse-firing condition. Note that one may represent a nonzero threshold by applying an external impulse bias b_i to the neuron i with $b_i = \theta_i$.

Interactions between two different neurons occur through a synapse serving as a duplex 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 a firing neuron i produces w_{ji}, amount of influence to 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:

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

Actually, in the human nervous system or any other biological implementations, neural networks make smooth continuous transitions when the neurons change their states. Therefore, we assume that neurons which receive a larger influence in magnitude will have a faster state

transition. Neuron processors are expected to operate in an asynchronous and random fashion.

Following the description of the behavior of an individual neuron, let us look at the overall network state transition behavior. First, a network state is denoted by the vector of neuron state variables. Given a network state vector $X = (x_1, x_2, \cdots, x_N)$, 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 concern 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. First, let us consider the case of an excitatory or positive input 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 negative energy, since by convention, a system is more stable when the energy level is lower. Likewise, an inhibitory or negative input to a neuron in the nonfiring state should also be considered as negative energy. On the other hand, positive energy is created if a firing neuron receives an inhibitory input or a nonfiring neuron receives an excitatory input, since the network is potentially destabilized by the input.

If $w_{ij} = w_{ji}$ for all i's and j's, the total energy E^{NN}, commonly known as the Lyapunov function [11], is given by

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

Note that the change of state by neuron i will result in an absolute decrease of $\Sigma_j w_{ij} s_j + f(s_i) b_i$ in energy, where $f(s_i) = 1$ if $s_i = 0$, and -1 otherwise. Together with the fact that the total energy is bounded, the network is guaranteed to reach a local minimal energy fixed point [12].

III. Neural Network Solutions

The idea of using a neural network to tackle combinatorial optimization problems was first proposed by Hopfield [12], who designed and fabricated a neural network for solving the classic *Traveling Salesman Problem* [5]. The objective (cost) function of a combinatorial optimization problem can be represented by Lyapunov's energy function of the neural network, and the network's property of convergence from a random initial state to a local minimal energy state can be utilized to reduce the cost function of the combinatorial problem. However, the neural computing approach given in [12] leads only to the design of a greedy gradient descent (GD) algorithm which stops searching at the first local minimal solution encountered; consequently, the neural network solution is generally of low quality.

In this section we study the convergence behavior of neural networks. We demonstrate that more powerful searching strategies can be developed by modifying the simple GD algorithm. By programming the neural network in an appropriate way, we find that the neural nets are capable of searching the solution by HC or tunneling. Thus, the probability of obtaining a globally optimal solution by using neural networks is very high.

A. Array Repair Problem Representation

Assume a memory array of size $N \times N$ with exactly p spare rows and q spare columns that can be utilized to replace the defective rows and columns. Assume an arbitrary fault pattern in which faulty cells randomly occur on m ($< N$) distinct rows and n ($< N$) distinct columns of the memory array, such that the compacted subarray of size $m \times n$ contains all relevant rows and columns which have at least one faulty cell. Let the matrix $D = \{d_{ij}\}$ of size $m \times n$ characterize the status of the memory cells, such that d_{ij}, which corresponds to the cell at row i and column j, is 1 if the cell is faulty, and $d_{ij} = 0$, otherwise. Normally, very few cells in the array are faulty, and the characteristic matrix, D, is highly sparse. A row repair scheme can be represented as a vector U of m bits, such that $u_i = 0$ if row i is to be replaced, otherwise $u_i = 1$, $0 \le i \le m - 1$. A column repair scheme is defined in the same fashion, and is denoted by a vector V of n bits. The numbers of 0's in U and V must be less than or equal to p and q, respectively. The memory is said to be repairable if the above constraints are satisfied, and essentially the repair problem is how to determine a pair of U and V, such that $U^T D V = 0$.

In addition to the characteristic matrix D, let the overall status of row i and column j of the array be characterized by r_i and k_j, respectively,

$$r_i = \begin{cases} 1, & \text{if row } i \text{ contains defective elements} \\ 0, & \text{otherwise} \end{cases}$$

$$k_j = \begin{cases} 1, & \text{if column } j \text{ contains defective elements} \\ 0, & \text{otherwise.} \end{cases}$$

To design a neural net for the memory repair problem, a neural net of size $M = m + n$ is used, where the states of the first m neurons are denoted by s_{1i}'s, $1 \le i \le m$, and the states of the remaining n neurons by s_{2j}'s, $1 \le j \le n$. For ease of interpretation, the first m neurons are addressed as *row neurons*, and the remaining n neurons as *column neurons*. The physical meaning of these neuron states are defined as follows:

$$s_{1i} = \begin{cases} 1, & \text{if row } i \text{ is suggested for replacement} \\ 0, & \text{otherwise} \end{cases}$$

$$s_{2j} = \begin{cases} 1, & \text{if column } j \text{ is suggested for replacement} \\ 0, & \text{otherwise.} \end{cases}$$

The cost function C^{MR} of the memory repair problem should represent both the nonfeasible repairing and incomplete coverage schemes by higher costs. These two

aspects can be modeled by the following two expressions:

$$C_1 = A/2\left[\left(\sum_{i=1}^{m} s_{1i}\right) - p\right]^2 + A/2\left[\left(\sum_{j=1}^{n} s_{2j}\right) - q\right]^2$$

$$C_2 = B/2\left[\sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij}(1 - s_{1i})(1 - s_{2j}) + \sum_{j=1}^{m}\sum_{i=1}^{n} d_{ji}(1 - s_{1j})(1 - s_{2i})\right]$$

so that the overall cost function is represented as an algebraic sum of C_1 and C_2. In the above expressions, the values of A and B are empirically decided and as shown in Section IV, their relative values are critical for obtaining the appropriate percent of successful repairs.

The expression for C_1 encourages exactly p spare rows and q spare columns to be used in the repair scheme. For those nonfeasible schemes that require more than the available spare rows and/or columns, the cost will increase quadratically. If necessary, the optimum usage of spares can be experimentally attempted by repeating the search operation with successively reduced amounts of available spares (i.e., progressively decrementing the value of p and/or q in the expression for C_1), until no successful repair scheme can be found.

The expression for C_2 monotonically decreases as more and more defective cells are covered by the spare rows and columns. If all defective elements are covered by at least one spare row or column, the above expression is then minimized to zero. Unsuccessful repairing schemes which fail to cover all defective elements will yield positive costs. The two double summation terms in C_2 are equivalent. As it will be seen later in the expression for the synaptic weight matrix, this duplication facilitates the later transformation to neural net energy functions.

