SIMULATED ANNEALING ALGORITHM FOR THE FAULT COVERING OF REDUNDANT RAMS

Yeow-Meng Chee* Hon-Wai Leong† Andrew Lim¶ Chor-Ping Low†

Abstract

We present a fast simulated annealing algorithm for the problem of fault covering of redundant RAMs. The algorithm has been implemented and tested on various benchmark problems and has performed very well. Our simulations also indicate that the algorithm is very fast. For all the repairable chips tested, it took less than one second to find a repair solution.

1 Introduction

One technique to increase the production yield of memory chips is by the use of *redundant random access memories* (RRAMs). In RRAMs, a small number of *spare rows* and *spare columns* of memory cells, together with the associated decoders, are included, which can be used to replace the defective elements [HD87]. A typical system that performs RRAM testing and repair is Mera II (Memory Error-Repair Analyzer) [TBM84]. The repair procedure of RRAMs proceed in three phases, namely, the testing, analysis, and repair phases. Mera II first tests the device and collects failure data. A *reconfiguration algorithm* then analyzes these data and generates repair information for the laser of other repair systems [TBM84].

Research on the memory repair problem has focused on the analysis phase, namely, the problem of determining a repair solution [TBM84, Day85, WL87, KF87, HL88, LT89]. In this paper, we present a new algorithm for this problem based on the method of simulated annealing. We propose a simple representation of repair solutions from which we derive a natural definition of moves and neighboring solutions. The algorithm has been implemented and tested on some benchmark problems. The results show that our method not only produces very good quality solutions, but also runs very fast. In fact, for all the repairable chips, our simulated annealing algorithm finds a repair solution within a second for various problem sizes.

*Information Technology Institute, National Computer Board, 71 Science Park Drive, S0511, Republic of Singapore.

†Department of Information Systems and Computer Science, National University of Singapore, Lower Kent Ridge Road, S0511, Republic of Singapore.

¶Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A.

2 Problem Definition and Previous Work

We first give a formal definition of the problem. A faulty RRAM is given as a rectangular array of memory cells with α *spare rows* and β *spare columns*, together with a list (i_1, j_1), (i_2, j_2), ..., (i_e, j_e) of the coordinates of *faulty cells*. A faulty cell (i, j) can be repaired by a spare row (which replaces row i), or by a spare column (which replaces column j).

The *memory fault covering problem* is that of finding a number of spare rows and spare columns so as to cover all the faulty cells. We say that the given array is *repairable* if a cover solution exists; otherwise, if the available spare rows and spare columns cannot fully cover the faulty cells no matter how they are reconfigured, then the array is said to be *irrepairable.*

This problem can be transformed into the problem of finding a vertex cover in a bipartite graph subject to certain additional constraints. Let $G = (V_1 \cup V_2, E)$ be a bipartite graph. An (α, β)-*vertex cover* (or simply an (α, β)-cover) or G is a set $C = C_1 \cup C_2$ with $V_1 \supseteq C_1$ and $V_2 \supseteq C_2$, such that the following conditions are satisfied:

(2.1) Every edge in E is incident with some vertex in C;

(2.2) $|C_1| \leq \alpha$, $|C_2| \leq \beta$.

A *minimum* (α, β)-cover is one of minimum cardinality.

The transformation is as follows. For the given array, we construct a bipartite graph $G = (V_1 \cup V_2, E)$ where the vertices in V_1 and V_2 correspond to the rows and columns of the array, respectively. The edge (i, j) is in E if and only if there is a faulty cell at coordinate (i, j) of the array. Then the memory fault covering problem is precisely the problem of finding a minimum (α, β)-cover in the graph G.

The memory fault covering problem has attracted some attention in the past few years [TBM84, KF87, HL88, LT89]. The problem has been proven to be NP-complete [KF87], and thus it is unlikely that efficient (polynomial time) algorithms will ever be found for the problem. The solution space for this problem is large — there are ${}^{m}C_{\alpha} \cdot {}^{n}C_{\beta}$ possible ways to asssign the α spare rows and β spare columns to a chip with m rows and n columns.

Previous work on the problem includes (1) problem reduction methods which attempt to identify rows or columns that must be in all fault covers, (2) constructive algorithms, most of which are greedy in nature, and does not guarantee finding a solution even if one exists, (3) tests for irrepairability, and (4) exhaustive search algorithms which uses a number of different branch-and-bound techniques to prune the search tree.

3 A Simulated Annealing Algorithm

We present in this section a fault covering algorithm that employs the method of *simulated annealing*. The simulated annealing methodology was introduced by Kirkpatrick, Gelatt, and Vecchi [KGV83] based on the observation that a combinatorial optimization problem is analogous to the problem of determining the lowest-energy ground state of a physical system with many interacting atoms. Intuitively, the simulated annealing optimization procedure can be viewed as an enhanced version of iterative improvement (see, for example, [PS82]).

```
Generic Simulated Annealing Procedure
begin
    $S$ = initial solution $S_0$;
    $T$ = initial temperature $T_0$;
    while (stopping criterion is not satisfied) do
        begin
            while (not yet in equilibrium) do
                begin
                    $S'$ = a random neighbor of $S$;
                    $\Delta$ = cost($S'$) – cost($S$);
                    prob = min(1, $e^{-\Delta/T}$);
                    if (random(0, 1) $\leq$ prob) then
                        $S = S'$;
                end;
            update $T$;
        end;
    output best solution;
end.
```

Simulated annealing has been widely used in VLSI design (see, for example, [WLL88], for details). We elaborate on the four key ingredients in applying the method of simulated annealing to the fault covering problem: the solution representation, the neighborhood structure, the cost function, and the annealing schedule. But first, we describe a preprocessing step.

