Prof. Petru Cașcaval
Universitatea Tehnică „Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare

LISTĂ DE LUCRĂRI ȘTIINȚIFICE REPREZENTATIVE

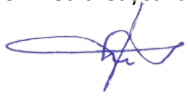
în domeniul de doctorat

Calculatoare și tehnologia informației

1. **Caşcaval, P.**; Leon, F., Optimization Methods for Redundancy Allocation in Hybrid Structure Large Binary Systems, Mathematics, Vol. 10 (19), 2022, https://doi.org/10.3390/math10193698 (**Q1**).
2. Leon, F., **Caşcaval, P.**, Bădica, C., Optimization Methods for Redundancy Allocation in Large Systems, Vietnam Journal of Computer Science, Vol. 7 (3), 281-299, 2020, https://doi.org/10.1142/S2196888820500165 (Q4).
3. **Caşcaval, P.**, Caşcaval, D., March test algorithm for unlinked static reduced three-cell coupling faults in random-access memories, Microelectronics Journal, Elsevier, Vol. 93, November 2019 (Q3) https://doi.org/10.1016/j.mejo.2019.104619.
4. **Caşcaval, P.**, Approximate Method to Evaluate Reliability of Complex Networks, Complexity, Wiley, Volume 2018, Article ID 5967604, https://doi.org/10.1155/2018/5967604 (**Q2**).
5. **Caşcaval, P**., Floria, S.A., SDP Algorithm for network reliability evaluation, IEEE Conf., INISTA, Gdynia, Poland, 3-5 July 2017, DOI: 10.1109/INISTA.2017.8001143 (**Best Paper award**).
6. Huzum, C., **Caşcaval, P.**, A Multibackground March Test for Static Neighborhood Pattern-Sensitive Faults in Random-Access Memories, Electronics and Electrical Engineering (Elektronika ir Elektrotechnika) – Section System Engineering, Computer Technology, Vol. 119 (3), 81-86, 2012, DOI10.5755/j01.eee.119.3.1369 (Q4).
7. **Caşcaval, P.**, Caşcaval, D., March SR3C: A Test for a reduced model of all static simple three-cell coupling faults in random-access memories, Microelectronics Journal, Elsevier, Vol. 41 (4), 212-218, 2010, doi:10.1016/j.mejo.2010.02.004 (Q3).
8. **Caşcaval, P.**, Silion, R., Caşcaval, D., A Logic Design for MarchS3C Memory Test BIST Implementation, Romanian Journal of Information Science and Technology, Vol. 12 (4), 2009, 440-454 (**Q2**).
9. **Caşcaval, P.**, Bennett, S., Huţanu, C., Efficient March Tests for a Reduced 3-Coupling and 4-Coupling Faults in Random-Access Memories, Journal of Electronic Testing: Theory and Applications, Springer, Vol. 20 (3), 227–243, 2004 (Q4), https://doi.org/10.1023/B:JETT.0000029457.21312.23.
10. **Caşcaval, P.**, Bennett, S., Efficient March Test for 3-Coupling Faults in Random Access Memories, Microprocessors and Microsystems, Elsevier Science, Vol. 24 (10), 501–509, 2001 (**Q2**), https://doi.org/10.1016/S0141-9331(00)00103-4.

25 septembrie 2023

Prof. Petru Cașcaval



mathematics

MDPI

Article

Optimization Methods for Redundancy Allocation in Hybrid Structure Large Binary Systems

Petru Cașcaval and Florin Leon *

Faculty of Automatic Control and Computer Engineering, "Gheorghe Asachi" Technical University of Iasi, Bd. Mangeron 27, 700050 Iasi, Romania

* Correspondence: florin.leon@academic.tuiasi.ro

Abstract: This paper addresses the issue of optimal redundancy allocation in hybrid structure large binary systems. Two aspects of optimization are considered: (1) maximizing the reliability of the system under the cost constraint, and (2) obtaining the necessary reliability at a minimum cost. The complex binary system considered in this work is composed of many subsystems with redundant structure. To cover most of the cases encountered in practice, the following kinds of redundancy are considered: active redundancy, passive redundancy, hybrid standby redundancy with a hot or warm reserve and possibly other cold ones, triple modular redundancy (TMR) structure with control facilities and cold spare components, static redundancy: triple modular redundancy or 5-modular redundancy (5MR), TMR/Simplex with cold standby redundancy, and TMR/Duplex with cold standby redundancy. A classic evolutionary algorithm highlights the complexity of this optimization problem. To master the complexity of this problem, two fundamentally different optimization methods are proposed: an improved evolutionary algorithm and a zero-one integer programming formulation. To speed up the search process, a lower bound is determined first. The paper highlights the difficulty of these optimization problems for large systems and, based on numerical results, shows the effectiveness of zero-one integer programming.

Keywords: redundancy allocation; hybrid structure binary systems; Markov chains; evolutionary algorithms; RELIVE algorithm; zero-one integer programming

MSC: 68M15; 68T20; 90C26

Citation: Cașcaval, P.; Leon, F. Optimization Methods for Redundancy Allocation in Hybrid Structure Large Binary Systems. *Mathematics* **2022**, *10*, 3698. https://doi.org/10.3390/math10193698

Academic Editor: Ioannis G. Tsoulos

Received: 7 September 2022
Accepted: 6 October 2022
Published: 9 October 2022

1. Introduction

The problem of reliability optimization in large hybrid systems mainly refers to the type of the system (binary or multi-state), type of solution (reliability allocation and/or redundancy allocation), or the kind of redundancy, which can be static (TMR or 5MR, for example), dynamic (active redundancy or standby redundancy), or hybrid (TMR/Simplex or TMR/Duplex with spare components, etc.). Useful overviews covering models and methods for these reliability optimization problems (ROPs), including reliability allocation, redundancy allocation, and reliability-redundancy allocation can be found in many works, such as [1–3].

The mathematical formulation of a reliability optimization problem requires the specification of three elements: decision variables, imposed constraints, and objective function(s).

The decision variables describe those elements that can be changed or adjusted or the decisions that can be made to improve system performance, as expressed by the objective function(s). As examples of decision variables one can mention the types of components and their characteristics (reliability, cost, etc.), the type of redundancy for each subsystem, the number of spare components for each subsystem, etc.

The constraints reflect practical design limitations, e.g., a required level of reliability or the available budget, which occur in almost all cases. But in practice there may be other limitations, related to the volume or weight of the system, for example.

The objective function measures the performance of the system for a set of values of the decision variables. Thus, by optimizing the objective function(s) under the specified constraints it is possible to identify the combination of values of the decision variables that leads to the best possible design solution for the studied system.

Usually for ROPs, the goal of optimization is to maximize system reliability or minimize system cost. In reliability engineering the problem of system reliability maximization under two or more constraints often arises, e.g., under cost constraints, but also under weight and/or volume constraints. When an analytical approach is possible (e.g., in the case of active-redundancy-only subsystems), to ensure that two or more constraints are satisfied, Lagrangian multipliers are often introduced as part of the objective function [4–6].

In this paper we address a class of redundancy allocation problems (RAPs) where the decision variable is the number of redundant components for each subsystem in a series redundant reliability model. RAP is one of the most studied reliability optimization problems, because it has been proven to be quite difficult to solve, and many different optimization approaches have been used to determine optimal or near-optimal solutions. As [7] demonstrates, RAPs belong to the NP-hard class of optimization problems.

The RAPs we consider involves hybrid structures with no less than eight types of redundancy; these are conditions where the optimization problems are difficult to solve, even if we limit ourselves to single-constraint optimization problems. More specifically, our goal is to highlight the difficulty of these RAPs for large systems, when the number of subsystems grows to the order of tens or even hundreds.

In order to master the complexity of RAPs in case of large systems, for which the difficulty of the problem increases, special research efforts have been made in recent years. In addition, to cover a wide range of techniques used to increase the reliability encountered in practice, many hybrid reliability models have been considered for which the RAPs get even more complicated. For example, [8] investigates a complex reliability-redundancy allocation problem with a component mixing strategy, which changes the traditional RAP model to a heterogeneous one. Moreover, in the hybrid reliability models proposed in [9], the choice of redundancy strategy is considered as a decision variable. So, for each subsystem, an active or cold standby redundancy may be considered. In addition, components of different types can be used in each subsystem, i.e., a component mixing strategy. Consequently, this RAP involves determining a solution that maximizes system reliability in terms of the type of redundancy and the number of spare components of each type (for each subsystem). To solve this RAP, a genetic algorithm is developed. Also, a reliability model based on cold standby redundancy combined with component mixing is investigated by [10]. For this complex problem, the author proposes a simplified swarm optimization method in which a multi-role resource sharing strategy is adopted to provide the diverse system components. Another reliability model based on active or cold standby redundancy combined with component mixing is investigated in [11]. To solve this RAP, the authors propose a parallel stochastic fractal search algorithm. Other RAPs involving a heterogeneous structure and/or component allocation strategy of a different type can be found in [12–14].

Such a hybrid reliability model is also considered in this paper. In the previously cited works, RAPs are formulated by considering redundant systems with hybrid redundancy strategies and/or reliability models with heterogeneous components, which means that each component of a subsystem can have its own failure rate. In this paper we limit ourselves to the case where subsystems include homogeneous components, but we extend RAPs to cover more redundancy strategies (not just active redundancy or cold standby), including static redundancy or reconfigurable structures such as TMR/Simplex or TMR/Duplex with cold standby redundancy.

To solve redundancy allocation problems of this type, several techniques can be applied, such as heuristic methods [15–18], Lagrange multiplier analytical methods, and branch-and-bound techniques, especially for active redundancy [5,6,19,20], dynamic programming [21–23], evolutionary algorithms [9,10,24–27], linear programming methods [28–30]

or a mix of integer and nonlinear programming [31]. As the RAPs we considered are complex, two evolutionary algorithms and a special model of zero-one integer programming are used.

As the highlights of our contribution we can mention:

- The formalization of two RAPs for binary systems with hybrid structure, which include no less than eight types of redundancy, where reliability modeling of redundant and reconfigurable structures is based on Markov chains;
- The design and implementation of two evolutionary algorithms and the formulation of a zero-one integer program for solving these complex optimization problems;
- Conducting an extensive performance evaluation study of the three proposed techniques on thousands of problems, which demonstrates the effectiveness of the zero-one integer programming approach for large systems with tens or even hundreds of subsystems.

This paper is organized as follows. Section 2 presents the issue addressed, whereas the types of redundancy considered here and the models or equations used for reliability evaluation are presented in detail in Section 3. Some related works are mentioned in Section 4. The algorithms used for these optimal allocation issues are described in Section 5. The objective functions adopted for the evolutionary algorithms and for the linear programming model are reported in Section 6, whereas in Section 7 a lower bound solution is proven. Experimental results are presented in Section 8. Further discussion is the subject of Section 9. The conclusions of the paper and several directions of future research are included in Section 10.

2. Problem Description

For systems with a large number of components without redundancy, reliability is often very low. To achieve the required reliability, a certain type of redundancy is applied to a certain element, depending on technical particularities, which can be static, dynamic, or hybrid redundancy. All of these types of redundancy are considered in this paper. The reliability model for this redundant system is a series-redundant one as presented in Figure 1.

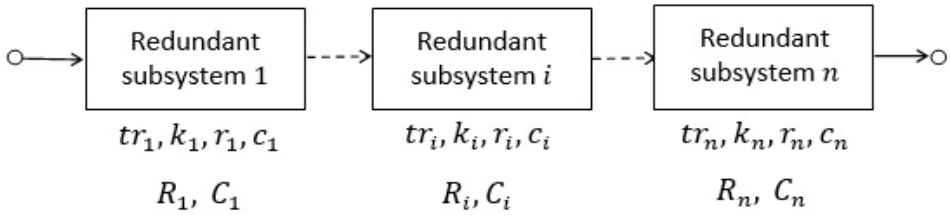


Figure 1. Series-redundant reliability model for a complex hybrid system.

The notations used to describe the redundant structures and their reliability evaluation models are presented at the end of the paper. Along with these notations we include a short nomenclature and some assumptions under which the reliability models are valid.

Typically, in this allocation process the criterion may be reliability, cost, weight, or volume. One or more criteria can be considered in an objective function, while the others may be considered constraints, as considered by [22] (pp. 331–338). In this paper, the criteria we consider are reliability and cost, and in this situation, two optimization problems are frequently encountered in practice:

1. Minimizing the cost of the redundant system for which a required reliability must be achieved;
2. Maximizing the reliability of the system within a maximum allowed cost.

In both cases, from the mathematical point of view, one must solve an optimization problem with an objective function and constrains. More exactly, for the first problem, one must minimize the cost function:

$$C_{rs} = f(C_1,\ C_2,\ \cdots, C_n) = \sum_{i=1}^{n} C_i \tag{1}$$

with the constraint of reliability:

$$R_{rs} = \prod_{i=1}^{n} R_i \geq R^*. \tag{2}$$

For the second problem, one must maximize the reliability function:

$$R_{rs} = f(R_1,\ R_2,\ \cdots, R_n) = \prod_{i=1}^{n} R_i \tag{3}$$

with the cost constraint:

$$\sum_{i=1}^{n} C_i \leq C^*. \tag{4}$$

For example, when for all the subsystems an active redundancy is considered, for the redundant system a series-parallel reliability model results. Thus, the cost and reliability functions can be expressed by the equations:

$$C_{rs} = \sum_{i=1}^{n} c_i k_i \tag{5}$$

$$R_{rs} = 1 - \prod_{i=1}^{n} (1 - r_i)^{k_i} \tag{6}$$

Thus, we have to determine the values $k_1, k_2, \ldots, k_n$ that minimize the cost function in Equation (5) with the reliability constraint in Equation (2), or maximize the reliability function in Equation (6) with the cost constraint in Equation (4), as the case may be.

3. Types of Redundancy

To cover most situations encountered in practice, the following types of redundancy are considered in this study, namely:

- active redundancy $(tr = \mathrm{A})$;
- passive redundancy (or cold standby redundancy) $(tr = \mathrm{B})$;
- hybrid standby redundancy with a hot reserve $(tr = \mathrm{C})$ or a warm one $(tr = \mathrm{D})$ and possibly other cold ones;
- hybrid redundancy consisting of a TMR structure with control facilities and possibly cold reserves $(tr = \mathrm{E})$;
- static redundancy: TMR or 5MR $(tr = \mathrm{F})$;
- reconfigurable TMR/Simplex type structure with possible other cold-maintained spare components $(tr = \mathrm{G})$;
- reconfigurable TMR/Duplex type structure with possible other cold-maintained spare components $(tr = \mathrm{H})$.

The reliability model and the equations used to evaluate the reliability for a subsystem, depending on the type of redundancy, are presented in this section. Since the time to failure for a component is assumed to have a negative exponential distribution, the following equations are valid:

$$r = e^{-\lambda T} \tag{7}$$

and

$$\lambda T = -\ln r \tag{8}$$

Remember that for any redundant subsystem the spare components are considered identical to the basic ones.

3.1. Active Redundancy (tr = A)

For this parallel reliability model where all components operate simultaneously, the well-known equation is applied:

$$R = 1 - (1 - r)^k,\ k = 2,\ 3,\ \ldots \tag{9}$$

3.2. Passive Redundancy (tr = B)

In this case, one component is in operation and all other identical $k - 1$ spare components are maintained in a cold state, which means that a spare component is switched off until it is needed to replace the defective one (i.e., a redundant component does not fail in cold standby mode). The following equation can be applied to this model:

$$R = \sum_{j=0}^{k-1} \frac{(\lambda T)^j}{j!} e^{-\lambda T} = r \sum_{j=0}^{k-1} \frac{(-\ln r)^j}{j!},\ k \geq 2 \tag{10}$$

Note that Equation (10) is the sum of the first k terms of the Poisson distribution of the parameter λT.

3.3. Hybrid Standby Redundancy with a Hot (tr = C) or a Warm (tr = D) Spare and Possibly Other Cold Ones

In this case of standby redundancy, a component is in operation, a spare component is active or kept in a warm state, and possibly other spare components are kept in cold conditions as illustrated in Figure 2.

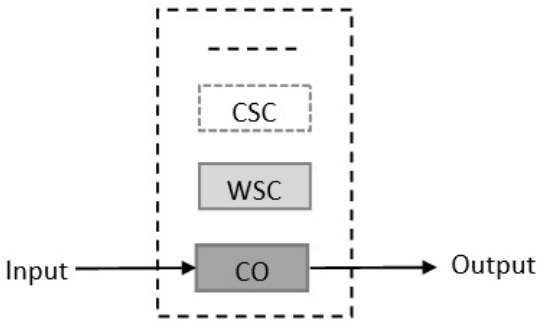


Figure 2. Standby redundancy with a hot/warm spare component and possibly other cold ones.

A warm component may fail before being put into operation and its failure rate is less than that of the same component in active mode. Therefore, let $\alpha\lambda,\ 0 < \alpha \leq 1$, be the failure rate for this reserve. For this type of redundancy, the subsystem reliability function is obtained based on the Markov method, depending on the total number of components, as shown below.

3.3.1. Case 1: $k = 2$

Consider a subsystem consisting of a component in operation and a warm-maintained reserve. The evolution of this redundant subsystem until failure is illustrated by the Markov chain presented in Figure 3.

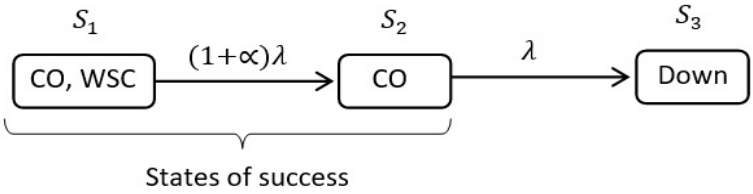


Figure 3. Markov chain for subsystem reliability evaluation ($k = 2$).

To begin with, let us refer to a general Markov model. Let $S_1,\ S_2,\ \ldots,\ S_N$ be the states of the Markov chain and $\mathbf{A} = \left[a_{x,y}\right]_{N\times N}$ be the matrix of state transition rates, where $a_{x,y},\ x \neq y$, represents the rate of transition from state S_y to state S_x, while an element of the main diagonal (i.e., $x = y$) is the negative value of the sum of all the other elements in the column.

Let $s(t)$ be the state of the subsystem at the time t, and

$$p_x(t) = \text{prob}(s(t) = S_x),\ \ x \in \{1,\ 2,\ \ldots,\ N\}. \tag{11}$$

To obtain the probability functions $p_x(t),\ x = 1 : N$, the following system of differential equations must be solved:

$$\mathbf{P'} = \mathbf{A} \times \mathbf{P}, \tag{12}$$

where $\mathbf{P} = \left[p_1(t)\ p_2(t)\ \cdots\ p_N(t)\right]^T$, and $\mathbf{P'} = \left[p'_1(t)\ p'_2(t)\ \cdots\ p'_N(t)\right]^T$.

Note that the state probabilities for $t = 0$ are also known.

Let us resume the analysis of the subsystem under study. In the Markov chain presented in Figure 3, S_1 and S_2 are successful states, while S_3 is a failure state. Thus, the reliability function of this redundant subsystem can be defined as

$$R(t) = p_1(t) + p_2(t),\ t \geq 0. \tag{13}$$

As the transition rate matrix is:

$$\mathbf{A} = \begin{bmatrix} -(1+\alpha)\lambda & 0 & 0 \\ (1+\alpha)\lambda & -\lambda & 0 \\ 0 & \lambda & 0 \end{bmatrix}, \tag{14}$$

to determine the probability functions $p_1(t)$ and $p_2(t)$, the following system of differential equations must be solved:

$$\begin{cases} p'_1(t) = -(1+\alpha)\lambda p_1(t) \\ p'_2(t) = (1+\alpha)\lambda p_1(t) - \lambda p_2(t) \end{cases} \tag{15}$$

With the initial values: $p_1(0) = 1$ and $p_2(0) = p_3(0) = 0$, by applying the Laplace transform ($\mathcal{L}$), the following system of algebraic equations results:

$$\begin{cases} sP_1(s) - 1 = -(1+\alpha)\lambda P_1(s) \\ sP_2(s) = (1+\alpha)\lambda P_1(s) - \lambda P_2(s) \end{cases} \tag{16}$$

where $P_i(s) = \mathcal{L}\left\{p_i(t)\right\},\ i \in \{1,\ 2\}$, are functions in the frequency domain, and s is the Laplace operator. Based on (16), after some algebraic operations, the following functions are obtained:

$$P_1(s) = \frac{1}{s + (1+\alpha)\lambda},\quad P_2(s) = \frac{(1+\alpha)\lambda}{s + (1+\alpha)\lambda} \cdot \frac{1}{s+\lambda} \tag{17}$$

After a partial-fraction-expansion, the function $P_2(s)$ can be expressed as follows:

$$P_2(s) = -\frac{1+\alpha}{\alpha}\frac{1}{s+(1+\alpha)\lambda} + \frac{1+\alpha}{\alpha}\frac{1}{s+\lambda} \tag{18}$$

As the function $\mathcal{R}(s) = \mathcal{L}\{R(t)\} = P_1(s) + P_2(s)$, the following expression results:

$$\mathcal{R}(s) = \frac{1+\alpha}{\alpha}\frac{1}{s+\lambda} - \frac{1}{\alpha}\frac{1}{s+(1+\alpha)\lambda} \tag{19}$$

The reliability function $R(t)$ can then be obtained by applying the inverse Laplace transform, $R(t) = \mathcal{L}^{-1}\{\mathcal{R}(s)\}$. Thus, the reliability function has the following form:

$$R(t) = \frac{1+\alpha}{\alpha}e^{-\lambda t} - \frac{1}{\alpha}e^{-(1+\alpha)\lambda t}, \quad t \geq 0, \quad 0 < \alpha \leq 1. \tag{20}$$

For a certain period of time T, the component reliability is $r = e^{-\lambda T}$, so that the subsystem reliability R as a function of r and α is given by the equation:

$$R(r,\ \alpha) = \frac{1+\alpha}{\alpha}r - \frac{1}{\alpha}r^{1+\alpha} = r + \frac{1}{\alpha}r(1-r^{\alpha}), \quad 0 < \alpha \leq 1. \tag{21}$$

For a redundancy subsystem with a larger number of components, the reliability function can be obtained based on the Markov method in the same way, but algebraic operations are more complicated. The results for the other two cases are presented below.

3.3.2. Case 2: $k = 3$

Take a redundant subsystem composed of an active component, a hot/warm spare component, and another one maintained in cold conditions. For this case, the following reliability function results:

$$R(r,\ \alpha) = \frac{(1+\alpha)^2}{\alpha^2}r - \left(\frac{1+2\alpha}{\alpha^2} - \frac{1+\alpha}{\alpha}\ln r\right)r^{1+\alpha}, \quad 0 < \alpha \leq 1. \tag{22}$$

3.3.3. Case 3: $k = 4$

For a redundant subsystem with an active component, a hot/warm spare component, and two other ones maintained in cold conditions, the reliability function is given by the following equation:

$$R(r,\ \alpha) = \frac{(1+\alpha)^3}{\alpha^3}r - \left(\frac{1+3\alpha+3\alpha^2}{\alpha^3} - \frac{1+3\alpha+2\alpha^2}{\alpha^2}\ln r + \frac{(1+\alpha)^2}{2\alpha}(\ln r)^2\right)r^{1+\alpha},\ 0 < \alpha \leq 1. \tag{23}$$

3.4. TMR Structure with Control Facilities and Cold Spare Components (tr = E)

In this case, another hybrid redundancy is considered. Thus, a redundant system is composed of a TMR structure with control facilities as a basic structure (i.e., static redundancy) and possibly one or more components maintained in cold conditions (i.e., standby redundancy). This type a hybrid redundancy is illustrated in Figure 4.

The decision logic works on the principle of majority logic, 2 out of 3, called voter and represented by the symbol V in Figure 4. When one of the three components in operation (CO_1, CO_2 or CO_3) fails, an error signal indicates the faulty component. Thus, the faulty component can be replaced with a cold-maintained standby one as soon as possible. In this way, this redundant hybrid subsystem can tolerate one or more defective components, as the case may be. For additional decision and control block the failure rate, denoted by λ_{dc}, is expressed based on the basic component rate, λ. In this study, the following expression is used:

$$\lambda_{dc} = \frac{\lambda}{\beta}, \quad \beta > 1. \tag{24}$$

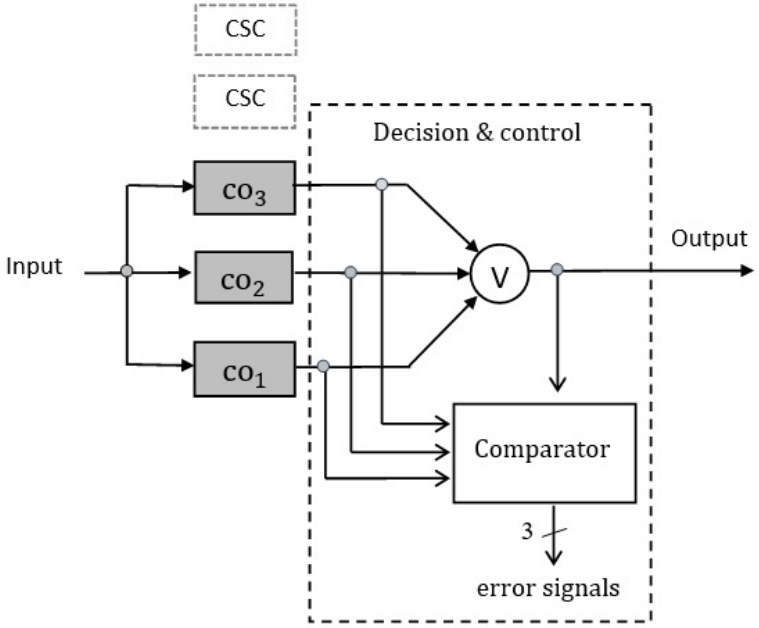


Figure 4. TMR structure with control facilities and cold spare components.

Consequently, the reliability function for logical decision and control block, denoted by r_{dc}, is expressed as:

$$r_{dc} = e^{-\lambda_{dc}T} = e^{-\frac{\lambda}{\beta}T} = \left(e^{-\lambda T}\right)^{\beta^{-1}} = r^{\beta^{-1}}, \quad \beta > 1. \tag{25}$$

3.4.1. Case 1: TMR Structure without Standby Redundancy

In case of a TMR structure without reserves (i.e., $k = 3$), the redundant subsystem can tolerate only one faulty component, so the subsystem reliability function is given by the well-known equation:

$$R(r, \beta) = (3r^2 - 2r^3)r_{dc} = (3r^2 - 2r^3)r^{\beta^{-1}}, \quad \beta > 1 \tag{26}$$

3.4.2. Case 2: TMR Structure and One Cold Spare Component

A redundant subsystem with hybrid redundancy composed of a TMR structure and one CSC (i.e., $k = 4$) may tolerate two faulty components. For a start, for the logical block of decision and control, the possibility of failure is neglected. The reliability evaluation is made based on the Markov graph given in Figure 5.

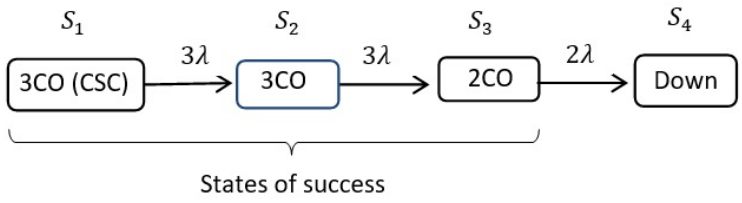


Figure 5. Markov chain for subsystem reliability evaluation ($k = 4$).

In this graph, S_1, S_2 and S_3 are successful states, while S_4 is a failure one. Given these aspects, the reliability function of this redundant subsystem is expressed as:

$$R(t) = p_1(t) + p_2(t) + p_3(t), \quad t \geq 0. \tag{27}$$

As the transition rate matrix is:

$$\mathbf{A} = \begin{bmatrix} -3\lambda & 0 & 0 & 0 \\ 3\lambda & -3\lambda & 0 & 0 \\ 0 & 3\lambda & -2\lambda & 0 \\ 0 & 0 & 2\lambda & 0 \end{bmatrix}, \tag{28}$$

by applying Equation (12) in order to determine the probability functions $p_1(t)$, $p_2(t)$ and $p_3(t)$, the next system of differential equations results:

$$\begin{cases} p_1'(t) = -3\lambda p_1(t) \\ p_2'(t) = 3\lambda p_1(t) - 3\lambda p_2(t) \\ p_3'(t) = 3\lambda p_2(t) - 2\lambda p_3(t) \end{cases} \tag{29}$$

With the initial values: $p_1(0) = 1$, and $p_2(0) = p_3(0) = 0$, by applying the Laplace transform, the following system of algebraic equations is obtained:

$$\begin{cases} sP_1(s) - 1 = -3\lambda P_1(s) \\ sP_2(\mathrm{s}) = 3\lambda P_1(s) - 3\lambda P_2(s) \\ sP_3(\mathrm{s}) = 3\lambda P_2(s) - 2\lambda P_3(s) \end{cases} \tag{30}$$

By solving the system, the following functions in the frequency domain result:

$$\begin{cases} P_1(\mathrm{s}) = \frac{1}{s+3\lambda} \\ P_2(s) = \frac{3\lambda}{(s+3\lambda)^2} \\ sP_3(s) = \frac{9\lambda^2}{(s+3\lambda)^2} \cdot \frac{1}{s+2\lambda} = \frac{9}{s+2\lambda} - \frac{9}{s+3\lambda} - \frac{9\lambda}{(s+3\lambda)^2}. \end{cases} \tag{31}$$

As the function

$$\mathcal{R}(s) = \mathcal{L}\{R(t)\} = P_1(s) + P_2(s) + P_3(s), \tag{32}$$

the following expression results:

$$\mathcal{R}(s) = \frac{9}{s+2\lambda} - \frac{8}{s+3\lambda} - \frac{6\lambda}{(s+3\lambda)^2} \tag{33}$$

The reliability function $R(t)$, obtained by applying the inverse Laplace transform, is of the form:

$$R(t) = 9\mathrm{e}^{-2\lambda t} - 8\mathrm{e}^{-3\lambda t} - 6\lambda t\mathrm{e}^{-3\lambda t}, \quad t \geq 0. \tag{34}$$

Finally, taking also into account the reliability of the decision and control logic, the subsystem reliability R as a function of r and β is given by the equation:

$$R(r,\ \beta) = (9r^2 - r^3(8 - 6\ln r))r_{dc} = (9r^2 - r^3(8 - 6\ln r))r^{\beta^{-1}}, \quad \beta > 1 \tag{35}$$

For a hybrid redundancy subsystem with a larger number of CSCs, the reliability function can be obtained by applying the Markov method in the same way, but algebraic operations are more complicated. A result obtained for another case is presented as follows.

3.4.3. Case 3: TMR Structure and Two Cold Spare Components

Take a redundant subsystem with hybrid redundancy composed of a TMR structure and two CSCs (i.e., $k = 5$). This redundant subsystem can tolerate three defective compo-

nents. A Markov-based approach similar to the one presented above gives the following subsystem reliability as a function of r and β:

$$R(r,\ \beta) = \left(27r^2 - r^3(26 - 24\ln r + 9(\ln r)^2)\right)r^{\beta^{-1}}, \quad \beta > 1 \tag{36}$$

3.5. Static Redundancy: TMR or 5MR (tr = F)

This type of redundancy refers to those subsystems for which a static redundancy with majority logic (TMR or 5MR) can be adopted, depending on the desired level of reliability. Thus, in the process of finding an optimal solution, the valid values for variable k are 1, 3 and 5.

3.5.1. Case 1: TMR Structure

This case where $k = 3$ was also considered in Section 3.4, Case 1, so that the reliability function for this redundant subsystem is given by Equation (25).

3.5.2. Case 2: 5MR Structure

When a 5MR redundancy is adopted (i.e., $k = 5$), as [22] (pp. 165–176) appreciates, the additional logic of decision and control is more complex than that used for TMR redundancy. Consequently, the failure rate, denoted by λ'_{dc}, expressed on the basis of the failure rate of the basic components, is considered of the form:

$$\lambda'_{dc} = \frac{\lambda}{\gamma}, \quad \gamma > 1 \tag{37}$$

where the reduction factor γ is lower than the reduction factor β used for the TMR redundancy. Because the 5MR structure can tolerate two defective components, the reliability of the subsystem can be calculated as follows:

$$\begin{aligned} R(r,\gamma) &= (r^5 + 5r^4(1-r) + 10r^3(1-r)^2)\lambda'_{dc} \\ &= \left(10r^3 - 15r^4 + 6r^5\right)r^{\gamma^{-1}},\ \gamma > 1. \end{aligned} \tag{38}$$

3.6. TMR/Simplex and Cold Standby Redundancy (tr = G)

This is another case of hybrid redundancy in which the basic structure is reconfigurable. Specifically, the redundant subsystem consists of a TMR structure with control and reconfiguration facilities and other possible CSCs, as shown in Figure 6.

If one of the three components in operation fails, the subsystem continues to operate successfully based on redundancy, and the control logic generates an error signal indicating the faulty component. The status of the active component (good or failed) is reflected by three dedicated flip-flops. For example, Figure 6 illustrates the case where components CO_1 and CO_3 work successfully and component CO_2 is defective.

When an error signal is activated, the defective component must be replaced with a spare one as soon as possible to restore the initial fault tolerance state. Let us suppose this replacement is done quickly enough so reliability is not significantly affected. When only two components remain in good state, in order to increase the reliability, it is preferable for only one component to continue to work, not both. This reconfigurable structure is known as TMR/Simplex [32] (p. 233) or TMR 3-2-1 [22] (p. 152). Note that after a component has failed, the control logic can no longer correctly indicate another fault, so the values of the status flip-flops must be preserved until the fault tolerance is restored. This is the role of the 3-input NAND logic gate in Figure 6.

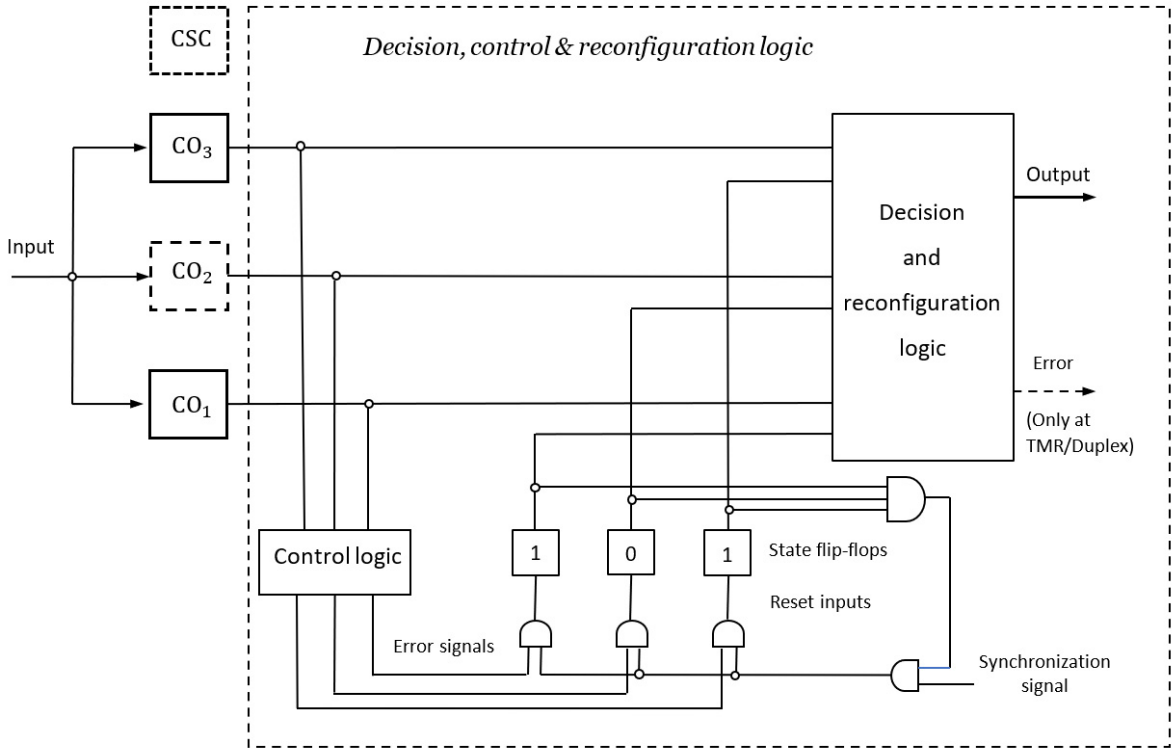


Figure 6. Reconfigurable TMR structure with cold redundancy.

For an additional decision, control and reconfiguration logic block, the faulty rate denoted by λ_{dcr} is expressed based on the basic component rate. In this study, the following equation is used:

$$\lambda_{dcr} = \frac{\lambda}{\delta}, \quad \delta > 1 \tag{39}$$

where the reduction factor δ is lower than the reduction factor β used for TMR redundancy. Consequently, the reliability function for the logic of decision, control and reconfiguration denoted by r_{dcr} is expressed as:

$$r_{dcr} = e^{-\lambda_{dcr}T} = e^{-\frac{\lambda}{\delta}T} = \left(e^{-\lambda T}\right)^{\delta^{-1}} = r^{\delta^{-1}}, \ \delta > 1. \tag{40}$$

The reliability of the redundant subsystem depends on the number of CSCs, as shown below.