Another cost function can be defined by modifying the C_1 expression, i.e.,

$$C_1' = A/2\left(\sum_{i=1}^{m} s_{1i}\right)^2 + A/2\left(\sum_{j=1}^{n} s_{2j}\right)^2.$$

Note that C_1' is actually a simplified C_1 with p and q set to zero. Without considering whether all defects will be covered or not, the expression for C_1' prefers repair schemes with fewer spares used. Intuitively, this way more efficient repair schemes may be encouraged. But the new $C_1' + C_2$ combination has a potential danger of having a large amount of unsuccessful repair schemes with a small number of uncovered defects mapped to local minima just because they request less spares. The differences between the neural networks defined by these two cost functions will be further illustrated later in the simulation studies.

B. The Gradient Descent (GD) Approach

It may be recalled that the neural network energy function is $E^{NN} = -1/2\ \Sigma_i\ \Sigma_j w_{ij} s_i s_j - \Sigma_i s_i b_i$. By rewriting C^{MR} in the form of E^{NN}, the weights of the interconnecting synapses and the intensities of the external biases to neurons can be determined through simple algebraic manipulations.

But in order to keep the main diagonal of the neural net's synaptic weight matrix zero, all $A/2 \cdot \Sigma s_i^2$ terms in the array repair cost expression are rewritten as $A/2 \cdot \Sigma s_i$. Since $s_i \in \{0, 1\}$, these two terms are mathematically equivalent. However, $A/2 \cdot \Sigma s_i^2$ corresponds to self-feedback through a synapse of strength $-A$ for all neurons, and $A/2 \cdot \Sigma s_i$ means a bias of $-A/2$ for every neuron. The resulting neural network is shown as follows:

$$w_{1i,1j} = -A(1 - \delta_{ij}) \qquad w_{2i,2j} = -A(1 - \delta_{ij})$$

$$w_{1i,2j} = -B \cdot d_{ij} \qquad w_{2i,1j} = -B \cdot d_{ji}$$

$$b_{1i} = (p - 1/2) \cdot A + B\Sigma_j d_{ij}$$

$$b_{2j} = (q - 1/2) \cdot A + B\Sigma_i d_{ij}$$

where $\delta_{ij} = 0$ if $i \neq j$, otherwise 1.

In order to illustrate how to construct the synaptic weight matrix from these values, let us consider a pathological defect pattern that consists of 16 faulty cells as shown in Fig. 1. If there are only four spare rows and four spare columns, a greedy algorithm such as RM cannot produce a successful repair scheme [1]. The only successful allocation of spares for replacement is indicated in the figure by horizontal and vertical stripes. But due to the presence of four defects in both row 1 and column 9, the RM algorithm will be tempted to make the first two repairs on them. The algorithm will thereby fail to repair the memory because, in addition to the available six spare rows and columns, the algorithm will require two more columns or rows to cover all the defects along the diagonal direction. The neural network algorithm described in this section can repair successfully such a defect pattern by finding the unique solution shown in Fig. 1. For such a memory fault pattern, the neural net synaptic weights will be estimated by adjusting $A/B = 1$ (the value $A = 2$ has been chosen empirically and is not critical for the performance of the algorithm) in the above expressions. The resulting neural net synaptic weight matrix and the biases are given in Fig. 2.

C. The Hill-Climbing (HC) Approach

We propose here a neural net with simplified HC behavior that provides high-quality solutions similar to the powerful combinatorial technique, called *Simulated Annealing* [13]. First, let us note that the solutions to the memory array repair problem are classified as *success* and *failure*. In fact, it is commonly known that all optimization problems can be transformed to a series of yes/no questions [5]. Due to this specific criterion, we do not consider any random move that may increase the energy of the system until the move is absolutely necessary, i.e., the search has reached an unsuccessful local energy minimum. If the current local minimum does not yield a successful repair, moves that increase the system energy are

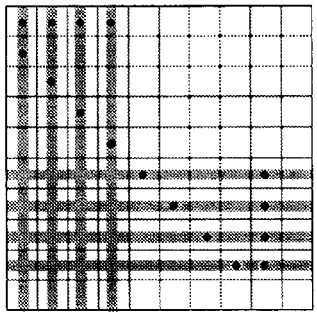


Fig. 1. A repairable faulty array with four spare rows and four spare columns.

$$W = \begin{bmatrix}
0 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2 & 0 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2 & -2 & 0 & -2 & -2 & -2 & -2 & -2 & -2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2 & -2 & -2 & 0 & -2 & -2 & -2 & -2 & -2 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2 & -2 & -2 & -2 & 0 & -2 & -2 & -2 & -2 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2 & -2 & -2 & -2 & -2 & 0 & -2 & -2 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & -2 & 0 \\
-2 & -2 & -2 & -2 & -2 & -2 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & -2 & 0 \\
-2 & -2 & -2 & -2 & -2 & -2 & -2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -2 & 0 \\
-2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 \\
-2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 \\
-2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -2 & -2 & -2 & -2 & -2 & -2 & -2 \\
-2 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 & -2 & -2 & -2 & -2 & -2 & -2 \\
-2 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & -2 & 0 & -2 & -2 & -2 & -2 & -2 \\
0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & -2 & -2 & -2 & 0 & -2 & -2 & -2 & -2 \\
0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & -2 & -2 & -2 & -2 & -2 & 0 & -2 & -2 & -2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & -2 & -2 & -2 & -2 & -2 & -2 & 0 & -2 & -2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & 0 & -2 \\
0 & 0 & 0 & 0 & 0 & -2 & -2 & -2 & -2 & 0 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & 0
\end{bmatrix}$$

b's = 15, 9, 9, 9, 9, 11, 11, 11, 11, 7, 11, 11, 11, 11, 9, 9, 9, 9, 15, 7.

Fig. 2. A *GD* neural net.

allowed by providing negative synaptic feedback to neurons themselves. The basic idea is that when the search has reached an unsuccessful local minimum, we can force the neural net to make a move by turning on a neuron, which appears to enter a lower energy state but, in fact, will increase the system energy due to the negative self-feedback. Next, the system is expected to turn off some other neuron to enter a new lower energy state, thus escaping the local minimum energy trap. Notice that with very low probability, it is possible for a neuron to fall into a loop of alternate *on* and *off* states, as it is similarly possible for simulated annealing to cycle through random moves. Consequently, network behavior is not admissible in the sense that it is guaranteed to converge to an optimal solution state. As in simulated annealing, which uses a predefined criterion to break the inner loop of optimization, a suitable timeout mechanism is necessary for HC neural nets to prevent excessively long searches.