3.1 Preprocessing

The algorithm starts by running a *forced-repair* procedure that identifies all the rows and the columns that must be covered in all coverings. A *must-repair row* is a row which has more than β faulty cells. This row must necessarily be covered by a spare row since there are not enough spare columns to cover the faulty cells in that row. A *must-repair column* is similarly defined. The forced-repair procedure repeatedly finds must-repair rows/columns and covers them until there are no more must repair rows/columns left; or when it has run out of spare rows or spare columns, in which case the memory chip is irrepairable.

3.2 Solution Representation

We view a cover as an assignment of each spare row and column to some row and column in the array, respectively. Of course, we allow the possibility that a spare row or column is not used in the cover. More formally, we represent a solution by a pair of vectors $(\mathcal{R}, \mathcal{C})$ where $\mathcal{R} = (r_1, r_2, ..., r_\alpha) \in \{0, 1, ..., m\}^\alpha$, and $\mathcal{C} = (c_1, c_2, ..., c_\beta) \in \{0, 1, ..., n\}^\beta$. The non-zero components of $\mathcal{R}$ and $\mathcal{C}$ gives the rows and columns that are to be covered, respectively. A zero component r_i means that the i^{th} spare row is not used in the repair solution, and similarly for c_j.

Thus, the solution space consists of all possible covers using at most α spare rows and β spare columns. Furthermore, it is not difficult to see that any repair solution can be transformed into this representation.

3.3 Neighboring Solutions

For a given solution $S = (\mathcal{R}, \mathcal{C})$, there is a natural definition of the *neighboring solutions* of S. A *move* selects one of the vectors $\mathcal{R}$ or $\mathcal{C}$ at random, and changes a component (also chosen at random) of the selected vector by the procedure which is described below. We consider only the vector $\mathcal{R}$. The change for $\mathcal{C}$ is similar. To change a component, we generate an integer between 0 and m (inclusive) randomly, and replace that component with the generated integer. We define the *neighbors* of S to be those solutions that are reachable from S in one move. Clearly, it is possible to go from any solution to any other solution via a finite sequence of moves. Let I be the discrete random variable denoting the outcome of the event of changing a component of $\mathcal{R}$. We considered the following condition on I,

$$Prob(I = 0) = \gamma Prob(1 \leq I \leq m),$$

and found that $\gamma = 1$ gives good results, since too low a value of γ often gives covers of large cardinality and too high a value of γ often gives infeasible solutions.

For a vector $\mathcal{R}$, there are α possible choices of a component to subject to change, and the range of change has cardinality $m + 1$. The consideration for $\mathcal{C}$ is similar. Hence, there are exactly

$$NS = \alpha(m + 1) + \beta(n + 1)$$

neighbors for any solution S, and this number is a fixed constant that is independent of S. If we restrict ourselves by allowing only vectors without non-zero repeating components, i.e. all non-zero components are distinct, then the number of neighbors of S is upper-bounded by NS.

3.4 Cost Function

The objective function is to minimize the number of spare rows and spare columns used to cover all the faulty cells. However, since at each instance of time, our solution may not be feasible, it is also important to take the number of faulty cells left uncovered into consideration. In the simulated annealing algorithm, the following cost function is used:

$$\text{cost}(S) = \lambda_1 r(S) + \lambda_2 c(S) + \lambda_3 u(S),$$

where $r(S)$ is the number of spare rows used by S, $c(S)$ is the number of spare columns used by S, and $u(S)$ is the number of faulty cells uncovered by S.

The values of λ_1, λ_2, and λ_3 control the relative importance of the three terms in the cost function. Setting λ_3 very large corresponds to ignoring the size of the cover. This is undesirable since it defeats the purpose of including the first two terms. On the other hand, setting λ_3 too small often results in solutions that are not feasible. Experimental results show that setting $\lambda_1 = \lambda_2 = \lambda_3/2$ produces solutions of good quality.

3.5 Annealing Schedule

For the annealing schedule, we start with a high temperature T_0 and cool exponentially, $T_k = \mu T_{k-1}$, for $k = 1, 2, \ldots$. To estimate reasonable values of the initial temperature T_0, we consider the following situation. Suppose that a move increases any of the quantities r, c, and u by one. Then the corresponding change in the cost function is $\Delta = \lambda_1$, $\Delta = \lambda_2$, and $\Delta = \lambda_3$, respectively. The probability of making such a move is $e^{-\Delta/T}$. Thus, for there to be any reasonable probability of making this move at the initial high temperature T_0, we must have $T_0 = \Omega(\max\{\lambda_1, \lambda_2, \lambda_3\})$. Therefore, the initial temperature T_0 is chosen to be of the form $\lambda_T \max\{\lambda_1, \lambda_2, \lambda_3\}$, where λ_T is used to control the initial temperature. The default value for λ_T that we used is 2.3. The value of μ varies from 0.8 to 0.9.