3.6.1. Case 1: TMR/Simplex without Standby Redundancy

In case of TMR/Simplex redundancy without spare components (i.e., $k = 3$), the subsystem reliability function is given by the well-known equation [32], (p. 233):

$$R(r, \delta) = (1.5r - 0.5r^3)r_{dcr} = (1.5r - 0.5r^3)r^{\delta^{-1}}, \quad \delta > 1. \tag{41}$$

3.6.2. Case 2: TMR/Simplex and One Cold Reserve

For this case of hybrid redundancy, the reliability evaluation is made by applying the Markov method. For starters, for the logical block of decision, control and configuration the possibility of failure is neglected. In this condition, the evolution of the redundant subsystem to failure is illustrated by the Markov chain shown in Figure 7.

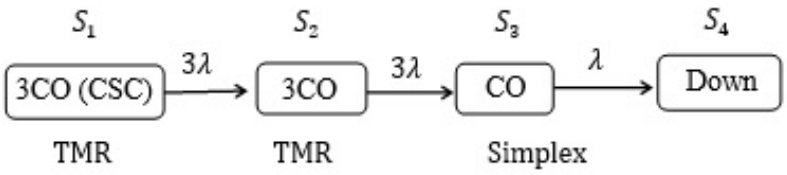


Figure 7. Markov chain for TMR/Simplex and one CSC ($k = 4$).

In this graph, S_1, S_2 and S_3 are states of success, while S_4 is a failure state. Consequently, the subsystem reliability is defined as:

$$R(t) = p_1(t) + p_2(t) + p_3(t), \ \ t \geq 0. \tag{42}$$

Since the transition rate matrix is:

$$\mathbf{A} = \begin{bmatrix} -3\lambda & 0 & 0 & 0 \\ 3\lambda & -3\lambda & 0 & 0 \\ 0 & 3\lambda & -\lambda & 0 \\ 0 & 0 & \lambda & 0 \end{bmatrix}, \tag{43}$$

based on (12), the following system of differential equations results:

$$\begin{cases} p_1'(t) = -3\lambda p_1(t) \\ p_2'(t) = 3\lambda p_1(t) - 3\lambda p_2(t) \\ p_3'(t) = 3\lambda p_2(t) - \lambda p_3(t) \end{cases} \tag{44}$$

With the initial values: $p_1(0) = 1$, and $p_2(0) = p_3(0) = 0$, by applying the Laplace transform, the following system of algebraic equations is obtained:

$$\begin{cases} sP_1(s) - 1 = -3\lambda P_1(s) \\ sP_2(s) = 3\lambda P_1(s) - 3\lambda P_2(s) \\ sP_3(s) = 3\lambda P_2(s) - \lambda P_3(s) \end{cases} \tag{45}$$

By solving this equation system, the following functions in the field of Laplace transform are obtained:

$$\begin{cases} P_1(s) = \frac{1}{s+3\lambda} \\ P_2(s) = \frac{3\lambda}{(s+3\lambda)^2} \\ sP_3(s) = \frac{9\lambda^2}{(s+3\lambda)^2} \cdot \frac{1}{s+\lambda} = \frac{9}{4(s+\lambda)} - \frac{9}{4(s+3\lambda)} - \frac{9\lambda}{2(s+3\lambda)^2} \end{cases} \tag{46}$$

The reliability function in the field of Laplace transform is:

$$\mathcal{R}(s) = P_1(s) + P_2(s) + P_3(s) = \frac{9}{4(s+\lambda)} - \frac{5}{4(s+3\lambda)} - \frac{3\lambda}{2(s+3\lambda)^2} \tag{47}$$

The reliability function $R(t)$, obtained by applying the inverse Laplace transform, is of the form:

$$R(t) = \frac{9}{4}e^{-\lambda t} - \frac{5}{4}e^{-3\lambda t} - \frac{3}{2}\lambda t e^{-3\lambda t}, \quad t \geq 0. \tag{48}$$

Finally, taking also into account the reliability of the logical block of decision, control and configuration, the reliability of the subsystem R as a function of r and δ is given by the equation:

$$\begin{aligned} R(r,\ \delta) &= \left(2.25r - r^3(1.25 - 1.5\ln r)\right) r_{dcr} \\ &= \left(2.25r - r^3(1.25 - 1.5\ln r)\right) r^{\delta^{-1}}, \quad \delta > 1 \end{aligned} \tag{49}$$

3.6.3. Case 3: TMR/Simplex and Two Cold Reserves

Take a reconfigurable subsystem with hybrid redundancy composed of a TMR/Simplex structure and two CSCs (i.e., $k = 5$). This redundant subsystem can tolerate three defective components. A Markov-based approach similar to the one presented above gives the following subsystem reliability as a function of r and δ:

$$R(r,\ \delta) = \left(\frac{27}{8}r - \frac{1}{8}\left(19 - 30\ln r + 18(\ln r)^2\right)r^3\right) r^{\delta^{-1}}, \quad \delta > 1 \tag{50}$$

3.7. TMR/Duplex and Cold Standby Redundancy ($tr = H$)

As in the previous case, the redundant subsystem has a hybrid redundancy consisting of a reconfigurable TMR structure and possibly other CSCs, as shown in Figure 6. But this reconfigurable structure also aims at high operational safety. Thus, when one component of the TMR structure fails, the other two good components are put into operation in duplex mode. Specifically, the two components operate in parallel and their outputs are compared continuously. When the two components no longer generate the same response, an error signal is activated (as shown in Figure 6), so that the operation is stopped in safe mode. This reconfigurable structure is called by [32] TMR/Duplex.

Regarding the reliability assessment, note that this redundant subsystem can tolerate the same number of faulty components as the TMR structure presented in Section 3.4 for type E redundancy. Consequently, depending on the total number of components (k), Equations (26), (35) or (36) are valid in this case as well, with the only difference that the reduction factor β is replaced by δ.

4. Related Work

The problems of maximizing reliability with a cost constraint or minimizing cost with a reliability constraint can be solved using various methods. One is by solving an analytical model based on Lagrange multipliers with an alternative indicator for reliability [4]. The resulting system of algebraic equations can be solved but involve some approximate relations which may impact the accuracy of the solution. Also, this method gives real-valued results which must be converted into integers, and this may have a strong impact on solution quality. Therefore, heuristic methods can be appropriate. For example, one such technique described by [22] (p. 335) is a greedy approach that tries to make an optimal choice at each step: starting with the minimum system design, the system reliability is increased by adding one component to the subsystem with the lowest reliability. This process is repeated as long as the cost constraint is met.

Another method described by [33] (pp. 499–532) tries to accelerate the allocation process by noticing that the subsystem with the highest reliability should have the smallest number of components, and the least reliable subsystem should have the greatest number of components. Starting with the initial system, the reliability is increased by adding one component to each subsystem as long as the cost constraint is met. For the most reliable subsystem, this is the final allocation. The process continues with the other subsystems, until no allocation is possible any longer.

Pairwise Hill Climbing (PHC) [29] adapts the idea of classic hill climbing to the reliability-cost problem. Two candidate solutions are generated for each pair of subsystems. The first candidate is created by adding one component to the first subsystem, i.e., the direct hill climbing operation. The second is created by adding one component to the first subsystem and subtracting one from the second subsystem, i.e., a swapping operation. A

hybrid approach starting from an approximate, but nearly-optimal solution given by the analytical approach, further improved by PHC was found to provide good results.

The problem can also be expressed as a quadratic unconstrained binary optimization (QUBO). This formulation has the potential of being solved by the D-Wave quantum computer as shown by [29] or [34].

The problem must be stated in the form of:

$$O(\mathbf{q}; \mathbf{a}, \mathbf{b}) = \sum_i a_i q_i + \sum_{(i,\ j)} b_{ij} q_i q_j. \tag{51}$$

The user needs to specify the parameters a_i (the weights associated with each qubit) and b_{ij} (the strengths of the couplers between qubits). The expression is minimized by quantum annealing when run on the quantum computer and the observed q_i values of either 0 or 1 represent the solution. A special procedure is required to transform the inequality constraint into additional terms to be optimized together with the main objective function in the same expression [29].

5. The Optimization Algorithms

The experimental studies presented in Section 9 are based on three approaches: a classical real-valued evolutionary algorithm, an improved evolutionary algorithm called RELIVE, that combines global search with local search, and a zero-one integer programming model, i.e., a special case of linear programming. While these techniques have been extensively used for various optimization problems, an original contribution of the current paper is the design of the objective functions corresponding to the problem under study, described in Section 6.

5.1. Classic Evolutionary Algorithm

Evolutionary algorithms (EAs) are inspired by biological natural selection [35,36]. They maintain a population of individuals (or chromosomes) which are potential solutions, i.e., different values of the $\mathbf{x}$ input of the objective function $f(\mathbf{x})$ that needs to be optimized. There are three main genetic operators which are repeatedly applied for a pre-specified number of generations or until a convergence criterion is satisfied: selection (which identifies "parents", such that individuals with better objective functions have a higher probability of being selected), crossover (which combines the genes of two parents and creates an offspring), and mutation (which may change some genes of a child before it is inserted into the new population). All these operators are stochastic, but the constant favoring of better individuals to reproduce drives the algorithm towards increasingly better solutions, while random changes in the chromosomes try to prevent it from convergence into local optima. For the experiments in Section 8, the standard evolutionary algorithm (SEA) uses the following types of operators and parameters:

- tournament selection with two individuals;
- elitism is used, i.e., the best individual is directly copied into the next generation;
- arithmetic crossover, where a child chromosome is a linear combination of the parent chromosomes, with a probability of 0.9;
- mutation by gene resetting, where the value of a randomly selected gene is set to a random number from a uniform distribution defined on its domain of definition, with a probability of 0.2;
- stopping criterion with a fixed number of generations; depending on the experiment 1000 or 10,000 generations are used.

5.2. RELIVE

The cross-generational evolutionary algorithm with local improvements (RELIVE) [4] is an original evolutionary algorithm which performs secondary local searches in addition to the main global search and includes the concept of personal improvement of individuals that survive for several generations, instead of just one. Since the lifespan of individuals is

no longer fixed, the size of the population is variable. Personal improvement is based on a number of hill climbing steps in each generation. During a generation, the individuals undergo the classic evolution based on selection, crossover and mutation. Another typical feature of RELIVE is the way in which it encourages exploration. This has proved particularly useful for difficult optimization problems such as the one addressed in our work. First, a few newly created chromosomes are added in each generation. Secondly, to generate a neighbor state in the hill climbing stage, three types of mutation are used with different probabilities: Gaussian mutation, resetting mutation, and pairwise mutation, where two genes exchange a unit, i.e., one's value is incremented and the other's value is decremented. The latter type is again specifically designed for problems involving integer solutions, such as the present one. For the experiments in Section 8, RELIVE uses the following parameter values:

- the initial size of the population is 50;
- the fraction of newly generated chromosomes in a generation is 0.25;
- the life span of an individual is 4;
- the number of neighbors generated for hill climbing is 20;
- the number of hill climbing steps is 20;
- the probability of overall mutation is 0.2, divided into:
    - Gaussian mutation, with a probability of 0.05, where the value of a randomly selected gene is set to a normal random number with the mean equal to the original gene value and a standard deviation of 2;
    - resetting mutation, with a probability of 0.05, where the value of a randomly selected gene is set to a random number from a uniform distribution defined on its domain of definition is 0.25;
    - pairwise mutation, with a probability of 0.1, where two genes exchange a unit.

For the rest of the operators RELIVE uses, like SEA, tournament selection with two individuals, elitism, arithmetic crossover, with a probability 0.9, and a maximum number of 100 or 1000 generations.

5.3. Linear Programming

Linear programming (LP) is an optimization method aimed at problems with a linear objective function and linear constraints. There are several specific LP algorithms implemented in various libraries and programs. For our experiments, *lpsolve* [28] was used, which implements an optimized version of the simplex algorithm proposed by [37]. Depending on the nature of the optimization problem, it can select either the primal or the dual method, with factorization and scaling procedures to increase numerical stability. The problem we address in this paper is in fact cast as a zero-one integer programming (01IP) problem, a special case of LP.

6. Designing the Objective Functions

6.1. Evolutionary Algorithms

6.1.1. Problem Definition

For the two evolutionary algorithms, the objective (or fitness) function closely follows the definition of the two correlated problems stated in Section 3 and repeated here for convenience.

The maximization of the reliability with a maximum cost limit can be expressed as:

$$\begin{cases} \text{Maximize} : \prod_{i=1}^{n} R_i \\ \text{subject to} : \sum_{i=1}^{n} C_i \leq C^* \end{cases} \tag{52}$$

The minimization of the cost of the redundant system with a required reliability can be expressed as:

$$\begin{cases} \text{Minimize} : \sum_{i=1}^{n} C_i \\ \text{subject to} : \prod_{i=1}^{n} R_i \geq R^* \end{cases} \tag{53}$$

As C_i and R_i are computed by means of the equations detailed in Section 3, which depend on the number of components for each subsystem, the optimization problem reduces to finding $k_1, k_2, \ldots,$ and k_n.

For the two evolutionary algorithms, the fitness functions are the expressions in (52) and (53) that need to be optimized. Since an EA maximizes the fitness function by default, in case of (53), the negative of the sum of costs is actually used as the fitness function. The encoding of the problem uses real values, thus the chromosomes have n real genes, corresponding to k_i. The domain of the genes is $[1,\ k_{max}]$, i.e., $1 \leq k_i \leq k_{max}$. It depends on the problem and therefore k_{max} needs to be chosen by the user.

6.1.2. Genotype-Phenotype Mapping

The real values involved in the evolutionary search are interpreted as integer values for k_i before the computation of the fitness function. Therefore, the first step is to round the real values to the nearest integer:

$$k_i^p = \left\lfloor k_i^g + 0.5 \right\rfloor \tag{54}$$

where k_i^g reflects the genotype (the actual value of the gene), and k_i^p reflects the phenotype (its interpretation for further use).

Because in our case studies, for some types of redundancy we limited ourselves to a certain number of spare components as sufficient, another important issue is related to the unsuitability of some values of k_i for certain subsystems. Therefore, the adjustment rules in Table 1 are used to interpret the values of k_i as valid ones.

Table 1. Adjustment rules for phenotype interpretation.

Redundancy Type	**Adjustment Rule**
$tr =$ A or $tr =$ B	No adjustment
$tr =$ C	if $k_i^p > 5$ then $k_i^p \leftarrow 5$
$tr =$ D	if $k_i^p > 4$ then $k_i^p \leftarrow 4$
$tr =$ E	if $k_i^p < 3$ then $k_i^p \leftarrow 3$ if $k_i^p > 5$ then $k_i^p \leftarrow 5$
$tr =$ F	if $k_i^p < 4$ then $k_i^p \leftarrow 3$ else $k_i^p \leftarrow 5$
$tr =$ G	if $k_i^p < 3$ then $k_i^p \leftarrow 3$ if $k_i^p > 5$ then $k_i^p \leftarrow 5$
$tr =$ H	if $k_i^p < 3$ then $k_i^p \leftarrow 3$ if $k_i^p > 5$ then $k_i^p \leftarrow 5$

It must be mentioned that trying to enforce a valid domain for each subsystem gene a priori would have caused discontinuities in the evolutionary search, would have decreased the genetic diversity, and thus would have led to inferior results.

6.1.3. Chromosom Repairing Procedure

Although expressed with a very simple equation, because of the possibly large size of a problem (e.g., $n = 50$ or $n = 100$, as considered in our case studies), the constraints are actually difficult to satisfy.

A naïve approach based on penalties for constraint violation decreases the genetic diversity to such an extent that the algorithms usually fail to find any solution at all, or find feasible solutions very far from the optimum.

Therefore, one can apply a repairing procedure for the chromosomes, such that even if a certain individual resulted from the application of the genetic operators is initially unfeasible, it can be slightly modified to become feasible. In this way, all the individuals in the population represent feasible solutions and the evolutionary algorithm focuses on optimizing the fitness function.

For the reliability maximization problem with cost constraints, a random repairing method is applied. Iteratively, a subsystem whose $k_i > 1$ is randomly selected and its k_i is decreased by 1, until the overall cost of the system becomes smaller than C^*.

Alternative methods were also attempted, but they were slower with no significant improvement of results:

- The selection of the subsystem with the highest cost. Because of the genotype-phenotype distinction, this could sometimes lead to infinite loops (e.g., the repairing procedure decrements a value, and the corresponding adjustment rule increments it);
- The selection of the subsystem with the highest reliability. This is even slower because it requires the recomputation of the system reliability after each k_i is decremented, with i from 1 to n.

The repairing procedure for the cost minimization with reliability constraints proved much more challenging. Eventually, a random repairing method was also applied in this case. Iteratively, a subsystem whose $k_i < k_{max}$ is randomly selected and its k_i is increased by 1, until the overall reliability of the system becomes greater than R^*. However, the way in which this increment affects the overall system reliability is nonlinear. Simple random selection may be very slow, because it may take several trials to choose the proper subsystem whose increased reliability may turn the overall reliability above the imposed threshold. That is why a specified number of repairing attempts trials is imposed (e.g., 10). If after these repeated trials the reliability does not exceed R^*, the individual is penalized with a very low value for its fitness function (e.g., -10^6) and thus becomes likely to be excluded from the evolutionary selection process.

Several other alternative methods were attempted as well, but they all had various drawbacks compared to the random method presented above:

- The selection of the subsystem with the lowest reliability. This method is slower and its results are not much better;
- A more elaborate method, where the number of components is increased on layers, with subsystems taken in a random order. When one layer of incrementation is completed, the next one begins. This method was the slowest, about an order of magnitude slower than random selection.

6.2. Linear Programming

The objective function is transformed in a different way in order to apply 01IP optimization. This is based on the idea proposed by [29]. The maximization of the product is equivalent to the maximization of the sum of logarithms. The desired solutions of the problem, i.e., k_i, $i = 1 : n$, are included as separate terms, one for each possible result, from 1 to k_{max}:

$$\text{Maximize} : \sum_{i=1}^{n} \sum_{j=1}^{k_{max}} x_{ij} \cdot \ln R_i(j) \tag{55}$$

where $x_{ij} \in \{0, 1\}$, $\forall i \in \{1, \ldots, n\}$, $\forall j \in \{1, \ldots, k_{max}\}$, is a binary variable that shows that for subsystem i, j components are needed to maximize reliability. The notation $R_i(j)$ signifies the reliability of subsystem i when it contains j redundant components.

For a subsystem i, only one solution is possible, i.e., its binary indicator must be 1, and the rest must be 0, and this can be written as an additional constraint:

$$\sum_{j=1}^{k_{max}} x_{ij} = 1, \forall i \in \{1, \ldots, n\} \tag{56}$$

The main constraint of the problem is also expressed by using a different term for each possible solution:

$$\sum_{i=1}^{n} \sum_{j=1}^{k_{max}} x_{ij} \cdot j \cdot c_i \ \leq C^* \tag{57}$$

For the cost minimization problem, the formulation becomes:

$$\begin{cases} \text{Maximize :} \\ \quad \sum_{i=1}^{n} \sum_{j=1}^{k_{max}} x_{ij} \cdot j \cdot c_i \\ \text{subject to :} \\ \quad \sum_{i=1}^{n} \sum_{j=1}^{k_{max}} x_{ij} \cdot \ln R_i(j) \geq R_* \\ \quad \sum_{j=1}^{k_{max}} x_{ij} = 1, \forall i \in \{1, \ldots, n\} \end{cases} \tag{58}$$

The genotype-phenotype mapping described above is also used here to compute the reliability of the subsystems by handling the k_i values that are not allowed for the corresponding subsystem type.

7. Lower Bound Solution

The minimum system design represents the first step toward achieving an optimized system design. Let us consider the optimization problem in which the required reliability R^* must be achieved at a minimum cost. To obtain a lower bound solution expressed by the values k_i', $i = 1 : n$, as the first step for optimization, an improved version of Albert's method [22,38] is used. Albert's method assumes that as spare elements are added, the reliability of the subsystems tends to become more uniform. This method involves the following steps:

Step 1. The components are renumbered so that the reliabilities are in increasing order:

$$r_1 \leq r_2 \leq \cdots \leq r_n. \tag{59}$$

Step 2. Let m be the lower limit to which all subsystems certainly require an additional allocation. According to Albert's method, the limit m is adopted so that

$$r_m \leq R^* < r_{m+1}, \tag{60}$$

or $m = n$ in case of $r_n \leq R^*$.

As an improved version, we propose that the limit m be adopted as the highest value for which the following condition is met:

$$r_m r_{m+1} \cdots r_n < R^*. \tag{61}$$

Let R be the reliability level that the first m subsystems must reach. Based on the condition that:

$$R^m r_{m+1} r_{m+2} \cdots r_n \geq R^*, \tag{62}$$

for R the following condition results:

$$R \geq \left(R^* / (r_{m+1} r_{m+2} \cdots r_n)\right)^{m^{-1}}. \tag{63}$$

Step 3. With this intermediate result (reliability value R), for each subsystem i, $i = 1{:}m$, depending on the redundancy type, the lower bound k_i' is then determined. For example, for a subsystem i with active redundancy ($tr = \mathrm{A}$), the following equations apply:

$$1 - (1 - r_i)^{k_i} \geq R = (1 - r_i)^{k_i} \leq 1 - R, \quad i = 1 : m \tag{64}$$

After applying the logarithm we get:

$$k_i \ln(1 - r_i) \leq \ln(1 - R), \tag{65}$$

and then:

$$k_i \geq \frac{\ln(1 - R)}{\ln(1 - r_i)}, \quad i = 1 : m \tag{66}$$

So, the lower bound as an integer value is:

$$k_i' = \left\lfloor \frac{\ln(1 - R)}{\ln(1 - r_i)} \right\rfloor + 1, \quad i = 1 : m \tag{67}$$

where the equations are too complicated, the lower bound is determined iteratively, and not algebraically.

For other components with higher reliability, the lower bound corresponds to the non-redundant variant, so that:

$$k_i' = 1, \quad i = m + 1 : n \tag{68}$$

Based on this lower bound solution, the search for an optimal solution can decrease significantly.

8. Experimental Results

In order to evaluate the effectiveness of the proposed algorithms, a large number of optimization problems of the order of thousands were analyzed. For all these optimization problems, all eight types of redundancy presented in Section 3 are considered. For any of the n subsystems, the type of redundancy is randomly generated based on the predetermined weights, as shown in Table 2.

Table 2. Weights for types of redundancy considered in experimental studies.

Type of Redundancy	A, B, C, D	E, F, G, H
Weight	15%	10%

Component reliabilities and costs are also randomly generated. In terms of cost, the values are in the range of $[1,\ 50]$ units for all n subsystems. In terms of reliability, the value ranges depend on the type of redundancy, as shown in Table 3.

Table 3. Value ranges for component reliability by type of redundancy.

Type of Redundancy	A, B, C, D	E, F, G, H
Weight	$[0.9,\ 1)$	$[0.95,\ 1)$

Regarding the coefficient α and the reduction factors β and δ, the values are randomly generated in the ranges:

$$0 < \alpha < 1, \quad 50 \leq \beta \leq 100, \quad 40 \leq \delta \leq 80 \tag{69}$$

In the case of type F redundancy subsystems, the value of the reduction factor γ is taken as half of the value for β ($\gamma = \beta/2$).

For the optimization problems we address, two levels of complexity were taken into account, when $n = 50$ and $n = 100$. For each case, extensive experimental studies were performed, including thousands of optimization problems.

For each reliability model, the proposed algorithms were tested taking into account both optimization problems. Specifically, for any reliability model, the study on the optimal allocation of redundancy was conducted in this way. First, the issue of redundancy allocation is considered to maximize system reliability at a maximum allowable cost $C^* = 3 \times C_{ns}$. Let R_{max} be the maximum system reliability obtained in this way. Then, another redundancy allocation problem is solved to obtain the required reliability $R^* = R_{max}$ at a minimum cost. In this way, either the solution from the first optimization problem is validated, or an improved solution is obtained.

This is the final allocation that we consider, reflected by the vector $\mathbf{k}$ and for which the reliability and cost are R_{rs} and C_{rs}, respectively. For any allocation solution, the redundancy efficiency is then calculated as follows:

$$Ef = \frac{1 - R_{sn}}{1 - R_{rs}}. \tag{70}$$

Efficiency is a more intuitive indicator that shows how often the risk of a failure for the redundant system decreases compared to the basic, non-redundant one.

To illustrate this approach, the numerical results of four experimental studies (problems $P_1 - P_4$) are presented below. First, two reliability models for a system with 50 subsystems are considered (problems P_1 and P_2). All the details of these models are presented in Tables 4 and 5.

Each problem is defined by a set of n tuples corresponding to the parameters of its subsystems. In Table 4, we define a problem with 50 subsystems, therefore we have 50 tuples. The first number in the tuple, i, goes from 1 to 50. The second item of a tuple is the subsystem type. It is identified by a letter following the convention defined in Section 3. For example, the first tuple (1: D, 0.989, 39; α = 0.55) has tr_1 = D, which corresponds to hybrid standby redundancy with a warm reserve and possibly other cold ones. The following two numbers identify the reliability and the cost of a single component. Again, for the first tuple, the reliability is r_1 = 0.989 and the cost is c_1 = 39.

Table 4. Problem P_1 for $n = 50$ subsystems.

Structural Details: Tuples of $(i: tr_i, r_i, c_i)$ Extended with Parameters α_i, β_i, or as Appropriate, i = 1:n.
(1: D, 0.989, 39; α = 0.55), (2: C, 0.958, 25), (3: C, 0.905, 41), (4: E, 0.952, 46; β = 50), (5: C, 0.975, 44), (6: A, 0.984, 14), (7: D, 0.939, 43; α = 0.86), (8: A, 0.944, 13), (9: G, 0.987, 48; δ = 74), (10: A, 0.914, 9), (11: H, 0.955, 32; δ = 65), (12: A, 0.986, 41), (13: D, 0.957, 16; α = 0.84), (14: D, 0.920, 1; α = 0.31), (15: C, 0.913, 27), (16: A, 0.985, 8), (17: A, 0.902, 9), (18: F, 0.956, 26; β = 80, γ = 40), (19: B, 0.910, 32), (20: F, 0.986, 42; β = 95, γ = 48), (21: F, 0.968, 47; β = 80, γ = 40), (22: D, 0.965, 47; α = 0.24), (23: H, 0.981, 31; δ = 72), (24: H, 0.982, 31; δ = 53), (25: F, 0.953, 45; β = 77, γ = 39), (26: B, 0.959, 18), (27: H, 0.962, 13; δ = 49), (28: E, 0.974, 46; β = 98), (29: C, 0.915, 26), (30: D, 0.983, 18; α = 0.74), (31: H, 0.975, 8; δ = 47), (32: A, 0.988, 12), (33: A, 0.971, 21), (34: C, 0.909, 17), (35: C, 0.953, 7), (36: C, 0.926, 7), (37: D, 0.989, 8; α = 0.74), (38: C, 0.906, 43), (39: H, 0.971, 11; δ = 66), (40: C, 0.944, 16), (41: E, 0.989, 21; β = 79), (42: A, 0.907, 36), (43: B, 0.942, 5), (44: C, 0.975, 18), (45: F, 0.961, 42; β = 95, γ = 48), (46: G, 0.979, 8; δ = 60), (47: E, 0.970, 38; β = 82), (48: H, 0.952, 23; δ = 48), (49: G, 0.958, 15; δ = 68), (50: C, 0.975, 7)
$C_{ns} = 1241$, $C^* = 3 \times C_{ns} = 3723$

Table 5. Problem P_2 for $n = 50$ subsystems.

Structural Details: Tuples of $(i: tr_i, r_i, c_i)$ Extended with Parameters α_i, β_i, or as Appropriate, i = 1:n.
(1: D, 0.925, 39; α = 0.42), (2: B, 0.985, 31), (3: E, 0.968, 29; β = 67), (4: A, 0.969, 35), (5: B, 0.904, 36), (6: A, 0.909, 18), (7: F, 0.973, 6; β = 92, γ = 46), (8: C, 0.976, 10), (9: C, 0.947, 19), (10: C, 0.940, 33), (11: A, 0.931, 22), (12: G, 0.970, 35; δ = 62), (13: H, 0.966, 14; δ = 69), (14: B, 0.989, 31), (15: A, 0.945, 41), (16: C, 0.974, 17), (17: B, 0.980, 47), (18: H, 0.972, 4; δ = 79), (19: C, 0.917, 44), (20: B, 0.902, 32), (21: B, 0.981, 1), (22: C, 0.983, 34), (23: F, 0.983, 12; β = 92, γ = 46), (24: G, 0.960, 12; δ = 54), (25: D, 0.936, 28; α = 0.41), (26: G, 0.965, 11; δ = 56), (27: F, 0.976, 7; β = 53, γ = 26), (28: B, 0.978, 10), (29: H, 0.972, 21; δ = 55), (30: C, 0.980, 2), (31: G, 0.975, 46; δ = 41), (32: B, 0.901, 46), (33: H, 0.972, 26; δ = 56), (34: C, 0.928, 7), (35: A, 0.909, 5), (36: A, 0.977, 49), (37: D, 0.973, 22; α = 0.72), (38: C, 0.918, 42), (39: A, 0.930, 29), (40: B, 0.986, 37), (41: G, 0.968, 37; δ = 60), (42: G, 0.977, 31; δ = 41), (43: F, 0.981, 41; β = 84, γ = 42), (44: G, 0.975, 33; δ = 40), (45: B, 0.975, 25), (46: E, 0.965, 37; β = 81), (47: B, 0.941, 20), (48: F, 0.979, 8; β = 86, γ = 43), (49: F, 0.964, 40; β = 58, γ = 29), (50: E, 0.967, 25; β = 64)
$C_{ns} = 1287$, $C^* = 3 \times C_{ns} = 3861$

The rest of the parameters depend on the subsystem type. They were defined in the mathematical description in Sections 3.1–3.7, but for convenience we include a summary here with the list of the parameters used for each type of subsystems:

- active redundancy ($tr = \mathrm{A}$), passive redundancy (or cold standby redundancy) ($tr = \mathrm{B}$), and hybrid standby redundancy with a hot reserve ($tr = \mathrm{C}$) and possibly other cold ones: no additional parameters;
- hybrid standby redundancy with a warm reserve ($tr = \mathrm{D}$) and possibly other cold ones: parameter α (the coefficient of reduction of the failure rate for a warm-maintained reserve compared to the failure rate of the component in operation);
- hybrid redundancy consisting of a TMR structure with control facilities and possibly cold reserves ($tr = \mathrm{E}$): parameter β (the reduction factor used to express the failure rate of the decision and control logic of a TMR structure based on the failure rate of the basic components);
- static redundancy: TMR or 5MR ($tr = \mathrm{F}$): parameters β (as above) and γ (the reduction factor used to express the failure rate of the decision and control logic of a 5MR structure based on the failure rate of the basic components);

- reconfigurable TMR/Simplex type structure with possible other cold-maintained spare components ($tr = \mathrm{G}$) and reconfigurable TMR/Duplex type structure with possible other cold-maintained spare components ($tr = \mathrm{H}$): parameter δ (the reduction factor used to express the failure rate of the decision, control and reconfiguration logic of a TMR/Simplex or a TMR/Duplex structure based on the failure rate of the basic components).

For example, in Table 4, since subsystem 1 is of type D, its parameter α_1 is 0.55. Since subsystem 4 is of type E, its parameter β_4 is 50. The subscripts were omitted to avoid cluttering the table, but the parameters have distinct values for each subsystem, i.e., they are α_i, β_i, γ_i or δ_i.

On the last line, one can see the cost of the non-redundant system C_{ns} and the maximum allowable cost of the system C^*, chosen to be three times greater than C_{ns}. C^* could have in fact any value, but greater values do not make the problem harder, because the main difficulty lies in finding the proper distribution of redundant components in the "upper" part of the allocation. Greater values for C^* would lead to a certain number of redundant components included for all subsystems, and then the main issue would also lie in this "upper" part of the allocation.

The redundancy allocation for these problems generated by the three proposed algorithms after the first optimization process, that tries to maximize system reliability at a maximum allowable cost C^*, is presented in Tables 6 and 7.

Table 6. Best solutions to problem P_1 after first optimization (maximizing reliability under cost constraint: $C^* = 3723$).

Algorithm	Optimal Allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{rs}	Ef
SEA	2, 2, 3, 4, 2, 2, 3, 4, 3, 5, 4, 2, 3, 3, 3, 3, 4, 3, 3, 3, 3, 2, 3, 3, 3, 2, 5, 4, 3, 2, 4, 3, 2, 3, 3, 4, 4, 3, 4, 3, 3, 3, 3, 3, 3, 4, 3, 4, 4, 2	3719	0.973398	33.714
RELIVE	2, 3, 3, 4, 2, 2, 3, 3, 3, 4, 4, 2, 3, 4, 3, 2, 4, 3, 3, 3, 3, 2, 3, 3, 3, 3, 4, 4, 3, 2, 4, 2, 2, 3, 3, 4, 2, 3, 4, 3, 3, 3, 3, 2, 3, 4, 4, 4, 4, 3	3719	0.977724	40.260
LP	2, 3, 3, 4, 2, 2, 3, 3, 3, 4, 4, 2, 3, 4, 3, 2, 4, 3, 3, 3, 3, 2, 3, 3, 3, 3, 4, 4, 3, 2, 4, 2, 2, 3, 3, 4, 2, 3, 4, 3, 3, 3, 3, 2, 3, 4, 4, 4, 4, 3	3719	0.977724	40.260

Table 7. Best solutions to problem P_2 after first optimization (maximizing reliability under cost constraint: $C^* = 3861$).

Algorithm	Optimal Allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{rs}	Ef
SEA	3, 2, 4, 2, 3, 3, 3, 2, 3, 3, 4, 4, 5, 2, 3, 3, 2, 5, 3, 3, 8, 2, 3, 4, 3, 4, 3, 2, 4, 5, 3, 3, 4, 5, 4, 2, 4, 3, 3, 2, 3, 3, 3, 3, 2, 4, 3, 3, 3, 4	3856	0.978930	41.911
RELIVE	3, 2, 4, 3, 3, 4, 3, 3, 3, 3, 3, 4, 4, 2, 3, 3, 2, 5, 3, 3, 4, 2, 3, 4, 3, 4, 3, 2, 4, 3, 3, 3, 4, 4, 4, 2, 2, 3, 3, 2, 4, 3, 3, 3, 2, 4, 3, 3, 3, 4	3861	0.981474	47.666
LP	3, 2, 4, 3, 3, 4, 3, 3, 3, 3, 3, 4, 4, 2, 3, 3, 2, 5, 3, 3, 4, 2, 3, 4, 3, 4, 3, 2, 4, 3, 3, 3, 4, 4, 4, 2, 2, 3, 3, 2, 4, 3, 3, 3, 2, 4, 3, 3, 3, 4	3861	0.981474	47.666

The solutions after the second optimization process trying to minimize the cost under the reliability constraint $R_{rs} \geq R^* = R_{max}$ are presented in Tables 8 and 9.

Table 8. Best solutions to problem P_1 after second optimization (minimizing cost under reliability constraint R^*).

Algorithm	Optimal Allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{rs}	Ef
SEA $R^* = 0.973398$	2, 3, 3, 4, 2, 2, 3, 3, 3, 3, 4, 2, 3, 4, 3, 3, 4, 3, 3, 3, 3, 2, 3, 3, 3, 3, 5, 3, 3, 2, 4, 2, 2, 3, 3, 3, 2, 3, 4, 3, 3, 3, 3, 3, 3, 4, 3, 4, 4, 3	3658	0.973465	33.798
RELIVE $R^* = 0.977724$	2, 3, 3, 4, 2, 2, 3, 3, 3, 4, 4, 2, 3, 4, 3, 2, 4, 3, 3, 3, 3, 2, 3, 3, 3, 3, 4, 4, 3, 2, 4, 2, 2, 3, 3, 4, 2, 3, 4, 3, 3, 3, 3, 2, 3, 4, 4, 4, 4, 3	3719	0.977724	40.260
LP $R^* = 0.977724$	2, 3, 3, 4, 2, 2, 3, 3, 3, 4, 4, 2, 3, 4, 3, 2, 4, 3, 3, 3, 3, 2, 3, 3, 3, 3, 4, 4, 3, 2, 4, 2, 2, 3, 3, 4, 2, 3, 4, 3, 3, 3, 3, 2, 3, 4, 4, 4, 4, 3	3719	0.977724	40.260

Table 9. Best solutions to problem P_2 after second optimization (minimizing cost under reliability constraint R^*).

Algorithm	Optimal Allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{rs}	Ef
SEA $R^* = 0.978930$	3, 2, 4, 3, 3, 4, 3, 2, 3, 3, 3, 3, 4, 2, 3, 2, 2, 5, 3, 3, 2, 2, 3, 4, 3, 4, 3, 2, 4, 4, 3, 3, 4, 3, 5, 2, 3, 3, 3, 2, 4, 3, 3, 3, 2, 4, 3, 3, 3, 4	3819	0.979250	42.556
RELIVE $R^* = 0.981474$	3, 2, 4, 3, 3, 4, 3, 3, 3, 3, 3, 4, 4, 2, 3, 3, 2, 5, 3, 3, 4, 2, 3, 4, 3, 4, 3, 2, 4, 3, 3, 3, 4, 4, 4, 2, 2, 3, 3, 2, 4, 3, 3, 3, 2, 4, 3, 3, 3, 4	3861	0.981474	47.666
LP $R^* = 0.981474$	3, 2, 4, 3, 3, 4, 3, 3, 3, 3, 3, 4, 4, 2, 3, 3, 2, 5, 3, 3, 4, 2, 3, 4, 3, 4, 3, 2, 4, 3, 3, 3, 4, 4, 4, 2, 2, 3, 3, 2, 4, 3, 3, 3, 2, 4, 3, 3, 3, 4	3861	0.981474	47.666

For the second experiment, more complex reliability models corresponding to a system with 100 subsystems are considered (problems P_3 and P_4). These models are presented in Tables 10 and 11.