We being the description of the new neural net by giving the synaptic weights and biases as follows:

$$w_{1i,1j} = -A; \qquad w_{2i,2j} = -A$$

$$w_{1i,2j} = -B \cdot d_{ij}; \qquad w_{2i,1j} = -B \cdot d_{ji}$$

$$b_{1i} = p \cdot A + B\Sigma_j d_{ij}; \qquad b_{2j} = q \cdot A + B\Sigma_i d_{ij}.$$

To examine the neural net, let us assume that $A \neq 0$ and $B = 0$. Thus, each row neuron will receive a positive $p \cdot A$ amount of bias, and each column neuron will receive a positive $q \cdot A$ amount of bias. Also, due to $B = 0$, the row neurons are independent of the column neurons, i.e., there are no synaptic connections between the two groups. Suppose there are $p' < p$ row neurons that are in firing states; then each row neuron will receive a positive $(p - p') \cdot A$ amount of influence, and all of them will be encouraged to fire. Similarly, if $p' > p$, all row neurons will receive a negative $(p - p') \cdot A$ amount of influence, and all will tend to turn off.

Next, let us assume that $A = 0$ and $B \neq 0$. In this case, the amount of influence received by row neuron i is given by

$$B\left(\sum_j d_{ij} - \sum_j d_{ij} s_{2j}\right)$$

i.e., B times the number of defective elements in row i that would be covered exclusively by a replacement. Those defective elements in row i that are currently covered by some column repairs will not contribute any urgency to force replacement of this row.

Consider the situation where all the spares have been utilized (hypothetically) for repair, but there is still a single defect left uncovered by the current repair scheme. At this point, the number of defects inside a row or column is the only influence on neurons. For the row and column that contains this yet uncovered defect, a positive total influence of B will be received by the two corresponding neurons. After this defect is covered by turning on, say, the row neuron, which causes the use of one more spare row than allowed, all the row neurons will now receive an additional negative influence of A due to the extra spare suggested. If we choose to have $B > A$, then the neural

net will be stuck at the present state, making the repair scheme unsuccessful. On the other hand, if we have $A > B$, all the current row spares will cover only one faulty element exclusively will now receive a net negative influence equal to $|B - A|$, thus causing the network to switch to a new state and give up on the current scheme.

IV. Simulation Studies

In this section, simulation results are provided to demonstrate the superiority of the proposed neural computing approach for the problem of automatic array repair. Six hundred *reduced array fault patterns* of size 10×10 (Case 1) and 20×20 (Case 2) randomly generated arrays with about 10, 15, and 20% faulty elements were used in the experiments. As was explained in Section III, these small size arrays represent the actual defective rows and columns in large-size memory arrays, some as large as a few million bits.

The performance of two GD neural nets *GD* and *GD'*, defined by cost functions $C_1 + C_2$ and $C_1' + C_2$, respectively, are compared. For both 10×10 and 20×20 arrays, the probabilities of finding a successful repair scheme versus the ratio B/A are depicted in Fig. 3. By controlling the value of B/A, the importance of the fault coverage over the spare usage can be manipulated.

According to the figure, the effectiveness of *GD'* is largely affected by how A and B are selected. When A is set equal to B, no successful repair scheme can be found by *GD'* for any one of the repairable fault patterns, just as was expected in Section III. For 10×10 arrays, the percentage of successful repairs improves to about 50% as B/A increases to 5, and thereafter converges to about 42%. *GD'* behaves similarly for 20×20 arrays, except that the peak performance happens at about $B/A = 12$. Now, let k be the maximum of p and q, where p and q are the maximum numbers of spare rows and columns allowed to form a successful repair scheme, respectively. The B/A ratio for a peak value to occur is found to be equal to k. Note that each firing row (column) neuron will send a $-A$ amount of influence to disable all other row (column) neurons, and a row or column neuron is encouraged to fire by as low as a $-B$ amount of influence to cover faults in a row or column. In order to allow the neural net to explore solutions using up to k spare rows and k spare columns, we must have $B \leq kA$ to keep up to k row or column neurons firing at the same time.

On the other hand, *GD* shows the advantage of its low sensitivity to how A and B are selected over the range of $B/A > 1$. In a hardware implementation of a neural network chip, the ratio of B/A is likely to vary over a wide range because of processing parameter variations and the wide tolerance of resistive elements. Thus the performance of the neural network will remain uniform from chip to chip if *GD* is implemented as opposed to *GD'*, whose behavior changes dramatically with different values of B/A. The percentage of successful repairs obtained by *GD* appears to have a peak value of about 50% at $A = B$ and declines asymptotically to about 42% as B/A increases. It is not surprising that the percentages of successful repairs obtained by *GD'* and *GD* are converging to the same value, since when B is much larger than A, the consideration for the spare usage is of little or no effect, compared with the consideration for fault coverage.

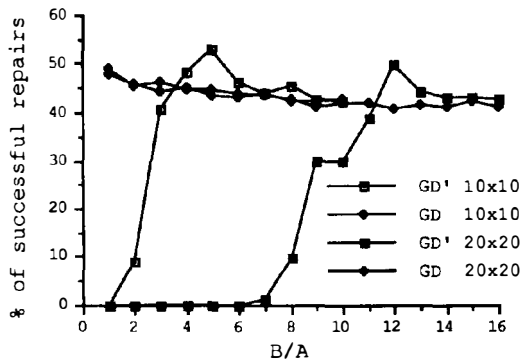


Fig. 3. Performance comparisons between *GD* and *GD'*.

For the second experiment, the effectiveness of the RM algorithm is compared with *GD*. In order to provide an equal starting situation, the programmed neural net is started with all neuron processors in nonfiring states instead of other random combinations. We will use *GD*-zero to denote this special case of the GD convergence. The performances of these two algorithms are compared in Table I. Random defect patterns are generated for both cases, and the algorithms are applied to repair the faulty arrays with three different sets of available spares. From the results it was seen that on average, *GD*-zero is twice as successful as RM when the defect pattern is not very large (represented by Case 1) and few spare rows and columns are used. But *GD*-zero is about three to five times more successful when the defect pattern is large (represented by Case 2) and a relatively large number of spare rows and columns are used. As for the number of steps taken to find a repair scheme, it is about the same for both algorithms.

