NOVEL GRAPH-BASED ALGORITHMS FOR RECONFIGURABLE ARRAYS*

Sung-Chuan Fang, Sao-Jie Chen, and S. L. Lee

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.

Abstract— In this paper, a generalized version of the repair-most method and an exhaustive search method for reconfigurable arrays in fabrication- or compile-time are presented. This generalized version of method, instead of removing the most possible edges as in the repair-most method, tries to "free" as many vertices as possible in the bipartite graph at each iteration. Therefore, it can overcome the major defects of the repair-most one and runs in equal time complexity. We call this generalized method as the free-most method. For evaluating the results generated from the free-most method, we also develop an exhaustive search method based on the vertex covering problem in graph theory and call it the vertex-cover method. This method can produce optimal solutions for all instances. Additionally, the lower bounds of instances are derived from and proved under the bipartite graph model. In all the test examples, the free-most method can generate optimal solutions and better results than the repair-most one.

Index terms— Fault tolerance, reconfigurable arrays, array processors, vertex covering, bipartite graph.

1.Introduction

Fabrication-time and *compile-time* reconfigurations are often used to enhance the yield. These two methods are very alike to each other. Both applications have the advantage of performing reconfiguration of arrays without interrupting the normal operation of the system. The array may be composed of identical memory cells or other function units. Since the reconfiguration does not occur in real time, therefore, the length of required time is not restricted. The key difference between fabrication- and compile-time reconfigurations is their applying time. The fabrication-time reconfiguration occurs after the fabrication of array during the production processes; the configuration is permanent and array cannot be reconfigured again at a later time. But, compile-time reconfiguration can be performed after the array has been used for some period of time; furthermore, the reconfiguration is not permanent. Consequently, compile-time reconfiguration can be done numerous times.

There were many reconfiguration strategies supporting reconfigurable arrays in fabrication- or compile-time, such as *rippling replacement* strategy [1], *fault stealing* technique [1] and *repair-most* algorithm [2]. The first two methods can be applied in the array with fewer faulty elements. The rippling replacement strategy cannot accommodate any two faulty elements in a same row. Speaking to the fault stealing technique, it will fail when there are two or more faulty elements located on each one of the two crossing branches of the '+' lines. The failed examples for first two methods are shown in Figure 1.

The repair-most method is more suitable to the array with more faulty elements and it replaces complete rows or columns by the spare rows or columns. The major benefit of the repair-most method is its ability to overcome any kind of fault patterns. But, it is inefficient for the array with fewer faulty elements under the fault patterns which can be reconfigured by the rippling replacement strategy or fault stealing technique. Additionally, the method does not utilize spare rows or columns in the most efficient manner [3]. The repair-most method uses the bipartite graph model to represent the *array repair* problem. (For details, please refer to [2].) As shown in Figure 2, if there is a faulty element $E(i,j)$ in the array, then there is an edge connecting vertices R_i and C_j in the associated bipartite graph. When one removes all edges from vertex Ri (Cj), it means that row i (column j) is replaced by a spare row (column). As an example in Figure 2, the repair-most method needs totally eight spare rows or columns for reconfiguration. But, it is clear and simple to find that six spares are enough to reconfigure the array.

Some methods were introduced to overcome the defects of the repair-most method. A comprehensive approach was proposed in [4]. But, this exhaustive search method becomes very inefficient according to its exponential time complexity when the problem size increases. If the numbers of spare rows and columns are constrained or limited, the array repair

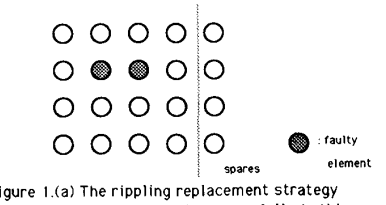


Figure 1.(a) The rippling replacement strategy will not work successfully in this case.

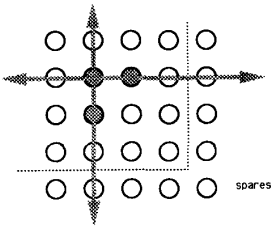


Figure 1.(b) This example cannot be reconfigured by fault stealing technique.