Table 10. Problem P_3 for $n = 100$ subsystems.

Structural Details: Tuples of $(i: tr_i, r_i, c_i)$ Extended with Parameters $\alpha_i, \beta_i, \gamma_i$ or as Appropriate, $i = 1{:}n$.
(1: F, 0.987, 9; $\beta = 75$, $\gamma = 38$), (2: H, 0.970, 16; $\delta = 40$), (3: F, 0.959, 20; $\beta = 82$, $\gamma = 41$), (4: C, 0.984, 40), (5: B, 0.919, 23), (6: D, 0.953, 9; $\alpha = 0.34$), (7: G, 0.985, 8; $\delta = 44$), (8: A, 0.966, 23), (9: H, 0.978, 38; $\delta = 48$), (10: B, 0.908, 5), (11: C, 0.919, 23), (12: A, 0.946, 18), (13: A, 0.969, 42), (14: F, 0.970, 15; $\beta = 94$, $\gamma = 47$), (15: B, 0.921, 17), (16: B, 0.913, 32), (17: D, 0.905, 15; $\alpha = 0.41$), (18: H, 0.958, 14; $\delta = 58$), (19: B, 0.963, 12), (20: B, 0.930, 29), (21: A, 0.954, 18), (22: C, 0.989, 27), (23: A, 0.990, 7), (24: C, 0.983, 23), (25: D, 0.928, 10; $\alpha = 0.22$), (26: E, 0.958, 13; $\beta = 93$), (27: A, 0.962, 25), (28: F, 0.967, 20; $\beta = 53$, $\gamma = 27$), (29: G, 0.970, 36; $\delta = 67$), (30: B, 0.972, 20), (31: C, 0.943, 23), (32: G, 0.982, 43; $\delta = 58$), (33: H, 0.978, 45; $\delta = 64$), (34: B, 0.952, 20), (35: A, 0.944, 7), (36: C, 0.969, 19), (37: F, 0.953, 43; $\beta = 57$, $\gamma = 29$), (38: G, 0.953, 18; $\delta = 47$), (39: H, 0.987, 25; $\delta = 54$), (40: A, 0.940, 25), (41: B, 0.962, 43), (42: H, 0.958, 31; $\delta = 77$), (43: A, 0.947, 26), (44: E, 0.984, 48; $\beta = 57$), (45: E, 0.969, 6; $\beta = 87$), (46: A, 0.900, 46), (47: C, 0.945, 47), (48: G, 0.967, 8; $\delta = 52$), (49: F, 0.961, 27; $\beta = 64$, $\gamma = 32$), (50: E, 0.971, 44; $\beta = 82$), (51: B, 0.912, 47), (52: F, 0.968, 34; $\beta = 52$, $\gamma = 26$), (53: G, 0.978, 19; $\delta = 51$), (54: E, 0.966, 32; $\beta = 69$), (55: B, 0.946, 35), (56: C, 0.983, 32), (57: H, 0.970, 10; $\delta = 50$), (58: D, 0.926, 46; $\alpha = 0.61$), (59: H, 0.975, 30; $\delta = 77$), (60: D, 0.902, 10; $\alpha = 0.99$), (61: D, 0.982, 33; $\alpha = 0.30$), (62: A, 0.940, 38), (63: C, 0.922, 37), (64: F, 0.986, 19; $\beta = 78$, $\gamma = 39$), (65: G, 0.975, 32; $\delta = 59$), (66: D, 0.938, 30; $\alpha = 0.22$), (67: B, 0.974, 22), (68: H, 0.958, 22; $\delta = 70$), (69: E, 0.951, 9; $\beta = 75$), (70: G, 0.969, 48; $\delta = 77$), (71: D, 0.905, 38; $\alpha = 0.21$), (72: E, 0.989, 47; $\beta = 64$), (73: H, 0.962, 38; $\delta = 63$), (74: B, 0.923, 37), (75: H, 0.976, 36; $\delta = 53$), (76: A, 0.937, 36), (77: B, 0.942, 2), (78: C, 0.913, 8), (79: E, 0.968, 18; $\beta = 69$), (80: C, 0.928, 14), (81: B, 0.962, 16), (82: C, 0.924, 17), (83: A, 0.913, 42), (84: A, 0.987, 41), (85: A, 0.960, 22), (86: D, 0.902, 39; $\alpha = 0.72$), (87: H, 0.953, 24; $\delta = 54$), (88: B, 0.925, 13), (89: H, 0.953, 35; $\delta = 65$), (90: E, 0.972, 24; $\beta = 86$), (91: D, 0.924, 9; $\alpha = 0.48$), (92: B, 0.971, 46), (93: H, 0.969, 37; $\delta = 66$), (94: D, 0.980, 15; $\alpha = 0.11$), (95: E, 0.972, 41; $\beta = 80$), (96: B, 0.922, 6), (97: E, 0.988, 44; $\beta = 54$), (98: C, 0.955, 7), (99: F, 0.960, 16; $\beta = 90$, $\gamma = 45$), (100: A, 0.904, 25)
$C_{ns} = 2579,\ C^* = 3 \times C_{ns} = 7737$

Table 11. Problem P_4 for $n = 100$ subsystems.

Structural Details: Tuples of $(i: tr_i, r_i, c_i)$ Extended with Parameters $\alpha_i, \beta_i, \gamma_i$ or as Appropriate, $i = 1{:}n$.
(1: D, 0.974, 45; $\alpha = 0.98$), (2: B, 0.902, 13), (3: C, 0.955, 24), (4: D, 0.958, 21; $\alpha = 0.91$), (5: E, 0.954, 39; $\beta = 82$), (6: A, 0.923, 46), (7: D, 0.952, 8; $\alpha = 0.23$), (8: B, 0.900, 33), (9: A, 0.926, 19), (10: D, 0.933, 3; $\alpha = 0.55$), (11: D, 0.973, 4; $\alpha = 0.13$), (12: E, 0.976, 2; $\beta = 100$), (13: D, 0.912, 12; $\alpha = 0.43$), (14: G, 0.963, 19; $\delta = 45$), (15: B, 0.975, 27), (16: D, 0.985, 11; $\alpha = 0.23$), (17: C, 0.984, 34), (18: B, 0.940, 47), (19: F, 0.981, 35; $\beta = 79$, $\gamma = 40$), (20: F, 0.961, 20; $\beta = 79$, $\gamma = 39$), (21: D, 0.929, 17; $\alpha = 0.36$), (22: H, 0.989, 7; $\delta = 63$), (23: E, 0.977, 1; $\beta = 57$), (24: A, 0.943, 44), (25: F, 0.965, 40; $\beta = 97$, $\gamma = 48$), (26: E, 0.982, 34; $\beta = 97$), (27: F, 0.974, 49; $\beta = 79$, $\gamma = 39$), (28: H, 0.969, 12; $\delta = 42$), (29: D, 0.949, 45; $\alpha = 0.44$), (30: G, 0.977, 11; $\delta = 56$), (31: D, 0.915, 2; $\alpha = 0.48$), (32: C, 0.975, 10), (33: A, 0.904, 10), (34: A, 0.928, 16), (35: H, 0.976, 49; $\delta = 65$), (36: E, 0.958, 25; $\beta = 55$), (37: D, 0.962, 47; $\alpha = 0.15$), (38: B, 0.909, 1), (39: H, 0.960, 37; $\delta = 44$), (40: B, 0.923, 49), (41: C, 0.907, 32), (42: E, 0.985, 49; $\beta = 63$), (43: B, 0.918, 4), (44: F, 0.964, 38; $\beta = 90$, $\gamma = 45$), (45: A, 0.952, 36), (46: B, 0.945, 41), (47: C, 0.906, 16), (48: D, 0.915, 24; $\alpha = 0.70$), (49: B, 0.905, 21), (50: A, 0.902, 20), (51: C, 0.969, 15), (52: H, 0.964, 24; $\delta = 51$), (53: D, 0.916, 44; $\alpha = 0.68$), (54: E, 0.973, 37; $\beta = 53$), (55: C, 0.945, 13), (56: D, 0.976,38; $\alpha = 0.23$), (57: D, 0.931, 13; $\alpha = 0.09$), (58: B, 0.912, 30), (59: F, 0.960, 31; $\beta = 71$, $\gamma = 35$), (60: A, 0.925, 5), (61: B, 0.958, 46), (62: E, 0.954, 46; $\beta = 57$), (63: F, 0.968, 38; $\beta = 85$, $\gamma = 43$), (64: B, 0.955, 8), (65: H, 0.958, 1; $\delta = 59$), (66: B, 0.988, 44), (67: D, 0.954, 42; $\alpha = 0.19$), (68: C, 0.974, 46), (69: G, 0.977, 19; $\delta = 47$), (70: D, 0.958, 3; $\alpha = 0.04$), (71: A, 0.922, 13), (72: A, 0.975, 33), (73: C, 0.918, 10), (74: D, 0.946, 36; $\alpha = 0.42$), (75: C, 0.918, 38), (76: H, 0.968, 18; $\delta = 70$), (77: F, 0.981, 3; $\beta = 93$, $\gamma = 46$), (78: H, 0.963, 12; $\delta = 78$), (79: A, 0.981, 8), (80: D, 0.980, 48; $\alpha = 0.97$), (81: B, 0.967, 19), (82: C, 0.939, 26), (83: F, 0.967, 40; $\beta = 55$, $\gamma = 27$), (84: C, 0.947, 25), (85: D, 0.982, 46; $\alpha = 0.07$), (86: E, 0.982, 28; $\beta = 84$), (87: G, 0.976, 15; $\delta = 66$), (88: D, 0.941, 22; $\alpha = 0.44$), (89: F, 0.983, 3; $\beta = 97$, $\gamma = 49$), (90: C, 0.972, 12), (91: A, 0.976, 13), (92: B, 0.950, 18), (93: D, 0.976, 20; $\alpha = 0.07$), (94: G, 0.989, 32; $\delta = 42$), (95: H, 0.974, 3; $\delta = 66$), (96: E, 0.989, 36; $\beta = 93$), (97: G, 0.967, 11; $\delta = 45$), (98: H, 0.974, 46; $\delta = 68$), (99: G, 0.956, 38; $\delta = 74$), (100: G, 0.974, 42; $\delta = 73$)
$C_{ns} = 2506,\ C^* = 3 \times C_{ns} = 7518$

The numerical results after the two optimization processes described above are presented in Tables 12–15.

Table 12. Best solutions to problem P_3 after first optimization (maximizing reliability under cost constraint: $C^* = 7737$).

Algorithm	**Optimal Allocation: $k_1, k_2, \ldots, k_n$**	C_{rs}	R_{rs}	Ef
SEA	5, 3, 3, 2, 4, 3, 5, 3, 3, 7, 4, 4, 2, 3, 3, 3, 4, 3, 3, 3, 2, 4, 7, 2, 3, 4, 3, 3, 3, 5, 2, 3, 3, 4, 3, 2, 3, 3, 5, 3, 2, 2, 3, 2, 3, 3, 3, 2, 4, 3, 3, 3, 3, 3, 4, 2, 2, 4, 3, 4, 4, 3, 2, 4, 3, 3, 2, 2, 4, 5, 3, 3, 3, 3, 3, 3, 3, 5, 3, 3, 5, 2, 3, 3, 2, 2, 3, 4, 2, 3, 3, 4, 2, 3, 3, 3, 4, 3, 5, 3, 3	7722	0.894261	9.375
RELIVE	3, 4, 3, 2, 3, 3, 4, 2, 3, 4, 3, 3, 3, 3, 3, 3, 4, 4, 3, 3, 3, 2, 2, 2, 3, 5, 2, 3, 3, 2, 3, 3, 4, 3, 2, 3, 3, 3, 3, 3, 2, 4, 3, 3, 5, 3, 2, 4, 3, 3, 3, 3, 3, 4, 3, 3, 2, 4, 2, 3, 4, 2, 3, 2, 3, 3, 2, 2, 4, 5, 4, 3, 3, 4, 2, 4, 3, 8, 4, 4, 4, 2, 3, 3, 2, 3, 3, 4, 3, 4, 4, 3, 2, 4, 2, 4, 3, 3, 3, 4	7737	0.927214	13.619
LP	3, 4, 3, 2, 3, 3, 4, 2, 3, 3, 3, 3, 2, 3, 3, 3, 3, 4, 2, 3, 3, 2, 2, 2, 3, 4, 3, 3, 3, 2, 3, 3, 3, 3, 3, 2, 3, 4, 3, 3, 2, 4, 3, 3, 4, 3, 3, 4, 3, 4, 3, 3, 3, 4, 2, 2, 4, 3, 4, 4, 2, 3, 3, 3, 3, 3, 2, 4, 4, 3, 3, 3, 4, 3, 3, 3, 3, 4, 4, 3, 2, 3, 3, 2, 3, 3, 4, 3, 4, 4, 3, 2, 4, 2, 4, 3, 3, 3, 3, 3	7737	0.947769	18.979

Table 13. Best solutions to problem P_4 after first optimization (maximizing reliability under cost constraint: $C^* = 7518$).

Algorithm	Optimal Allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{rs}	Ef
SEA	2, 3, 3, 3, 4, 3, 4, 4, 3, 3, 4, 4, 4, 3, 3, 3, 2, 2, 3, 3, 4, 3, 4, 2, 3, 4, 3, 4, 2, 4, 4, 4, 4, 5, 3, 4, 2, 5, 4, 3, 3, 3, 8, 3, 3, 2, 3, 3, 4, 4, 3, 3, 3, 3, 3, 2, 4, 2, 3, 3, 2, 3, 3, 2, 5, 2, 2, 2, 5, 3, 3, 4, 5, 2, 3, 5, 3, 5, 3, 2, 3, 5, 3, 3, 3, 3, 3, 3, 2, 3, 2, 3, 3, 2, 4, 5, 3, 5, 4, 3, 3	7499	0.930610	14.281
RELIVE	3, 3, 2, 3, 4, 3, 3, 3, 4, 4, 2, 5, 4, 4, 2, 2, 2, 2, 3, 3, 3, 4, 3, 2, 3, 3, 3, 5, 3, 4, 4, 3, 5, 4, 3, 4, 2, 6, 4, 3, 4, 3, 4, 3, 3, 2, 4, 3, 3, 4, 3, 4, 3, 3, 3, 2, 3, 3, 3, 4, 3, 4, 3, 2, 5, 2, 3, 2, 4, 4, 4, 2, 4, 3, 3, 4, 5, 5, 3, 2, 2, 3, 3, 3, 2, 4, 4, 3, 3, 3, 3, 3, 2, 3, 5, 3, 4, 3, 4, 3	7518	0.952116	20.695
LP	2, 3, 3, 3, 4, 3, 3, 3, 3, 4, 3, 5, 3, 4, 2, 2, 2, 3, 3, 3, 3, 4, 5, 3, 3, 3, 3, 4, 3, 4, 4, 3, 4, 3, 4, 4, 2, 4, 4, 3, 3, 3, 3, 3, 3, 3, 4, 3, 3, 4, 3, 4, 3, 4, 3, 2, 3, 3, 3, 4, 2, 4, 3, 3, 5, 2, 2, 2, 4, 3, 4, 2, 3, 3, 3, 4, 3, 2, 2, 3, 3, 3, 2, 3, 4, 3, 3, 3, 3, 3, 2, 3, 4, 3, 4, 4, 4, 3	7518	0.962884	26.699

Table 14. Best solutions to problem P_3 after second optimization (minimizing cost under the constraint of reliability R^*).

Algorithm	Optimal Allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{rs}	Ef
SEA $R^* = 0.894261$	3, 5, 3, 2, 2, 2, 3, 3, 4, 2, 2, 3, 2, 5, 3, 3, 4, 4, 2, 2, 3, 2, 3, 2, 4, 5, 4, 5, 3, 2, 2, 3, 3, 2, 7, 2, 3, 5, 3, 4, 2, 5, 3, 3, 4, 3, 2, 5, 3, 3, 3, 3, 3, 3, 3, 2, 2, 3, 2, 3, 3, 3, 4, 3, 3, 3, 4, 2, 3, 5, 3, 3, 3, 4, 2, 3, 2, 8, 3, 4, 5, 3, 3, 3, 2, 2, 3, 5, 4, 4, 3, 4, 2, 3, 3, 3, 6, 3, 5, 3, 3	7735	0.896609	9.588
RELIVE $R^* = 0.927214$	3, 4, 3, 2, 3, 4, 3, 2, 3, 3, 3, 3, 2, 3, 3, 3, 4, 4, 3, 3, 3, 2, 2, 2, 3, 5, 3, 3, 3, 2, 3, 3, 3, 3, 3, 2, 3, 4, 3, 3, 2, 4, 3, 3, 5, 3, 2, 5, 3, 5, 2, 3, 3, 4, 2, 2, 5, 3, 4, 4, 2, 2, 3, 3, 3, 2, 2, 4, 4, 3, 3, 3, 4, 2, 3, 2, 3, 5, 5, 3, 3, 3, 3, 2, 3, 3, 4, 3, 4, 5, 4, 2, 3, 2, 3, 3, 3, 5, 3, 3	7622	0.927251	13.626
LP $R^* = 0.947769$	3, 4, 3, 2, 3, 3, 4, 2, 3, 3, 3, 3, 2, 3, 3, 3, 3, 4, 2, 3, 3, 2, 2, 2, 3, 4, 3, 3, 3, 2, 3, 3, 3, 3, 3, 2, 3, 4, 3, 3, 2, 4, 3, 3, 4, 3, 3, 4, 3, 4, 3, 3, 4, 2, 2, 4, 3, 4, 4, 2, 3, 3, 3, 3, 3, 2, 4, 4, 3, 3, 3, 4, 3, 3, 3, 3, 4, 4, 3, 2, 3, 3, 2, 3, 3, 4, 3, 4, 4, 3, 2, 4, 2, 4, 3, 3, 3, 3, 3	7737	0.947769	18.979

Table 15. Best solutions to problem P_4 after second optimization (minimizing cost under the constraint of reliability R^*).

Algorithm	Optimal Allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{rs}	Ef
SEA $R^* = 0.930610$	2, 3, 5, 2, 4, 3, 4, 4, 2, 4, 2, 5, 3, 4, 4, 3, 3, 3, 5, 3, 3, 5, 4, 3, 3, 3, 3, 3, 3, 5, 4, 5, 6, 5, 5, 5, 2, 8, 3, 3, 3, 3, 8, 3, 3, 2, 5, 4, 4, 4, 5, 4, 3, 3, 4, 3, 2, 4, 3, 4, 2, 3, 3, 5, 5, 2, 2, 3, 5, 3, 4, 2, 4, 2, 3, 4, 5, 5, 3, 4, 2, 5, 3, 4, 2, 5, 3, 2, 5, 4, 2, 5, 2, 4, 3, 3, 5, 4, 3, 3	8213	0.932688	14.722
RELIVE $R^* = 0.952116$	2, 3, 3, 3, 4, 3, 4, 3, 3, 4, 4, 4, 3, 4, 2, 2, 2, 2, 3, 3, 3, 5, 5, 3, 3, 3, 3, 5, 3, 5, 4, 3, 4, 4, 3, 4, 2, 3, 4, 2, 3, 3, 3, 3, 3, 3, 2, 3, 3, 4, 3, 3, 4, 3, 4, 3, 2, 3, 3, 3, 3, 2, 4, 3, 3, 4, 2, 2, 2, 4, 3, 4, 2, 3, 3, 3, 5, 5, 4, 3, 2, 3, 3, 3, 3, 2, 3, 5, 3, 5, 3, 3, 3, 2, 3, 4, 3, 4, 3, 4, 3	7384	0.952135	20.703
LP $R^* = 0.962884$	2, 3, 3, 3, 4, 3, 3, 3, 3, 4, 3, 5, 3, 4, 2, 2, 2, 3, 3, 3, 3, 4, 5, 3, 3, 3, 3, 4, 3, 4, 4, 3, 4, 3, 4, 4, 2, 4, 4, 3, 3, 3, 4, 3, 3, 3, 4, 3, 3, 4, 3, 4, 3, 4, 3, 2, 3, 3, 3, 4, 2, 4, 3, 3, 5, 2, 2, 2, 4, 3, 3, 2, 4, 3, 3, 4, 3, 2, 2, 3, 3, 3, 2, 3, 4, 3, 3, 3, 3, 3, 2, 3, 4, 3, 4, 4, 4, 3	7518	0.962884	26.699

For each algorithm, the reliability found by maximization was set as the threshold for the minimization problem. The goal is to find systems with a lower total cost for the same reliability. During maximization, if there are more candidates with the same reliability

but different costs, the choice between them is indifferent from the point of view of the objective function. Therefore, a solution with a higher cost but still less or equal to the cost threshold can be selected. The second optimization can identify better solutions from both points of view. This is often the case for SEA, which usually gives suboptimal results for the first optimization. On the contrary, LP likely finds the optimal solution every time, and therefore, the results of the second optimization are the same as for the first.

The three optimization algorithms considered in our study generate different solutions. The following three examples illustrate how we can determine whether one is superior to the other:

- Consider problem P_1 for which the best solutions generated by the three optimization algorithms are shown in Table 6. All three solutions require 3719 cost units, but the solution given by SEA achieves lower reliability (0.973398) compared to that given by RELIVE and LP (0.977724);
- Consider problem P_3 for which the best solutions generated by the three optimization algorithms are shown in Table 12. The solutions given by RELIVE and LP both require 7737 cost units, but the solution generated by LP achieves higher reliability (0.947769) compared to that given by RELIVE (0.927214);
- Consider problem P_4 for which the best solutions generated by the three optimization algorithms are shown in Table 15. Please note that the solution given by SEA requires the highest cost and offers the lowest reliability compared to the solutions given by RELIVE and LP.

For a better comparison of the three proposed optimization algorithms, 1000 randomly generated problems were considered for both $n = 50$ and $n = 100$. The corresponding results are presented in Figures 8–11. Each graph presents the mean values as the height of the bars, with the standard deviations represented as two sigmas (one up from the mean, and one down from the mean).

First, the reliability maximization case for $n = 50$ was considered. Figure 8 shows some statistics of the final system reliability obtained by the algorithms. Since the performance of the evolutionary algorithm greatly depends on the number of generations, two versions were considered: 1000 and 10,000 generations for SEA, and 100 and 500 generations for RELIVE.

It must be mentioned that RELIVE performs additional function evaluations during the hill climbing procedure, therefore it is normal that its number of generations be less than for SEA. Figure 8a presents the actual efficiency values obtained by the algorithms. Figure 8b includes a comparison relative to LP, where in each of the 1000 trials the efficiency found by LP was considered to correspond to 100% and the efficiency found by the other algorithms is represented as a percentage of that found by LP. It can be seen that the results of LP and RELIVE are very close, with LP being slightly better, while those of SEA are of a lower quality.

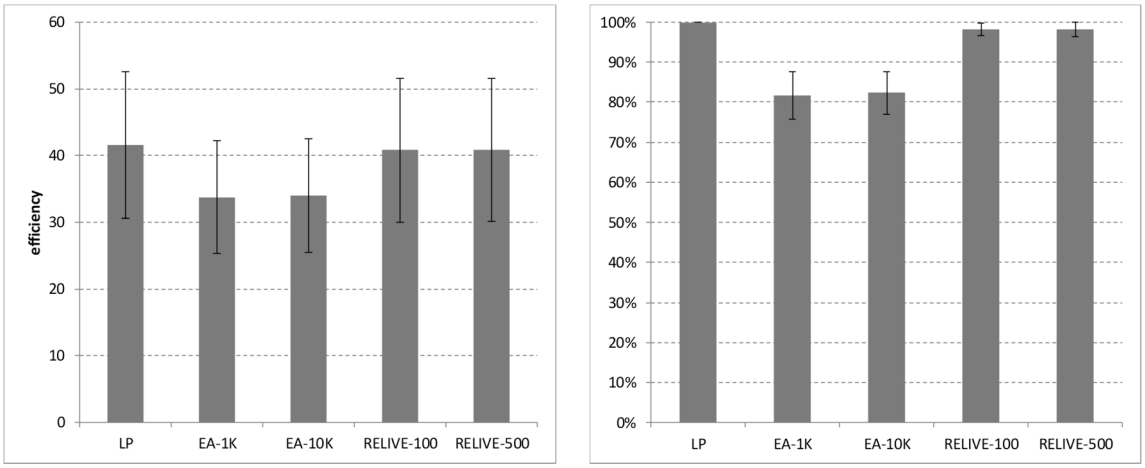
It can also be seen that there is no significant difference between the results of SEA and RELIVE with different numbers of generations: most likely, 1000 and 100 generations, respectively, are sufficient for such problems.

Similar statistics are displayed in Figure 9 for systems with $n = 100$. In this case, since the problems are more difficult, there are greater differences between algorithms. LP remains the best, while the relative average efficiency of RELIVE solutions is around 75%, and that of SEA is around 45%.

Figures 10 and 11 show the results obtained for the cost minimization problems. Since an increase in the number of generations does not seem to be a decisive factor, only 100 and 500 generations were considered for SEA and RELIVE, respectively. The relative performance of algorithms is similar: LP provides the best results, RELIVE results are comparable, slightly worse especially for $n = 100$, while SEA gives an average minimum cost around 120–130% higher than the optimal solution.

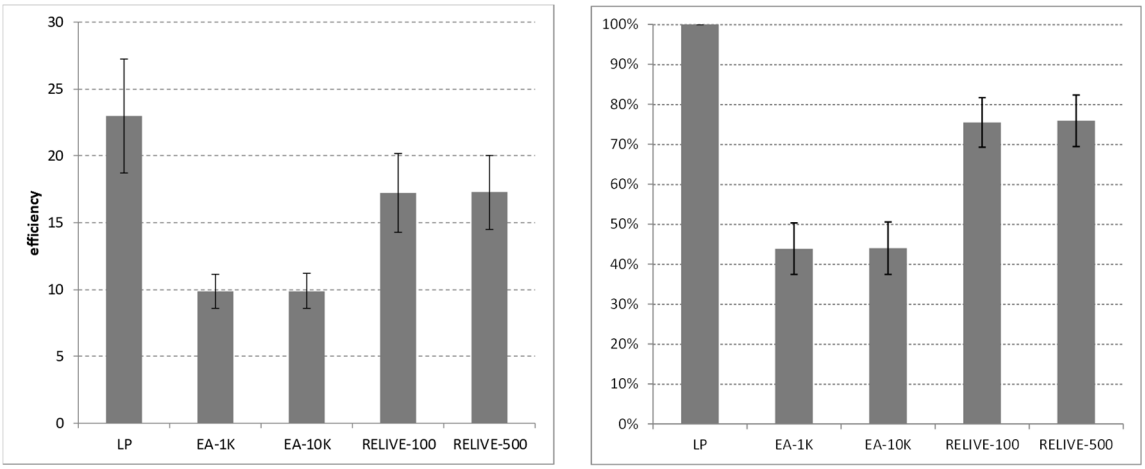
In addition, in order to better verify the effectiveness of the proposed algorithms, for the 2000 problems studied, the results obtained for the initial variant were compared

with those for two other variants in which the order of the subsystems changed, being sorted by reliability. The LP algorithm provided the same results for all 2000 problems checked, which highlights its stability for this type of stress. This is not the case with the two evolutionary algorithms, RELIVE and SEA, but the differences that occurred were not statistically significant.



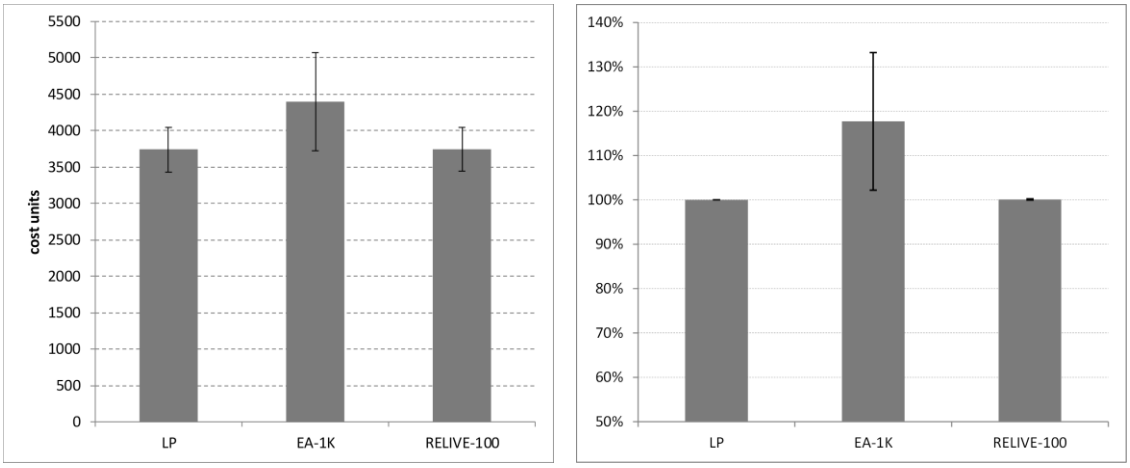
(**a**) (**b**)

Figure 8. Comparison between algorithms performance for reliability maximization on systems with $n = 50$: (**a**) the average efficiency for 1000 random problem instances; (**b**) the results relative to the maximum efficiency found by LP considered as 100%.



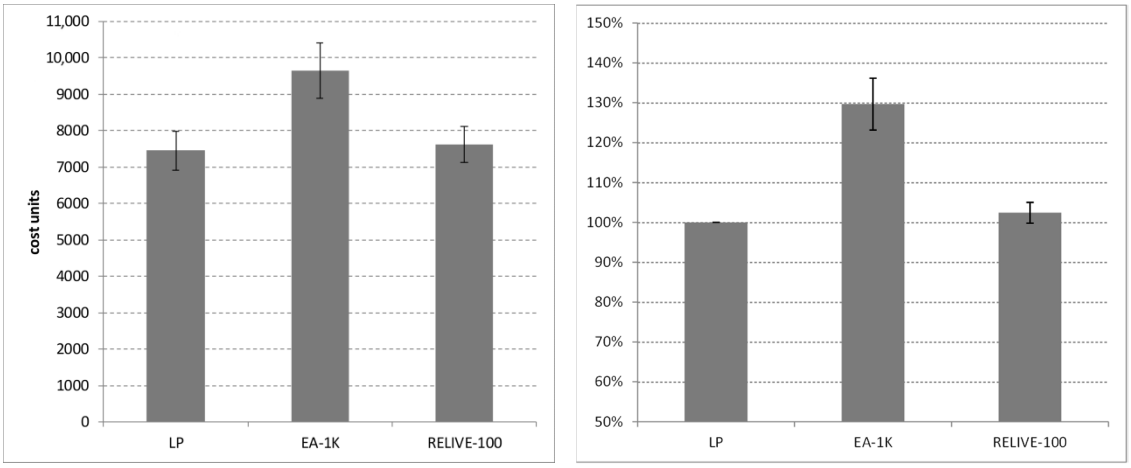
(**a**) (**b**)

Figure 9. Comparison between algorithms performance for reliability maximization with systems with $n = 100$ subsystems: (**a**) average efficiency; (**b**) results relative to LP.



(**a**) (**b**)

Figure 10. Comparison between the performance of the algorithms for cost minimization with systems of $n = 50$ subsystems: (**a**) average cost; (**b**) results relative to LP.



(**a**) (**b**)

Figure 11. Comparison between the performance of the algorithms for cost minimization with systems of $n = 100$ subsystems: (**a**) average cost; (**b**) results relative to LP.

9. Discussion

In the mathematical model, we assume that the time to failure of a component follows a negative-exponential distribution. For electronic components or electronic modules, especially for integrated circuits, the time to failure is usually considered to have such a distribution. This means that, for a given operating regime, the average failure rate is constant (and not a function of time). But for mechanical elements, for example, this assumption must be accepted with caution because of the physical wear and tear that can occur during system operation. In this case, a Weibull distribution may be more appropriate.

This assumption is important only for specifying the reliability of the redundant system. Only under this assumption the reliability function for most of the redundant structures we considered can be determined analytically, using Markov models, as presented in Section 3. For other distributions, the evaluation of subsystem reliability is more complicated and can be done in other ways, e.g., by using a Monte Carlo simulation.

The optimization methods used in this study are not fundamentally affected by this simplifying assumption. The only change concerns the calculation of the objective function,

which otherwise should be done in a different way. Thus, we appreciate that the comparative performance results of the three optimization methods presented in this article are not significantly affected by this simplifying assumption.

The systems discussed in this paper are all series-aligned subsystems. Our study does not cover cases where a system component may have a redundant structure composed of elements other than the base component, as shown in Figure 12.

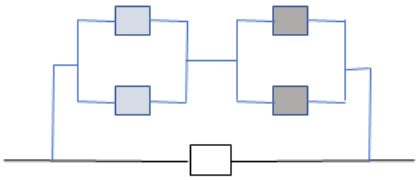


Figure 12. Alternative redundant subsystem structure.

In this situation, the optimization problem must be formulated differently, and it involves the inclusion of more types of components than those that form the non-redundant system.

Such cases are encountered in complex systems, e.g., with a network structure. Unfortunately, the conclusions regarding the performance of the three optimization algorithms compared in this paper cannot be extended to these more general cases. There is no evidence to support this.

Another point of discussion is needed about the number of generations used by the two evolutionary algorithms. The specific number of generations used in the study are powers of ten so that the reader can have an intuitive view about the results. A fairer comparison would need to assess their performance, e.g., with the same number of objective function evaluations, a common setting in the area of biologically-inspired optimization algorithms. The number of function evaluations is easy to determine in case of SEA. If the population consists of 50 chromosomes and 10,000 generations are used, then 500,000 evaluations are needed. However, RELIVE does not have a constant population size. Additional function evaluations are performed in the hill climbing step, although at most one of these solutions will be actually used subsequently in the next generation, i.e., the best local improvement. It was empirically estimated that RELIVE with 100 generations needs about 27 times more function evaluations than SEA with 1000 generations. Thus, a comparison could be made with SEA with about 27,000 generations. Still, from the statistical analysis presented above, we hypothesize that the poorer results of SEA are not caused by a smaller number of generations than required. The performance in both cases with 1000 and 10,000 generations is quite similar. Also, the main issue is not execution time, because this is not a real-time application, but the fact that SEA usually gets stuck into a local optimum because, e.g., at the "top" part of the allocation, one cannot include any more components without exceeding the cost limit. It would require one to add one component to a subsystem and remove one component from another subsystem in order to improve the optimization. SEA lacks any mechanisms to do so, and such improvements can come only from "lucky" mutations and removals of components during the chromosome repairing procedure. On the other hand, RELIVE has an especially designed mutation for this situation, based on exchanging a unit between a pair of genes. Because of this, we eventually chose to use the lower number of generations, i.e., 1000 for SEA and 100 for RELIVE, because in this case the optimization is faster and it seems to show the hierarchy of the used methods quite well.

Since evolutionary algorithms are stochastic, more runs may be necessary to obtain a good solution. In the case studies presented above, we used the following methodology:

- For the results presented in Figures 8–11, each algorithm was run a single time for a problem and 2000 problems were used, i.e., 1000 problems for $n = 50$ and another 1000 problems for $n = 100$. Due to the high number of problems, the results are

statistically significant to assess the performance of the algorithms. These figures show this statistical analysis in terms of mean and standard deviation;

- For the results presented in Tables 6–9 and 12–15, the best out of ten runs was selected for SEA and RELIVE, because we were interested in the best solution. The LP algorithm was run only once.

10. Conclusions

Extensive experimental studies on the allocation of redundancy in large binary systems with a hybrid structure, which include a number of optimization problems of the order of thousands, highlight the difficulty of these optimization problems as the number of subsystems increases. Three algorithms were used for optimization: zero-one integer programming, a classic evolutionary algorithm and an original evolutionary algorithm, RELIVE, which combines global search with local fine tuning and includes a number of mutation strategies in order to escape from local optima.

The proposed algorithms are compared, but their effectiveness was also verified by solving two optimization problems, properly correlated. Specifically, a converse problem of minimizing cost for the reliability threshold found in the first case was also attempted as a means to verify the optimality of the solution and when the solution was not optimal, to attempt to improve it from either the cost or reliability perspectives, and possibly both. Experimental results demonstrate that for large instances of the reliability maximization problem, zero-one integer programming yields the best results, followed by RELIVE. The differences become apparent when the number of subsystems is large, e.g., when $n = 100$.

As future research, the authors intend to extend the study on the optimal allocation of reliability in hybrid structure binary systems in several directions, as shown below.

For the optimization issues considered in this paper, the type of redundancy is predetermined for all subsystems, as shown in Table 4, Table 5, Table 10 or Table 11. But for certain reliability models this condition may be relaxed. For example, if a redundancy technique based on majority logic is appropriate for a subsystem, then one of the following solutions can be adopted: TMR, TMR/Simplex or 5MR, with or without cold-maintained spare components. The same is true for dynamic redundancy, where active redundancy or hybrid standby redundancy with a hot component and other passive spare ones can be adopted. Therefore, the optimization process can be extended to find an optimal solution that refers to both the type of redundancy and the number of components for each of the n subsystems.

On the other hand, some redundant structures often adopt the technical solution in which the components are functionally compatible but different in design to avoid common errors. For example, this idea applies to majority logic structures (TMR, TMR/Simplex and 5MR) or duplex structure. A future direction of research also refers to these redundant subsystems with heterogeneous structure.

In reliability engineering the problem of system reliability maximization under two or more constraints often arises; for example, under cost constraints, but also under weight and/or volume constraints. We intend to extend the research to also cover this important problem of maximizing system reliability under two or more constraints.

We also plan to study the transformation of the problem into a multi-objective optimization problem, e.g., maximize the system's reliability while minimizing the associated cost. The solutions to be considered would be the solutions around the imposed threshold for cost or reliability. Previously we saw that an increase in the cost limit of only 5% can lead to a larger increase in system reliability. By using a multi-objective optimization approach, such analysis could be more principled.