Second, tradeoffs between the use of two neural computing methods are examined, and the results are shown in Table II. For each defect pattern, a number of random trials are performed by each method. Average performance reaches consistency within one hundred random trials. As is expected, the simulation indicates that the HC approach is almost perfect in locating a successful repair scheme for repairable arrays due to its ability to perform global searches. The probability for the *GD* to reach a successful repair in a random trial is about 0.5. The average number of steps needed by HC is about two to three times that needed by *GD*. The runtime for the *GD* algorithm varies very little over a large number of experiments, but the HC algorithm is found to have a wide variance in the number of steps to execute the search successfully.

The chances of getting a successful scheme achieved by the four methods are shown in Fig. 4 as percentages.

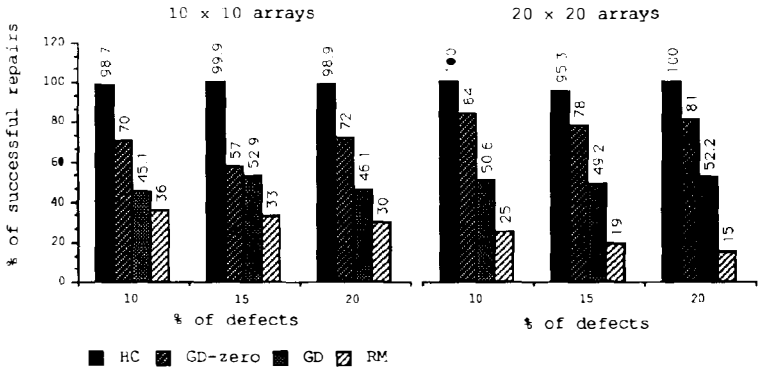


Fig. 4. Performance summary of the four repair methods.

TABLE I
AVERAGE % OF SUCCESSFUL REPAIRS DONE BY RM AND *GD*-ZERO

% of Defects	Case 1 Spares	Case 1 RM	Case 1 *GD*-zero	Case 2 Spares	Case 2 RM	Case 2 *GD*-zero
10	(3, 3)	36	70	(9, 9)	25	84
15	(3, 4)	33	57	(9, 12)	19	78
20	(4, 5)	30	72	(12, 12)	15	81

TABLE II
PERFORMANCE COMPARISONS BETWEEN GD AND HC NEURAL NETS

	% of Defects	Spares	% of Success GD	% of Success HC	Avg # of Steps GD	Avg # of Steps HC	σ GD	σ HC
Case 1	10	(3, 3)	45.1	98.7	7.0	14.4	2.0	11.5
	15	(3, 4)	52.9	99.9	6.6	14.0	2.0	10.4
	20	(4, 5)	46.1	98.9	6.6	22.6	1.9	21.8
Case 2	10	(9, 9)	50.6	100.0	13.0	22.6	2.6	13.1
	15	(9, 12)	49.2	95.3	12.6	40.2	2.7	25.3
	20	(12, 12)	52.2	100.0	12.7	20.4	2.6	16.8

It can be seen that for 10 × 10 arrays HC achieves almost 100% success in all cases. The *GD*-zero performs better than *GD* where the neurons are randomly turned on at the initiation of the algorithm. Greedy techniques such as RM fail to cover more than 67% cases on the average. For 20 × 20 arrays, RM's performance deteriorates, and on average in more than 80% of the cases, it fails to repair the memory.

To summarize the simulation results, it may be pointed out that *GD* is fast in reaching a spare allocation, and has small variance in the number of search steps. On the other hand, the number of search steps used by the HC can vary widely, depending on the initial random starting states and how difficult the problem instances are. But on average, the number of search steps needed by HC does not escalate exponentially. For four out of six types of settings, the average number of search steps for HC is limited to about twice the number for *GD*. For the other two types of settings, the ratios of the average number of search steps are no worse than four. In fact, if we compare the *equivalent average number of search steps* expected for the GD approach to attain the same level of success as the HC approach does within 1%, the HC approach is found to be more efficient. The actual numbers can be seen in Table III.

TABLE III
GD AND HC IN AVERAGE NUMBER OF SEARCH STEPS

% of Defects	Case 1 GD	Case 1 HC	Case 2 GD	Case 2 HC
10	49.0	14.4	91.0	22.6
15	39.6	14.0	63.0	40.2
20	46.2	22.6	88.9	20.4

V. ELECTRONIC IMPLEMENTATION

In this section, we demonstrate how to implement an electronic neural network that can repair a faulty memory by using the HC searching strategy. Electronic neural nets can be built by both analog and digital techniques. Murray and Smith have discussed their tradeoffs and devel-

oped a digital neural net representing the interactions by pulse stream of different rates [22]. Even though the digital network has better noise immunity, it requires large silicon area for its pulse generators and digital multipliers. Analog neural nets using current summing principles were developed by Graf *et al.* [7] and Sivilotti *et al.* [24]. Carver Mead's book on analog VLSI presents the design strategies for several types of analog neural systems [20]. These analog neural nets require small area and their intrinsic robustness and ability to compute correctly even in the presence of component failures, are particularly useful features for large-scale VLSI implementation.

In this design, a neuron is realized as a difference amplifier which produces binary output voltages. The firing neuron state is represented by the high voltage output, and the nonfiring state by the low voltage output. The synaptic influences between neurons, as well as the biases to neurons, are represented by electrical currents. Thus, an excitatory influence propagated to a neuron can be realized by injecting current to the corresponding amplifier input. Similarly, an inhibitory influence propagated to a neuron can be realized by sinking current from the corresponding amplifier input. As for the synaptic amplification, it is simulated by the branch resistance regulating the amount of current that passes through.

Note that for the array repair spare allocation problem discussed here, all the synaptic weights of the interconnection matrix are negative. Moreover, between two connected neurons, the synaptic weight of the connecting link can only be one of two different values, $-A$ or $-B$. If we assume that there are equal numbers of row and column neurons, an example electronic neural net with eight neurons can be implemented, as shown in Fig. 5.

According to Fig. 5, a neuron fires a negative influence to another neuron by turning on a transistor which sinks current from the target neuron's amplifier input. The amount of current involved is controlled by the size of the transistor. Note that owing to the assumption that equal numbers of neurons are reserved for row and column repairs, we are able to divide the interconnection matrix into four equal parts. It is not hard to find that only the first and third quadrants, as indicated in Fig. 5, need to be programmed based on the variable memory array defect pattern. For each programmable link, an additional transistor controlled by a memory cell is added to the pull-down transistor. A programmable link can be disconnected (connected) by storing a 0 (1) in the memory cell.