At each temperature, enough moves are attempted until either $\varepsilon_1 NS$ moves are accepted or $\varepsilon_2 NS$ moves are attempted. In our experiments, even values as low as $0.1 \le \varepsilon_1 \le 0.2$ and $0.02 \le \varepsilon_2 \le 0.04$ produce results of good quality.

The annealing process is terminated if the number of accepted moves is less than two percent of all the moves made at that temperature, or if the temperature is low enough.

4 Results and Conclusion

The simulated annealing algorithm has been implemented in the C programming language on a HP9000/300 machine and tested on various benchmark problems from [KF87] and [WL87]. The results are shown in Tables 4.1 and 4.2, respectively. For all the repairable cases, our simulated annealing algorithm obtained a solution in less than one second. Furthermore, the running times are all in the order of a few seconds. For example, in Table 4.1, our algorithm found a cover of size 38 for a 512 × 512 memory chip, with 20 spare rows and 20 spare columns, and 45 faulty cells, in about 0.6 seconds. Irrepairable memory chips are marked with an "IR" in the column indicating the size of the covers. This compares well with a number of exhaustive search algorithms [Day85, KF87, HL88].

In summary, we proposed in this paper, a fault covering algorithm based on simulated annealing. Our algorithm uses a simple solution representation and a natural neighborhood structure. The algorithm is appealing because it runs extremely fast and it produces good results. We are currently conducting more exhaustive testing of the algorithm on more random data.

Acknowledgement
Research of the second author (HWL) is partially supported by the National University of Singapore under grant RP880610, and by the National Science Foundation (USA) under grant INT-8713879.

References

[Day85] R. J. Day. A fault-driven comprehensive redundancy algorithm. *IEEE Design and Test* **2** (1985) 35–44.

[HD87] R. W. Haddad and A. T. Dahbura. Increased throughput for the testing and repair of RAMs with redundancy. *Proceedings of the IEEE International Conference on Computer Aided Design* (1987) 230–233.

[HL88] N. Hasan and C. L. Liu. Minimum fault coverage in reconfigurable arrays. *Proceedings of the International Symposium on Fault-Tolerant Computing* , 1988.

[KF87] S. Y. Kuo and W. K. Fuchs. Efficient spare allocation for reconfigurable arrays. *IEEE Design and Test* **4** (1987) 24–37.

[KGV83] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. Optimization by simulated annealing. *Science* **220** (1983) 671–680.

[LT89] H. W. Leong and G. W. Tan. Reconfigurable techniques for memory chip repair. *Proceedings of the 2nd International Symposium on the Physical and Failure Analysis of Integrated Circuits* , 1989.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Prentice Hall, New York, 1982.

[TBM84] M. Tarr, D. Boudreau and R. Murphy. Defect analysis system speeds test and repair of redundant memories. *Electronics* (1984) 175–179.

[WL87] C. L. Wey and F. Lombardi. On the repair of redundant RAMs. *IEEE Transactions on Computer-Aided Design* **6** (1987) 222–231.

[WLL88] D. F. Wong, H. W. Leong and C. L. Liu. *Simulated Annealing for VLSI Design.* Kluwer Academic Publishers, Boston, Lancaster, Dordrecht, 1988.

Table 4.1 : Performance on data from [KF87]

Data	Array Size	$\alpha \times \beta$	e	Size of Cover	CPU time
KF1	128	4×4	5	5	0.0s
KF2	128	4×4	15	IR	0.0s
KF3	256	5×5	10	9	0.0s
KF4	256	5×5	30	IR	0.1s
KF5	512	5×5	15	5	0.0s
KF6	512	10×10	19	18	0.2s
KF7	512	10×10	45	IR	0.4s
KF8	512	20×20	45	38	0.6s
KF9	1024	20×20	40	36	0.7s
KF10	1024	20×20	60	IR	1.0s
KF11	1024	20×20	200	IR	3.9s
KF12	1024	20×20	400	IR	8.2s

Table 4.2 : Performance on data from [WL87]

Data	Array Size	$\alpha \times \beta$	e	Size of Cover	CPU time
WL1	1024	5×5	27	10	0.2s
WL2	512	5×5	36	IR	0.1s
WL3	512	8×8	58	15	0.8s
WL4	256	6×6	31	11	0.3s
WL5	256	4×4	28	IR	0.1s
WL6	1024	4×4	27	IR	0.1s
WL7	1024	7×7	61	14	0.5s
WL8	1024	8×8	81	IR	0.6s
WL9	512	4×4	27	IR	0.1s
WL10	256	4×4	25	8	0.0s
WL11	512	7×7	42	14	0.5s
WL12	256	4×4	24	8	0.0s