Another direction of investigation would be to assess the effect of integer-based representation for the evolutionary algorithms instead of the real-valued representation used so far.

Author Contributions: Conceptualization, P.C. and F.L.; methodology, P.C. and F.L.; software, F.L.; validation, P.C.; formal analysis, P.C.; investigation, F.L.; writing—original draft preparation, P.C. and

F.L.; writing—review and editing, P.C. and F.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All data used for the experimental studies are available on request.

Conflicts of Interest: The authors declare no conflict of interest.

Nomenclature

Reliability	The probability that a component or a system works successfully within a given period of time
Binary system	A system in which each component can be either operational or failed
Series-redundant model	A reliability model that reflects a redundant system composed of subsystems consisting of basic components or redundant structures, and possibly other spare components
Notations	
n	The number of components in the non-redundant system or the number of subsystems in the redundant system, as appropriate
T	A certain period of time for which reliability is assessed
r_i	The reliability of a component of type i, $i \in \{1, \ldots, n\}$, for a given period of time T
c_i	The cost of a component of type i
λ_i	The failure rate for a component of type i
k_i	The number of components that make up the redundant subsystem i
R_i	The reliability of subsystem i (subsystem with redundant structure)
C_i	The cost of subsystem i
tr_i	The type of redundancy for subsystem i
α, $0 < \alpha < 1$	The coefficient of reduction of the failure rate for a warm-maintained reserve compared to the failure rate of the component in operation
β, $\beta > 1$	The reduction factor used to express the failure rate of the decision and control logic of a TMR structure based on the failure rate of the basic components
γ, $\gamma > 1$	The reduction factor used to express the failure rate of the decision and control logic of a 5MR structure based on the failure rate of the basic components
δ, $\delta > 1$	The reduction factor used to express the failure rate of the decision, control and reconfiguration logic of a TMR/Simplex or a TMR/Duplex structure based on the failure rate of the basic components
R_{ns}	The reliability of the non-redundant system (system with series reliability model)
C_{ns}	The cost of the non-redundant system
R_{rs}	The reliability of the redundant system (system with series-redundant reliability model)
R_{rs}	The reliability of the redundant system (system with series-redundant reliability model)
C_{rs}	The cost of the redundant system
R^*	The required level of reliability of the system
C^*	The maximum allowable cost of the system
CO	A component in operation (active component)
WSC	A warm-maintained spare component
CSC	A cold-maintained spare component

Note: For notations r_i to tr_i, when the subsystem is not indicated the index is not necessary, therefore the notations used are r, c, λ and so on.

Assumptions

- For any redundant subsystem, the spare components are considered identical to the basic one/ones.
- For the components in operating mode or for the spare components maintained in warm conditions, the time to failure has a negative-exponential distribution.
- The events of failure that may affect the components of the system are stochastically independent.

References

1. Coit, D.W.; Zio, E. The evolution of system reliability optimization. *Reliab. Eng. Syst. Saf.* **2019**, *192*, 106259. [CrossRef]
2. Soltani, R. Reliability optimization of binary state non-repairable systems: A state of the art survey. *Int. J. Ind. Eng. Comput.* **2014**, *5*, 339–364. [CrossRef]
3. Kuo, W.; Wan, R. *Recent Advances in Optimal Reliability Allocation, Computational Intelligence in Reliability Engineering*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 1–36.
4. Leon, F.; Caşcaval, P.; Bădică, C. Optimization Methods for Redundancy Allocation in Large Systems. *Vietnam. J. Comput. Sci.* **2020**, *7*, 281–299. [CrossRef]
5. Kuo, W.; Lin, H.H.; Xu, Z.; Zhang, W. Reliability optimization with the Lagrange-multiplier and branch-and-bound technique. *IEEE Trans. Reliab.* **1987**, *36*, 624–630. [CrossRef]
6. Misra, K.B. Reliability Optimization of a Series-Parallel System, part I: Lagrangian Multipliers Approach, part II: Maximum Principle Approach. *IEEE Trans. Reliab.* **1972**, *21*, 230–238. [CrossRef]
7. Chern, M. On the Computational Complexity of Reliability Redundancy Allocation in Series System. *Oper. Res. Lett.* **1992**, *11*, 309–315. [CrossRef]
8. Dobani, E.R.; Ardakan, M.A.; Davari-Ardakani, H.; Juybari, M.N. RRAP-CM: A new reliability-redundancy allocation problem with heterogeneous components. *Reliab. Eng. Syst. Saf.* **2019**, *191*, 106–563. [CrossRef]
9. Gholinezhad, H.; Hamadani, A.Z. A new model for the redundancy allocation problem with component mixing and mixed redundancy strategy. *Reliab. Eng. Syst. Saf.* **2017**, *164*, 66–73. [CrossRef]
10. Hsieh, T.J. A simple hybrid redundancy strategy accompanied by simplified swarm optimization for the reliability–redundancy allocation problem. *Eng. Optim.* **2022**, *54*, 369–386. [CrossRef]
11. Ali Najmi, K.B.; Ardakan, M.A.; Javid, A.Y. Optimization of reliability redundancy allocation problem with component mixing and strategy selection for subsystems. *J. Stat. Comput. Simul.* **2021**, *91*, 1935–1959. [CrossRef]
12. Peiravi, A.; Karbasian, M.; Ardakan, M.A.; Coit, D.W. Reliability optimization of series-parallel systems with K-mixed redundancy strategy. *Reliab. Eng. Syst. Saf.* **2019**, *183*, 17–28. [CrossRef]
13. Feizabadi, M.; Jahromi, A.E. A new model for reliability optimization of series-parallel systems with non-homogeneous components. *Reliab. Eng. Syst. Saf.* **2017**, *157*, 101–112. [CrossRef]
14. Hsieh, T.-J.; Yeh, W.C. Penalty guided bees search for redundancy allocation problems with a mix of components in series–parallel systems. *Comput. Oper. Res.* **2012**, *39*, 2688–2704. [CrossRef]
15. Sadjadi, S.J.; Soltani, R. An efficient heuristic versus a robust hybrid meta-heuristic for general framework of serial–parallel redundancy problem. *Reliab. Eng. Syst. Saf.* **2009**, *94*, 1703–1710. [CrossRef]
16. Coit, D.W.; Konak, A. Multiple weighted objectives heuristic for the redundancy allocation problem. *IEEE Trans. Reliab.* **2006**, *55*, 551–558. [CrossRef]
17. Prasad, V.R.; Nair, K.P.K.; Aneja, Y.P. A Heuristic Approach to Optimal Assignment of Components to Parallel-Series Network. *IEEE Trans. Reliab.* **1992**, *40*, 555–558. [CrossRef]
18. Shi, D.H. A new heuristic algorithm for constrained redundancy optimization in complex system. *IEEE Trans. Reliab.* **1987**, *36*, 621–623.
19. Caşcaval, P.; Leon, F. Active Redundancy Allocation in Complex Systems by Using Different Optimization Methods. In Proceedings of the 11th International Conference on Computational Collective Intelligence (ICCCI 2019), Hendaye, France, 4–6 September 2019; Nguyen, N., Chbeir, R., Exposito, E., Aniorte, P., Trawinski, B., Eds.; Computational Collective Intelligence, Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11683, pp. 625–637.
20. Everett, H. Generalized Lagrange Multiplier Method of Solving Problems of Optimal Allocation of Resources. *Oper. Res.* **1963**, *11*, 399–417. [CrossRef]
21. Yalaoui, A.; Châtelet, E.; Chu, C. A new dynamic programming method for reliability & redundancy allocation in a parallel-series syste. *IEEE Trans. Reliab.* **2005**, *54*, 254–261.
22. Shooman, M. *Reliability of Computer Systems and Networks*; John Wiley & Sons: New York, NY, USA, 2002.
23. Misra, K.B. Dynamic programming formulation of the redundancy allocation problem. *Int. J. Math. Educ. Sci. Technol.* **1971**, *2*, 207–215. [CrossRef]

24. Sahoo, L.; Bhunia, A.K.; Roy, D. Reliability optimization with high and low level redundancies in interval environment via genetic algorithm. *Int. J. Syst. Assur. Eng. Manag.* **2014**, *5*, 513–523. [CrossRef]
25. Tavakkoli-Moghaddam, R.; Safari, J.; Khalili-Damghani, F.; Abtahi, K.; Tavana, A.-R. A new multi-objective particle swarm optimization method for solving reliability redundancy allocation problems. *Reliab. Eng. Syst. Saf.* **2013**, *111*, 58–75.
26. Coelho, L.D.S. Self-organizing migrating strategies applied to reliability-redundancy optimization of systems. *IEEE Trans. Reliab.* **2009**, *58*, 501–510. [CrossRef]
27. Agarwal, M.; Gupta, R. Genetic Search for Redundacy Optimization in Complex Systems. *J. Qual. Maint. Eng.* **2006**, *12*, 338–353. [CrossRef]
28. Berkelaar, M.; Eikland, K.; Notebaert, P. lpsolve, Mixed Integer Linear Programming (MILP) Solver. 2021. Available online: https://sourceforge.net/projects/lpsolve (accessed on 1 September 2022).
29. Leon, F.; Caşcaval, P. 01IP and QUBO: Optimization Methods for Redundancy Allocation in Complex Systems. In Proceedings of the 2019 23rd International Conference on System Theory, Control and Computing, Sinaia, Romania, 9–11 October 2019; pp. 877–882.
30. Misra, K.B.; Sharma, U. An efficient algorithm to solve integer-programming problems arising in system-reliability design. *IEEE Trans. Reliab.* **1991**, *40*, 81–91. [CrossRef]
31. Floudas, C.A. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*; Oxford University Press: New York, NY, USA, 1995.
32. Trivedi, K.S. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*; John Wiley & Sons: New York, NY, USA, 2002.
33. Misra, K.B. (Ed.) *Handbook of Performability Engineering*; Springer: London, UK, 2008.
34. McGeoch, C.C.; Harris, R.; Reinhardt, S.P.; Bunyk, P. Practical Annealing-Based Quantum Computing, Whitepaper, D-Wave Systems. 2019. Available online: https://www.dwavesys.com/media/vh5jmyka/ (accessed on 1 September 2022).
35. Holland, J.H. Genetic Algorithms. *Sci. Am.* **1992**, *267*, 66–73. Available online: http://www.jstor.org/stable/24939139 (accessed on 1 September 2022). [CrossRef]
36. De Jong, K.A. *Evolutionary Computation: A unified Approach*; MIT Press: Cambridge, MA, USA, 2006.
37. Dantzig, G.B.L. *Linear Programming and Extensions*; Princeton University Press: Princeton, NJ, USA, 1963.
38. Albert, A.A. *A Measure of the Effort Required to Increase Reliability*; Technical Report, No. 43; Stanfort University, Applied Mathematics and Statistics Laboratory: Stanford, CA, USA, 1958.

Vietnam Journal of Computer Science
Vol. 7, No. 3 (2020) 281–299

DOI: 10.1142/S2196888820500165

World Scientific
www.worldscientific.com

Optimization Methods for Redundancy Allocation in Large Systems

Florin Leon*,‡, Petru Caşcaval*,§ and Costin Bădică†,¶

*Department of Computers
"Gheorghe Asachi" Technical University of Iaşi
Bd. Mangeron 27A, Iaşi 700050, Romania

†Department of Computers and Information Technology
University of Craiova
Bd. Decebal 107, Craiova 200776, Romania
‡florin.leon@tuiasi.ro
§cascaval@tuiasi.ro
¶cbadica@software.ucv.ro

Received 15 January 2020
Accepted 28 February 2020
Published 14 April 2020

This paper addresses the issue of optimal allocation of spare modules in large series-redundant systems in order to obtain a required reliability under cost constraints. Both cases of active and standby redundancy are considered. Moreover, for a subsystem with standby redundancy, two cases are examined: in the first case, all the spares are maintained in cold state (cold standby redundancy) and, in the second one, to reduce the time needed to put a spare into operation when the active one fails, one of the spares is maintained in warm conditions. To solve this optimization problem, for the simpler case of active redundancy an analytical method based on the Lagrange multipliers technique is first applied. Then the results are improved by using Pairwise Hill Climbing, an original fine-tuning algorithm. An alternative approach is an innovative evolutionary algorithm, RELIVE, in which an individual lives for several generations and improves its fitness based on local search. These methods are especially needed in case of very large systems.

Keywords: Reliability allocation; series-parallel reliability models; Lagrange multipliers; RELIVE algorithm; evolutionary algorithm; Pairwise Hill Climbing.

1. Introduction

The reliability design of a complex system is one of the most studied topics in the literature. The problems mainly refer to the kind of solution (reliability allocation and/or redundancy allocation), the kind of redundancy (active, standby, etc.), the type of system (binary or multi-state), the levels of redundancy (multi-level system

‡Corresponding author.

or multiple component choice), etc. All these issues have practical applications and provide good directions of research. An excellent overview of such problems can be found in Refs. 1–4.

According to the decision variables,[3,4] a reliability optimization problem may belong to the following types:

(a) reliability allocation, when the decision variables are component reliabilities;
(b) redundancy allocation, when the variable is the number of component units;
(c) reliability-redundancy allocation, when the decision variables include both the component reliabilities and the redundancies.

In this paper, we address the problem of redundancy allocation in which the number of redundant units in a series-parallel reliability model is the only decision variable. This problem belongs to the NP-hard category.[5]

We have limited ourselves to the binary systems in which each component is either operational or failed. Regarding the kind of redundancy, for each component of the system we consider that the reserves can be active or in standby, as necessary from the technical point of view.

To solve this optimization problem of redundancy allocation, more methods or techniques can be applied, such as intuitive engineering methods,[6] heuristic search algorithms,[7–10] analytical methods based on Lagrange multipliers,[11–13] or dynamic programming.[14] Other metaheuristic methods based on genetic algorithms are also appropriate.[15–18] In this work, we focus on metaheuristic methods combined with an analytical approach based on Lagrange multipliers.

Other distinct approaches for this problem are based on Zero-One Integer Programming (01IP) and Quadratic Unconstrained Binary Optimization (QUBO).[19]

This paper is an extended version of Ref. 20, in which the analytical method is explained in more detail and an original evolutionary algorithm (EA) is proposed that, unlike the approach previously used, seems to be able to solve large instances of the problem under consideration.

The rest of this work is organized as follows: Section 2 presents a brief nomenclature, notations and some assumptions, while Sec. 3 describes the optimization problem we deal with. An analytical approach based on Lagrange multipliers is presented in Sec. 4. In Sec. 5, we mention the Pairwise Hill Climbing algorithm used in Ref. 20, while in Sec. 6, we describe a novel evolutionary algorithm, RELIVE, that combines global and local search. In Sec. 7, we include the experimental results, and Sec. 8 contains the conclusions of this work.

2. Nomenclature, Notations and Assumptions

2.1. *Nomenclature*

(a) *Reliability* — The probability that a component or a system works successfully within a given period of time.

(b) *Series-redundant model* — A reliability model corresponding to a redundant system consisting of basic components and possibly other active or standby spare components, as the case may be.

2.2. *Notations*

(a) n — The number of components in the non-redundant system or the number of subsystems in the redundant system, as the case may be;
(b) r_i — The reliability of a component of type i, $i \in \{1, 2, \ldots, n\}$, for a given period of time T;
(c) λ_i — The failure rate for a component of type i;
(d) x_i — Another reliability indicator for a component of type i, $i \in \{1, 2, \ldots, n\}$;
(e) c_i — The cost of a component of type i;
(f) R_{ns} — The reliability of the non-redundant system (system with series reliability model);
(g) C_{ns} — Cost of the non-redundant system;
(h) k_i — The number of components that compose the redundant subsystem i,$i \in \{1, 2, \ldots, n\}$;
(i) R_i — The reliability of subsystem i (a subsystem consisting of a basic component and possibly one or more identical spare components);
(j) C_i — The cost of subsystem i;
(k) R_{rs} — The reliability of the redundant system (system with series-redundant reliability model);
(l) C_{rs} — The cost of the redundant system;
(m) R^* – The required level of reliability for the system;
(n) C^* — The maximum allowed cost for the system;
(o) η — A Lagrange multiplier.

2.3. *Assumptions*

(a) Each component in the system can be either operational or failed (i.e. binary system).
(b) For any redundant subsystem, the spare components are considered identical to the basic one.
(c) For a subsystem with standby redundancy, for the component in operating mode or for the spare maintained in warm conditions, the time to failure has a negative-exponential distribution law.
(d) The events of failure that may affect the components of the system are stochastically independent.

3. Problem Description

Let us first consider a non-redundant system composed of n basic components for which the reliability model is a series one. The reliability of this system without

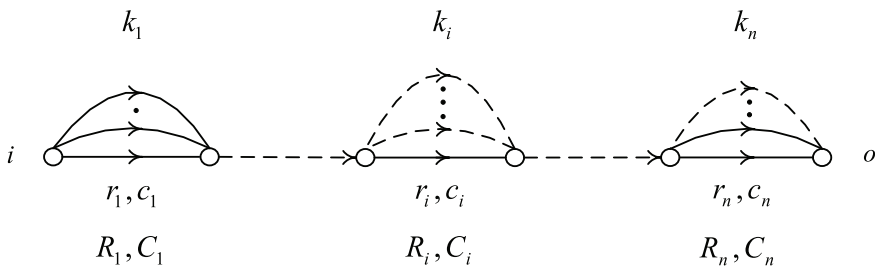


Fig. 1. Reliability model for a series-redundant system with spare components.

redundancy is $R_{\text{ns}} = \prod_{i=1}^{n} r_i$ and for systems with a large number of components this value is often quite low. To reach a required reliability, spare components are added so that the reliability model becomes series-redundant as presented in Fig. 1.

For a subsystem, the spare components can be active or passive (in standby), as the case may be. Moreover, for a subsystem with standby redundancy, in this work, two cases are considered: in the first case, all the spares are maintained in cold state (cold standby redundancy) so that an inactive component does not fail (this failure rate is 0), or in the second one, in order to reduce the time needed to put a spare into operation when the active one fails, one of the spares is maintained in warm or even hot condition, as the case may be; if any, the other spare components are maintained in cold state. Consequently, in this second case, one inactive component may fail before it is put into operation and its failure rate is less or equal to the failure rate of the component in active mode (AC). In Fig. 1, a cold spare is illustrated by a broken line.

Typically, in this allocation process, the criterion is reliability, cost, weight or volume. One or more criteria can be considered in an objective function, while the others can be considered as constraints.[2,6,14] From a mathematical point of view, one must solve an optimization problem of an objective function with constraints.

In this work, we address the issue of maximizing system reliability within the cost constraint. Thus, we have to determine the values $k_i, i \in \{1, 2, \ldots, n\}$ that maximize the reliability function

$$R_{\text{rs}} = f(R_1, R_2, \ldots, R_n) = \prod_{i=1}^{n} R_i \tag{1}$$

under the cost constraint:

$$\sum_{i=1}^{n} C_i \leq C^*. \tag{2}$$

For a subsystem i with active redundancy, as the spare components are identical to the basic one, the reliability function can be expressed by the following equation:

$$R_i = 1 - (1 - r_i)^{k_i}, \quad i \in \{1, 2, \ldots, n\}. \tag{3}$$

For a subsystem i with cold standby redundancy, composed of one active component and other $k_i - 1$ identical components in cold state, under the assumption

that for the component in AC, the time to failure has a negative-exponential distribution law, the reliability function R_i can be expressed by the following equation (see, e.g. p. 111 of Ref. 3):

$$R_i = \sum_{j=0}^{k_i-1} \frac{(\lambda_i T)^j}{j!} e^{-\lambda_i T}, \quad i \in \{1, 2, \ldots, n\}. \tag{4}$$

As $r_i = e^{-\lambda_i T}$, the equation becomes

$$R_i = r_i \sum_{j=0}^{k_i-1} \frac{(-\ln(r_i))^j}{j!}, \quad i \in \{1, 2, \ldots, n\}. \tag{5}$$

For a subsystem with standby redundancy, in which a spare component is maintained in a warm state, the reliability function is obtained based on a Markov chain. Let λ be the failure rate for the component in operating mode, and let $\alpha\lambda$, $0 < \alpha \leq 1$, be the failure rate for the warm spare (WS). To illustrate how the Markov model is applied, let us consider a subsystem composed of a component in AC, a WS and a cold one (CS) (i.e. $k_i = 3$). The evolution of this redundant subsystem until failure is illustrated by the Markov chain presented in Fig. 2.

Let $s(t)$ be the state of the subsystem at time t, and $p_i(t) = \text{Prob}(s(t) = S_i)$, $i \in \{S_1, S_2, S_3, S_4\}$. As S_1, S_2 and S_3 are successful states, and S_4 is an absorbent one, the reliability function of this redundant subsystem can be defined as $R(t) = p_1(t) + p_2(t) + p_3(t), t \geq 0$. To determine the probability functions $p_i(t)$, $i \in \{1, 2, 3\}$, the following system of differential equations has to be solved:

$$P' = A \times P, \tag{6}$$

where $P = [p_1(t)p_2(t)p_3(t)]^T$, $P' = [p_1'(t)p_2'(t)p_3'(t)]^T$ and $\mathbf{A}$ is the matrix of state transition rates.

In this case, matrix $\mathbf{A}$ has the form

$$\mathbf{A} = \begin{bmatrix} -(1+\alpha)\lambda & 0 & 0 \\ (1+\alpha)\lambda & -(1+\alpha)\lambda & 0 \\ 0 & (1+\alpha)\lambda & -\lambda \end{bmatrix}. \tag{7}$$

Thus, the following system of differential equations results:

$$\begin{cases} p_1'(t) = -(1+\alpha)\lambda p_1(t), \\ p_2'(t) = (1+\alpha)\lambda p_1(t) - (1+\alpha)\lambda p_2(t), \\ p_3'(t) = (1+\alpha)\lambda p_2(t) - \lambda p_3(t). \end{cases} \tag{8}$$

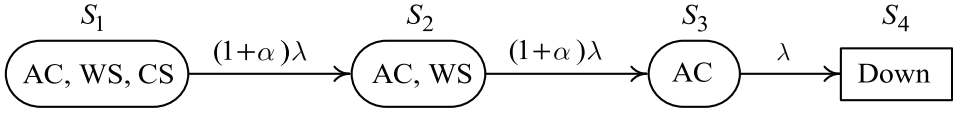


Fig. 2. Markov chain for subsystem reliability evaluation (subsystem with a WS and a cold one).

The initial values are: $p_1(0) = 1, p_2(0) = p_3(0) = p_4(0) = 0$. Thus, by applying the Laplace transform ($\mathcal{L}$), the following system of algebraic equations can be written:

$$\begin{cases} sP_1(s) - 1 = -(1+\alpha)\lambda P_1(s), \\ sP_2(s) = (1+\alpha)\lambda P_1(s) - (1+\alpha)\lambda P_2(s), \\ sP_3(s) = (1+\alpha)\lambda P_2(s) - \lambda P_3(s), \end{cases} \tag{9}$$

where $P_i(s) = \mathcal{L}\{p_i(t)\}, i \in \{1,2,3\}$, are functions in the frequency domain, and s is the Laplace operator.

Based on (9), after some algebraic operations, the following equations result:

$$\begin{cases} P_1(s) = \dfrac{1}{s + (1+\alpha)\lambda}, \\ P_2(s) = \dfrac{(1+\alpha)\lambda}{(s + (1+\alpha)\lambda)^2}, \\ P_3(s) = \dfrac{((1+\alpha)\lambda)^2}{(s + (1+\alpha)\lambda)^2} \dfrac{1}{s+\lambda}. \end{cases} \tag{10}$$

After a partial-fraction-expansion, the function $P_3(s)$ can be expressed in this way

$$P_3(s) = -\frac{(1+\alpha)^2}{\alpha^2}\left(\frac{\alpha\lambda}{(s + (1+\alpha)\lambda)^2} - \frac{1}{s + (1+\alpha)\lambda} + \frac{1}{s+\lambda}\right). \tag{11}$$

As the function $\Re(s) = \mathcal{L}\{R(t)\} = P_1(s) + P_2(s) + P_3(s)$, the following equation results:

$$\Re(s) = -\frac{(1+\alpha)\lambda}{\alpha}\frac{1}{(s + (1+\alpha)\lambda)^2} - \frac{1+2\alpha}{\alpha^2}\frac{1}{s + (1+\alpha)\lambda} + \frac{(1+\alpha)^2}{\alpha^2}\frac{1}{s+\lambda}. \tag{12}$$

The mean time to failure (MTTF) for the redundant subsystem can be obtain based on the function $\Re(s)$ by giving the value 0 to s. Thus,

$$\begin{aligned} \text{MTTF} = \Re(0) &= -\frac{(1+\alpha)\lambda}{\alpha}\frac{1}{((1+\alpha)\lambda)^2} - \frac{1+2\alpha}{\alpha^2}\frac{1}{(1+\alpha)\lambda} + \frac{(1+\alpha)^2}{\alpha^2}\frac{1}{\lambda} \\ &= \frac{3+\alpha}{(1+\alpha)\lambda}. \end{aligned} \tag{13}$$

As a check, based on the state transitions presented in Fig. 1, one can write

$$\text{MTTF} = \frac{1}{(1+\alpha)\lambda} + \frac{1}{(1+\alpha)\lambda} + \frac{1}{\lambda} = \frac{3+\alpha}{(1+\alpha)\lambda}. \tag{14}$$

The reliability function $R(t)$ can be obtained by applying the inverse Laplace transform, $R(t) = \mathcal{L}^{-1}\{\Re(s)\}$. Thus, the reliability function has the following form:

$$R(t) = \frac{(1+\alpha)^2}{\alpha^2}e^{-\lambda t} - \left(\frac{1+2\alpha}{\alpha^2} + \frac{1+\alpha}{\alpha}\lambda t\right)e^{-(1+\alpha)\lambda t}. \tag{15}$$

Let us consider now the subsystem i, where for the operating component the failure rate is λ_i and, for a given period of time $T, r_i = e^{-\lambda_i T}$. As $\lambda_i T = -\ln r_i$, the

subsystem reliability function is given by the following equation:

$$R_i = \frac{(1+\alpha)^2}{\alpha^2} r_i - \left(\frac{1+2\alpha}{\alpha^2} - \frac{1+\alpha}{\alpha} \ln r_i \right) r_i^{1+\alpha}. \tag{16}$$

As a check, for $T = 0, r_i = 1$ and

$$R_i = \frac{(1+\alpha)^2}{\alpha^2} - \frac{1+2\alpha}{\alpha^2} = \frac{\alpha^2}{\alpha^2} = 1. \tag{17}$$

For a redundancy subsystem i with a different number of components, the reliability function can be obtained based on a Markov chain in the same manner. For example, if the redundant subsystem is composed of a component in active mode and a spare maintained in warm conditions (i.e. $k_i = 2$), by applying the method based the Markov chain, the reliability function is

$$R_i = r_i - r_i^{\alpha}(r_i - 1). \tag{18}$$

For another subsystem i composed of a component in active mode (AC), a warm spare (WS) and two spares in a cold state (2 CS) (i.e. $k_i = 4$), the algebraic operations are more complicated but one can show in the same manner that the subsystem reliability function is given by the following equation:

$$R_i(r_i, \alpha) = \frac{(1+\alpha)^3}{\alpha^3} r_i - \left(\frac{1+3\alpha+3\alpha^2}{\alpha^3} - \frac{1+3\alpha+2\alpha^2}{\alpha^2} \ln r_i + \frac{(1+\alpha)^2}{2\alpha} (\ln r_i)^2 \right) r_i^{1+\alpha}. \tag{19}$$

Note that, compared to the case of active redundancy (Eq. (3)), the expression of the reliability function is more complex, therefore an analytical approach to this optimization problem is more difficult to apply.

In case of active redundancy, to reduce the computation time, we choose to first apply an analytical method based on Lagrange multipliers in order to quickly obtain an approximate solution, and then, this approximate solution is improved with other original optimization methods.

4. Analytical Approach for Active Redundancy

As the spares are identical to the basic component, for the series-parallel reliability model presented in Fig. 1, the reliability function can be expressed by the following equation:

$$R_{\text{rs}} = \prod_{i=1}^{n} (1 - (1 - r_i)^{k_i}). \tag{20}$$

We have to determine the values $k_1, k_2, \ldots, k_n$ that maximize the function R_{rs} with the cost constraint:

$$\sum_{i=1}^{n} c_i k_i \leq C^*. \tag{21}$$

Based on (20) and (21), the following Lagrangian function results:

$$L(k_1, k_2, \ldots, k_n, \eta) = \prod_{i=1}^{n} (1 - (1 - r_i)^{k_i}) + \eta \left(\sum_{i=1}^{n} c_i k_i - C^* \right), \tag{22}$$

where η is the Lagrange multiplier.

Thus, instead of maximizing the function R_{rs} given by (20) within the cost constraint (21), we have to maximize the Lagrange function given by (22) without constraints. For this purpose, the following system with partial derivatives must be solved:

$$\begin{cases} \dfrac{\partial L}{\partial k_i} = 0, \quad i = 1, 2, \ldots, n, \\ \dfrac{\partial L}{\partial \eta} = 0. \end{cases} \tag{23}$$

It is easy to observe that by applying the partial derivatives presented in (23), the system of algebraic equations that results is very difficult to solve because of the products that appear. For example, the partial derivative $\frac{\partial L}{\partial k_i} = 0$ leads to the complex equation:

$$-\prod_{j \neq i} (1 - (1 - r_j)^{k_j}) \times (1 - r_i)^{k_i} \ln(1 - r_i) + \eta c_i = 0. \tag{24}$$

For this reason, another way to express the reliability of a system is more appropriate. The reliability of a system can be expressed by the reliability indicator (R) or the non-reliability indicator ($1 - R$), i.e. by a point within the unit segment $[0, 1]$. Another indicator of reliability can also be used, namely,

$$x = \frac{1 - R}{R} = \frac{1}{R} - 1. \tag{25}$$

It should be noted that if $R \to 1$ then $x \to 0$, and if $R \to 0$ then $x \to \infty$. To return to the initial reliability identifier, R, the following equation is applied:

$$R = \frac{1}{x + 1}. \tag{26}$$

Let us now consider a system with a series reliability model, composed of n components of reliability $x_i = \frac{1}{r_i} - 1, i = 1, 2, \ldots, n$. The system reliability can be expressed by the following equation:

$$x_s(n) = \frac{1}{R_s(n)} - 1 = \frac{1}{\prod_{i=1}^{n} \frac{1}{x_i + 1}} - 1 = \prod_{i=1}^{n} (x_i + 1) - 1. \tag{27}$$

This relationship is rather complicated, but for values $x_i \ll 1$, which reflect high reliability, the resulting products can be neglected and the reliability of the system

can be approximated with good accuracy by the following equation:

$$x_s(n) \approx \sum_{i=1}^{n} x_i. \tag{28}$$

In the same manner, one can be shown that for a parallel reliability model with n components, when $x_i \ll 1$, the reliability of the redundant system can be approximated with good accuracy by the equation

$$x_P(n) \approx \prod_{i=1}^{n} x_i. \tag{29}$$

Based on this reliability indicator and when approximate equations (28) and (29) can be accepted, the optimization problem reduces to the maximization of the reliability function:

$$x_{\text{rs}} = \sum_{i=1}^{n} x_i^{k_i} \tag{30}$$

with the cost constraint (21). The Lagrangian function in this case is

$$L(k_1, k_2, \ldots, k_n, \eta) = \sum_{i=1}^{n} x_i^{k_i} + \eta\left(\sum_{i=1}^{n} c_i k_i - C^*\right). \tag{31}$$

By applying the partial derivatives presented in (23), the following system of algebraic equations results:

$$\begin{cases} x_i^{k_i} \ln x_i + \eta c_i = 0, \quad i = 1, 2, \ldots, n, \\ \sum\limits_{i=1}^{n} c_i k_i = C^*, \end{cases} \tag{32}$$

and then

$$\begin{cases} x_i^{k_i} = -\dfrac{\eta c_i}{\ln x_i}, \quad i = 1, 2, \ldots, n, \\ \sum\limits_{i=1}^{n} c_i k_i = C^*. \end{cases} \tag{33}$$

With the notation $\alpha_i = -c_i / \ln x_i, i = 1, 2, \ldots, n$, and when $\alpha_i > 0$ (i.e. $x_i < 0$ and, consequently, $r_i \geq 0.5$), by applying the logarithm function, the following system results:

$$\begin{cases} k_i \ln x_i = \ln \eta + \ln \alpha_i, \quad i = 1, 2, \ldots, n, \\ \sum\limits_{i=1}^{n} c_i k_i = C^*, \end{cases} \tag{34}$$

and then

$$\begin{cases} k_i = \dfrac{\ln \eta}{\ln x_i} + \dfrac{\ln \alpha_i}{\ln x_i}, \quad i = 1, 2, \ldots, n, \\ \displaystyle\sum_{i=1}^{n} c_i k_i = C^*. \end{cases} \tag{35}$$

With other two notations, $\pi_i = 1/\ln x_i = -\alpha_i/c_i$ and $\gamma_i = \pi_i \ln \alpha_i$, the system becomes

$$\begin{cases} k_i = \pi_i \ln \eta + \gamma_i, \quad i = 1, 2, \ldots, n, \\ \displaystyle\sum_{i=1}^{n} c_i(\pi_i \ln \eta + \gamma_i) = C^*, \end{cases} \tag{36}$$

and finally,

$$\begin{cases} \eta = e^v, \quad \text{where } v = \dfrac{C^* - \sum_{i=1}^{n} \gamma_i c_i}{\sum_{i=1}^{n} \pi_i c_i}, \\ k_i = \pi_i \ln \eta + \gamma_i. \end{cases} \tag{37}$$

The only impediment for solving the problem remains that the values k_i obtained by applying (37) are real values, and the allocation is, by its nature, a discrete problem. Therefore, it is necessary to determine the optimal allocation in integer numbers starting from the actual solution.

An approximate solution results by adopting the integers closest to the real values. But in this case, it is possible that the new solutions for k_i no longer satisfy the cost constraint. Starting from the value of the Lagrange multiplier η given by (37), a search process is carried out in a neighborhood of this value in order to obtain a better approximate solution for k_i while satisfying the cost constraint. Unfortunately, this approximate solution may not be accurate enough. Consequently, this approximate solution is further improved by applying other methods of refining the search process, as shown in the following sections.

The analytical approach is applied only to the case of active redundancy because for the case of standby spares, the Lagrangian function is more complicated and the system of partial derivatives is very difficult to solve. But the following optimization methods are extended to cover the case in which some subsystems may have active spares and the other ones may have other forms of redundancy.

5. Pairwise Hill Climbing

The analytical methods described in Sec. 4 only give an approximation of the solution since the final result has to contain integer values, and the they provide real values. In order to obtain the actual correct solution, or at least closer to the global optimum, one needs to perform an additional local search.

In Ref. 20, we introduced an original algorithm called "**P**airwise **H**ill **C**limbing" (PHC) to address this issue. For our particular problem, a classic hill climbing method cannot be applied because simply adding components to the subsystems would likely cause the violation of the cost constraint.

In the proposed PHC algorithm, two candidate solutions are generated for each pair of subsystems SS_1 and SS_2. The first candidate is created by adding one component to SS_1, i.e. the direct hill climbing operation. The second is created by adding one component to SS_1 and subtracting one from SS_2, i.e. a swapping operation. There is no ordering constraint for the subsystems being paired, so eventually, any subsystem will have the role of both SS_1 and SS_2.

Another specific optimization is the generation of the starting points of the search. Unlike classic hill climbing, which starts from a single initial point, in PHC there are several intertwined searches. One starting point is the approximate solution provided by the analytical method, let us call it I_0. Another set of starting points are the n candidate solutions I_{1i} created by subtracting one component from the subsystem i of I_0. Yet another starting point is the candidate solution I_2 created by subtracting one component from each subsystem of I_0. Of course, the subtraction procedure is applied only if the number of components of the targeted subsystem is greater than two, because a valid allocation requires that each subsystem has at least one component.

All the searches proceed by means of a priority queue. The initial solutions I_0, I_{1i} and I_2 are inserted into the priority queue. At a certain step, the solution with the best reliability is extracted from the priority queue and expanded using the two candidate solutions created as explained above. Since each child solution also has the information about its parent, the generated solutions can also be viewed as the results of a tree search.

One can reach a balance between the execution time and the quality of the final solution by limiting the search to a maximum number of levels in the tree $tl_{\max}$ and to a maximum number of solutions taken from the queue $ns_{\max}$.

6. RELIVE — An Innovative Evolutionary Algorithm

In Ref. 20, it was discovered that the best results were given by a hybrid method that combined the analytical approach and PHC. However, real-valued evolutionary algorithms (EA) were also investigated to see whether they could provide the optimal solution by themselves. Despite using them in various ways, e.g. optimizing the offspring generation process, steady-state evolution, extending the number of generations and imposing a limit on the search domain by using the estimations of the analytical method, it was found that the EAs were unable to match or surpass the results of the hybrid method.

Beside the reported results, several EA libraries available on the Internet were tried, together with non-evolutionary, differential-based methods. Even if they claimed to be able to handle constraints, they completely failed to provide useful

solutions — if not optimal, at least close to the solution discovered by our methods, likely because of the high dimensionality of the considered problems. Most of the time, they did not converge at all.

Therefore, in this work, a new evolutionary algorithm called "C**r**oss-Generational **E**volutionary Algorithm with **L**ocal **I**mpro**ve**ments" (RELIVE) is proposed, which combines the principles of a classic evolutionary algorithm with ideas inspired by alternative optimization methods, such as hill climbing and particle swarm optimization.