Fig. 6 shows the essential steps in programming the neural net's interconnection network. Let the memory array defect pattern be given in Fig. 6(a), with faulty cells represented by black dots. Next, by deleting fault-free rows and columns, a compressed defect pattern is obtained in Fig. 6(b), with a faulty cell denoted by a 1. Then the compressed defect pattern is used to program the first quadrant of the neural net interconnection matrix, and the third quadrant is programmed by the transposed compressed defect pattern, as is shown in Fig. 6(c).

Finally, the possibility of building the neural network

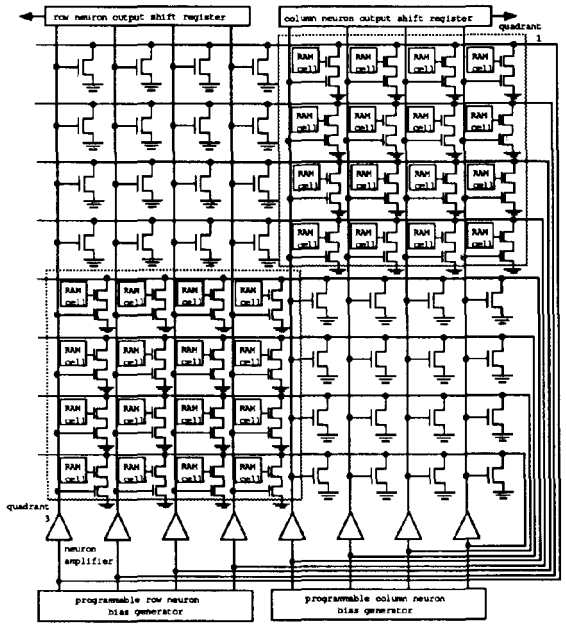


Fig. 5. An example electronic neural net for memory repair

alongside an embedded memory to achieve this self-repair purpose is demonstrated by the schematic diagram shown in Fig. 7.

The design to be discussed assumes that a built-in tester is available to provide fault diagnosis information. First of all, the status of each memory row and column is determined after the testing is done, and this information is stored in the faulty-row-indicator shift-register (FRISR) and faulty-column-indicator shift-register (FCISR), with a faulty row or column indicated by a 1. Then, to compress the memory array defect pattern, the detailed defect pattern of each faulty row is provided to the row-defect-pattern shift-register (RDPSR) one row at a time. As mentioned in Section III, the characteristic matrix D is highly sparse, and the fault pattern can be stored in the form of a compressed matrix each row and column of which will contain at least one faulty element. This is obtained by shifting those bits of RDPSR that correspond to the nonzero bits of FCISR, into the compressed-row-defect-pattern shift-register (CRDPSR). The content of CRDPSR is then used to program a row (column) in the first (third) quadrant of the neural net's interconnection network. The row (column) address of the quadrant is obtained by counting the nonzero bits in the FRISR. The stimulating biases to the row and column neuron inputs are generated by counting the total number of faults in each row (column) in the row (column) fault-count shift-registers. After a repair scheme is obtained by reading the outputs of the neuron amplifiers, the information is expanded in reverse order to the compression of defect patterns, and passed on to control logic of actual reconfiguration circuits. Normally, laser zapping [23], [3], focused ion-beam [21], or electron beam [6] techniques are used to restructure a

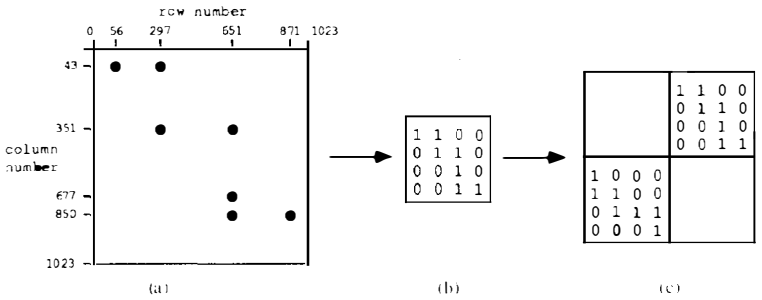


Fig. 6. Example programming of the neural net for memory repair. (a) Memory array defect pattern. (b) Compressed defect pattern. (c) Neural network RAM cell bitmap.

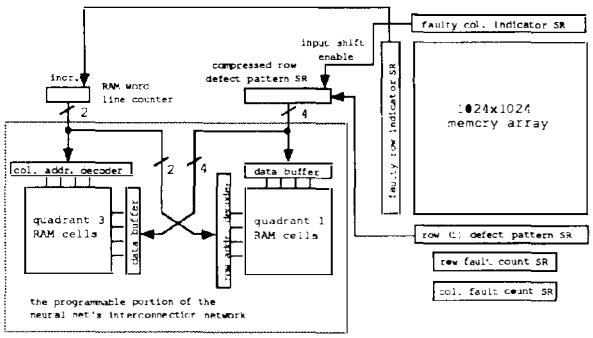


Fig. 7. Schematic of a neural net for embedded memory.

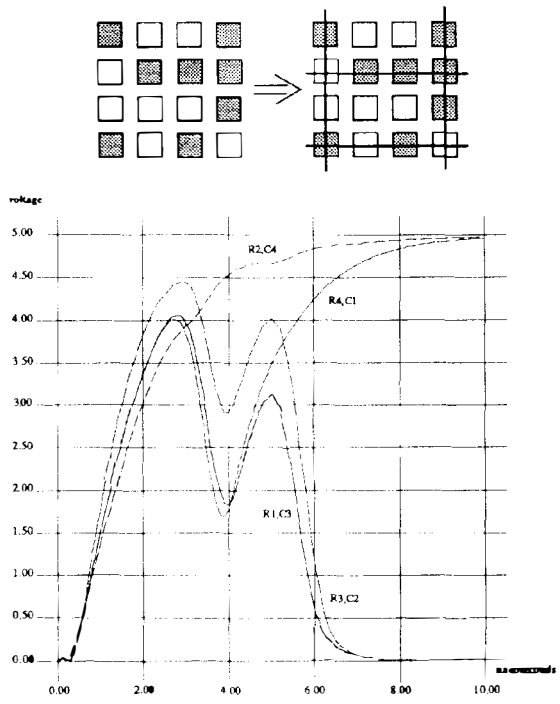


Fig. 8. Example memory repair SPICE simulation.

faulty memory array where the laser or charged beams are used for blowing out the programmable fuses to disconnect the faulty rows and columns. These schemes cannot be employed in automatic restructuring. Two viable techniques for self-restructuring are i) electronically programmable amorphous silicon fuses, which can be programmed by applying a 20-V pulse [27]; and ii) programmable electronic reconfiguration switches, which usually impose a small amount of penalties on circuit speed and area.