*This work was partially supported by the National Science Council, Taipei, Taiwan, R.O.C., under Grants #NSC79-0416-E002-02 and NSC79-0404-E002-33.

CH 3006-4/91/0000 – 2866 $1.00 © IEEE

problem was proved to be *NP-complete* [3]. To deal with the problem, two algorithms were proposed in [3]. The first algorithm is a branch-and-bound method with exponential time consumption. The other one is a heuristic one bounded almost in polynomial time $O(4n^2)$, where n is the dimension of the array. Both algorithms are more complicated and more time consuming than the repair-most method.

In this paper, a generalized version of the the repair-most method is developed. We call this generalized method as the *free-most* method [5]. The free-most method attempts to overcome the defects of the repair-most method. Moreover, it takes almost equal time with respect to the repair-most method and is easier to implement. For comparison, an exhaustive search method (the *vertex-covering* method) based on the *vertex covering* problem in the graph theory field is also developed. This method always guarantees the optima and is used to evaluate the results generated from the free-most one.

In the following section, the lower bounds of instances based on the bipartite graph model are derived. Section 3 addresses the free-most method in details. The vertex-covering method is presented in Section 4. Section 5 shows the experimental results and the conclusion is given in the final section.

2. Lower Bound Derivation

Lemma 1. The number R of required spare rows or columns must exceed or equal the number N of connected components in the associated bipartite graph G_b of the repair instance I.

Proof: The repair method removes all edges from a vertex in G_b at each step until all edges in G_b are removed completely. In each connected component Q_i of G_b, we need at least one cut to remove all edges from Q_i. Consequently, this lemma is clear to be proved.#

In a bipartite graph G_b, we define the most number of degrees of vertices in a connected component Q_i as $D_{max,i}$. In addition, the total number of edges in the component Q_i is E_i. For example, in Figure 3, the $D_{max,i}$ is four and E_i is nine for the component Q_i.

Assume that there are N components in G_b, we have:

Theorem 1. The lower bound of the number of required spare rows and columns is

$$\sum \lceil Ei/D_{max,i} \rceil, \text{ for } i = 1, 2, ..., N; \quad (1)$$

where $[x]$ represents the smallest integer greater or equal to x.

Proof : From Lemma 1, $R \geq N$. But in each component Q_i, each cut can remove at most $D_{max,i}$ edges. That is, at least $\lceil E_i/D_{max,i} \rceil$ cuts are needed to remove all edges in Q_i. Therefore, $R \geq \lceil E_1/D_{max,1} \rceil + \lceil E_2/D_{max,2} \rceil + ... + \lceil E_N/D_{max,N} \rceil$. Hence the theorem is complete.#

3. Algorithm Using the Free-Most Method

The same bipartite graph model is used to represent our array repair problem. In each step, a vertex with maximal cost in the bipartite graph is chosen to remove all edges from it. The algorithm terminates when all edges in the bipartite graph are removed.

In the following, the algorithm using free-most method is addressed.

ALGORITHM Free-Most
1. Input the instance I;
2. Construct the associated bipartite graph G_b;
3. Calculate all costs of vertices with at least one edge in G_b (ignore the isolated vertices);
4. **While** (there is at least one edge in G_b) **do**
5. Select one vertex with maximal cost and remove all its edges;
6. Update the costs of the vertices which are changed according to Step 5 (ignore the isolated vertices);
7. **end**;
8. Output the repair sequence;

END Free-Most

It is very simple to prove that this algorithm runs in $O(|V|^2/4)$ $(=O(n^2))$ time, where $|V|$ is the number of vertices in G_b; n is the dimension of the array. That is to say, the algorithm Free-Most has the same time complexity as the repair-most method. The proof is skipped here.

3-1. The Cost Function

The cost function corresponding to a non-isolated vertex V_i consists of three elements. To describe it, we need to define:

f_i : the number of isolated vertices resulted from removing all edges of V_i.
d_i : the number of degrees of V_i.
$A(V_i)$: the adjacent list of V_i (the set of vertices with edges conneting to V_i).
r_i : the total number of edges of vertices in $A(V_i)$ minus the number of edges of V_i.