The encoding of the problem is real-valued, and each chromosome has n real genes, corresponding to k_i. The domain of the genes $[1, k_{\max}]$ depends on the problem. $k_{\max}$ is the maximum estimated value for any k_i.

The fitness function is the expression in Eq. (1). Since the genes are real numbers and k_i are integers, the gene values are rounded when computing the reliability.

The selection method is tournament selection with two individuals. We use arithmetic crossover and mutation by gene resetting. The stopping criterion is a fixed number of generations.

The main feature of the RELIVE algorithm is that an individual can live for a certain number of generations, and in each generation it improves its fitness by performing several hill climbing steps. In each generation it can participate in the selection process in the normal way, in order to have a chance to generate offspring. Thus, the size of the population is no longer fixed; it can vary as some individuals are "born" and others "die" in each generation.

More formally, the life length of an individual is initialized when it is created as a natural number $l = l_f + r_v$, where l_f is a fixed parameter and r_v is a natural random number uniformly drawn from the $\{0, 1, \ldots, l_f - 1\}$ set.

In each generation, l is decremented until it reaches 0, and then the individual is removed from the population.

A mechanism similar to elitism is also used. It has the goal of saving the best individual from a generation to the next, so as the best solution is never lost. In the RELIVE algorithm, the life length of the best chromosome c_{best} in a generation is set to $l(c_{\text{best}}) \leftarrow \max(l(c_{\text{best}}), l_f)$.

In Ref. 20, an additional post-child-generation procedure was considered, where the fitness values of the child, its mother and its father are compared and the best of the three chromosomes are added to the next generation. This was shown to improve the algorithm performance.

In this approach, this is no longer necessary because the mother and the father may survive anyway into the next generation because of their life spans. Thus, the child is directly inserted into the population.

The proposed algorithm also uses a new method to introduce new genetic information into the population. Let s_0 be the initial size of the population. In each generation, a number $f \cdot s_0$ of new, randomly generated chromosomes are inserted

into the population, where $f \in (0, 1)$. The life span l'_f of these individuals is initialized in a similar way, but with a slightly smaller value than l_f.

As mentioned above, in each generation, an individual performs a hill climbing procedure in order to improve its fitness. Hill climbing is achieved using only a mutation operator to generate n_n potential solution in the neighborhood of the current one. If a potential solution has a better fitness than the current one, the former becomes the current solution and the process is repeated for n_i iterations.

To generate a neighbor solution, three types of mutation are used with different probabilities as follows:

(1) *Gaussian mutation*: the new chromosome is generated by changing the value of a gene to a new one using a normal distribution with the mean in the current solution and a user-chosen standard deviation σ. Since the problems have multiple dimensions, the new values are generated independently for each dimension using a univariate Gaussian distribution;
(2) *Resetting mutation*: the new chromosome is generated by changing the value of a gene to a randomly generated one using a uniform distribution from the allowed domain of the gene $[1, k_{\max}]$;
(3) *Pairwise mutation*: in the new chromosome, two genes g_1 and g_2 are randomly selected using a uniform distribution, such as $g_1 \geq 2$ and $g_2 < k_{\max}$. Then, the values for these genes change by exchanging one unit: $(g'_1, g'_2) \leftarrow (g_1 - 1, g_2 + 1)$, i.e. the number of components decreases in one subsystem and increases in the other one. This type of mutation is inspired from the pairwise hill climbing idea, but simplified; it proved to be crucial for the success of the new evolutionary algorithm.

Let us call the probabilities for using these types of mutation p_{mg}, p_{mr} and p_{mp}, respectively, with $p_{\text{mp}} = 1 - (p_{\text{mg}} - p_{\text{mr}})$. It must be mentioned that these probabilities are only used to choose the type of mutation. The Gaussian and the resetting mutations also use a mutation probability p_m with the same meaning as in a standard evolutionary algorithm, i.e. the probability of actually changing a specific gene in the chromosome.

Since the problem has a constraint (Eq. (2)), there must be a way to enforce it in the evolutionary algorithm. Previous experiments showed that adding a penalty to the fitness function of the chromosomes that do not satisfy the constraint has a negative impact on the quality of the solution because, in the beginning, very few or no individuals may satisfy the cost constraint and this often leads to no solution at all, or to very poor solutions. Therefore, a repairing method was considered for the chromosomes that violate the cost constraint. In Ref. 20, the repairing procedure for a chromosome was to randomly remove one redundant component at a time until the remaining components satisfied the constraint. Each subsystem must have at least one component, so only redundant components may be removed.

In this work, we also optimized the repairing method. The components to be removed are no longer selected randomly. Instead, the subsystem with the highest

component cost and the number of components greater than one is chosen. An alternative method was also tried, i.e. choosing the subsystem with the highest reliability. However, the computational cost of finding such a system for each unfeasible chromosome proved to be too high for practical purposes. The cost-based repairing method is fast and it was empirically found to work better than the random choice.

7. Experimental Results

In this section, some numerical results of the previously described methods are presented. To compare the optimization methods, two experimental studies were performed. In the first study, only the active redundancy allocation is considered, and in the second one, a mixed redundancy strategy is considered, to increase the system reliability. More precisely, some subsystems use active redundancy, other ones use cold standby redundancy, and the rest use a warm–cold standby redundancy as presented in Sec. 3. For both studies, a system composed of 50 subsystems ($n = 50$) is considered, and two problems are solved. For the first problem, the values for reliability and cost (r_i and c_i) are randomly generated (initial input values), whereas, for the second one, for some components the values are chosen in such a way as to make the optimization more difficult (stressed input values).

7.1. *Experiment I. Active redundancy allocation*

We first consider this particular (rather hypothetical) case for which an analytical approach based on Lagrange multipliers is available, as presented in Sec. 4. This analytical method gives an approximate solution because the real values are converted to integer values. Consequently, this approximate solution is further improved by applying another method of refining the search process, the PHC algorithm, resulting in a hybrid method: Analytical + PHC. The results of the analytical method are a very good starting point for the PHC, because the component allocation is likely less than the optimal one and a procedure based on the hill climbing idea can easily add new components and thus improve the objective function.

Table 1 presents the initial values for randomly generated reliability and cost (Problem 1), and then the stressed input values, in which for some components, with a very similar reliability, highlighted in italic font, the cost is much different (Problem 2).

The best solutions found by the considered methods for these two problems are presented in Tables 2 and 3, respectively. We include below the parameter values used in the experiments, organized by algorithm:

(1) Pairwise Hill Climbing (PHC): $tl_{\max} = 20$ and $ns_{\max} = 1000$;
(2) Global search in the evolutionary algorithm: $k_{\max} = 15$, $s_0 = 30$ (this is the initial population size; throughout the evolution the population size is variable, about 190–200), $n_{\text{gen}} = 500$ (although in our experiments, the algorithm usually reached the final value after 200–300 generations), $f = 0.25$, $l_f = 4$ and $l'_f = 3$;

Table 1. The optimization problems considered for Experiment I. Active redundancy allocation.

Problem	Component reliability and cost (r_i, c_i) pairs	$C*$
Problem 1: initial input values	(0.974, 8), (0.601, 32), (0.928, 44), (0.981, 14), (0.961, 25), (0.979, 17), (0.972, 26), (0.975, 21), (0.978, 49), (0.770, 13), (0.931, 21), (0.975, 7), (0.873, 47), (0.995, 28), (0.999, 2), (0.906, 44), (0.999, 4), (0.999, 42), (0.995, 42), (0.581, 45), (0.890, 22), (0.644, 2), (0.990, 15), (0.557, 5), (0.971, 25), (0.971, 31), (0.936, 44), (0.972, 6), (0.978, 20), (0.976, 23), (0.972, 18), (0.964, 17), (0.728, 28), (0.677, 34), (0.954, 4), (0.820, 26), (0.995, 23), (0.634, 45), (0.943, 49), (0.872, 48), (0.979, 20), (0.842, 8), (0.999, 10), (0.975, 29), (0.830, 50), (0.990, 22), (0.988, 44), (0.842, 26), (0.756, 42), (0.986, 7)	4000
Problem 2: stressed input values	(0.974, 8), (0.601, 32), (0.928, 44), (0.981, 14), (0.961, 25), (0.979, 17), (*0.972, 46*), (*0.975, 41*), (0.978, 49), (0.770, 13), (0.931, 21), (0.975, 7), (0.873, 47), (0.995, 28), (0.999, 2), (0.906, 44), (0.999, 4), (0.999, 42), (0.995, 42), (*0.557, 45*), (0.890, 22), (0.644, 2), (0.990, 15), (*0.581, 5*), (*0.971, 45*), (0.971, 31), (0.936, 44), (0.972, 6), (0.978, 20), (0.976, 23), (*0.972, 8*), (0.964, 17), (*0.728, 8*), (*0.677, 44*), (0.954, 4), (0.820, 26), (0.995, 23), (0.634, 45), (0.943, 49), (*0.873, 10*), (0.979, 20), (*0.842, 50*), (0.999, 10), (0.975, 29), (0.830, 50), (0.990, 22), (0.988, 44), (*0.842, 8*), (0.756, 42), (0.986, 7)	4000

Table 2. The best solutions found by the considered methods: Experiment I, Problem 1.

Method	Optimal allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{sr}	Efficiency
Hybrid (Analytical + PHC)	3, 7, 3, 2, 2, 2, 2, 2, 2, 5, 3, 3, 3, 2, 2, 3, 2, 1, 2, 7, 3, 9, 2, 10, 2, 2, 3, 3, 2, 2, 2, 3, 5, 6, 3, 4, 2, 6, 2, 3, 2, 4, 2, 2, 4, 2, 2, 4, 4, 2	4000	0.9629	26.89
RELIVE	3, 7, 3, 2, 2, 2, 2, 2, 2, 5, 3, 3, 3, 2, 2, 3, 2, 1, 2, 7, 3, 9, 2, 10, 2, 2, 3, 3, 2, 2, 2, 3, 5, 6, 3, 4, 2, 6, 2, 3, 2, 4, 2, 2, 4, 2, 2, 4, 4, 2	4000	0.9629	26.89
Standard EA	3, 6, 2, 3, 2, 2, 2, 2, 2, 4, 4, 2, 4, 2, 3, 3, 3, 1, 2, 6, 3, 7, 3, 10, 2, 2, 3, 5, 2, 2, 2, 4, 5, 5, 3, 5, 1, 5, 3, 4, 2, 8, 2, 3, 3, 2, 2, 3, 4, 3	4000	0.9410	16.91

Table 3. The best solutions found by the considered methods: Experiment I, Problem 2.

Method	Optimal allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{sr}	Efficiency
Hybrid (Analytical + PHC)	3, 7, 3, 2, 2, 2, 2, 2, 2, 5, 3, 3, 3, 2, 2, 3, 2, 1, 2, 7, 3, 8, 2, 9, 2, 2, 3, 3, 2, 2, 3, 2, 6, 5, 3, 4, 2, 6, 2, 4, 2, 4, 2, 2, 4, 2, 2, 4, 4, 2	4000	0.9617	26.01
RELIVE	3, 7, 3, 2, 2, 2, 2, 2, 2, 5, 3, 3, 3, 2, 2, 3, 2, 1, 2, 7, 3, 8, 2, 9, 2, 2, 3, 3, 2, 2, 3, 2, 6, 5, 3, 4, 2, 6, 2, 4, 2, 4, 2, 2, 4, 2, 2, 4, 4, 2	4000	0.9617	26.01
Standard EA	3, 6, 2, 2, 2, 2, 2, 2, 3, 4, 3, 3, 3, 2, 11, 3, 6, 1, 2, 6, 5, 12, 2, 15, 2, 2, 2, 5, 2, 2, 3, 4, 10, 4, 5, 4, 2, 5, 3, 3, 2, 3, 3, 2, 3, 3, 2, 5, 4, 6	3991	0.9331	14.90

(3) Local search in the evolutionary algorithm: $n_n = 20$, $n_i = 20$, $\sigma = 2$, $p_{\text{mg}} = 0.25$, $p_{\text{mr}} = 0.25$, $p_{\text{mp}} = 0.5$ and $p_m = 0.2$.

The settings of the standard evolutionary algorithm are similar to those used for the global search in RELIVE, but with 50 chromosomes in the population and 1000 generations as a stopping criterion. For all the evolutionary approaches, the best result out of 10 runs is presented.

Beside the actual reliability value of the redundant system R_{rs}, we included an additional, more intuitive measure of the results, namely the redundancy efficiency defined as follows:

$$\text{Ef} = \frac{1 - R_{\text{ns}}}{1 - R_{\text{rs}}}. \tag{38}$$

The efficiency shows how many times the risk of a failure decreases for the redundant system, compared to the baseline, non-redundant one.

The experimental results for the two problems presented in Tables 2 and 3, respectively, show that proposed method (RELIVE) gives the same solution as the hybrid method (Analytical + PHC), which is a reference for this particular case in which only the active redundancy allocation is used. On the other hand, the standard EA is not able to reach an optimal solution, neither for Problem 1 nor for Problem 2.

7.2. *Experiment II. Mixed redundancy strategy*

In this experiment, a mixed redundancy strategy is considered. More precisely, for some subsystems an active redundancy (denoted by symbol "■") is used, for other ones, a cold standby redundancy (denoted by symbol "*") is applied, and for the rest, a warm–cold standby redundancy (denoted by symbol "□") is preferred, as presented in Sec. 3.

Table 4 presents the initial values for reliability, cost and type of redundancy randomly generated (Problem 1), and then with stressed input values (Problem 2).

The best solutions found by the considered methods for these two problems are presented in Tables 5 and 6, respectively.

These experimental results confirm that the proposed method (RELIVE) offers better solutions than those provided by the standard EA. Thus, the innovative evolutionary algorithm that combines global and local search proves to be a good choice for these problems. Despite the overhead incurred by local search in each generation, it is faster and often converges to the optimal solutions.

As a qualitative verification, the reliability of the redundant system (R_{sr}) is better when a mixed redundancy strategy is applied than when only the active redundancy allocation is used.

Table 4. The optimization problems considered for Experiment II. Mixed redundancy strategy.

Problem	Component reliability, cost and type: (r_i, c_i, t_i) tuples	$C*$
Problem 1: initial input values	(0.974, 8, □), (0.601, 32, ■), (0.928, 44, ■), (0.981, 14, □), (0.961, 25, ■), (0.979, 17, □), (0.972, 26, □), (0.975, 21, □), (0.978, 49, □), (0.770, 13, *), (0.931, 21, *), (0.975, 7, □), (0.873, 47, ■), (0.995, 28, □), (0.999, 2, □), (0.906, 44, ■), (0.999, 4, □), (0.999, 42, ■), (0.995, 42, □), (0.581, 45, ■), (0.890, 22, *), (0.644, 2, ■), (0.990, 15, □), (0.557, 5, ■), (0.971, 25, □), (0.971, 31, □), (0.936, 44, *), (0.972, 6, □), (0.978, 20, □), (0.976, 23, □), (0.972, 18, □), (0.964, 17, ■), (0.728, 28, *), (0.677, 34, *), (0.954, 4, *), (0.820, 26, ■), (0.995, 23, □), (0.634, 45, ■), (0.943, 49, *), (0.872, 48, *), (0.979, 20, □), (0.842, 8, ■), (0.999, 10, *), (0.975, 29, □), (0.830, 50, *), (0.990, 22, □), (0.988, 44, □), (0.842, 26, *), (0.756, 42, *), (0.986, 7, □)	4000
Problem 2: stressed input values	(0.974, 8, □), (0.601, 32, ■), (0.928, 44, ■), (0.981, 14, □), (0.961, 25, ■), (0.979, 17, □), (*0.972, 46,* □), (*0.975, 41,* □), (0.978, 49, □), (0.770, 13, *), (0.931, 21, *), (0.975, 7, □), (0.873, 47, ■), (0.995, 28, □), (0.999, 2, □), (0.906, 44, ■), (0.999, 4, □), (0.999, 42, ■), (0.995, 42, □), (*0.557, 45,* ■), (0.890, 22, *), (0.644, 2, ■), (0.990, 15, □), (*0.581, 5,* ■), (*0.971, 45,* □), (0.971, 31, □), (0.936, 44, *), (0.972, 6, □), (0.978, 20, □), (0.976, 23, □), (*0.972, 8,* □), (0.964, 17, ■), (*0.728, 8,* *), (*0.677, 44,* *), (0.954, 4, *), (0.820, 26, ■), (0.995, 23, □), (0.634, 45, ■), (0.943, 49, *), (*0.873, 10,* *), (0.979, 20, □), (*0.842, 50,* ■), (0.999, 10, *), (0.975, 29, □), (0.830, 50, *), (0.990, 22, □), (0.988, 44, □), (*0.842, 8,* *), (0.756, 42, *), (0.986, 7, □)	4000

Table 5. The best solutions found by the considered methods: Experiment II, Problem 1.

Method	Optimal allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{sr}	Efficiency
RELIVE	3, 8, 3, 2, 3, 2, 2, 2, 2, 4, 3, 3, 4, 2, 2, 3, 2, 1, 2, 7, 3, 10, 2, 10, 2, 2, 3, 3, 2, 2, 2, 3, 4, 4, 3, 5, 2, 7, 2, 3, 2, 5, 2, 2, 3, 2, 2, 3, 4, 2	4000	0.9837	61.07
Standard EA	2, 7, 3, 2, 3, 3, 2, 2, 2, 4, 3, 3, 3, 2, 1, 3, 3, 1, 2, 6, 4, 10, 4, 11, 2, 2, 3, 2, 4, 2, 2, 3, 4, 4, 9, 5, 2, 6, 2, 3, 2, 12, 2, 2, 3, 2, 2, 4, 3, 2	3997	0.9737	37.92

Table 6. The best solutions found by the considered methods: Experiment II, Problem 2.

Method	Optimal allocation: $k_1, k_2, \ldots, k_n$	C_{rs}	R_{sr}	Efficiency
RELIVE	2, 7, 3, 2, 3, 2, 2, 2, 2, 4, 3, 3, 4, 2, 2, 3, 2, 1, 2, 8, 3, 9, 2, 10, 2, 2, 3, 3, 2, 2, 3, 3, 4, 4, 3, 4, 2, 6, 2, 3, 2, 4, 2, 2, 3, 2, 2, 4, 4, 2	4000	0.9815	53.77
Standard EA	3, 7, 2, 2, 2, 2, 2, 2, 2, 5, 3, 3, 4, 2, 2, 3, 1, 1, 1, 7, 3, 13, 2, 13, 2, 2, 2, 3, 3, 2, 3, 2, 6, 4, 9, 4, 2, 7, 2, 3, 2, 3, 3, 4, 3, 2, 2, 5, 5, 2	3996	0.9645	28.10

8. Conclusions

In this paper, some optimization methods were presented for the problem of maximizing the reliability of a redundant system, with a series-redundant reliability model, in the presence of cost-related constraints. For the redundant system, a mixed

redundancy strategy is considered: with active spares, cold standby spares or warm-cold standby spares. Beside an analytical solution based on Lagrange multipliers, other general techniques were applied: an original algorithm, PHC, based on the hill climbing idea, but using swaps and a priority queue in addition to the incremental greedy improvements, in order to fine-tune the approximate solutions found by the analytical model and an innovative evolutionary algorithm, RELIVE, that combines global search, in the same manner as a classic EA, and local search, in the form of an original hill climbing process used to gradually improve the fitness of an individual as it can survive for a specified number of generations.

References

1. F. A. Tillman, C. L. Hwang and W. Kuo, Optimization techniques for system reliability with redundancy, A review, *IEEE Trans. Reliab.* **26** (1977) 148–155.
2. F. A. Tillman, C. L. Hwang and W. Kuo, *Optimization of System Reliability* (Marcel Dekker, New York, 1980).
3. M. Shooman, *Reliability of Computer Systems and Networks* (John Wiley & Sons, New York, 2002), pp. 331–383.
4. K. B. Misra (ed.), Optimal reliability design of a system, B. K. Lad, M. S. Kulkarni and K. B. Misra, *Handbook of Performability Engineering* (Springer-Verlag, London, 2008), pp. 499–519.
5. M. Chern, On the computational complexity of reliability redundancy allocation in series system, *Ope. Res. Lett.* **11** (1992) 309–315.
6. A. A. Albert, A measure of the effort required to increase reliability, Technical Report No. 43, Stanfort University, Applied Mathematics and Statistics Laboratory (1958).
7. K. B. Misra, A simple approach for constrained redundancy optimization problems, *IEEE Trans. Reliability* **R-20**(3) (1971) 117–120.
8. K. K. Agarwal and J. S. Gupta, On minimizing the cost of reliable systems, *IEEE Trans. Reliab.* **R-24** (1975) 205–206.
9. V. Rajendra Prasad, K. P. K. Nair and Y. P. Aneja, A heuristic approach to optimal assignment of components to parallel-series network, *IEEE Trans. Reliability* **40**(5) (1992) 555–558.
10. E. El-Neweihi, F. Proschan and J. Sethuraman, Optimal allocation of components in parallel-series and series-parallel systems, *J. Appl. Probability* **23**(3) (1986) 770–777, R10.
11. H. Everett, Generalized lagrange multiplier method of solving problems of optimal allocation of resources, *Ope. Res.* **11** (1963) 399–417.
12. K. B. Misra, Reliability optimization of a series-parallel system, part i: Lagrangian multipliers approach, part ii: Maximum principle approach, *IEEE Trans. Reliability* **R-21** (1972) 230–238.
13. W. R. Blischke and D. N. Prabhakar, *Reliability: Modelling, Prediction, and Optimization* (Wiley, New York, 2000).
14. R. Belmann and S. Dreyfus, Dynamic programming and the reliability of multi-component devices, *Ope. Res.* **6**(2) (1958) 200–206.
15. D. W. Coit and A. E. Smith, Reliability optimization of series-parallel systems using a genetic algorithm, *IEEE Trans. Reliability* **45** (1996) 254–260.
16. M. Marseguerra and E. Zio, System design optimization by genetic algorithms, in *Proc. Annual Reliability and Maintainability Symposium*, IEEE, Los Angeles, CA, 2000, pp. 222–227.

17. M. Agarwal and R. Gupta, Genetic search for redundacy optimization in complex systems, *J. Qual. Maintenance Eng.* **12**(4) (2006) 338–353.
18. R. Romera, J. E. Valdes and R. I. Zequeira, Active redundancy allocation in systems, *IEEE Trans. Reliability* **53**(3) (2004) 313–318. https://ieeexplore.ieee.org/document/1331673.
19. F. Leon and P. Cascaval, 01IP and QUBO: Optimization methods for redundancy allocation in complex systems, in *Proc. 2019 23rd Int. Conf. System Theory, Control and Computing (ICSTCC 2019)*, Sinaia, Romania, 2019, pp. 877–882, doi: 10.1109/icstcc.2019.8885826.
20. P. Caşcaval and F. Leon, Active redundancy allocation in complex systems by using different optimization methods, in *Proc. 11th Int. Conf. Computational Collective Intelligence (ICCCI 2019)*, Hendaye, France, eds. N. Nguyen, R. Chbeir, E. Exposito, P. Aniorte and B. Trawinski, Computational Collective Intelligence, Lecture Notes in Computer Science, Vol. 11683, 2019, pp. 625–637, doi: 10.1007/978-3-030-28377-3_52.

Microelectronics Journal 93 (2019) 104619

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

March test algorithm for unlinked static reduced three-cell coupling faults in random-access memories

P. Caşcaval [a,*], D. Caşcaval [b]

[a] "Gheorghe Asachi" Technical University of Iaşi, Department of Computer Science and Engineering, Bd. D. Mangeron, nr. 27, Iaşi, 700050, Romania
[b] "Gheorghe Asachi" Technical University of Iaşi, Department of Industrial Engineering, Bd. D. Mangeron, nr. 27, Iaşi, 700050, Romania

ABSTRACT

A memory fault model regarding the unlinked static three-cell coupling faults in $n \times 1$ random-access memories is discussed. This model is an extension of the well-known model of unlinked static two-cell coupling faults. Because this model of three-cell coupling is limited to the physically neighbouring memory cells, it can also be considered a neighbourhood pattern-sensitive model. An efficient march test algorithm able to cover this reduced model of three-cell coupling is presented in this letter.

1. Introduction

The fault model of unlinked static three-cell coupling faults in $n \times 1$ random-access memories as presented in Ref. [1] is discussed.

Based on the model of all static simple two-cell coupling faults presented in Ref. [2], a fault primitive (FP) based-model of three-cell coupling is defined in Ref. [1]. A set of 72 FPs completely covers this model of three-cell coupling faults.

It is said that two or more FPs are not linked when they do not influence each other. According to the fault classification presented in Refs. [2,3], the class of 'static faults' refers to those faults which are sensitized by performing at most one operation in the memory, whereas the class of 'dynamic faults' refers to those faults which can be sensitized by performing more than one operation sequentially. As in Ref. [1], we only address the class of static coupling faults.

The first test algorithm dedicated to a model of three-cell coupling is proposed by Nair, Thatte and Abraham [4]. Other two more efficient test algorithms are given by Cockburn (S3CTEST and S3CTEST2) [5]. Because the authors assume that the coupling cells can be anywhere in the memory, all these test algorithms are quite long. For example, the test algorithm S3CTEST needs $5n \log_2 n + 22.5n$ operations. To reduce the length of the tests, Caşcaval and Bennett [6] have limited to the more realistic case where only the physically neighbouring memory cells may form a three-cell coupling. For this model, they proposed a march memory test algorithm (MT) with a length of $38n$. An improved algorithm dedicated to this model (MT-R3CF) with a length of $30n$ is given by Caşcaval, Bennett and Huţanu [7]. Both test algorithms, MT and MT-R3CF, assume that a memory fault can be sensitized only by a transition write operation into a cell. In Ref. [1], the model of three-cell coupling is extended by considering other classes of faults, such as the faults sensitized by a non-transition write or a read operation, namely: disturb coupling, read destructive coupling, deceptive read destructive coupling or incorrect read, as defined in Ref. [2] and in other works. This is the model we consider in this work.

As in Refs. [1,6,7], the authors restricted their study to the case where only the physically neighbouring memory cells may be affected by a three-cell coupling fault. As in Ref. [1], six coupling patterns of three physically neighbouring cells (denoted by CP_1, CP_2, …, CP_6) are considered in this work, as presented in Fig. 1.

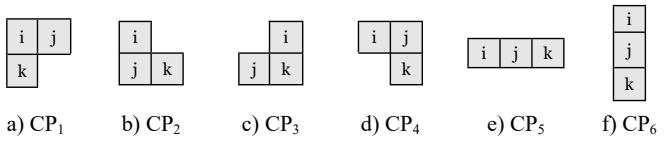
This model is known as 'reduced three-cell coupling'. For a better comparison, all the preliminary considerations presented in Ref. [1] are also accepted in this work. Note that the model we discuss can also be viewed as a neighbourhood pattern-sensitive fault model (NPSF). Nevertheless, in this model, any cell in the group may be a victim cell of the other two aggressor cells, not just the central cell, as it is usually considered in the NPSF model. As in any work dedicated to NPSFs, we assume that the scramble map is completely known, so we can run the test using this physical address information. Naturally, in the memory under testing, one or more groups of coupled cells may exist. As in Ref. [1–7], we assume that the groups of coupled cells are disjoint.

For this model, the test algorithm March SR3C with a length of $66n$ operations is proposed in Ref. [1]. In this letter, we present a more efficient march test (MT-SR3C) with only $54n$ operations able to cover this model of unlinked static three-cell coupling faults.

Notation: The following notations are used to describe a memory test algorithm:

* Corresponding author.
E-mail addresses: cascaval@cs.tuiasi.ro (P. Caşcaval), cascaval@tex.tuiasi.ro (D. Caşcaval).

https://doi.org/10.1016/j.mejo.2019.104619
Received 24 March 2019; Received in revised form 27 July 2019; Accepted 16 September 2019
Available online 19 September 2019



a) CP_1 b) CP_2 c) CP_3 d) CP_4 e) CP_5 f) CP_6

Fig. 1. Coupling patterns of three physically neighbouring cells denoted by i, j, and k.

r – a read operation from a memory cell; when the expected value is explicitly indicated, the operation is denoted by r0 or r1 as the case;
w0 (w1) – a write 0 (1) operation into a cell;
0w1 (1w0) – an up (down) transition write operation;
0w0, 1w1 – non-transition write operations;
w_t – a transition write operation into a cell when the transition type is not explicitly indicated (i.e., 0w1 or 1w0 as appropriate);
w_{nt} – a non-transition write operation into a cell when the logical value of operation is not indicated (i.e., 0w0 or 1w1 as appropriate).

The memory test March MT-SR3C: To detect the faults of this model of three-cell coupling we propose the march test algorithm presented in Fig. 2, where BGC_1, BGC_2, BGC_3 and BGC_4 are test sequences for memory checking and background change.

The sequence BGC_1 fills the odd columns in the memory with 0 and the even ones with 1, and BGC_3 does it the other way around, whereas BGC_2 and BGC_4 perform a checkerboard data background and its complement, respectively. The four patterns used for data background changes denoted by BP_1, BP_2, BP_3, and BP_4 are presented in Fig. 3.

The fourteen test sequences that compose this march algorithm are identified with the superscript (x), where $x \in \{0, 1, \ldots, 13\}$. In the second part of the test, composed of the sequences (5)–(12), before a march element is applied, the memory data background is changed with a different pattern. As a remark, this technique allows to apply march tests even for NPSFs (see, for example [8,9]). An excellent overview on this topic is presented in Ref. [10]. One can observe in Fig. 2 that any data background change affects half of the n cells. Because each write operation for changing the state of a memory cell is preceded by a read operation (to check the cell state), during a test sequence BGC_i, $i \in \{1, 2, 3, 4\}$, n operations are carried out: $n/2$ read operations and $n/2$ write operations. Thus, the memory test MT-SR3C has a length of $n+4 \times 4 \times n+4 \times (1+8) \times n+n=54n$ operations.

$\{\Updownarrow (w0)^{(0)};$
$\Uparrow (r0, \mathbf{0w1}, 1w1, r1)^{(1)}; \Uparrow (r1, \mathbf{1w0}, 0w0, r0)^{(2)};$
$\Downarrow (r0, \mathbf{0w1}, 1w1, r1)^{(3)}; \Downarrow (r1, \mathbf{1w0}, 0w0, r0)^{(4)};$
$BGC_1{}^{(5)}; \Uparrow (r, \mathbf{w_t}, w_{nt}, r, r, \mathbf{w_t}, w_{nt}, r)^{(6)};$
$BGC_2{}^{(7)}; \Uparrow (r, \mathbf{w_t}, w_{nt}, r, r, \mathbf{w_t}, w_{nt}, r)^{(8)};$
$BGC_3{}^{(9)}; \Uparrow (r, \mathbf{w_t}, w_{nt}, r, r, \mathbf{w_t}, w_{nt}, r)^{(10)};$
$BGC_4{}^{(11)}; \Uparrow (r, \mathbf{w_t}, w_{nt}, r, r, \mathbf{w_t}, w_{nt}, r)^{(12)};$
$\Updownarrow (r0)^{(13)}\}$

Fig. 2. Memory test algorithm MT-SR3C.

BP_1 BP_2 BP_3 BP_4

Fig. 3. Patterns used for data background changes (BP_1, BP_3 – column stripe; BP_2, BP_4 – checkerboard).

Table 1
The expected value at a read operation during data background changes.

BGC_1	BGC_2	BGC_3	BGC_4
$A_c[0]$	$A_c[0] \oplus A_r[0]$	$\overline{A_c[0]}$	$\overline{A_c[0] \oplus A_r[0]}$

These four patterns (BP_i, $i \in \{1, 2, 3, 4\}$) allow easy implementation of the sequences for data background changes. Specifically, during the execution of a test sequence BGC_i, $i \in \{1, 2, 3, 4\}$, the memory cells which must be changed are selected based on the LSB of the column address ($A_c[0]$) and on the LSB of the row address ($A_r[0]$). Also, the expected value at a read operation which precedes the transition write for changing the state of the cell is determined based on $A_c[0]$ and $A_r[0]$, as presented in Table 1.

Regarding the march elements (6), (8), (10), and (12), the expected value of a read operation (r) and the value used for a write operation (w_t or w_{nt}) are also determined on the basis of $A_c[0]$ and $A_r[0]$.

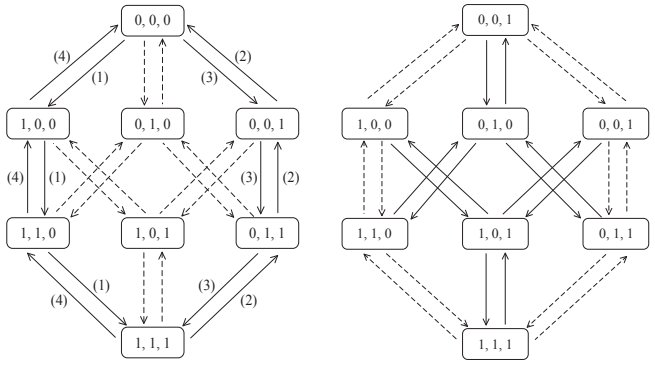
Theorem. *The march test algorithm MT-SR3C is able to detect all unlinked static faults of reduced three-cell coupling.*

Proof. To detect a fault in the memory under testing we need to be able to sensitize the fault by a suitable memory operation, and then to observe the changed value of the cell affected by the fault. It is well-known that if a memory test covers all the FPs that define a fault model, it also covers all unlinked faults of this model. In other words, the set of unlinked faults dominates the set of FPs. Consequently, we can limit the proof to the set of FPs that define the model of reduced three-cell coupling. Consider an arbitrary group G of three neighbouring memory cells that matches to one of the six coupling patterns presented in Fig. 1. Taking into consideration the order in which these cells are accessed during the memory testing, we refer to the cells in group G by i, j and k as illustrated in Fig. 1. As follows, we show that the test algorithm MT-SR3C is able to sensitize and observe any FP that may affect a cell in the group of coupled cells G = {i, j, k}.

2. MT-SR3C sensitizes any FP in group G of coupled cells

We have to show that during the memory testing, MT-SR3C is able to perform all possible operations in the group of cells G = {i, j, k}. In other words, we have to show that, during the memory testing, MT-SR3C is able to entirely cover the graph of states that describes the normal operation of these cells. The Proof consists of two stages: in the first stage we focus on the transition write operations, and in the second one, on the read and the non-transition write operations.

Stage 1. Transition writes: A transition write operation changes the state of the memory cell. Such an operation is highlighted in boldface in



a) March elements (1)–(4) b) March elements: (6), (8), (10), (12)

Fig. 4. The graph of states for a group of cells G = {i, j, k} and the transitions carried out by the test MT-SR3C.

the test algorithm description (Fig. 2). With the first memory initialisation, any group of three cells is brought to the state (0,0,0). By applying the march elements (1)−(4), the test MT-SR3C performs the transitions presented in Fig. 4a and highlighted with solid line in each group of cells G = {i, j, k}.

A more detailed analysis is necessary regarding the operations carried out by the other march elements (6), (8), (10), and (12). After a data background change, the state of a group G depends both on the pattern used for memory initialisation and on the coupling pattern of this group of cells. Fig. 5 illustrates all possible cases.

Table 2 presents the initial states of a group of cells G after data background changes, depending on the coupling pattern, as highlighted in Fig. 5.

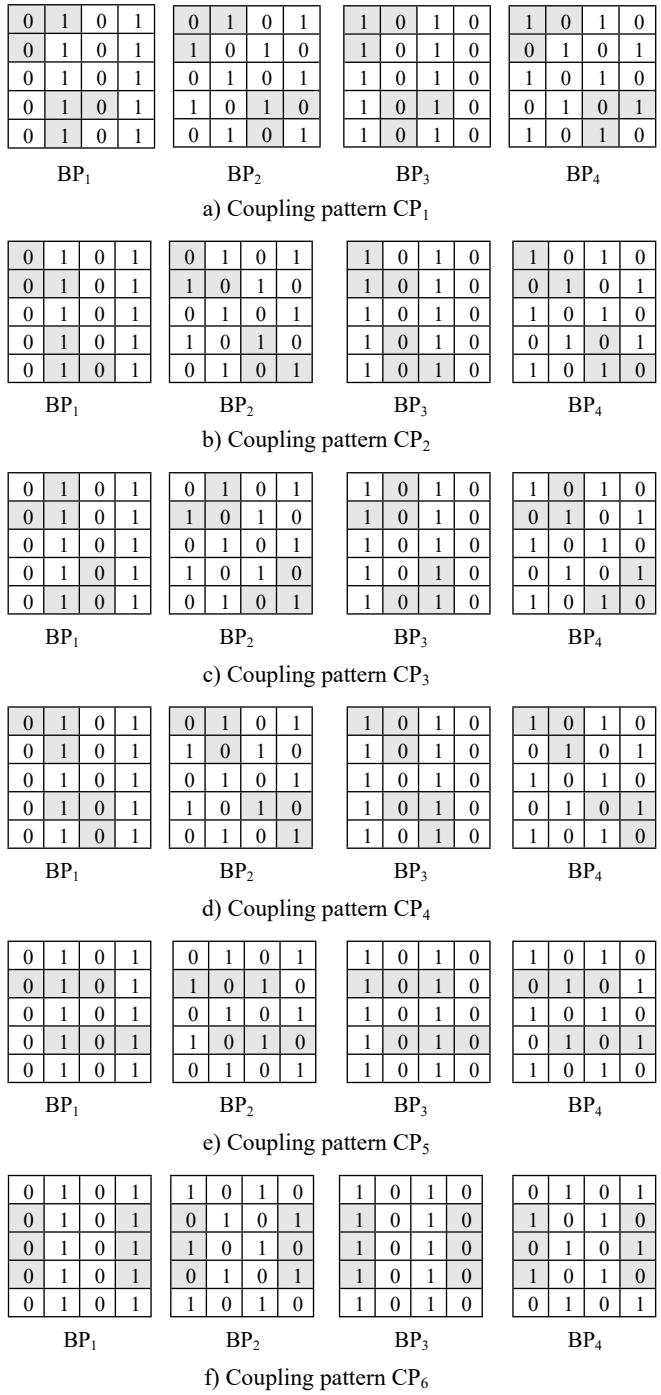


Fig. 5. The initialisation of a group of cells by changing the memory data background.