The circuit behavior was verified through SPICE simulations. Simulation output for a compressed 4 × 4 defect pattern with eight defective memory cells is shown in Fig. 8 as an example. The defects are represented by shaded squares. The initial state of the neural net was zero, i.e., there was no allocation of spares for any faulty rows or columns. Since every neuron represents either a faulty row or a faulty column in the compressed defect pattern, to cover the defects all neurons initially began transition toward the firing state. After 3 ns, the neurons representing row 2 and column 4 were successful in competition and remained in the firing state. This can be explained by the fact that row 2 and column 4 each have three defects to give the corresponding neuron the largest positive bias toward the firing state. All other neurons started to turn off, due to the mutual discouragement propagated through the negative synapses. When the remaining neurons were becoming low in activity, the mutual discouragement factor was also weakening, and the neurons started to move toward the firing state again. This time the neurons representing row 4 and column 1 were successful. The other four neurons then continued to the off state, since there was no remaining defect to cover and the spares were all used up.

VI. Neural Net Behavior in Fault Conditions

One unavoidable problem in adding extra hardware for BISR is that the extra hardware itself may be subject to component failures. In this section, we demonstrate the intrinsic fault-tolerant capability of neural networks as a reconfiguration control unit in the memory repair circuit.

Three types of component failures have been identified in neural networks, namely synapse-stuck faults, bias fluctuations, and neuron-stuck faults to serve as the fault model. For each faulty synapse, either of the synaptic

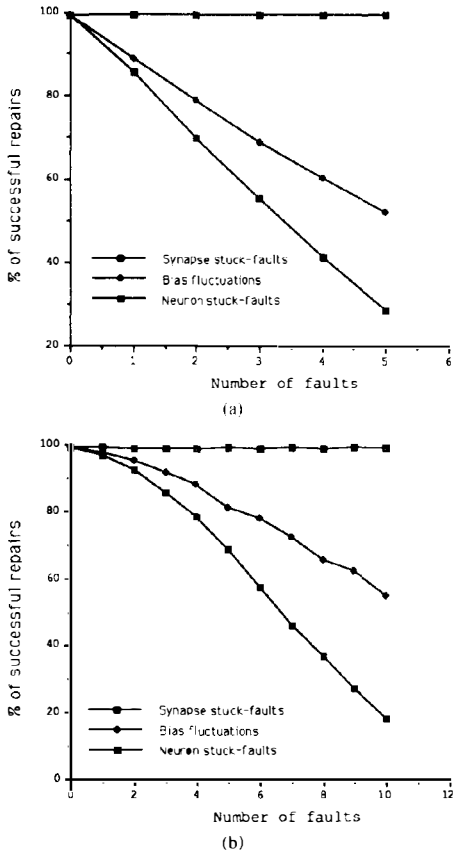


Fig. 9. Neural net performances in fault conditions

weights w_{ij} or w_{ji} can be assumed to be stuck-at-x, where x is a positive number in the range of synaptic strength values, due to transistor-stuck faults or defective memory cells that control the programmable synapses. Faulty bias generators are modeled to fluctuate within one unit of the predetermined biases, and faulty neurons will have stuck-at-firing or stuck-at-nonfiring states.

Rather than rendering the entire neural network useless, these faults will only reshape the energy (cost) distribution over the set of spare allocation schemes. Allocation schemes, which correspond to complete covers of all defective memory cells, may no longer be mapped to acceptable local energy minima, thus prolonging the search for an acceptable solution. On the other hand, incomplete allocation schemes may be mapped to false acceptable local energy minima, causing the neural net to stop searching without success.

Identical memory defect patterns prepared in Section IV are used here. Random faults are incrementally injected into a HC type of neural net to examine the achievable percentages of repairs for repairable defect patterns. To distinguish the degrees of seriousness among different types of faults, we inject only faults of the same type. The simulation results are shown in Fig. 9.

The results indicate that a small number of synapse-stuck faults, up to five in the 20-neuron network and ten in the 40-neuron network, have almost no effect on the average performance. But the bias fluctuation faults and the neuron-stuck faults cause the average percentage of repairs out of repairable defect patterns to decrease steadily to zero when over one-fourth of total neurons are affected. The difference between a bias fluctuation fault and a neuron-stuck fault is in the amount of influence given to a particular faulty row or column for repair. While a bias fluctuation fault will encourage or discourage the corresponding row or column substitution slightly by one unit of influence, a neuron-stuck fault will actually insist the substitution be made.

VII. Yield Analysis

In this section, a quantitative analysis of yield enhancement due to neural network's self-repair capability is done. Faults are injected into memory arrays, spares, and neural networks to compute the resulting yield. The overhead of the self-repair logic is also estimated.

A well-known yield formula due to Stapper [25], [26] is used here to calculate the original yield, Y_0, without the neural-net-controlled self-repair,

$$Y_0 = (1 + A\delta/\alpha)^{-\alpha}$$

where δ is the defect density, A is the memory array area, and α is some clustering factor of the defects. Let P_r be the probability function for a defect pattern to be repairable with respect to the remaining fault-free spares and the fault condition of the self-repair circuitry, and B be the area of overhead. Then, the yield, Y_N, with neural-net-controlled self-repair can be calculated as follows:

$$Y_N = Y_0' + (1 - Y_0')\,P_r,$$

where

$$Y_0' = [1 + (A + B)\delta/\alpha]^{-\alpha}.$$

By minor algebraic manipulation, it can be seen that

$$Y_0' = \left(\frac{Y_0}{1 + B\delta Y_0/\alpha}\right)^{\alpha}.$$

If $\alpha = 1$ then, the new yield formula can be simplified to

$$Y_N = P_r + \frac{(1 - P_r)\,Y_0}{1 + B\delta Y_0}.$$

We used 256 × 256 (64-Kbit) memory arrays for simulation with $\alpha = 1$ and $A\delta = 0.25, 0.5, 1, 2, 3, 4$, so that $Y_0 = 80\%, 66.7\%, 50\%, 33.3\%, 25\%$, and 20%, respectively. Six hundred memory arrays, each with at least one defect, are generated accordingly. For each set of memory arrays, up to four spare rows and four spare columns were provided to examine the neural net repairability with respect to the number of available spares. The spares and the neural net components are all subject to the same degree of defect density as the memory array. The resulting function values of P_r are shown in Fig. 10. Finally, the corresponding calculated yields are given in Fig. 11.