Therefore, the cost function of a vertex Vi is

$$C(V_i) = c1*f_i + c2*d_i + (c3-r_i); \quad (2)$$

where $c1$, $c2$, and $c3$ are constants, but $c1 >> c2*d_i$ and $c2 >> (c3 - r_i)$ for all i's.

A triple (f_i, d_i, r_i) is used to represent the cost of V_i. All these three parameters tend to "*free*" as many vertices as

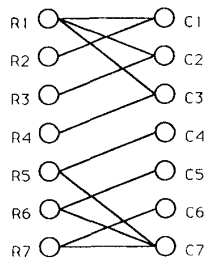


Figure 2. Bipartite graph prior to reconfiguration

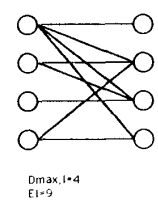


Figure 3. A component QI and its Dmax,I and EI

possible. That is, they try to make as many vertices isolated as possible by removing all edges from vertex V_i. Figure 4 demonstrates the notation of the cost function.

3-2. Handling Constraints on Numbers of Spare Rows and Columns

If the number of spare rows and columns are constrained or limited, we must identify the overuse of spare rows or columns in the Free-Most algorithm and try to avoid this condition. It is easy to modify the algorithm Free-Most under such consideration. That is, if there is no spare row (column) to be used, the costs of all vertices corresponding to rows (columns) will be set to negative infinity. In this way, the overuse of spare rows or columns can be avoided. Of course, when the costs of all vertices of the bipartite graph are all negative infinity, it means the original array cannot be reconfigured under the constraints on the numbers of spare rows and columns.

3-3. A Test Example

As an example illustrated in Figure 2, the algorithm Free-Most selects one vertex with the most cost and removes all its edges in each iteration. The selected vertices in each iteration are listed below :

Iteration 1: $C(R5) = (2, 2, 2)$ is maximal.
Iteration 2: $C(R6) = (2, 2, 1)$ is maximal.
Iteration 3: $C(R7) = (3, 2, 0)$ is maximal.
Iteration 4: $C(C1) = (2, 2, 2)$ is maximal.
Iteration 5: $C(C2) = (2, 2, 1)$ is maximal.
Iteration 6: $C(C3) = (3, 2, 0)$ is maximal.

Hence the replace or repair sequence is *R5 --> R6 --> R7 --> C1 --> C2 --> C3*. Only six spare rows and columns are required to reconfigure the array. This outcome is better than the one generated by the repair-most method. Remember that the number of required spare rows or columns is eight for the repair-most method under the same test example (see Introduction).

4. Algorithm using the Vertex-Covering Method

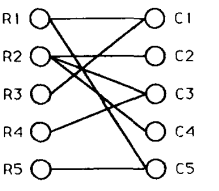
Before introducing the algorithm, some definitions and background theorems are required.

Definition 1. A set $S \subseteq V$ is a *vertex cover* for an undirected graph $G = (V, E)$, if every edge of G is incident with some vertex in S.

Theorem 2. The problem of determining whether an undirected graph $G = (V; E)$ has a vertex cover of size l is *NP*-complete.

Corollary 1. Vertex covering problem (the problem of determining a minimum vertex cover, $S_{min}(G)$, of an undirected graph $G = (V, E)$) is *NP*-complete.

These definition, theorem, and corollary above can be found in books of graph theory or combinatorial algorithms, e.g., in [6], [7]. As an example in Figure 5, the minimum vertex cover of an undirected graph G is illustrated, that is, $S_{min}(G) = \{1, 3, 7\}$. If we remove edges from all the vertices in $S_{min}(G)$, all vertices in G become isolated. Following this concept and assuming the graph to be bipartite, then we have immediately:



C(R1) = (1, 2, 2)
C(C5) = (2, 2, 1)

Figure 4. The costs for R1 and C5

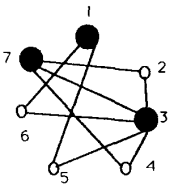


Figure 5. A graph G with a minimum vertex cover shown in boldface

Theorem 3. The array repair problem is equivalent to the vertex covering problem based on the bipartite graph model.