Table 2
The initial states of a group of cells G = {i, j, k} after data background changes.

Coupling pattern	Background patterns	
	BP_1 and BP_3	BP_2 and BP_4
CP_1	(0,1,0), (1,0,1)	(0,1,1), (1,0,0)
CP_2	(0,0,1), (1,1,0)	(0,1,0), (1,0,1)
CP_3	(0,1,0), (1,0,1)	(0,0,1), (1,1,0)
CP_4	(0,1,1), (1,0,0)	(0,1,0), (1,0,1)
CP_5	(0,1,0), (1,0,1)	(0,1,0), (1,0,1)
CP_6	(0,0,0), (1,1,1)	(0,1,0), (1,0,1)

As shown in Table 2, any group of cells G = {i, j, k} reaches the states (0,1,0) and (1,0,1) by two data background changes. As highlighted with solid lines in Fig. 4b, starting with one of the two states, by applying the march element ⇑(r, w_t, w_{nt}, r, r, w_t, w_{nt}, r) three neighbouring nodes are reached, by going back and forth, and finally the group of cells is left in its initial state. Note that for two neighbouring nodes the states differ by a single bit, whereas for two non-neighbouring nodes, the states differ by at least two bits. We can conclude that, by applying the march elements (6), (8), (10), and (12), the test MT-SR3C assures execution of the transitions highlighted with solid lines in the graph in Fig. 4b in every group of cells, regardless of the coupling pattern. As highlighted in the two graphs, MT-SR3C assures execution of all possible transition write operations in any group of cells G = {i, j, k}.

Stage 2. Reads and non-transition writes: One can observe in the test description that, after a new state has been reached by a transition write operation, MT-SR3C performs other two operations without leaving the current state: a non-transition write operation and a read operation. This means that, during the memory testing, all normal operations are carried out in any group of cells G = {i, j, k}. Therefore, the test MT-SR3C is able to sensitize any FP in a group G of coupled cells.

3. MT-SR3C observes any FP sensitized in group G of coupled cells

We have to show that the test MT-SR3C fulfils two conditions: (a) after a sensitizing operation into a cell, the test reads the cell to check its state, before a new transition write operation into the cell is allowed to happen, and (b) after one or more operations have been carried out on a cell in group G (a possible aggressor cell), the test reads the other two cells in group G (possible victim cells) to check if state is changed.

Looking at the test description, one can observe that the test MT-SR3C satisfies condition (a) because it carries out a read operation before any transition write operation. Regarding condition (b), note that without the first memory initialisation, any march element or any sequence for data background changing starts with a read operation. Taking also into account the final checking, we can conclude that condition (b) is also satisfied. In any case we can easily check that this assessment is true if we write down the operations carried out in the cells of group G during the memory testing. Therefore, MT-SR3C is able to detect any sensitized FP in a group G of three coupled cells. This result has also been checked by means of a simulation study. □

Remark 1. The test MT-SR3C is also able to cover the model of unlinked static two-cell coupling faults presented in Ref. [2]. Specifically, the first five march elements of this test completed with a final checking are sufficient for this two-cell coupling model. That implies a number of 18*n* operations. Other march tests of the same length dedicated to the two-cell coupling model can be found in Ref. [11]. Note that the memory test March SS given by Hamdioui et al. [2] to cover this model has a length of 22*n*.

Remark 2. For the coupling patterns CP_1, CP_3, and CP_5, the test sequences (7), (8), (11), and (12) are not necessary so that these sequences can be removed from the memory test algorithm. Likewise for the

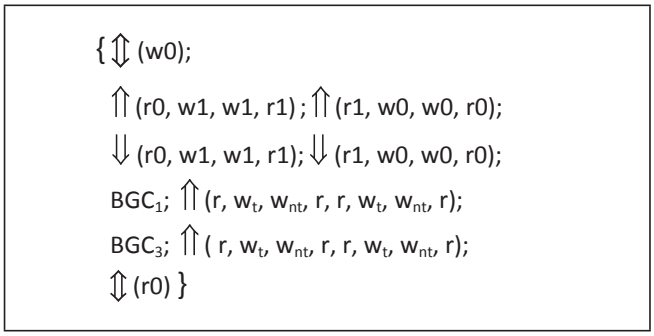


Fig. 6. Memory test algorithm MT-SR3C-1.

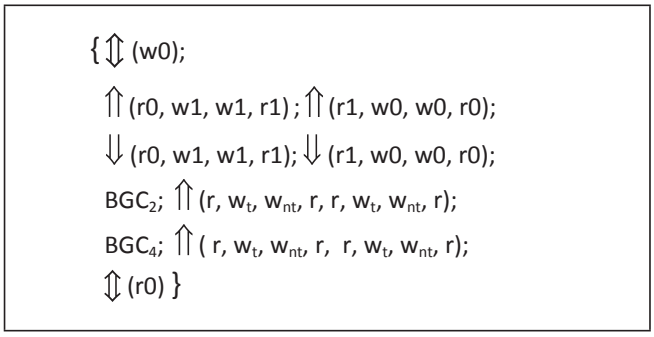


Fig. 7. Memory test algorithm MT-SR3C-2.

coupling patterns CP_2, CP_4, and CP_6, the test sequences (5), (6), (9), and (10) are not necessary. Consequently, for the subset of coupling patterns {CP_1, CP_3, CP_5}, the test algorithm is composed of the sequences (0)–(6), (9), (10), and (13), whereas for the subset of coupling patterns {CP_2, CP_4, CP_6}, the test algorithm is composed of the test sequences (0)−(4), (7), (8), (11)−(13). These reduced march memory tests, called MT-SR3C-1 and MT-SR3C-2, are explicitly presented in Fig. 6 and Fig. 7, respectively.

Note that to change the data background from BP_1 to BP_3 (by the sequence BGC_3 in case of test MT-SR3C-1), or from BP_2 to BP_4 (by the sequence BGC_4 in case of test MT-SR3C-2), the value of each memory cell must be changed. Both test sequences require $2n$ operations. Thus, for such a subset of coupling patterns, the memory test is reduced from $54n$ to $n+4\times4\times n+(1+8)\times n+(2+8)\times n+n = 37n$ operations. It is important to note that, except for the operations used to change the data background, in such a case, any arc in the graph of states is crossed only once during the memory testing (Fig. 4). That means that for such a subset of coupling patterns, MT-SR3C-1 and MT-SR3C-2 (of length $37n$) are almost optimal memory tests.

4. Conclusion

Compared to the test March SR3C, the test MT-SR3C is more efficient, as it is 18.5% shorter. The main idea used in this work for devising a shorter test is that, between two consecutive write operations of a march element, a read operation is not necessary when the second write operation is a non-transition one.

References

[1] P. Caşcaval, D. Caşcaval, March SR3C: a Test for a reduced model of all static simple three-cell coupling faults in random-access memories, Microelectron. J. 41 (4) (2010) 212–218.
[2] S. Hamdioui, A.J. van de Goor, M. Rodgers, March SS: a test for all static simple RAM faults, in: Proceeding of the IEEE Workshop on Memory Technology, Design and Testing, Isle of Bendor, July 2002, pp. 95–100. France.
[3] S. Hamdioui, Z. Al-Ars, A.J. van de Goor, M. Rodgers, Dynamic faults in random-access-memories: concept, fault models and tests, J. Electron. Testing Theory and Appl. 19 (2) (2003) 195–205.
[4] R. Nair, S. Thatte, J. Abraham, Efficient algorithms for testing semiconductor random-access memories, IEEE Trans. Comput. C- 27 (6) (1978) 572–576.
[5] B.F. Cockburn, Deterministic testing for detecting single V-coupling faults in RAMs, J. Electron. Testing Theory and Appl. 5 (1) (1994) 91–113.
[6] P. Caşcaval, S. Bennett, Efficient march test for 3-coupling faults in random-access memories, Microprocess. Microsyst. 24 (10) (2001) 501–509.
[7] P. Caşcaval, S. Bennett, C. Huţanu, Efficient march tests for a reduced 3-coupling and 4-coupling faults in RAMs, J. of Electron. Testing Theory and Appl. 20 (2) (2004) 227–243.
[8] K.L. Cheng, M.F. Tsai, C.W. Wu, Neighbourhood pattern-sensitive fault testing and diagnostics for random-access memories, IEEE Trans. Comput. Aided Des. Integr Circuits Syst. 21 (11) (2002) 1328–1336.
[9] C. Huzum, P. Caşcaval, 'March test for static neighborhood pattern-sensitive faults in random-access memories', elektronika ir elektrotechnika – section system engineering, Computer Technol. 119 (3) (2012) 81–86.
[10] I. Mrozek, Multi-run Memory Tests for Pattern Sensitive Faults, Springer Verlag, 2019.
[11] G. Harutunyan, V.A. Vardanian, Y. Zorian, Minimal march tests for unlinked static faults in random access memories, in: Proceeding of the 23rd IEEE VLSI Symposium, May 2005, pp. 53–59. Palm Springs, CA, USA.

Hindawi
Complexity
Volume 2018, Article ID 5967604, 11 pages
https://doi.org/10.1155/2018/5967604

WILEY | Hindawi

Research Article

Approximate Method to Evaluate Reliability of Complex Networks

Petru Caşcaval

Department of Computer Science and Engineering, "Gheorghe Asachi" Technical University of Iaşi, Dimitrie Mangeron Street, 27, 700050 Iaşi, Romania

Correspondence should be addressed to Petru Caşcaval; cascaval@cs.tuiasi.ro

Received 20 April 2018; Revised 12 July 2018; Accepted 31 July 2018; Published 12 November 2018

Academic Editor: Ireneusz Czarnowski

This paper deals with the issue of reliability evaluation in complex networks, in which both link and node failures are considered, and proposes an approximate method based on the minimal paths between two specified nodes. The method requires an algorithm for transforming the set of minimal paths into a sum of disjoint products (SDP). To reduce the computation burden, in the first stage, only the links of the network are considered. Then, in the second stage, each term of the set of disjoint link-products is separately processed, taking into consideration the reliability values for both links and adjacent nodes. In this way, a reliability expression with a one-to-one correspondence to the set of disjoint products is obtained. This approximate method provides a very good accuracy and greatly reduces the computation for complex networks.

1. Introduction

The network reliability theory is extensively applied in many real-world systems that can be modeled as stochastic networks, such as communication networks, sensor networks, social networks, etc. A variety of tools are used for system modeling and computation of reliability or availability indices that describe in a certain way the ability of a network to carry out a desired operation. Most tools are based on algorithms described in terms of minimal path set or minimal cut set (see, for example, [1–8]). Unfortunately, the problem of computing the network reliability based on the set of the minimal paths or cuts is NP-hard [7, 9]. For this reason, in case of more complex networks, other techniques for approximate reliability evaluation are also applied, such as those based on network decomposition or on Monte Carlo simulations (see, for example, [10–16]).

In this work, we deal with the problem of evaluation of two-terminal reliability or availability indices in medium-to-large networks, based on SDP algorithms, in which both link and node failures are considered.

Many authors address this problem assuming that the nodes of the system are perfectly reliable (see, for example, [1, 4–6]). However, in a communication system, nodes also have certain probability of failure so that the reliability evaluation assuming perfect nodes is not realistic.

The failure of a node inhibits the work of all links connected to it. Based on this concept, starting from the given network with unreliable nodes, reduced models with perfect nodes but with links having increased failure probabilities can be obtained. This method is simple, but not so accurate. Because the failure of a node inhibits the work of all adjacent links, the work of the links connected to it depends on the state of this common node. However, a reduced model is solved under the hypothesis according to which the failures that affect the network are independent. For this reason, the reliability estimation must be accepted with caution. Indeed, the estimation error of two-terminal network reliability could be unacceptable in many cases, especially when the failure probabilities of the nodes have high values.

To highlight this aspect, let us consider a simple network with unreliable nodes as presented in Figure 1(a). The reliability of the connection between nodes 1 and 4 has to be evaluated. These two terminal nodes are considered in series with the rest of the network. Three reduced models with perfect nodes and links having increased probabilities of failure are presented in Figures 1(b)–1(d). With dashed line, it is indicated that the failure of a node can be modeled by a cut

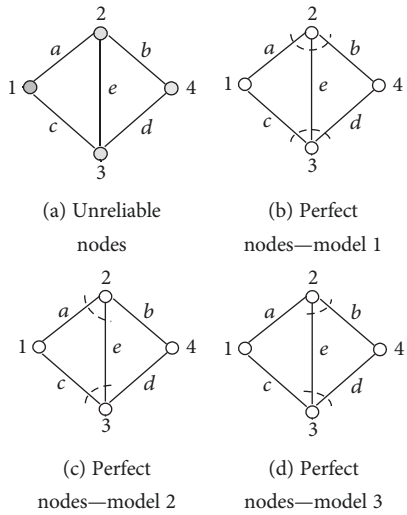


Figure 1: A simple network: (a) initial model; (b–d) reduced models.

of the links connected to it. Consequently, in case of a reduced model with perfect nodes, the reliability of the links in the network must be adjusted accordingly. It is easy to observe that other reduced models are also possible.

For the reduced models presented in Figure 1, Table 1 shows how the reliability of each link in the network is adjusted to capture the fact that the nodes of the given network are also unreliable.

For a numerical evaluation of these approximate models, let us consider a network with the following reliability values for the nodes and the links: $p_1 = 0.99$, $p_2 = 0.98$, $p_3 = 0.97$, $p_4 = 0.96$, $p_a = 0.99$, $p_b = 0.98$, $p_c = 0.97$, $p_d = 0.96$, and $p_e = 0.95$. The numerical results expressing the reliability estimation of the connection between nodes 1 and 4 $(R_{1\text{-}4})$ are presented in Table 2.

These numerical results show that the reduced models with perfect nodes might be useful in a way, but the reliability estimation is not so accurate, even for this simple network.

A better solution that makes a link-based reliability evaluation algorithm adaptable to communication systems is given by Aggarwal et al. [2]. Thus, based on a SDP expression obtained with the assumption of perfect nodes, the node reliability values are taken into account in a specific mode for each term of the set of disjoint link-products. However, the authors do not completely address aspects of the influence of a node on the links connected to it, as it will be seen in Section 5. Moreover, the method is limited to the SDP expressions generated with a so-called "single variable inversion" (SVI) technique. But, for complex networks, "multiple variable inversion" (MVI) techniques are required [1, 4, 14, 16].

In this work, a new approximate method for two-terminal network reliability evaluation with a much better accuracy is proposed. The method is based on algorithms described in terms of minimal path set and covers both SVI and MVI expressions. Just like in [2], in the first stage, the method is focused only on the links of the network. For the any two given nodes, all the minimal paths are enumerated,

Table 1: Adjusted reliability values for the links in the network.

Reduced model	New reliability values
Model 1	$p'_a = p_a p_2, p'_b = p_b p_2, p'_c = p_c p_3, p'_d = p_d p_3, p'_e = p_e p_2 p_3$
Model 2	$p'_a = p_a p_2, p'_b = p_b, p'_c = p_c p_3, p'_d = p_d, p'_e = p_e p_2 p_3$
Model 3	$p'_a = p_a, p'_b = p_b p_2, p'_c = p_c, p'_d = p_d p_3, p'_e = p_e p_2 p_3$

Table 2: Numerical results$(R_{1\text{-}4})$.

Exact result	Approximate results obtained based on the reduced models		
	Model 1	Model 2	Model 3
0.9467	0.9466	0.9477	0.9476

and then this set of minimal paths is transformed into a set of disjoint products. In the second stage, each term of the sum of disjoint products including state variables associated to the links is processed distinctly by considering both links and adjacent node reliability values.

This new approximate method reduces the computation time for large networks to a great extent, compared with an exact method. This reduction in computation time is explained by the fact that the node failures are taken into account only in the second stage when the computation process is simpler, belonging to the $O(n \times m)$ class of complexity, where n is the number of disjoint link-products and m is the number of the network components.

The rest of this paper is organized as follows. Section 2 introduces notations, assumptions, and a short nomenclature, while Section 3 presents general issues regarding the problem of network reliability evaluation. Section 4 provides a method for exact evaluation of two-terminal network reliability when both node and link failures are considered. Section 5, the most extensive one, presents a new approximate method that reduces the complexity of this problem in medium-to-large networks. Section 6 presents some obtained numerical results. The paper ends with some final remarks presented in Section 7.

2. Notations and Preliminary Considerations

2.1. Nomenclature

(a) *Reliability*: the two-terminal reliability of a stochastic network expresses the probability that there exists at least one path between any two specified nodes (let us say a source node and a target one) which operate successfully

(b) *Connected nodes*: two nodes which can communicate with each other are connected; otherwise, they are disconnected

(c) *Minimal path*: a minimal set of links and their adjacent nodes whose good operation ensures that two given nodes are connected. For a minimal path, any proper subset is no longer a path

(d) *Uniproduct*: Boolean product composed only of distinct uncomplemented variables

(e) *Subproduct*: part of a Boolean product that is a complemented or an uncomplemented uniproduct

(f) *Mixproduct*: product of one uncomplemented subproduct and one or more complemented subproducts

(g) *Disjoint products*: a set of products expressing mutually exclusive states

2.2. Notations

(a) $G(V, E)$ is a network model with node set $V = \{y_1, y_2, \dots, y_k\}$ and link set $E = \{x_1, x_2, \dots, x_m\}$

(b) $s, t \in V, s \neq t$, are the source and target nodes

(c) p_x is the reliability of node $x \in V$ or link $x \in E$, and $q_x = 1 - p_x$

(d) R_{s-t} is the two-terminal reliability of network $G(V, E)$ with s and t the source and target nodes ($s - t$ network reliability)

(e) $P(A)$ denotes the probability of the event A

2.3. Assumptions

(a) Each component in the network (i.e., node or link) is either operational or failed, so a logical variable is used to indicate its state. The same notations $y_1, y_2, \dots, y_k$ and $x_1, x_2, \dots, x_m$ are used to denote these logical variables

(b) The events of failure that affect the nodes or the links in network are stochastically independent

3. Considerations on Network Reliability Evaluation

Consider $G(V, E)$ the network under study and $s, t \in V, s \neq t$, the source and target nodes. For this model, consider the minimal path set MPS $= \{P_1, P_2, \dots, P_{np}\}$. Note that a minimal path $P_i \in$ MPS is expressed by a product of distinct logical variables associated with some links or nodes of the network, and the reliability of this path is given by

$$P(P_i) = \prod_{c \in P_i} p_c. \tag{1}$$

Starting from this minimal path set, a structure function $S = \bigcup_{i=1}^{np} P_i$ is defined, and the two-terminal network reliability of this model is calculated by

$$R_{s-t} = P(S) = P\left(\bigcup_{i=1}^{np} P_i\right). \tag{2}$$

Efficient methods for enumerating all minimal paths are presented in [14, 17, 18]. To compute the network reliability R_{s-t} based on (2), the well-known rule of sum of disjoint products is recommended:

$$P\left(\bigcup_{i=1}^{n} A_i\right) = P(A_1) + P\left(\bar{A}_1 \cap A_2\right) + P\left(\bar{A}_1 \cap \bar{A}_2 \cap A_3\right) + \cdots + P\left(\bar{A}_1 \cap \bar{A}_2 \cap \cdots \cap \bar{A}_{n-1} \cap A_n\right). \tag{3}$$

For this purpose, the structure function S is transformed into an equivalent form S', composed only of disjoint products (DP), so that the two-terminal network reliability R_{s-t} is given by

$$R_{s-t} = P\left(S'\right) = \bigcup_j \mathrm{DP}_j = \sum_j P\left(\mathrm{DP}_j\right). \tag{4}$$

Observe that (4) is easy to compute, so that the problem of computing the two-terminal network reliability essentially boils down to generating a new set of disjoint products starting from the set MPS of minimal paths. Unfortunately, this task falls in the NP-hard category.

The first computerized SDP algorithm was proposed by Aggarwal et al. [3], but one of the best known SDP algorithms for transforming the structure function to a sum of disjoint products is given by Abraham [4].

If P and Q are two undisjoint products, and $x_1, x_2, \dots, x_s \in P \setminus Q$, according to Abraham's theorem, the following logical expression can be written as follows:

$$P + Q = P + \bar{x}_1 Q + x_1 \bar{x}_2 Q + x_1 x_2 \bar{x}_3 Q + \cdots + x_1 x_2 \cdots x_{s-1} \bar{x}_s Q. \tag{5}$$

Note that, to ensure that two products are disjoint, only a single complemented variable is added with each new term. Abraham's algorithm is a reference for the so-called SVI algorithms. Two improved SVI algorithms are presented in [19, 20].

To reduce the computation time, other approaches based on the so-called MVI technique have been devised (see, for example, [5, 6, 21–23]).

When an MVI technique is applied, a product may contain distinct logical variables (complemented or not) but also one or more complemented subproducts. For instance, take seven variables representing a network state where links 2 and 4 are not both operational, link 6 is operational, link 7 is in the failed state, and links 1, 3, and 5 are in a do-not-care state. In an MVI approach, this network state is represented by the Boolean expression $\overline{x_2 x_4} x_6 \overline{x_7}$, whereas in an SVI approach, by the expression $\overline{x_2}\, x_6\, \overline{x_7} + x_2\, \overline{x_4}\, x_6\, \overline{x_7}$, so that the advantage of the MVI approach is obvious.

An excellent survey on MVI techniques can be found in [16]. A new MVI technique, called NMVI, is proposed by Cașcaval and Floria in [1].

According to the NMVI method, in order to expand a product Q in relation to a given uniproduct P, so that any new generated product to be disjoint with P, the following two MVI rules are applied.

Rule 1. Type I expansion
If $x_1, x_2, \ldots, x_s \in P \setminus Q$, the following equation can be written as follows:

$$P + Q = P + x_1 x_2 \cdots x_s Q + \overline{x_1 x_2 \cdots x_s} Q. \tag{6}$$

When P and Q are both uniproducts, for the new term $x_1 x_2 \cdots x_s Q$, the absorption law is applicable, so that a reduced logical expression with two disjoint products is obtained:

$$P + Q = P + \overline{x_1 x_2 \cdots x_s} Q. \tag{7}$$

Rule 2. Type II expansion
Consider $P = x_1 x_2 \cdots x_i R_1$, and $Q = \overline{x_1 x_2 \cdots x_i x_{i+1} \cdots x_s} R_2$. By applying the Boolean rule $\overline{xy} = \bar{x} + x\bar{y}$, the following logical expression results are as follows:

$$\begin{aligned} P + Q &= P + \overline{x_1 x_2 \cdots x_i x_{i+1} \cdots x_s} R_2 \\ &= P + \overline{x_1 x_2 \cdots x_i} R_2 + x_1 x_2 \cdots x_i \overline{x_{i+1} \cdots x_s} R_2. \end{aligned} \tag{8}$$

When $R_1 \in R_2$, the term $x_1 x_2 \ldots x_i \overline{x_{i+1} \cdots x_s} R_2$ is absorbed by product P, so that a reduced logical expression composed of two disjoint products is obtained:

$$P + Q = P + \overline{x_1 x_2 \cdots x_i} R_2. \tag{9}$$

As shown in [1], NMVI is an efficient method, providing fewer disjoint products compared with other well-known MVI methods, as CAREL [5], VT [6], or KDH88 [21].

In the next two sections, we address the problem of two-terminal network reliability evaluation, in which both link and node failures are considered. First, an exact method of reliability evaluation is discussed. Then, a new approximate method is presented, with the advantage of being much faster and able to offer a very good accuracy.

4. Exact Evaluation of Network Reliability

For two given nodes, s and t, an exact evaluation of two-terminal network reliability can be obtained based on the set of minimal paths that include both links and adjacent nodes. Compared with the case in which the study is limited to the links of the network, when the nodes are also considered, the number of the minimal paths is unchanged, but any term is extended by also including the adjacent nodes. To illustrate this method, let us analyze the network N_1 presented in Figure 2(a), where the source and target nodes are 1 and 5. These two terminal nodes are considered in series with the rest of the network. For these two given nodes, the set of minimal paths is

$$\begin{aligned} \text{MPS} = \{&3bf, 34beg, 24adg, 23acf, \\ &234aceg, 234adef, 234bcdg\}. \end{aligned} \tag{10}$$

By applying the NMVI method, the following set of disjoint products results as follows:

$$\begin{aligned} \text{DPS} = \Big\{&3bf, 34beg\bar{f}, 24adfg\overline{3b}, 24adgf\overline{3be}, 23acf\bar{b}\overline{4dg}, \\ &234aceg\bar{b}\bar{d}\bar{f}, 234adef\bar{b}\bar{c}\bar{g}, 234bcdg\bar{a}\bar{e}\bar{f}\Big\}. \end{aligned} \tag{11}$$

Finally, the reliability R_{1-5} is given by

$$\begin{aligned} R_{1-5} = p_1 p_5 \Big(& p_3 p_b p_f + p_3 p_4 p_b p_e p_g (1 - p_f) \\ & + p_2 p_4 p_a p_d p_g \Big(p_f (1 - p_3 p_b) \\ & + (1 - p_f)(1 - p_3 p_b p_e) \Big) \\ & + p_2 p_3 p_a p_c p_f (1 - p_b)(1 - p_4 p_d p_g) \\ & + p_2 p_3 p_4 \Big(p_a p_c p_e p_g (1 - p_b)(1 - p_d)(1 - p_f) \\ & + p_a p_d p_e p_f (1 - p_b)(1 - p_c)(1 - p_g) \\ & + p_b p_c p_d p_g (1 - p_a)(1 - p_e)(1 - p_f) \Big) \Big). \end{aligned} \tag{12}$$

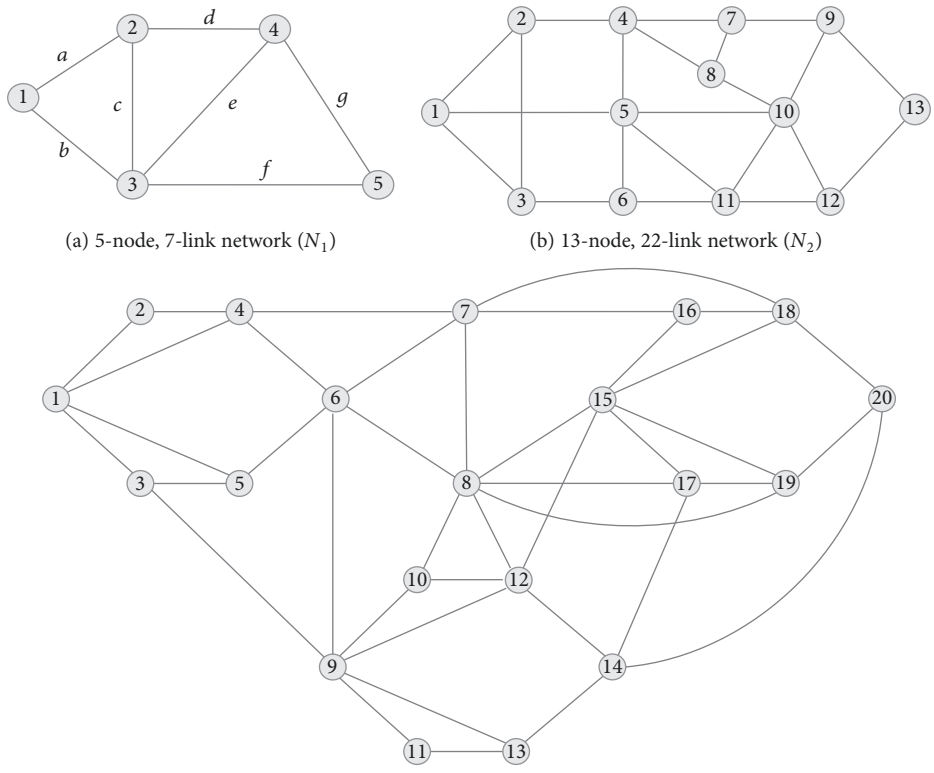
The same network is analyzed in [2], example 2. So, we compared the numerical result obtained based on this equation with the result generated with equation (21) presented in [2]. These results are identical. For example, assuming for all the nodes a reliability of 0.98 and for all the links a reliability of 0.95, both methods give a reliability value $R_{1-5} = 0.969611$.

To cover both nodes and links, much more logical variables are used. The problem that arises in this case is that the number of disjoint products increases to a large extent when complex networks are evaluated. To highlight this aspect, comparative results with respect to the network models N_2 and N_3 given in Figure 2 are presented in Table 3.

Compared with the case in which the study is limited to the links of the network, when the adjacent nodes are also considered, the number of disjoint products increases significantly. The relative growth with respect to the number of disjoint products is about 39% for network N_2, but for the more complex network N_3, this relative growth reaches 86%.

5. Approximate Approach for Network Reliability Evaluation

The process of generating the set of disjoint products is a difficult one, of NP-hard complexity. In order to reduce the computation time, the process of enumerating the minimal paths and their development as a sum of disjoint products is focused only to the links of the network. For this purpose, for a link $x_i \in E$, let X_i be a logical variable that reflects the event of successful communication through that branch—that means that the link x_i and the two adjacent nodes are operational. Thus, the structure function S can be expressed in terms of these logical variables $X_1, X_2, \ldots, X_m$.



(a) 5-node, 7-link network (N_1)

(b) 13-node, 22-link network (N_2)

(c) 20-node, 40-link network (N_3)

FIGURE 2: Network models with unreliable nodes: (a) 5-node, 7-link network (N_1); (b) 13-node, 22-link network (N_2); (c) 20-node, 40-link network (N_3).

TABLE 3: The number of disjoint products. Comparative results.

Network models	Number of minimal paths	Number of disjoint products generated by NMVI, including	
		Only links	Both links and nodes
N_2	281	2269	3151
N_3	16618	1799888	3353457

In the second stage, each term of the sum of disjoint products is processed distinctly by considering both links and adjacent node reliability values. The node reliability values are taken into account in a specific mode for each term of the set of disjoint link-products, when only the adjacent nodes of the links that compose the current product are considered. This is the starting point for this approximate approach.

Based on the set of disjoint products DPS = $\{DP_1, DP_2, \ldots, DP_n\}$, the two-terminal network reliability is computed by applying (4).

A term DP in the set of disjoint products is a mixproduct that includes one uniproduct, noted with U, and one or more complemented subproducts. Figure 3 shows such a complex mixproduct.

As illustrated in Figure 3, the uniproduct U reflects a state of operability of a part of the network that ensures the connection between the source and target nodes. Let SAN be the set of all adjacent nodes of the links that compose the uniproduct U. All these links and all the nodes that belong to SAN are operational. Consequently, the probability of the network state described by U is given by

$$P(U) = \prod_{x \in U} p_x \prod_{y \in \text{SAN}} p_y. \tag{13}$$

The main problem is how to compute or at least evaluate with a good accuracy the probability of a network state described by a complemented subproduct (such a subproduct is illustrated in Figure 3 with a dashed line).

A complemented subproduct reflects a state of inoperability of a branch or of a bigger portion of the network. To begin with, consider the case where such a portion of the network is independent of that portions described by the other complemented subproducts. Under these circumstances, the current term can be independently evaluated. Two cases are distinguished.

Case 1 (a single complemented variable (an SVI term)). Consider a single complemented variable $\overline{X_i}$ (an SVI term) associated with the link x_i that connects two nodes; let us say y_i and y_j (for instance, the variable $\overline{X_{11}}$ in Figure 3). The probability of this event is

$$P\left(\overline{X_i}\right) = 1 - p'_{x_i}, \tag{14}$$

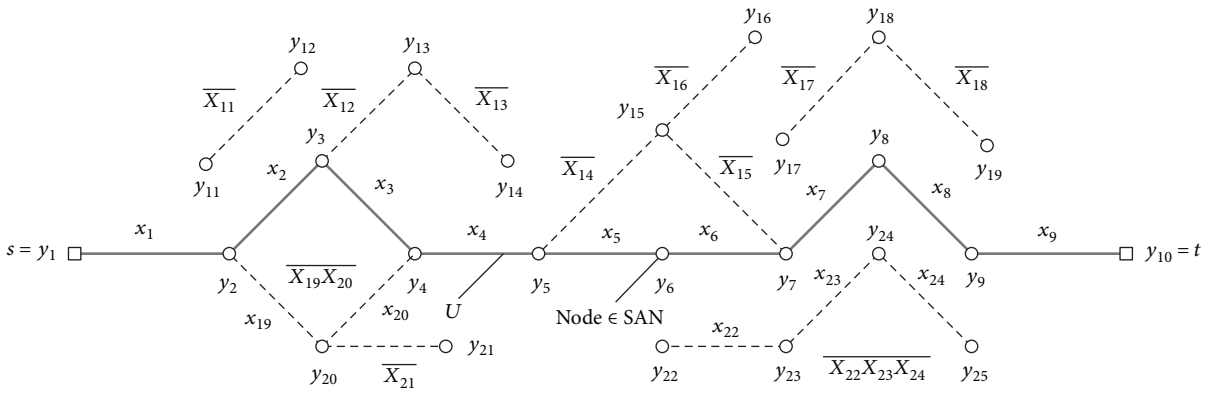


Figure 3: Illustration of a complex mixproduct: $\mathrm{DP} = X_1X_2 \cdots X_9\overline{X_{11}}\,\overline{X_{12}} \cdots \overline{X_{18}}\,\overline{X_{19}X_{20}}\,\overline{X_{21}}\,\overline{X_{22}X_{23}X_{24}}$.

where

$$p'_{x_i} = \begin{cases} p_{x_i}p_{y_i}, & \text{if } y_j \in \text{SAN}, y_i \notin \text{SAN}, \\ p_{x_i}p_{y_j}, & \text{if } y_i \in \text{SAN}, y_j \notin \text{SAN}, \\ p_{x_i}p_{y_i}p_{y_j}, & \text{if } y_i, y_j \notin \text{SAN}. \end{cases} \tag{15}$$

Equations (13) and (14) are found in another form in [2] where the same problem of network reliability evaluation is treated. Remember that the method presented in [2] covers only SDP expressions composed of SVI terms.

Case 2 (an MVI term). Consider a complemented subproduct $\overline{X_1X_2 \cdots X_k}$ (an MVI term) that describes a state of inoperability of a portion of the network as illustrated in Figure 4. The probability that this portion of the network to be inoperable is

$$P\left(\overline{X_1X_2 \cdots X_k}\right) = 1 - Q, \tag{16}$$

where the product Q includes not only the reliability of the corresponding links but also the reliability of the adjacent nodes that do not belong to SAN, considered only once. More exactly, the probability Q is computed by the following sequence of steps presented in Pseudocode 1.

Even though the two portions of the network described by two complemented subproducts may not have any common link, they may have one or even more common nodes. Consequently, the state of inoperability of these two portions of the network may be due to the failure of such a common node. Of course, we refer to a common node that does not belong to SAN.

In the first stage, the analysis of these dependencies given by the common nodes is limited to the level of pairs of complemented subproducts. The following three cases are distinguished.

Case 3 (two SVI terms with a common node). Consider two complemented variables $\overline{X_i}$ and $\overline{X_j}$ that describe a state of inoperability for two branches x_i and x_j that have

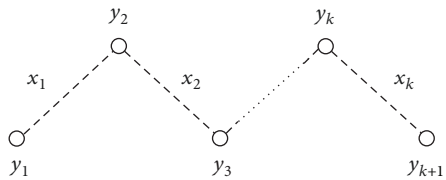


Figure 4: Illustration of a MVI term: $\overline{X_1X_2 \cdots X_k}$.

```
Q = 1;
for i = 1 : k
    Q = Q* p_{x_i};
    if y_i ∉ SAN then Q = Q* p_{y_i};
end
if y_{k+1} ∉ SAN then Q = Q* p_{y_{k+1}};
```

Pseudocode 1: Computing the product Q.

y_k as a common node, as illustrated in Figure 5. The node $y_k \notin$ SAN.

Let us define the probabilities p'_{x_i} and p'_{x_j} associated with the links x_i and x_j, given as follows:

$$p'_{x_i} = \begin{cases} p_{x_i}, & \text{if } y_i \in \text{SAN}, \\ p_{x_i}p_{y_i}, & \text{if } y_i \notin \text{SAN}, \end{cases}$$
$$p'_{x_j} = \begin{cases} p_{x_j}, & \text{if } y_j \in \text{SAN}, \\ p_{x_j}p_{y_j}, & \text{if } y_j \notin \text{SAN}. \end{cases} \tag{17}$$

By applying the theorem of total probability to the event space $\{y_k, \bar{y}_k\}$, the following equation can be written as follows:

$$P\left(\overline{X_i}\,\overline{X_j}\right) = P(X) = p_{y_k}P(X \mid y_k) + \left(1 - p_{y_k}\right)P(X \mid \bar{y}_k). \tag{18}$$

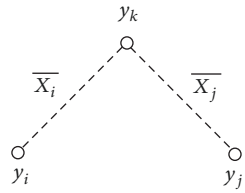


Figure 5: Illustration of two inoperable links with a common node.

As

$$\begin{aligned} P(X \mid \bar{y}_k) &= 1, \\ P(X \mid y_k) &= \left(1 - p'_{x_i}\right)\left(1 - p'_{x_j}\right), \end{aligned} \tag{19}$$

finally, the equation becomes

$$P\left(\overline{X_i}\,\overline{X_j}\right) = 1 - p_{y_k}\left(p'_{x_i} + p'_{x_j} - p'_{x_i}p'_{x_j}\right). \tag{20}$$

Note that this case is not treated in [2].