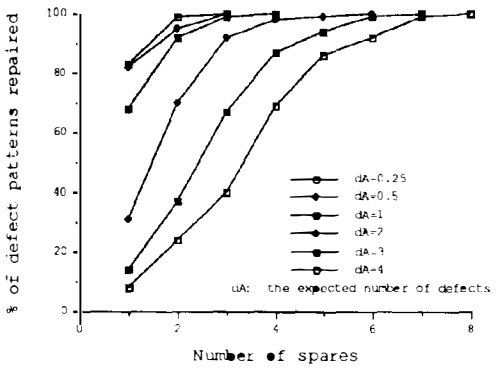


Fig. 10. Repairability of a defective 256 × 256 (64K) memory arrays.

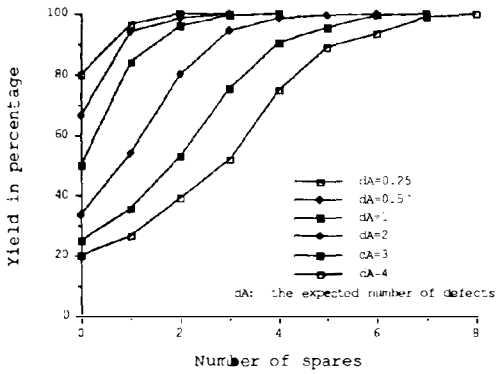


Fig. 11. Chip yield w.r.t. fault density and available spares.

According to the schematic diagram shown in Fig. 7, except for the FC/RISR and the RDPSR, which are proportional to the dimension of the memory array, the complexity of the remaining self-repair logic basically depends on the dimensions of the compressed defect patterns. For instance, an $m \times n$ compressed defect pattern will require a neural net of $m + n$ neurons. Since the mean and variance of the number of faulty columns (rows) can be calculated according to the fault distribution function, an estimate on the required neural net size can be easily made to accommodate nearly all possible compressed defect patterns. For the case of 256 × 256 (64K) memory arrays with $A\delta$ as high as 4, it is a near-certainty that the compressed array will not be larger than 10 × 10. Given an $N \times N$ DRAM, let the size of the maximum compressed memory defect pattern be n^2. An itemized account of the dynamic memory array and self-repair hardware based on transistor count is given in Table IV, and the percentages of overhead are listed in Table V for various memory and neural net sizes. As indicated by the results, the overhead is insignificant, compared with the yield improvement, and the overhead can be even smaller if static RAM's, with $6N^2$ transistors in the memory array, are considered. Here the overhead of the BIST circuit is not included. Typically this overhead is very low (about 28 flip-flops and ten gates for a 256-K RAM), as shown in Mazumder's earlier papers [16], [17].

TABLE IV
AN ITEMIZED ACCOUNT OF AN $N \times N$ RAM WITH BISR CIRCUITS

Component	Number of transistors
Memory cells	$4N^2$
Row decoder	$2N$
Column decoder	$2N$
Sense amp's	$6N$
Faulty column indicator SR	$8N$
Faulty row indicator SR	$8N$
Row defect pattern SR	$8N$
Row fault count SR	$8n \log n$
Column fault count SR	$8n \log n$
RAM word line counter	$8 \log n$
Compressed row defect pattern SR	$8n$
Programmable synapses SR's	$16n^2$
Non-programmable synapses	$2n^2$
Programmable synapses	$4n^2$
Neurons	$8n$
Bias generators	$2n(4 \log n + n)$

TABLE V
SELF-REPAIR HARDWARE OVERHEAD IN % OF RAM SIZE

	$n = 8$	$n = 12$	$n = 16$	$n = 20$
$N = 256$ (64 Kbit)	5.14	5.66	6.35	7.20
$N = 512$ (256 Kbit)	2.46	2.59	2.76	2.97
$N = 1024$ (1 Mbit)	1.20	1.23	1.28	1.33

VIII. FINAL REMARKS

The biological nervous system's ability to solve perceptual optimization problems is imitated here to tackle the VLSI array repair problem. In contrast to the current sequential repair algorithms, which run slowly on the conventional digital computers, the neural network's collective computational property provides very fast solutions. Methods to transform the problem instance into neural network computation model are demonstrated in detail. Of the two types of neural nets studied in this paper, the GD neural network has been found to be two to four times better than the RM algorithm in obtaining successful repair schemes. The GD minimizes the energy function of the network only in the locality of the starting energy value. The performance of the neural network is further improved by introducing an HC technique that allows the search to escape the traps of local minima. By generating random defect patterns and experimenting with a large number of arrays, it is seen that the HC algorithm finds a solution in a repairable memory array with near certainty (with a probability of 0.98 or more). For the same fault patterns, simple commercial algorithms, like RM, can yield feasible solutions for only 20% of the patterns. On the average, about twice as many search steps are used by the HC as opposed to GD.

Both the HC and GD neural networks can be implemented in hardware using very small overhead, typically less than 3% if the memory size is 256 Kbit or more. The payoff of this BISR approach is very high: the VLSI chip yield can increase from 10% without the BISR circuit to about 100% by using the proposed neural nets. The paper also proves that the neural networks are much more robust and fault-tolerant than the conventional logic circuits, and

thereby are ideally suited for self-repair circuits. The proposed designs of neural networks can operate correctly even in the presence of multiple faulty synaptic circuit elements, and as more numbers of neurons become permanently stuck to a firing or a nonfiring state, the networks gracefully degrade in their abilities to repair faulty arrays.

The paper shows how to solve a vertex-cover problem of graph theory, generally known to be an intractable problem, by using neural networks. A large number of problems in other domains, which can be modeled in similar graph-theoretic terms, can also be solved by the neural-network designs discussed in this paper. In the memory repair problem, an entire row or column is replaced to eliminate a defective cell. Such an approach is easy to implement and is cost-effective in redundant memory designs because the cells are very small in size. But a large class of array networks in systolic and arithmetic logic circuits employ a different strategy where a defective cell is exactly replaced by a spare or redundant cell. An appropriate graph model for such an array repair problem will be to construct a maximum matching of pairs between the vertices of a bipartite graph representing the set of faulty cells and the set of available spare cells. The technique described in this paper can be extended to solve the maximum matching algorithm by neural networks [18].