Theorem 3 is right to say, one can use the algorithms for vertex covering problem in a bipartite graph to solve the corresponding array repair problem. For example of Figure 2, it is easy to verify that the minimum vertex cover $S_{min}(G_b) = \{R5, R6, R7, C1, C2, C3\}$. It has the same solution with respect to the one shown in Subsection 3-3.

Though there was a heuristic algorithm proposed in [8] for the vertex covering problem. Here, we introduce an exhaustive search algorithm to guarantee of optima.

The vertex covering problem in an undirected graph G is the dual one of determining the maximum clique in the complement of G [7]. We denote the complement of G as G'. For example, the complement of the graph in Figure 5 is depicted in Figure 6; and its maximum clique, $CLI(G') = \{2, 4, 5, 6\}$, is also mentioned.

It is concise to state now:

Theorem 4. For an undirected graph $G = (V, E)$,

$$S_{min}(G) = V - CLI(G') \quad (3)$$

In Figure 6, we have $S_{min}(G) = V - CLI(G') = \{1, 3, 7\}$, which gives the same result as described above.

There is a most efficient algorithm for determining all maximal cliques in a general graph G ([7] pp. 353-359). The detail of the algorithm is omitted here, but, for convenience, it is denoted as CLIQUES(G) and will be used in our vertex-covering algorithm.

After introducing the concepts of minimum vertex cover and maximum clique, the algorithm using vertex-covering method can be readily depicted as follows:

Algorithm Vertex-Covering

1. Input the instance I;
2. Construct the associated bipartite graph G_b;
3. Search all connected components and assume them to be $Q_1, Q_2, ..., Q_k$ (ignore the isolated vertices);
4. **For** i = 1 **to** k **do**
5. **Call** CLIQUES(Q_i ');
6. Select a maximum clique CLI(Q_i ') from all maximal cliques in Q_i';
7. $S_{min}(Q_i) = V_i$ - CLI(Q_i'), where V_i is the set of vertices inQ_i;
8. **end**;
9. Output the solution from $S_{min}(Q_i)$, for i = 1, 2, ...,

k;
END Vertex-Covering

According to Step 5, it comes that an exhaustive search for all maximal cliques is processed, we can guarantee a true selection of the maximum clique for each connected component Q_i ' in Step 6. That is, a corresponding minimum vertex cover is selected for Q_i . Hence this algorithm can always generate an optimal solution for any problem instance. Even there are constraints on spare rows and columns, with some appropriate modifications, it still promises the optima for the array repair problem. These modifications are omitted in this paper. Since it is inherently an exhaustive search method, it will become very inefficient according to its exponential time complexity when the problem size increases.

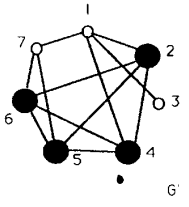
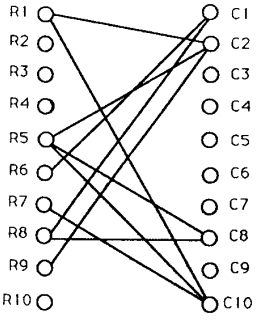


Figure 6. The complement of the graph G in Figure 5 and its maximum clique is shown in boldface.



Repair sequences :
Free most : C2->C10->C1->C8
Repair most : R5->R1->R8->R6->R7->R9

Figure 7. Example test2

5. Experimental Results

The above presented algorithms are coded in C language and implemented on SUN 3/110 workstation. There is a summary of results listed in Table I. The test examples are arrays with dimensions from 10 to 100 and the fault patterns are all generated randomly. In all algorithms, no constraint is set on the number of spare rows or columns. We hope that all methods can run in the most free manner, hence, we can compare the intrinsic abilities among them.

On an average, our method gives a 12.2% of improvement with respect to the repair-most one. Figure 7 illustrates the example test2. The repair sequences for both methods are listed simultaneously. According to the experimental results, the free-most method can generate better outcomes than the repair-most one. Since the vertex-covering method guarantees the optima for all test examples, we can find the free-most method operates in a very excellent way from Table I. For all the test examples, the free-most method produces the optimal solutions as the vertex-covering method does.