Case 4 (an MVI term and an SVI one with a common node). Consider, for example, $\overline{X_1X_2X_3}$ and $\overline{X_4}$ to be two terms in DP describing a state of inoperability of two portions of the network as illustrated in Figure 6. The common node $y_2 \notin$ SAN.

Let us define the probabilities p'_{x_1}, p'_{x_2}, p'_{x_3}, and p'_{x_4} given as follows:

$$\begin{aligned} p'_{x_1} &= \begin{cases} p_{x_1}, & \text{if } y_1 \in \text{SAN}, \\ p_{x_1}p_{y_1}, & \text{if } y_1 \notin \text{SAN}, \end{cases} \\ p'_{x_2} &= \begin{cases} p_{x_2}, & \text{if } y_3 \in \text{SAN}, \\ p_{x_2}p_{y_3}, & \text{if } y_3 \notin \text{SAN}, \end{cases} \\ p'_{x_3} &= \begin{cases} p_{x_3}, & \text{if } y_4 \in \text{SAN}, \\ p_{x_3}p_{y_4}, & \text{if } y_4 \notin \text{SAN}, \end{cases} \\ p'_{x_4} &= \begin{cases} p_{x_4}, & \text{if } y_5 \in \text{SAN}, \\ p_{x_4}p_{y_5}, & \text{if } y_5 \notin \text{SAN}. \end{cases} \end{aligned} \tag{21}$$

By applying the theorem of total probability to the event space $\{y_2, \bar{y}_2\}$, the following equation can be written as follows:

$$P\left(\overline{X_1X_2X_3}\,\overline{X_4}\right) = P(X) = p_{y_2}P(X \mid y_2) + \left(1 - p_{y_2}\right)P(X \mid \bar{y}_2). \tag{22}$$

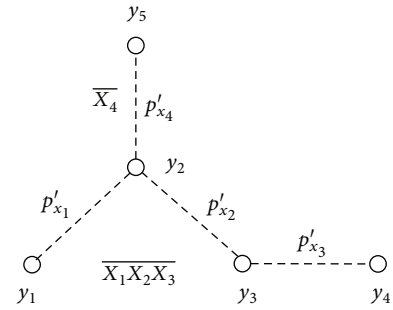


Figure 6: Illustration of an MVI term and an SVI one with a common node.

As

$$\begin{aligned} P(X \mid \bar{y}_2) &= 1, \\ P(X \mid y_2) &= \left(1 - p'_{x_1}p'_{x_2}p'_{x_3}\right)\left(1 - p'_{x_4}\right), \end{aligned} \tag{23}$$

finally, the following equation results in

$$P\left(\overline{X_1X_2X_3}\,\overline{X_4}\right) = 1 - p_{y_2}\left(p'_{x_1}p'_{x_2}p'_{x_3} + p'_{x_4} - p'_{x_1}p'_{x_2}p'_{x_3}p'_{x_4}\right). \tag{24}$$

Note that, if the two events were treated independently, the following equation would result in

$$\begin{aligned} P\left(\overline{X_1X_2X_3}\right)P\left(\overline{X_4}\right) &= \left(1 - p'_{x_1}p'_{x_2}p'_{x_3}p_{y_2}\right)\left(1 - p'_{x_4}p_{y_2}\right) \\ &= 1 - p_{y_2}\left(p'_{x_1}p'_{x_2}p'_{x_3} + p'_{x_4} - p'_{x_1}p'_{x_2}p'_{x_3}p'_{x_4}p_{y_2}\right) \\ &< P\left(\overline{X_1X_2X_3}\,\overline{X_4}\right). \end{aligned} \tag{25}$$

Remark 1. Equation (25) shows that when a common node is not taken into account, the reliability estimation is a pessimistic one.

Case 5 (two MVI terms with a common node). Consider, for example, $\overline{X_1X_2}$ and $\overline{X_3X_4}$ to be the two MVI terms describing a state of inoperability of two parts of the network, as illustrated in Figure 7, where the communication between nodes 1 and 2 and between nodes 3 and 4 is not possible. The common node $y_5 \notin$ SAN.

Let us define the probabilities p'_{x_1}, p'_{x_2}, p'_{x_3}, and p'_{x_4} given by the following:

$$p'_{x_i} = \begin{cases} p_{x_i}, & \text{if } y_i \in \text{SAN}, \\ p_{x_i}p_{y_i}, & \text{if } y_i \notin \text{SAN}, \\ i = 1, 2, 3, 4. \end{cases} \tag{26}$$

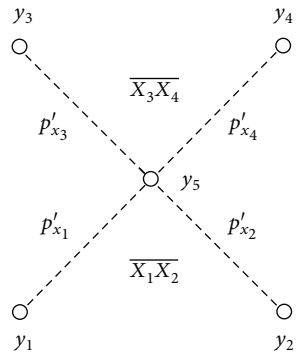


Figure 7: Illustration of two MVI terms with a common node.

The probability of this state, $P(\overline{X_1X_2}\,\overline{X_3X_4})$, can be determined by applying the rule of total probability to the event space $\{y_5, \bar{y}_5\}$.

If $X = \overline{X_1X_2}\,\overline{X_3X_4}$, the following equation can be written as follows:

$$P(X) = p_{y_5}P(X \mid y_5) + \left(1 - p_{y_5}\right)P(X \mid \bar{y}_5). \tag{27}$$

Obviously, $P(X/\bar{y}_5) = 1$.

$$\begin{aligned} P(X \mid y_5) &= \left(1 - p'_{x_1}p'_{x_2}\right)\left(1 - p'_{x_3}p'_{x_4}\right) \\ &= 1 - p'_{x_1}p'_{x_2} - p'_{x_3}p'_{x_4} + p'_{x_1}p'_{x_2}p'_{x_3}p'_{x_4}. \end{aligned} \tag{28}$$

Finally, the equation becomes

$$P\left(\overline{X_1X_2}\,\overline{X_3X_4}\right) = 1 - p_{y_5}\left(p'_{x_1}p'_{x_2} + p'_{x_3}p'_{x_4} - p'_{x_1}p'_{x_2}p'_{x_3}p'_{x_4}\right). \tag{29}$$

To exemplify these 5 rules previously defined, consider the mixproduct

$$\mathrm{DP} = X_1X_2 \cdots X_9\overline{X_{11}}\,\overline{X_{12}} \cdots \overline{X_{18}}\,\overline{X_{19}X_{20}}\,\overline{X_{21}}\,\overline{X_{22}X_{23}X_{24}}, \tag{30}$$

as illustrated in Figure 3. The mixproduct DP includes the uniproduct $U = X_1X_2 \cdots X_9$ and for this operable path, the set of adjacent nodes is $\mathrm{SAN} = \{y_1, y_2, \ldots, y_{10}\}$.

Taking into account the common nodes for the SVI and MVI terms, the probability of the mixproduct DP can be evaluated with a good accuracy by the following:

$$\begin{aligned} P(\mathrm{DP}) = P(U)P\left(\overline{X_{11}}\right)P\left(\overline{X_{12}}\,\overline{X_{13}}\right)P\left(\overline{X_{14}}\,\overline{X_{15}}\right)P\left(\overline{X_{16}}\right) \\ \times P\left(\overline{X_{17}}\,\overline{X_{18}}\right)P\left(\overline{X_{19}X_{20}}\,\overline{X_{21}}\right)P\left(\overline{X_{22}X_{23}X_{24}}\right). \end{aligned} \tag{31}$$

By applying the rules presented before, the following equations result in

$$\begin{aligned} P(U) &= p_{x_1}p_{x_2} \cdots p_{x_9}p_{y_1}p_{y_2} \cdots p_{y_{10}}, \\ P\left(\overline{X_{11}}\right) &= 1 - p_{y_{11}}p_{x_{11}}p_{y_{12}}, \\ P\left(\overline{X_{12}}\,\overline{X_{13}}\right) &= 1 - p_{y_{13}}\left(p_{x_{12}} + p_{x_{13}}{}' - p_{x_{12}}p_{x_{13}}{}'\right), \end{aligned} \tag{32}$$

where $p'_{x_{13}} = p_{x_{13}}p_{y_{14}}$.

$$\begin{aligned} P\left(\overline{X_{14}}\,\overline{X_{15}}\right) &= 1 - p_{y_{15}}\left(p_{x_{14}} + p_{x_{15}} - p_{x_{14}}p_{x_{15}}\right), \\ P\left(\overline{X_{16}}\right) &= 1 - p_{y_{15}}p_{x_{16}}p_{y_{16}}\,(\text{an approximate evaluation}), \\ P\left(\overline{X_{17}}\,\overline{X_{18}}\right) &= 1 - p_{y_{18}}\left(p'_{x_{17}} + p'_{x_{18}} - p'_{x_{17}}p'_{x_{18}}\right), \end{aligned} \tag{33}$$

where $p'_{x_{17}} = p_{x_{17}}p_{y_{17}}$ and $p'_{x_{18}} = p_{x_{18}}p_{y_{19}}$.

$$P\left(\overline{X_{19}X_{20}}\,\overline{X_{21}}\right) = 1 - p_{y_{20}}\left(p_{x_{19}}p_{x_{20}} + p'_{x_{21}} - p_{x_{19}}p_{x_{20}}p'_{x_{21}}\right), \tag{34}$$

where $p'_{x_{21}} = p_{x_{21}}p_{y_{21}}$.

$$P\left(\overline{X_{22}X_{23}X_{24}}\right) = 1 - p_{y_{22}}p_{x_{22}}p_{y_{23}}p_{x_{23}}p_{y_{24}}p_{x_{24}}p_{y_{25}}. \tag{35}$$

Observe that, related to the probability of this mixproduct, an approximation is made with respect to the terms $\overline{X_{14}}$, $\overline{X_{15}}$, and $\overline{X_{16}}$, because the links x_{14}, x_{15}, and x_{16} have a common node, $y_{15} \notin \mathrm{SAN}$. This case is discussed in more detail below.

Case 6 (many terms with a common node). Consider three links x_1, x_2, and x_3 with a common node and the network state reflected by the SVI terms $\overline{X_1}$, $\overline{X_2}$, and $\overline{X_3}$, as illustrated in Figure 8.

Suppose that $y_4 \notin \mathrm{SAN}$. Let us first define the probabilities p'_{x_1}, p'_{x_2}, and p'_{x_3} by the following:

$$p'_{x_i} = \begin{cases} p_{x_i}, & \text{if } y_i \in \mathrm{SAN}, \\ p_{x_i}p_{y_i}, & \text{if } y_i \notin \mathrm{SAN}, \\ i = 1, 2, 3. \end{cases} \tag{36}$$

In order to evaluate the probability $P(\overline{X_1}\,\overline{X_2}\,\overline{X_3})$, the theorem of total probability is applied to the event space $\{y_4, \bar{y}_4\}$. The following equation results in

$$P\left(\overline{X_1}\,\overline{X_2}\,\overline{X_3}\right) = P(X) = p_{y_4}P(X \mid y_4) + \left(1 - p_{y_4}\right)P(X \mid \bar{y}_4). \tag{37}$$

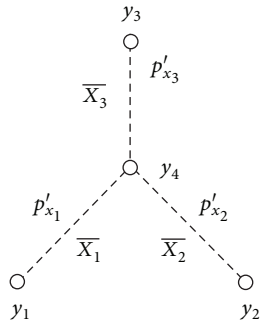


Figure 8: Illustration of three SVI terms with a common node.

Obviously, $P(X/\bar{y}_4) = 1$. When the node y_4 is operational, $\overline{X_1}$, $\overline{X_2}$, and $\overline{X_3}$ reflect independent events, so that

$$P\left(\overline{X_1}\,\overline{X_2}\,\overline{X_3}\right) = P\left(\overline{X_1}\right)P\left(\overline{X_2}\right)P\left(\overline{X_3}\right). \tag{38}$$

Consequently, we have

$$\begin{aligned} P(X \mid y_4) &= \left(1 - p'_{x_1}\right)\left(1 - p'_{x_2}\right)\left(1 - p'_{x_3}\right) \\ &= 1 - p'_{x_1} - p'_{x_2} - p'_{x_3} + p'_{x_1}p'_{x_2} + p'_{x_1}p'_{x_3} \\ &\quad + p'_{x_2}p'_{x_3} - p'_{x_1}p'_{x_2}p'_{x_3}. \end{aligned} \tag{39}$$

Finally, the equation becomes

$$\begin{aligned} P\left(\overline{X_1}\,\overline{X_2}\,\overline{X_3}\right) = 1 - p_{y_4}\Big(&p'_{x_1} + p'_{x_2} + p'_{x_3} - p'_{x_1}p'_{x_2} \\ &- p'_{x_1}p'_{x_3} - p'_{x_2}p'_{x_3} + p'_{x_1}p'_{x_2}p'_{x_3}\Big). \end{aligned} \tag{40}$$

Observe that, in case of an exact evaluation, the equation is composed of $2^3 = 8$ terms. This result can be generalized for the case with more events $\overline{X_1}, \overline{X_2}, \ldots, \overline{X_m}$ that depend on the state of a common node. The equation for computing the probability $P(\overline{X_1}\,\overline{X_2}\cdots\overline{X_m})$ comprises 2^m terms. To reduce the computation time, the method we propose is limited to the pairs of events depending on the state of a common node, for which the equation (20, 24, or 29 as is the case) comprises only four terms.

This is the approximation that may slightly affect the result given by the method we propose. Also, this work does not treat those cases when two MVI terms have two or more common nodes, considering that these cases are very rare. Nevertheless, this method provides a network reliability evaluation with a very good accuracy, as can be seen in the next section. In all the checks we have made, this method has generated exact values or slightly pessimistic results. So, the reliability value given by this method can be interpreted as a lower limit and it can be explained on the basis of (25) and Remark 1. This aspect is very important for a reliability study.

Before ending this section, an answer to the next question is required: why did this method focus only on minimal paths and not on minimal cuts at all? Indeed, it is well known that for networks with high reliability, the approaches based on minimal cuts are generally more appropriate for an approximate evaluation. However, in our case, an approach similar to that applied to the minimal paths is no longer appropriate, because when both node and link failures are considered, one must take into account a set of cuts consisting of nodes, a set of cuts consisting of links, and another one that comprises both nodes and links. So, when the nodes are also considered, the number of minimal cuts increases very much. For this reason, the proposed method is focused only on minimal paths.

6. Numerical Results

To illustrate the efficiency of this approximate method, we consider the network models N_2 and N_3 presented in Figure 2. For these networks, Table 4 presents comparative results, by assuming for all the nodes a reliability value of 0.98 and for all the links a reliability value of 0.95. Observe that the proposed method gives an accurate result for the network N_2, and it gives a slightly approximate value with five accurate decimal places for the larger network N_3.

Computing time for reliability evaluation is presented in Table 5. Compared to the exact method presented in Section 5, for network model N_3, the proposed approximate method greatly reduces the computational time, from 57 min 17 s to 18 min 43 s.

The values presented in Table 5 highlight the rapid growth of the computation time for the reliability evaluation with an increasing network size. The methods of network reliability evaluation based on SDP algorithms fall in the NP-hard category and, consequently, are difficult to apply for very large networks, such as social networks. In these cases, other techniques for approximate evaluation can also be applied, especially the Monte Carlo simulation (see, for example, [24–27]). Even so, the methods based on SDP algorithms are still necessary for the validation of simulation programs.

7. Final Remarks

In this work, the problem of two-terminal network reliability evaluation in which both link and node failures are considered is discussed. An approximate method that provides a very good accuracy is proposed. Compared to an exact method, this approximate method greatly reduces the computation time for complex networks.

The proposed solution can be applied with any SDP algorithm, but the accuracy of the reliability estimation depends on the method used for transforming the set of minimal paths into a set of disjoint products. When the number of disjoint products is lower, the reliability estimation is better. For this reason, efficient MVI algorithms as NMVI or the hybrid algorithm given by Chaturvedi and Misra [23] are recommended. Another approach based on binary decision diagrams (BDDs) is also recommended [28].

Table 4: Network reliability evaluation (R_{s-t}). Comparative results.

Network model	Source and target nodes	Exact method based on NMVI	The proposed method
N_2	$s = 1$, $t = 13$	0.982883	0.982883
N_3	$s = 1$, $t = 20$	0.959579	0.959575

Table 5: Computing time for reliability evaluation.

Network model	Exact method based on NMVI	The proposed method
N_2	0.1 s	0.06 s
N_3	57 min 17 s	18 min 43 s

Data Availability

The paper presents a method for network reliability evaluation for which mathematical proofs are included. It can be applied for any network, and therefore, it does not depend on specific data.

Disclosure

The same issue of two-terminal reliability evaluation in large networks is addressed by the paper "SDP Algorithm for Network Reliability Evaluation", authors P. Caşcaval and S. A. Floria, presented at the IEEE Conference INISTA 2017 [1]. In that paper, an efficient SDP method (called NMVI) for transforming algebraically a structure function (expressed in terms of minimal paths or cuts) into a sum of disjoint products is proposed. This new method is based on an MVI technique and provides better solutions, with fewer disjoint products, compared with other well-known MVI methods. The author asserts that some general issues, such as notations, nomenclature, or other general considerations on network reliability evaluation, are similar to those outlined in [1].

Conflicts of Interest

The author declares that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

The author thanks his colleague Sabina-Adriana Floria for the useful and fruitful discussions. Also, many thanks are due to Dr. Florin Leon and Dr. Marius Kloetzer for the helpful suggestions which helped improve the readability of this paper.

References

[1] P. Caşcaval and S. A. Floria, "SDP algorithm for network reliability evaluation," in *2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, Gdynia, Poland, July 2017.

[2] K. Aggarwal, J. Gupta, and K. Misra, "A simple method for reliability evaluation of a communication system," *IEEE Transactions on Communications*, vol. 23, no. 5, pp. 563–566, 1975.

[3] K. K. Aggarwal, K. B. Misra, and J. S. Gupta, "A fast algorithm for reliability evaluation," *IEEE Transactions on Reliability*, vol. R-24, no. 1, pp. 83–85, 1975.

[4] J. A. Abraham, "An improved algorithm for network reliability," *IEEE Transactions on Reliability*, vol. R-28, no. 1, pp. 58–61, 1979.

[5] S. Soh and S. Rai, "CAREL: computer aided reliability evaluator for distributed computing networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 2, pp. 199–213, 1991.

[6] M. Veeraraghavan and K. S. Trivedi, "An improved algorithm for symbolic reliability analysis," *IEEE Transactions on Reliability*, vol. 40, no. 3, pp. 347–358, 1991.

[7] J. S. Provan and M. O. Ball, "Computing network reliability in time polynomial in the number of cuts," *Operations Research*, vol. 32, no. 3, pp. 516–526, 1984.

[8] M. O. Ball and J. S. Provan, "Disjoint products and efficient computation of reliability," *Operations Research*, vol. 36, no. 5, pp. 703–715, 1988.

[9] K. S. Trivedi, *Probability and Statistics with Reliability*, Queueing and Computer Science Applications, John Wiley & Sons, New York, NY, USA, 2002.

[10] F. Beichelt and L. Spross, "An effective method for reliability analysis of complex systems," *Journal of Information Processing and Cybernetics*, vol. 23, pp. 227–235, 1987.

[11] P. Caşcaval and B. F. Romanescu, "Complementary approaches for the network reliability evaluation: network decomposition and Monte Carlo simulation," in *Bul. Inst. Polit. Iaşi, Tomul L (LIV), Fasc. 1–4*, pp. 123–131, Automatică şi Calculatoare, 2004.

[12] P. Caşcaval and A. R. Macovei, "Reliability evaluation by network decomposition," in *Bul. Inst. Polit. Iaşi, Tomul XLIX (LIII), Fasc. 1–4*, pp. 56–65, Automatică şi Calculatoare, 2003.

[13] P. Caşcaval and B. A. Botez, "Recursive algorithm for two-terminal network reliability evaluation," in *Bul. Inst. Polit. Iasi, LI (LV), Fasc. 1–4*, pp. 137–146, Automatică şi Calculatoare, 2005.

[14] K. B. Misra, *Reliability Analysis and Prediction: A Methodological Oriented Treatment*, Elsevier, Amsterdam, Oxford, New York, Tokyo, 1992.

[15] M. Shooman, *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design*, John Wiley & Sons, New York, NY, USA, 2002.

[16] S. K. Chaturvedi, *Network Reliability: Measures and Evaluation*, Scrivener Publishing-Wiley, Hoboken, NJ, USA, 2016.

[17] Y. Shen, "A new simple algorithm for enumerating all minimal paths and cuts of a graph," *Microelectronics Reliability*, vol. 35, no. 6, pp. 973–976, 1995.

[18] R. Mishra, M. A. Saifi, and S. K. Chaturvedi, "Enumeration of minimal cutsets for directed networks with comparative reliability study for paths or cuts," *Quality and Reliability Engineering International*, vol. 32, no. 2, pp. 555–565, 2016.

[19] F. Beichelt and L. Spross, "An improved Abraham-method for generating disjoint sums," *IEEE Transactions on Reliability*, vol. R-36, no. 1, pp. 70–74, 1987.

[20] M. O. Locks, "A minimizing algorithm for sum of disjoint products," *IEEE Transactions on Reliability*, vol. R-36, no. 4, pp. 445–453, 1987.

[21] K. D. Heidtmann, "Smaller sums of disjoint products by subproduct inversion," *IEEE Transactions on Reliability*, vol. 38, no. 3, pp. 305–311, 1989.

[22] T. Luo and K. S. Trivedi, "An improved algorithm for coherent-system reliability," *IEEE Transactions on Reliability*, vol. 47, no. 1, pp. 73–78, 1998.

[23] S. K. Chaturvedi and K. B. Misra, "A hybrid method to evaluate reliability of complex networks," *International Journal of Quality & Reliability Management*, vol. 19, no. 8/9, pp. 1098–1112, 2002.

[24] K. F. Tee, L. R. Khan, and H. Li, "Application of subset simulation in reliability estimation of underground pipelines," *Reliability Engineering & System Safety*, vol. 130, pp. 125–131, 2014.

[25] K. M. Zuev, S. Wu, and J. L. Beck, "General network reliability problem and its efficient solution by subset simulation," *Probabilistic Engineering Mechanics*, vol. 40, pp. 25–35, 2015.

[26] H.-S. Li, Y.-Z. Ma, and Z. Cao, "A generalized subset simulation approach for estimating small failure probabilities of multiple stochastic responses," *Computers & Structures*, vol. 153, pp. 239–251, 2015.

[27] A. Birolini, *Reliability Engineering*, Theory and practice, Springer-Verlag, Berlin Heidelberg, 2014.

[28] X. Zang, H.-R. Sun, K. S. Trivedi, and D. R. Avresky, Eds., "A BDD approach to dependability analysis of distributed computer systems with imperfect coverage," in *Dependable Network Computing*, pp. 167–190, Kluwer Academic Publishers, Amsterdam, Netherlands, 1999.

SDP Algorithm for Network Reliability Evaluation

Petru Caşcaval[1], Sabina-Adriana Floria[2]
[1,2]Department of Computer Science and Engineering
"Gheorghe Asachi" Technical University of Iaşi
Dimitrie Mangeron Street, 27, 700050, Iaşi, Romania
cascaval@cs.tuiasi.ro, sabina.floria@cs.tuiasi.ro

***Abstract*—This paper addresses the issue of the two-terminal reliability evaluation of medium-to-large networks and proposes a new method to solve the problem of 'sum of disjoint products' (SDP) algebraically. This method uses a 'multiple variables inversion' (MVI) technique for transforming a structure function into a sum of disjoint products which has a one-to-one correspondence with the reliability expression. The results of our method for several network models are compared with those obtained by means of other well-known MVI techniques. For large-scale networks, our method offers a better solution.**

Keywords— two-terminal network reliability; SDP algorithm; minimal paths; minimal cuts; multiple variable inversion

I. INTRODUCTION

The network reliability theory is applied extensively in many real-word systems, which can be modeled as stochastic networks, such as computer and communications systems, distributed systems, networks of sensors, social networks etc. The reliability evaluation approaches use a variety of tools for system modeling and computation of reliability or availability indices that, in a certain way, describe the ability of a network to carry out a desired operation. Most tools are based on algorithms described in terms of minimal path set or minimal cut set (see, for example, [1]-[5]). Unfortunately, the problem of computing the network reliability based on the set of minimal paths or cuts is NP-hard [4], [6]. For this reason, in case of complex networks, other techniques for approximate reliability evaluation must be applied, such as those based on network decomposition or Monte Carlo simulation (see, for example, [7]-[10]).

In this work, we deal with the problem of exact evaluation of two-terminal reliability or availability indices in medium-to-large networks, and we propose an efficient SDP algorithm able to compute the two-terminal network reliability based on the minimal path set or the minimal cut set.

In order to reduce the computation burden, as in most such works, we assume that each node of the network is perfectly reliable. Because the failure of a node inhibits the work of all arcs connected to it, starting from the given network with unreliable nodes, an equivalent reliability model with perfect nodes but with links that have greater failure probabilities can be obtained [11, 12]. For example, let us consider two adjacent nodes denoted by 1 and 2. Let p_1 and p_2 be the reliabilities of these nodes, and p_{12} be the reliability of the link between them. Fig. 1 presents two equivalent reliability models: one model with unreliable nodes, and another one with perfect nodes.

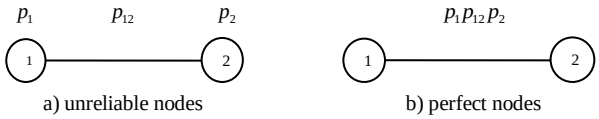


Fig. 1. Equivalent reliability models.

Moreover, to reduce the computation time for the minimal paths or cuts enumeration, once the source and destination nodes are known, the study can be simplified by artificially transforming the reliability graph into a directed one, as though only the communication from the source to the target node were of interest.

II. NOTATIONS AND PRELIMINARY CONSIDERATIONS

A. Nomenclature

a) *Reliability*. The two-terminal reliability of a stochastic network expresses the probability that there exists at least one path between any two specified nodes (let's say a source node and a target one), which operates successfully.

b) *Connected nodes*. Two nodes which can communicate with each other are connected, otherwise they are disconnected.

c) *Minimal path.* A minimal set of links (arcs) whose operation ensures that two given nodes are connected. For a minimal path, any proper subset is no longer a path.

d) *Minimal cut.* A set of links whose failure disconnects two given nodes. For a minimal cut, any proper subset is no longer a cut.

e) *Uniproduct.* A Boolean product composed only of distinct uncomplemented variables.

f) *Subproduct.* A part of a Boolean product which is a complemented or an uncomplemented uniproduct.

g) *Mixproduct.* A product of one uncomplemented subproduct and one or more complemented subproducts.

h) *Disjoint products.* A set of products expressing mutually exclusive states.

i) *Cube*. A vector of symbols representing a Boolean product.

B. Notations

a) $G(V, E)$ is a network model with node set $V = \{1, 2, \cdots, n\}$ and arc set $E = \{ x_1, x_2, \cdots, x_k \}$;

b) $s, t \in V, s \neq t$, are the source and target nodes;

c) p_x is the reliability of arc $x \in E$ and $q_x = 1 - p_x$;

d) R_{s-t} is the two-terminal reliability of network $G(V, E)$ with s and t the source and target nodes ($s-t$ network reliability).

C. Assumptions

a) Each node is perfect, as mentioned in section 1;

b) Each arc is either operational or failed, so a logical variable can be used to denote its state (the same notations $x_1, x_2, \cdots, x_k$ are used to denote these logical variables);

c) All failures that may affect the network under study are stochastically independent.

III. PROBLEM DESCRIPTION

Consider $G(V, E)$ the network under study and $s, t \in V, s \neq t$, the source and target nodes. For this model, consider the minimal path set $MPS = \{P_1, P_2, \ldots, P_m\}$. Note that, a minimal path $P_i \in MPS$ is expressed by a product of distinct logical variables associated with the arcs of this network, and the reliability of this path is given by the following equation:

$$Prob(P_i) = \prod_{c \in P_i} p_c \tag{1}$$

Starting from this minimal path set, a structure function $S = \bigcup_{i=1}^{m} P_i$ is defined, and the two-terminal network reliability of this model, R_{s-t}, is computed by applying the equation:

$$R_{s-t} = Prob(S) = Prob\left(\bigcup_{i=1}^{m} P_i \right). \tag{2}$$

When working with minimal cuts, let $MCS = \{C_1, C_2, \ldots, C_m\}$ be the cut set for the two-terminal network under study. The structure function in this case is $S = \bigcup_{i=1}^{m} C_i$, and the network reliability R_{s-t} is computed by applying the equation:

$$R_{s-t} = 1 - Prob(S) = 1 - Prob\left(\bigcup_{i=1}^{m} C_i \right). \tag{3}$$

Note that, for a minimal cut $C_i \in MCS$:

$$Prob(C_i) = \prod_{c \in P_i} q_c \ . \tag{4}$$

An efficient algorithm for enumerating all minimal paths and cuts of a graph is presented in [13]. Another method for the enumeration of minimal cuts in large network models and a comparative study for paths and cuts are presented in [14].

To compute the network reliability R_{s-t}, based on (2) or (3), the well-known rule of sum of disjoint products is recommended:

$$Prob\left(\bigcup_{i=1}^{m} A_i \right) = Prob(A_1) + Prob(\bar{A}_1 \cap A_2) + Prob(\bar{A}_1 \cap \bar{A}_2 \cap A_3) + \cdots + Prob(\bar{A}_1 \cap \bar{A}_2 \cap \cdots \cap \bar{A}_{m-1} \cap A_m) \ . \tag{5}$$

For this purpose, the structure function S must be transformed to an equivalent form S', composed only of disjoint products (DP), so that the network reliability R_{s-t} is computed by applying the equation:

$$R_{s-t} = Prob(S) = Prob\left(\bigcup_{i=1}^{m} P_i \right) =$$

$$= Prob(S') = Prob\left(\bigcup_{i=1}^{n} DP_i \right) = \sum_{i=1}^{n} Prob(DP_i). \tag{6}$$

Observe that (6) is much easier to compute than (2) or (3). In conclusion, the problem of computing the two-terminal network reliability essentially boils down to finding the disjoint products that define an equivalent structure function S'. Unfortunately, this task falls in the NP-hard category [4, 6, 12]. The space of solutions for this kind of functions is very large, so that the aim of an SDP algorithm is to identify the best possible solution for an equivalent function S'. That means that the number of disjoint products and the computing time for determining them should be as small as possible.

One of the best known SDP algorithms for expanding a structure function to another one, composed only of disjoint products, is given by Abraham [1]. Let us consider two undisjoint products, P and Q, and $X = \{x_1, x_2, \ldots, x_k\}$ be the set of logical variables included in P that do not belong to Q. Notice that P is an uniproduct whereas Q can also be a mixproduct. According to Abraham's theorem, the following logical expression can be written:

$$P + Q = P + \bar{x}_1 Q + x_1 \bar{x}_2 Q + x_1 x_2 \bar{x}_3 Q + \cdots + x_1 x_2 \cdots x_{k-1} \bar{x}_k Q \ . \tag{7}$$

To ensure that two products are disjoint, only a single complemented variable is added with each new term. Abraham's algorithm is a reference for the so-called 'single variable inversion' (SVI) algorithms. Remember that a product comprises a set of distinct variables, complemented or not. Two SVI products are disjoint if both contain at least one variable which reflects, in the two cases, complementary logical states (for example, x_1 and $\bar{x}_1$). Improved SVI methods are presented in [15] and [16].

To reduce the computation burden, other approaches based on the so-called MVI technique have been devised (see, for example, [2], [3], [17], and [18]). An excellent survey on MVI methods can be found in [12]. A new SDP algorithm based on an MVI technique is presented in the following section.

IV. A NEW MVI METHOD (NMVI)

A. Preliminaries

The method we propose uses the following laws from Boolean algebra:

- $x + \bar{x} = 1$ (Complementation law);
- $x + xy = x$ (Absorption law);
- $x + \bar{x}y = x + y$ (Idempotent law);
- $\overline{xy} = \bar{x} + x\bar{y} = \bar{x} + \bar{y}$ (De Morgan's law).

When an MVI technique is applied, a product may contain distinct logical variables (complemented or uncomplemented) but also one or more complemented groups of logical variables (complemented subproducts). For instance, a Boolean expression of six variables representing link 1 or 4 in the failed state, link 5 operational, link 6 in the failed state and links 2 and 3 in a don't care state is represented by $\overline{x_1 x_4} x_5 \overline{x_6}$. Note that in an SVI approach, the same network state is represented by the Boolean expression $x_1 \overline{x_4} x_5 \overline{x_6} + \overline{x_1} x_4 x_5 \overline{x_6}$. The advantage of an MVI approach is obvious.

To describe a product, we use a vector of length k (also called a cube) with the following meanings:

- an uncomplemented variable is indicated in cube by the symbol '1';
- a complemented variable is indicated by '-1';
- the absence of a variable is indicated by the symbol '–';
- a complemented subproduct is indicated by using a negative number smaller or equal to '-2', as illustrated in TABLE I.

To illustrate this new method, let us consider two undisjoint products, P and Q, for which the absorption law is not applicable (i.e., $P \not\subset Q$). Note that, to verify that $P \not\subset Q$ it is necessary to check that at least one variable from P (indicated in cube by '1') does not belong to Q in the same form (indicated in cube by any symbol $\neq$ '1'). Also, P and Q are two undisjoint products if there is no complemented subproduct in Q (indicated in cube by '-1', '-2', '-3', etc.) that also belongs to P in an uncomplemented form (indicated in cube by '1' in the same positions). Examples of two undisjoint products are presented in TABLE II. As illustrated in this table, three cases must be taken into account.

In order to expand a product Q in relation to a given uniproduct P, so that any new generated product to be disjoint with P, we propose the following two rules.

Rule 1. Let $X = \{x_1, x_2, \ldots, x_k\}$ be the set of logical variables included in P that do not belong to Q.

TABLE I. PRODUCT DESCRIPTION

Product	**Cube**										**Comment**
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	
$x_3 x_5 x_8$	–	–	1	–	1	–	–	1	–	–	Uniproduct
$\bar{x}_1 x_3 \bar{x}_5 x_7 x_9$	-1	–	1	–	-1	–	1	–	1	–	SVI product
$\overline{x_1 x_2 x_4 x_5} x_6 \overline{x_7 x_{10}}$	-1	-2	-	-2	-2	1	-3	–	–	-3	MVI product

TABLE II. EXAMPLES OF UNDISJOINT PRODUCTS

Two undisjoint products (P and Q)	**Cube**										**Comment**
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	
$P = x_2 x_5 x_8$	–	1	–	–	1	–	–	1	–	–	Case 1
$Q = x_3 x_4 x_5 x_7$	–	–	1	1	1	–	1	–	–	–	
$P = x_1 x_9$	–	1	–	–	–	–	–	–	1	–	Case 2
$Q = \overline{x_2 x_5}\ \overline{x_8 x_9 x_{10}}$	–	-2	–	–	-2	–	–	-3	-3	1	
$P = x_1 x_3 x_7 x_8$	1	–	1	–	–	–	1	1	–	–	Case 3
$Q = \overline{x_2 x_3 x_5}\ \overline{x_6 x_7 x_8 x_{10}}$	–	1	-2	–	-2	-3	-3	-3	–	1	

The following logical expression can be written based on the complementation law:

$$P+Q=P+Q(x_1x_2\cdots x_k+\overline{x_1x_2\cdots x_k})$$

$$=P+x_1x_2\cdots x_kQ+\overline{x_1x_2\cdots x_k}Q. \quad (8)$$

As follows, we will refer to the rule presented in (8) as the "type I expansion". When *P* and *Q* are both uniproducts (case 1 in TABLE II), the new product $x_1x_2\cdots x_kQ$ includes all the variables from *P* so that the absorption law is applicable. A reduced logical expression composed of two disjoint products is obtained:

$$P+Q=P+\overline{x_1x_2\cdots x_k}Q. \quad (9)$$

Otherwise, the new product $x_1x_2\cdots x_kQ$ must be expanded again until all the new generated products are disjoint with *P* and also between them.

Rule 2. Consider again two products, *P* and *Q*, in which $P=x_1x_2\cdots x_kR_1$, and $Q=\overline{x_1x_2\cdots x_kx_{k+1}\cdots x_m}R_2$. By applying the Boolean rule $\overline{xy}=\bar{x}+x\bar{y}$, the following logical expression can be written:

$$P+Q=P+\overline{x_1x_2\cdots x_kx_{k+1}\cdots x_m}R_2$$

$$=P+(\overline{x_1x_2\cdots x_k}+x_1x_2\cdots x_k\overline{x_{k+1}\cdots x_m})R_2$$

$$=P+\overline{x_1x_2\cdots x_k}R_2+x_1x_2\cdots x_k\overline{x_{k+1}\cdots x_m}R_2 \quad (10)$$

We will refer to the rule presented in (10) as the "type II expansion". Observe that, when $R_1\in R_2$ the product $x_1x_2\cdots x_k\overline{x_{k+1}\cdots x_m}R_2$ is absorbed by product *P*, so that a reduced logical expression composed of two disjoint products is obtained:

$$P+Q=P+\overline{x_1x_2\cdots x_k}R_2 \quad (11)$$

When $R_1\notin R_2$, the new mixproduct $x_1x_2\cdots x_k\overline{x_{k+1}\cdots x_m}R_2$ must be expanded again until all the new generated products are disjoint with *P* and also between them.

Remark 1. Based on (9) and (11), during the process of generating equivalent disjoint products, the absorption law is applicable in fewer cases. This is the main idea we have had in view for accelerating this time-consuming process.