The overall goal of the proposed BISR circuits is to improve the chip yield by reconfiguring the faulty components at the time of chip fabrication, and also to extend the life span of the chip by automatically testing and repairing it whenever a failure is detected during the chip's normal operation. In space, avionics, and oceanic applications where manual maintenance, namely field testing and replacement, is not feasible, such an auto-repair technique will be very useful in improving the reliability and survivability of computer and communication equipment.

REFERENCES

[1] J. R. Day, "A fault-driven comprehensive redundancy algorithm for repair of dynamic RAMs," *IEEE Design and Test*, vol. 2, pp 35-44, June 1985

[2] R. C. Evans, "Testing repairable RAMs and mostly good memories," in *Proc. Int. Test Conf.*, Oct. 1981, pp. 49-55.

[3] B. F. Fitzgerald and E. P. Thoma, "Circuit implementation of fusible redundant addresses of RAMs for productivity enhancement," *IBM J. Res. Develop.*, vol. 24, pp. 291-298, 1980.

[4] W. K. Fuchs and M.-F. Chang, "Diagnosis and repair of large memories: a critical review and recent results," *Defect and Fault Tolerance in VLSI Systems*, I. Koren, Ed. New York: Plenum, 1989, pp. 213-225.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[6] P. Girard, F. M. Roche, and B. Pistoulet, "Electron beam effects on VLSI MOS: conditions for testing and reconfiguration," in *Wafer-Scale Integration*, G. Saucier and J. Trihle, Eds. New York: Elsevier, 1986, pp. 301-310.

[7] H. P. Graf and P. deVegvar, "A CMOS implementation of a neural network model," in *Advanced Research in VLSI*, pp. 351-367, 1987.

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

[9] R. W. Haddad and A. T. Dahbura, "Increased throughput for the testing and repair of RAMs with redundancy," in *Proc. Int. Conf. Computer-Aided Design*, 1987, pp. 230-233.

[10] N. Hasan and C.-L. Liu, "Minimum fault coverage in reconfigurable arrays," in *Proc. Int. Symp. Fault-Tolerant Comput.*, June 1988, pp. 348-353

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

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

[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, 1983.

[14] S. Y. Kuo and W. K. Fuchs, "Efficient spare allocation in reconfigurable arrays," *IEEE Design and Test*, vol. 4, pp 24-31, 1987.

[15] W.-K. Huang *et al.*, "Approaches for the repair of VLSI/WSI RAMs by row/column deletion," in *Proc. Int. Symp. Fault-Tolerant Comput.*, June 1988, pp. 342 347.

[16] P. Mazumder, J. H. Patel, and W. K. Fuchs, "Methodologies for testing embedded content addressable memories," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 11-20, Jan. 1988.

[17] P. Mazumder and J. H. Patel, "An efficient built-in self-testing algorithm for random-access memory," *IEEE Trans. Ind. Electron.*, vol. 36, May 1989, pp. 394-407.

[18] P. Mazumder and Y. S. Jih, "Processor array self-reconfiguration by neural networks," in *Proc. IEEE Wafer-Scale Integration*, Jan. 1992.

[19] 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.

[20] C. Mead, "Analog VLSI and Neural Systems," Reading, MA: Addison-Wesley, 1989.

[21] J. Melngailis, "Focused ion beam technology and applications," *J. Vac. Sci. Technol. B*, vol. 5, pp. 469-495, 1987.

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

[23] G. Nicholas, "Technical and economical aspect of laser repair of WSI memory," in *Wafer-Scale Integration*, G. Saucier and J. Trihle, Eds. New York: Elsevier, 1986, pp. 271-280.

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

[25] C. H. Stapper, A. N. McLaren, and M. Dreckman, "Yield model for productivity optimization of VLSI memory chips with redundancy and partially good product," *IBM J. Res. Develop.*, vol. 24, no. 3, pp. 398-409, May 1980.

[26] C. H. Stapper, "On yield, fault distributions, and clustering of particles," *IBM J. Res. Develop.*, vol. 30, no. 3, pp. 326-338, May 1986.

[27] H. Stopper, "A wafer with electrically programmable interconnections," *Dig. IEEE Int. Solid-State Circuits Conf.*, 1985, pp. 268-269.

[28] M. Tarr, D. Boudreau, and R. Murphy, "Defect analysis system speeds test and repair of redundant memories," *Electronics*, pp. 175-179, Jan. 1984

[29] C.-L. Wey *et al.*, "On the repair of redundant RAM's," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 222-231, 1987.

Pinaki Mazumder (S'84-M'87) received the B.S.E.E. degree from the Indian Institute of Science in 1976, the M.Sc. degree in computer science from the University of Alberta, Canada in 1985, and the Ph.D. degree in electrical and computer engineering from the University of Illinois, Urbana-Champaign, in 1987. Presently he is an Assistant Professor at the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor.

Prior to this, he was a research assistant at the Coordinated Science Laboratory, University of Illinois, and for over six years was with the Bharat Electronics Ltd., India, (a collaborator of RCA)

where he developed several types of analog and digital integrated circuits for consumer electronics products. During the summers of 1985 and 1986, he was Member of the Technical Staff in the Naperville branch of AT&T Bell Laboratories. His research interest includes VLSI testing, physical design automation, ultrafast digital circuit design, and neural networks. He is a recipient of Digital's Incentives for Excellence Award, National Science Foundation Research Initiation Award, and Bell Northern Research Laboratory Faculty Award. He has published over 50 papers in IEEE and ACM archival journals and reviewed international conference proceedings. He is a Guest Editor of the IEEE DESIGN AND TEST MAGAZINE's special issue on memory testing, to be published in 1993.

Dr. Mazumder is a member of Sigma Xi, Phi Kappa Phi and ACM SIGDA.

Yih-Shyr Jih (S'85–M'91) received the B.S. degree in computer science and information engineering from Nation Taiwan University, Taipei, Taiwan, Republic of China, in 1981, the M.S. degree in computer science from Michigan State University, East Lansing, in 1985, and the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, in 1991.

He is currently a Research Staff Member in the IBM T. J. Watson Research Center, Yorktown Heights, NY. His research interests include VLSI computer-aided design, fault-tolerant computing, and high-bandwidth computer communication.