6. Conclusion

In this paper, we develop a new method, the free-most method, to reconfigure the arrays in fabrication- and compile-time. The free-most method can overcome the defects of the repair-most method. Moreover, it consumes almost equal time with comparison to the repair-most method and is easier to implement. With respect to the repair-most strategy, our method is more efficient to generate good results and with 12.2% of improvement on an average. For comparison, an exhaustive search method, the vertex-covering method, is also developed to obtain optimal solutions. From the experimental results, our free-most method generated optimal solutions for all test examples.

In the future, we will make use of planning method in the AI field for further smart reconfiguration when there are constraints on numbers of spare rows and columns. Besides, instead of bipartite graph, the free-most method can be generalized easily to the corresponding problem of general graph and solve the equivalent *vertex covering* problem in the graph theory field.

References

[1] R, Negrini, M. Sami and R. Stefanelli, "Fault tolerance techniques for array structures used in supercomputing," *Computer*, Vol. 19, No. 2, Feb. 1986, pp. 78-87.

[2] M. Tarr, D. Boudreau, and R. Murphy, "Defect analysis system speeds test and repair of redundant Memories," *Electronics*, Jan. 12, 1984, pp. 175-179.

[3] S. Y. Kuo and W. K. Fuchs, "Efficient spare allocation in reconfigurable arrays," *IEEE Design & Test of Computers*, Vol. 4, No.1, Feb. 1987, pp. 24-31.

[4] J. Day, "A fault-driven comprehensive redundancy algorithm," *IEEE Design & Test of Computers*, Vol. 2, No. 3, Jun. 1985, pp. 35-49.

[5] S. C. Fang, S. J. Chen, S. L. Lee, and J. C. Liou, "A new strategy for reconfigurable arrays using free-most method," *Proc. International Electron Devices and Materials Symposium*, Hsinchu, Taiwan, R.O.C., Nov. 14-16, 1990, pp. 362-365.

[6] N. Deo, *Graph theory with applications to engineering and computer science*, Prentice-Hall Press, Inc., Englewood Cliffs, N. J., 1974.

[7] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial algorithms: theory and practice*, Prentice-Hall Press, Inc., Englewood Cliffs, N. J., 1977.

[8] K. Clarkson, "A modification for the greedy algorithm for vertex cover," *Info. proc. Letters*, Vol. 16, Jan. 1983, pp. 23-35.

Table I. Results of the test examples

				free-most / repair-most			vertex-covering
examples	rows	columns	faults	spare_R	spare_C	total	total*
test1	10	10	5	1 / 4	2 / 0	3 / 4	3
test2	10	10	10	0 / 6	4 / 0	4 / 6	4
test3	10	10	12	1 / 5	4 / 1	5 / 6	5
test4	10	10	40	0 / 7	10 / 6	10 / 13	10
test5	20	20	10	4 / 8	3 / 0	7 / 8	7
test6	20	20	20	5 / 14	7 / 0	12 / 14	12
test7	20	20	30	6 / 14	7 / 1	13 / 15	13
test8	30	30	15	3 / 7	5 / 2	8 / 9	8
test9	30	30	30	8 / 17	8 / 1	16 / 18	16
test10	40	40	25	12 / 20	6 / 0	18 / 20	18
test11	40	40	50	13 / 19	10 / 6	23 / 25	23
test12	50	50	50	17 / 27	10 / 5	27 / 32	27
test13	50	50	60	18 / 34	16 / 3	34 / 37	34
test14	70	70	50	17 / 28	13 / 4	30 / 32	30
test15	70	70	120	22 / 44	28 / 12	50 / 56	50
test16	80	80	30	16 / 20	3 / 2	19 / 22	19
test17	80	80	120	24 / 50	35 / 18	59 / 68	59
test18	80	80	160	30 / 56	31 / 12	61 / 68	61
test19	90	90	80	30 / 44	10 / 6	40 / 50	40
test20	100	100	120	31 / 54	25 / 10	56 / 64	56

spare_R: No. of required spare rows
spare_C: No. of required spare columns
total: No. of total spare rows and columns
total*: Optimal no. of total spare rows and columns