Remark 2. When both type I and type II expansion rules are applicable (case 3 in TABLE II), we propose that the type II expansion rule be applied prior to the type I expansion.

The method we propose implies the following steps:

- The initial products (uniproducts) are sorted by length in ascending order;
- The first term is left unchanged;
- The second term is expanded with respect to the variables that appear only in the first one, by applying one or both rules previously defined (type I or type II expansion), as applicable.
- The operation is repeated for all other initial terms so that all the newly generated products are disjoint from each other and are also disjoint with all the terms already obtained in the previous steps.

For an efficient implementation of this method, we use three arrays, namely:

- the first one with the initial terms (uniproducts);
- a working array to store and process the terms generated by expanding a certain uniproduct;
- a collector array to store the final terms (disjoint products).

This method is illustrated by the following example. Take a set of uniproducts $MPS=\{adk, bfl, cek, adgl, cegl\}$. The process of expanding these terms by applying this method until all the products are disjoint from each other is presented in Fig. 2. Finally, the set of disjoint products obtained by this method is $DPS=\{adk, bfl\,\overline{adk}, cek\overline{ad}\,\overline{bfl}, adgl\,\bar{k}\,\overline{bf}, cegl\overline{ad}\,\bar{k}\,\overline{bf}\}$. Based on this set of disjoint products, by applying (6), the following two-terminal reliability expression is obtained:

$$R_{s-t}=p_ap_dp_k+p_bp_fp_l(1-p_ap_dp_k)+$$
$$p_cp_ep_k(1-p_ap_d)(1-p_bp_fp_l)+$$
$$p_ap_dp_gp_lq_k(1-p_bp_f)+$$
$$p_cp_ep_gp_l(1-p_ap_d)\,q_k(1-p_bp_f)\,. \quad (12)$$

V. COMPARATIVE STUDIES

A. Comparison with Abraham's method

To demonstrate the correctness of the new method we propose comparative results related to two-terminal network reliability evaluation obtained by applying NMVI and the SVI method given by Abraham [1] are presented in this section. On the other hand, the aim of this first comparative study is to illustrate the ability of the new MVI method to anticipate well enough the terms for which the absorption law could be applicable (see *Remark* 1), and to neglect them in order to reduce the number of terms that must be processed.

For this comparative study, four medium-to-large stochastic network models have been considered. To check the implementation of the two methods, the numerical results for the case with all the links having the same reliability (e.g., $p=0.95$) are computed. The comparative results are presented in TABLE III.

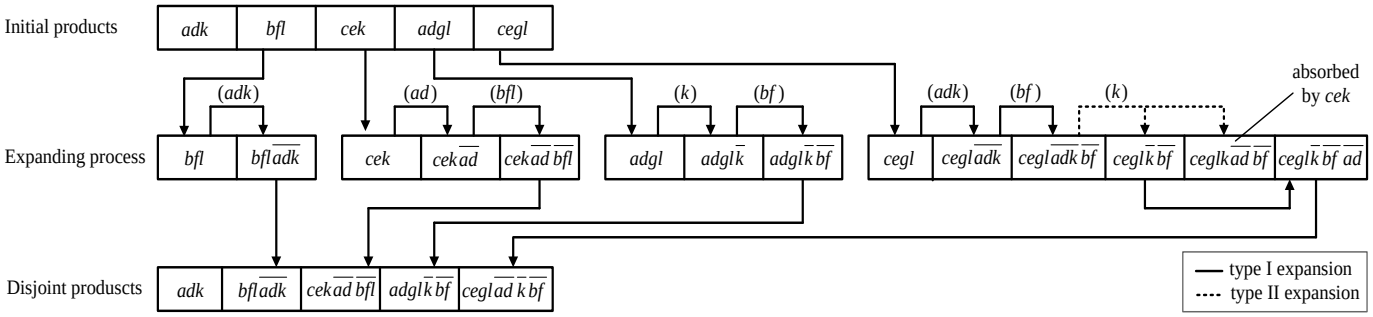
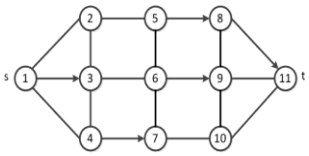
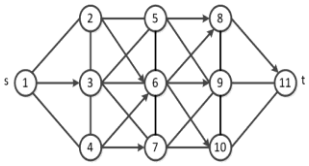
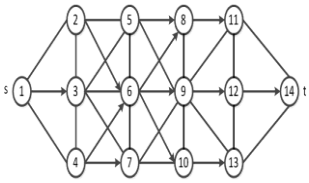
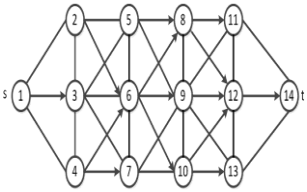


Fig. 2. Example of applying the method NMVI.

TABLE III. COMPARATIVE RESULTS: ABRAHAM'S METHOD VERSUS NMVI

Network model	Number of minimal paths	Applied method	Number of DPs	Processed terms	Absorption cases	R_{s-t} ($p = 0.95$)
	87	Abraham	2012	5472	675	0. 9994239
		NMVI	545	3458	546	
	993	Abraham	39395	174728	35687	0. 9997488
		NMVI	15502	136043	20384	
	5543	Abraham	897373	3961978	941589	0. 9997475
		NMVI	239101	2399387	456521	
	8716	Abraham	1835413	8057515	1817042	0. 9997488
		NMVI	574274	5318472	894473	

As shown in TABLE III, NMVI gives better solutions with fewer disjoint products. Also, the number of processed terms decreases significantly. Observe that the absorption rule is applicable in fewer cases (for the large networks, to less than half). This is an important merit for NMVI.

B. Comparison with other MVI methods

In this section our method is compared with other well-known MVI methods. For this comparative study the following MVI methods are considered:

- the method given by Veeraraghavan and Trivedi [3] (VT in this paper);

- CAREL method (PII option) given by Soh and Rai [2];
- KDH88 algorithm given by Heidtmann [17];
- The hybrid method (HM) given by Chaturvedi and Misra [19] that combines the best features of KDH88 and CAREL;
- NMVI.

The comparison criterion is the number of disjoint products generated, able to cover all the cases in which the two given nodes (*s* and *t*) are connected.

The network models we have considered for this comparative study are presented in Fig. 3. This selection includes network models typically used for comparing the efficiency of the methods dedicated to the problem of two-terminal network reliability evaluation. More exactly, we have considered those network models which are presented in at least two articles dedicated to the MVI techniques ([2], [3], [17] or [19]). Note that, another MVI algorithm dedicated to coherent-system reliability is presented in [18], but the paper does not report the number of disjoint products generated for the evaluated cases.

The comparative results are presented in TABLE IV. As shown in this table, compared to VT, CAREL or KDH88, NMVI gives better solutions with fewer disjoint products. When comparing NMVI with HM, their efficiency is found to be quite close.

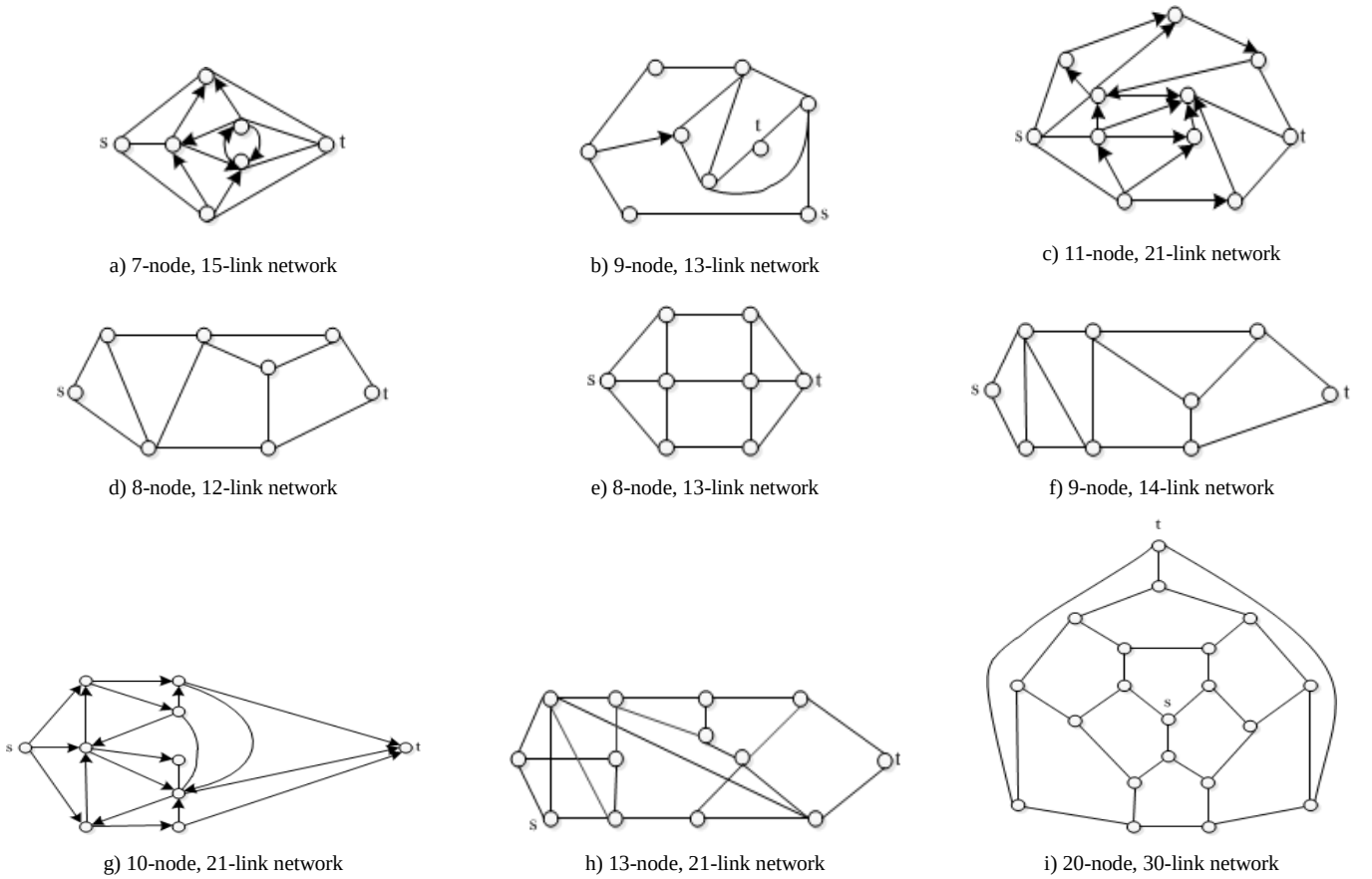
VI. CONCLUSIONS

In this paper, the process of algebraically extracting the disjoint products from a sum of products is discussed, and a new MVI method (NMVI) is proposed. In comparison with other well-known MVI methods (VT, CAREL or KDH88), NMVI gives better solutions with fewer disjoint products. At the same time, NMVI seems to be comparable with the hybrid method HM.

In case of complex networks, two or more such methods should be considered to obtain the best results. The hybrid methods that combine the best features of different MVI techniques must be taken into account.

Unfortunately, all the algorithms for exact reliability evaluation in network models fall in the NP-hard category, being difficult to apply for large networks, such as social networks. In these cases, other techniques for approximate evaluation can be applied, especially the Monte Carlo simulation (see, for example, [20]-[22]). Even so, the SDP algorithms for exact evaluation of two-terminal network reliability are still necessary for the validation of simulation programs.

Another approach for this SDP problem is based on binary decision diagrams (BDDs), as presented in [12], [23] [24]. This is a subject for a future work.



a) 7-node, 15-link network
b) 9-node, 13-link network
c) 11-node, 21-link network
d) 8-node, 12-link network
e) 8-node, 13-link network
f) 9-node, 14-link network
g) 10-node, 21-link network
h) 13-node, 21-link network
i) 20-node, 30-link network

Fig. 3. Network models for a comparative study of the MVI methods.

TABLE IV. COMPARATIVE RESULTS FOR SOME MVI METHODS

Network models presented in Fig. 3	Number of minimal paths	Number of disjoint products				
		VT [3]	CAREL [2]	KDH88 [17]	HM [19]	NMVI
a) 7-node, 15-link	14	–	23	27	–	23
b) 9-node, 13-link	18	27	30	31	–	25
c) 11-node, 21-link	18	82	94	101	–	82
d) 8-node, 12-link	24	41	39	41	38	38
e) 8-node, 13-link	29	77	76	75	77	77
f) 9-node, 14-link	44	90	82	87	80	82
g) 10-node, 21-link	64	305	309	–	–	298
h) 13-node, 21-link	281	–	2491	–	2302	2269
i) 20-node, 30-link	780	–	54032	–	46707	48696

REFERENCES

[1] J.A. Abraham, "An improved algorithm for network reliability", IEEE Trans. Reliability, vol. 28, pp. 58-61, April 1979.

[2] S. Soh, S. Rai, "Carel: Computer aided reliability evaluator for distributed computing networks", IEEE Trans. Parallel and Distributed Systems, vol 2, No. 2, pp. 199-213, April 1991.

[3] M. Veeraraghavan, K.S. Trivedi, "An improved algorithm for sysmbolic reliability analysis", IEEE Trans. Reliability, vol. 40, pp. 347-358, August 1991.

[4] J.S. Proven, M.O. Ball, "Computing network reliability in time polynomial in the number of cuts", Operations Research, vol. 32, pp. 516–526, 1984.

[5] M.O. Ball, J. S. Proven, "Disjoint products and efficient computation of reliability", Operations Research, vol. 36, pp. 703-715, 1988.

[6] K. S. Trivedi, Probability and Statistics with Reliability, Queueing and Computer Science Applications, John Wiley & Sons, New York, 2002.

[7] F. Beichelt, L. Spross, "An efficient method for reliability analysis of complex systems", J. Information Processing and Cybernetics EIK, vol. 23, pp. 227-235, 1987.

[8] P. Caşcaval, B.F. Romanescu, "Complementary Approaches for the Network Reliability Evaluation: Network Decomposition and Monte Carlo Simulation", Bul. Inst. Polit. Iaşi, Tomul L (LIV), Fasc. 1-4, Automatică şi Calculatoare, pp. 123-131, 2004.

[9] P. Caşcaval, A.R. Macovei, "Reliability Evaluation by Network Decomposition", Bul. Inst. Polit. Iaşi, Tomul XLIX (LIII), Fasc. 1-4, Automatică şi Calculatoare, pp. 56-65, 2003.

[10] P. Caşcaval, B.A. Botez, "Recursive Algorithm for Two-Terminal Network Reliability Evaluation", Bul. Inst. Polit. Iasi, LI (LV), Fasc.1-4, Automatică şi Calculatoare, pp. 137-146, 2005.

[11] M. Shooman, Reliability of computer systems and networks, John Wiley & Sons, New York, 2002, pp. 297-301.

[12] S.K. Chaturvedi, Network Reliability: Measures and Evaluation, Scrivener Publishing - Wiley, New Jersey, 2016.

[13] Y. Shen, "A new simple algorithm for enumerating all minimal paths and cuts of a graph", Microelectronics Reliability, vol. 35 (6), pp. 973–976, 1995.

[14] R. Mishra, M.A. Saifi, S.K. Chaturvedi, "Enumeration of minimal cutsets for directed networks with comparative reliability study for paths or cuts", Quality and Reliability Engineering International, vol. 32, pp. 555–565, February 2016.

[15] F. Beichelt, L. Spross, "An improved Abraham-method for generating disjoint products", IEEE Trans. Reliability, vol. 36, pp. 70-74, April 1987.

[16] M. O. Locks, "A minimizing algorithm for sum of disjoint products", IEEE Trans. Reliability, vol. 36, pp. 445-453, October 1987.

[17] K. Heidtmann, "Smaller sums of disjoint products by subproduct inversion", IEEE Trans. Reliability, vol. 38, No. 3, pp. 305-311, August 1989.

[18] T. Luo, K.S. Trivedi, "An improved algorithm for coherent-system reliability", IEEE Trans. Reliability, vol. 47, pp. 73-78, March 1998.

[19] S.K. Chaturvedi, K.B. Misra, "A hybrid method to evaluate reliability of complex networks", International Journal of Quality & Reliability Management, vol. 19, pp. 1098–1112, December 2002.

[20] K.F. Tee, L.R. Khan, H. Li, "Application of subset simulation in reliability estimation of underground pipelines", Reliability Engineering & System Safety, vol. 130, pp. 125–131, October 2014.

[21] K. Zuev, S. Wu, J. Beck, "General network reliability problem and its efficient solution by subset simulation", Probabilistic Engineering Mechanics, vol. 40, pp. 25–35, April 2015.

[22] H.-S. Li, Y.-Z. Ma, Z. Cao, "A generalized subset simulation approach for estimating small failure probabilities of multiple stochastic responses", Computers & Structures, vol. 153, pp. 239–251, June 2015.

[23] X. Zang, H.-R. Sun, and K.S. Trivedi, "A BDD approach to dependability analysis of distributed computer systems with imperfect coverage", Dependable Network Computing, D. R. Avresky (ed.), Kluwer Academic Publishers, Amsterdam, December 1999, pp. 167-190.

[24] A. Birolini, Reliability engineering. Theory and practice, Springer-Verlag, Berlin Heidelberg, 2014, pp. 57-60.

INISTA2017

2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications
3-5 July 2017, Gdynia, Poland

INISTA 2017 is over!

Thank you to everyone involved! We hope you enjoyed it and keep in touch with the people you met here.

INISTA 2017

The 2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA 2017) has been organized since 2005. It aims to bring together the researchers from the entire spectrum of the multi-disciplinary fields of intelligent systems and establish effective means of communication between them. In particular, it focuses on all aspects of intelligent systems and the related applications, from the points of view of both theory and practice. Apart of the main track it includes work-shops, tutorials, special sessions and plenary talks by invited speakers.

The conference will be hosted in Gdynia, known for its picturesque seashore. Gdynia is a part of Tricity, which also includes thousand-year-old Gdansk, Poland's cultural gem, and Sopot, a health - resort.

INISTA 2017 is organized by Gdynia Maritime University, Poland, in cooperation with Yildiz Technical University, Turkey, IEEE Poland Section, Polish Chapter IEEE Systems, Man, and Cybernetics Society and IEEE SMC TC on

NEWS

The conference is over!

Look at the **INISTA 2017** official photos or download **INISTA 2017** group photo or see other photos.

The **Best Session** of INISTA 2017 was awarded to **Semantic Intelligence (SINTEL 2017)**, organized by **Kambiz Badie and Maryam Tayefeh Mahmoudi**.

The **Best Paper** award went to "SDP Algorithm for Network Reliability Evaluation" by Petru Caşcaval and Sabina-Adriana Floria. The paper was presented by **Sabina-Adriana Floria.**

The **Best Student Paper** award went to "Applying Map-Reduce to Imbalanced Data Classification" by Joanna Jędrzejowicz, Jakub Neumann, Piotr Synowczyk and Magdalena Zakrzewska. The paper was presented by **Magdalena Zakrzewska**

All papers can be found in IEEE eXplore.

Next INISTA conference will be in Thessaloniki. We hope to meet you there!

ELECTRONICS AND ELECTRICAL ENGINEERING
ISSN 1392 – 1215 2012. No. 3(119)
ELEKTRONIKA IR ELEKTROTECHNIKA

SYSTEM ENGINEERING, COMPUTER TECHNOLOGY
T 120
SISTEMŲ INŽINERIJA, KOMPIUTERINĖS TECHNOLOGIJOS

A Multibackground March Test for Static Neighborhood Pattern-Sensitive Faults in Random-Access Memories

C. Huzum, P. Cascaval
Department of Computer Science and Engineering,"Gheorghe Asachi" Technical University of Iaşi,
Prof. dr. doc. Dimitrie Mangeron Str., no. 27, Iaşi, România, phone: +40-232-231343, e-mail: chuzum@cs.tuiasi.ro

crossref http://dx.doi.org/10.5755/j01.eee.119.3.1369

Introduction

Rapid increase of density in the integrated circuits has an immediate effect upon memory testing. On one hand, the capacity of random-access memories chips enhances, thus increasing the test time and cost; on the other hand, the density of memory circuits grows, therefore more failure modes and faults need to be taken into account in order to obtain a good quality product. Accordingly, there are two conflicting constraints that need to be dealt with when considering a test algorithm: reducing the number of memory operations in order to permit large capacity memories to be tested in an appropriate period of time and covering a larger variety of memory faults [1, 2].

As a result of the increasing coupling effect triggered by the growing density of memory circuits, the pattern-sensitive fault (PSF) is becoming an important fault model [3–5]. The PSF model is a type of coupling fault, with several aggressor cells (4, 9 etc.) and only one victim cell. In this work, the neighborhood PSF (NPSF) has been considered. This is a particular PSF, in which the aggressor cells are located in the physical neighborhood of the victim cell. The NPSF model was first defined by Hayes in 1980 [3]. He also devised a memory test for this model [3]. Soon after that, Suk and Reddy have proposed a new memory test [6] based on a bipartite method. This test divides the memory cells into two partitions and applies a sequence of transitions to cover all possible victim-aggressor combinations. Unfortunately, for the memory chips currently used, the test proposed by Suk and Reddy needs a long time to perform. In 2002, other more efficient March tests were given by Cheng, Tsai, and Wu, namely: CM-79N and March-100N [7]. More recently, Julie, Wan Zuha and Sidek use a modified version of March-100N for diagnosis of SRAM [8]. In all these papers the authors have limited themselves only to the class of simple faults. In this work, we have also focused on the problem of testing the linked neighborhood pattern-sensitive faults.

Notations, definitions and fault classifications

An operation sequence that results in a difference between the observed and the expected memory behaviour is called a *sensitizing operation sequence* (*S*). The observed memory behaviour that deviates from the expected one is called *faulty behaviour* (*F*). In order to specify a certain fault, one has to specify the *S*, together with the corresponding faulty behaviour *F*, and the read result (*R*) of *S*, in case it is a read operation. The combination of *S*, *F* and *R* for a given memory failure is called a *Fault Primitive* (FP), and is usually denoted as <*S*/*F*/*R*>. The concept of FPs allows for establishing a complete framework of all memory faults. Some classifications of FPs can be made based on different and independent factors of *S* [9]:

a) Depending on the number of simultaneous operations required in the *S*, FPs are classified into *single-port* and *multi-port* faults:

- *single-ports faults:* These are FPs that require at the most one port in order to sensitize a fault. Note that single-port faults can be sensitized in single-port as well as in multi-port memories,
- *multi-port faults:* These are FPs that can only sensitize a fault by performing two or more operations simultaneously via different ports.

b) Depending on the number of sequential operations required in the *S*, FPs are classified into *static faults* and *dynamic faults*. Let #*O* be the number of different operations carried out sequentially in a *S*:

- *static faults:* These are FPs which sensitize a fault by performing at most one operation in the memory (#*O*=0 or #*O*=1),
- *dynamic faults*: These are FPs that perform more than one operation sequentially in order to sensitize a fault (#*O*>1).

c) Depending on the way FPs manifest themselves, they can be divided into *simple faults* and *linked faults*:

- *simple faults:* These are faults which cannot be influenced by another fault. That means that the behaviour of a simple fault cannot change the behaviour of another one; therefore masking cannot occur,
- *linked faults:* These are faults that do influence the behaviour of each other. That means that the behaviour of a certain fault can change the behaviour of another one such that masking can occur. Note that linked faults consist of two or more simple faults.

In this work, single-port, static faults are considered. From here on, the term 'fault' refers to a single-port, static, simple fault and the term 'linked fault' means single-port, static, linked fault.

The following notations are usually used to describe operations on RAMs:

- $r0$ ($r1$) denotes a read 0 (1) operation from a cell;
- $w0$ ($w1$) denotes a write 0 (1) operation into a cell;
- $\uparrow$ ($\downarrow$) denotes an up (down) transition due to a certain sensitizing operation.

The neighborhood pattern-sensitive fault model

RAM faults can also be divided into *single-cell* and *multi-cell* faults. Single-cell faults consist of FPs involving a single cell, while multi-cell faults consists of FPs involving more than one cell. In this work, we consider a particular class of multi-cell faults (also called *coupling faults*), namely the pattern sensitive faults (PSF). The PSF is a coupling fault, which affects the content of a memory cell (called the *victim cell* or the *base cell*), or the ability to change its content, when other memory cells (called *aggressor cells*) have certain patterns. It is unnecessary and unrealistic to consider all possible patterns of all the memory cells, therefore simplified models of neighborhood pattern sensitive faults (NPSF) have been introduced. In these models, the aggressor cells are limited to the physical neighborhood of the victim cell. Depending on the number of aggressor cells, NPSF can be divided into several types, but only two of those are more commonly used: Type-1 NPSF and Type-2 NPSF, with four and eight aggressor cells, respectively, as illustrated in Fig. 1 [10].

Like in the most previous works, in this paper only the Type-1 NPSF has been considered.

Due to the features of the NPSF model, the general notation for a FP is particularized, thus in the rest of this paper a FP is denoted as <N W E S; B/B_f> [7], where:

- N, W, E, S describes the sensitizing value or operation in the aggressor cells, placed as presented in Fig. 1a;
- B describes the correct value or transition in the base cell;
- B_f shows the faulty value or transition of the base cell.

Note that N, W, E, S, B and $B_f \in \{0, 1, \uparrow, \downarrow\}$.

Depending on the behavior of the fault, the NPSFs can be divided into three classes [10], namely:

- Static NPSF (SNPSF): the base cell is forced to a certain value when the aggressor cells have a certain pattern. An example of a static NPSF is FP_1=<0100; 0/1>, where the base cell is forced to 1 when the aggressor cells have the pattern 0100;
- Passive NPSF (PNPSF) reflects the impossibility of the base cell to execute a transition due to the existence of a certain pattern in the aggressor cells. An example of a PNPSFs is FP_2=<1100; $\downarrow$/1>, where the base cell cannot switch from 1 to 0 because the aggressor cells have the pattern 1100;
- Active NPSF (ANPSF): a certain transition in one of the aggressor cells forces the victim cell to change its value when the other aggressor cells (also called *enabling cells*) have a certain pattern. An example of this class of faults is FP_3=<10$\uparrow$0; 0/1>, where a transition in the E cell causes the base cell to flip from 0 to 1 when the N, W and S cells have the pattern 100.

The model of NPSFs we have considered can be entirely described by the set of FPs presented in Table 1. There are 192 fault primitives of which 32 SNPSFs, 32 PNPSFs, and 128 ANPSFs.

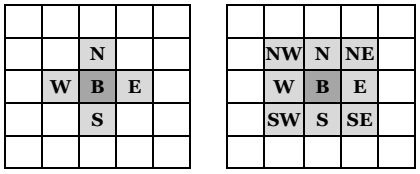


Fig. 1. Common types of neighborhood pattern sensitive faults: a – Type-1 NPSF; b – Type-2 NPSF

The linked neighborhood pattern-sensitive faults are NPSFs that influence the behavior of each other, such that masking can occur. Therefore, they are more difficult to detect.

Table 1. List of NPSF Primitives

Fault primitives		Fault type
<$x\ y\ z\ t$; 0 / 1>	$x, y, z, t \in \{0, 1\}$	SNPSF
<$x\ y\ z\ t$; 1 / 0>		
<$x\ y\ z\ t$; $\uparrow$ / 0>	$x, y, z, t \in \{0, 1\}$	PNPSF
<$x\ y\ z\ t$; $\downarrow$ / 1>		
<$x\ y\ z\ \uparrow$; 0 / 1>	$x, y, z \in \{0, 1\}$	ANPSF
<$x\ y\ z\ \downarrow$; 0 / 1>		
<$x\ y\ z\ \uparrow$; 1 / 0>		
<$x\ y\ z\ \downarrow$; 1 / 0>		
<$x\ y\ \uparrow\ z$; 0 / 1>		
<$x\ y\ \downarrow\ z$; 0 / 1>		
<$x\ y\ \uparrow\ z$; 1 / 0>		
<$x\ y\ \downarrow\ z$; 1 / 0>		
<$x\ \uparrow\ y\ z$; 0 / 1>		
<$x\ \downarrow\ y\ z$; 0 / 1>		
<$x\ \uparrow\ y\ z$; 1 / 0>		
<$x\ \downarrow\ y\ z$; 1 / 0>		
<$\uparrow\ x\ y\ z$; 0 / 1>		
<$\downarrow\ x\ y\ z$; 0 / 1>		
<$\uparrow\ x\ y\ z$; 1 / 0>		
<$\downarrow\ x\ y\ z$; 1 / 0>		

A linked fault consists of two or more FPs (even number) with contrary effects on the same victim (base) cell. For example, take a NPSF fault in which an up transition into cell W changes the state of cell B from 1 to 0, when the enabling cells have the pattern 100, whereas an up transition into cell S changes the state of cell B from 0 to 1, when the enabling cells have the pattern 011. This is a linked fault that can be modeled by two FPs: FP_1=<1↑00; 1/ 0>, and FP_2=<011↑; 0/1>.

Even though there are many papers that deal with the issue of linked faults, such as [11] for two-cell linked faults, or [14] for three-cell linked faults, we do not know any work treating this model of linked NPSFs.

A new memory test – March-76N

In order to describe the memory test, first some notations regarding the March tests are given. Usually, a complete March test is delimited by '{…}' bracket pair, while a March element is delimited by the '(…)' bracket pair. March elements are separated by semicolons, and the operations within a March element are separated by commas. Note that all operations of a March element are performed at a certain address, before proceeding to the next address. The whole memory is checked homogeneously in either one of two orders: ascending address order (⇑) or descending address order (⇓). When the address order is not relevant, the symbol ⇳ is used.

As stated in [7], if a certain March element is applied to a solid background (all ones or all zeros), when we are reading from or writing to a cell C, then all the cells with higher address than C have the same state, while those with lower address than C also have the same state. So, most of the NPSF faults cannot be activated by applying March elements to these solid backgrounds. Therefore, the necessity of a March test that runs under several different backgrounds arises. These are called multibackground March tests [12, 13].

The test we propose, called March-76N, uses sixteen different backgrounds, denoted by BG_0, BG_1, …, BG_{15}, as presented in Fig. 2. For a certain cell, let *a* be the value within the background, and *b* the complement of *a*. For example, when initializing the memory with BG1, *a* is 0 for the lines 1, 3, 4, 6, 7, 9 etc. of the memory, and 1 for lines 2, 5, 8 etc. These backgrounds were selected for the test in order to create all victim-aggressor combinations necessary for activating all the faults in the NPSF model.

March-76N forms an alternating series of March elements and background changes (as Cockburn proposed in [15]) and has the following structure:

$$\{ ⇳ (w0); [⇑(ra, wb); ⇑(rb, wa); CBG_i], i = 1, 2, \ldots, 16 \}, \quad (1)$$

where CBG_i is a sequence for checking and changing from the current background to the next. More exactly, the sequence CBG_i has the following meaning:

- CBG_i, i =1, 2, …, 15, changes the background from BG_{i-1} to BG_i. Note that when changing from one background to the next, only the cells that must change states are written. Each write operation is also preceded (for checking) by a read operation. We can observe in Fig. 2 that the backgrounds were ordered such as the difference between every two successive backgrounds consists of three out of the nine cells so that all background changes affect only a trinity of the cells.
- CBG_{16} reads the whole memory for the finally checking (CBG_{16} = ⇳(*ra*)).

Consequently, March-76N comprises the following operations:

- N operations for the first initialization;
- $[2N + 2N + 2\times3N/9] \times 15 = 70N$ operations, for the first fifteen series of March elements and background changes;
- $2N+2N+N=5N$, for the sixteenth series of March elements and final checking.

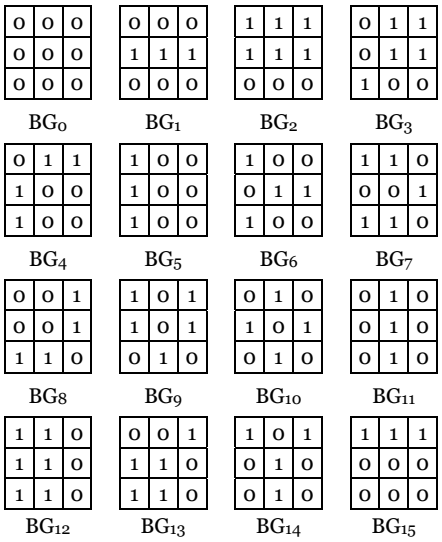


Fig. 2. The 16 backgrounds for the test March-76N

Therefore, the length of March-76N is $N + 70N + 5N = 76N$.

A simulation study has been made in order to determine whether March-76N covers all simple and also all linked NPSFs. Taking into account that the patterns of the backgrounds used by this test are composed of 3×3 cells, for the simulation study, the memory cells have been divided into nine mutually disjoint subsets. These are denoted by S_1, S_2, …, S_9, depending on their location. Let *r* and *c* be the row address and the column address, respectively, of a memory cell. The cell with the address (*r*, *c*) belongs to a certain subset according to the following rule:

$$\text{the cell } (r,c) \in \begin{cases} S_1, & \text{if } c \,\%\, 3 = 0 \text{ and } r \,\%\, 3 = 0, \\ S_2, & \text{if } c \,\%\, 3 = 0 \text{ and } r \,\%\, 3 = 1, \\ S_3, & \text{if } c \,\%\, 3 = 0 \text{ and } r \,\%\, 3 = 2, \\ S_4, & \text{if } c \,\%\, 3 = 1 \text{ and } r \,\%\, 3 = 0, \\ S_5, & \text{if } c \,\%\, 3 = 1 \text{ and } r \,\%\, 3 = 1, \\ S_6, & \text{if } c \,\%\, 3 = 1 \text{ and } r \,\%\, 3 = 2, \\ S_7, & \text{if } c \,\%\, 3 = 2 \text{ and } r \,\%\, 3 = 0, \\ S_8, & \text{if } c \,\%\, 3 = 2 \text{ and } r \,\%\, 3 = 1, \\ S_9, & \text{if } c \,\%\, 3 = 2 \text{ and } r \,\%\, 3 = 2, \end{cases} \quad (2)$$

where "%" denotes the modulo operation.

For a memory array with 8 rows and 8 columns, these nine subsets of cells are illustrated in Fig. 3.

	0	1	2	3	4	5	6	7
0	S_1	S_4	S_7	S_1	S_4	S_7	S_1	S_4
1	S_2	S_5	S_8	S_2	S_5	S_8	S_2	S_5
2	S_3	S_6	S_9	S_3	S_6	S_9	S_3	S_6
3	S_1	S_4	S_7	S_1	S_4	S_7	S_1	S_4
4	S_2	S_5	S_8	S_2	S_5	S_8	S_2	S_5
5	S_3	S_6	S_9	S_3	S_6	S_9	S_3	S_6
6	S_1	S_4	S_7	S_1	S_4	S_7	S_1	S_4
7	S_2	S_5	S_8	S_2	S_5	S_8	S_2	S_5

Fig. 3. The cell subsets for an 8×8 memory chip array

Note that every memory cell that belongs to the same cell subset will support the same operations during the test March-76N. Moreover, if two base cells belong to the same subset, their aggressor cells will support, respectively, the same initializations. Hence, for the simulation study, only nine locations (one for each subset) have to be considered for the base cell.

For the linked fault coverage evaluation, the linked faults consisting of two simple faults have been considered. There are 96 NPSFs that flip the base cell from 0 to 1 and 96 that flip it from 1 to 0. Consequently, a total of 96 × 96 = 9216 linked faults have been considered for the simulation study.

The conclusion of the study is that March-76N covers entirely the model of simple and linked NPSFs.

Theorem: March-76N is able to detect all simple and linked NPSFs in our model.

Table 2. Bit sequence for each background and for each subset of base cell

	S_1	**S_2**	**S_3**	**S_4**	**S_5**	**S_6**	**S_7**	**S_8**	**S_9**
BG_0	000 0 / 000 000 / 000 0 — I_1	0 / 000 — I_1	0 / 000 — I_1	0 / 000 — I_1	0 / 000 — I_1	0 / 000 — I_1	0 / 000 — I_1	0 / 000 — I_1	0 / 000 — I_1
BG_1	000 0 / 111 000 / 000 1 — I_2	0 / 111 — I_{16}	1 / 000 — I_3'	0 / 000 — I_2	0 / 111 — I_{16}	1 / 000 — I_3'	0 / 000 — I_2	0 / 111 — I_{16}	1 / 000 — I_3'
BG_2	111 0 / 111 111 / 000 1 — I_3	1 / 111 — I_2'	1 / 000 — I_{16}'	0 / 111 — I_3	1 / 111 — I_2'	1 / 000 — I_{16}'	0 / 111 — I_3	1 / 111 — I_2'	1 / 000 — I_{16}'
BG_3	011 1 / 011 101 / 100 0 — I_4	0 / 101 — I_7'	0 / 010 — I_5'	0 / 011 — I_9'	1 / 011 — I_8	1 / 100 — I_{14}	0 / 110 — I_{10}	1 / 110 — I_{11}'	1 / 001 — I_{15}'
BG_4	011 1 / 100 101 / 100 1 — I_5	0 / 010 — I_4'	1 / 010 — I_7	0 / 011 — I_{14}'	1 / 100 — I_9	0 / 100 — I_8'	0 / 110 — I_{15}	1 / 001 — I_{10}'	0 / 001 — I_{11}
BG_5	100 1 / 100 010 / 100 1 — I_6	1 / 010 — I_6	1 / 010 — I_6	0 / 100 — I_{13}'	0 / 100 — I_{13}'	0 / 100 — I_{13}'	0 / 001 — I_{12}	0 / 001 — I_{12}	0 / 001 — I_{12}
BG_6	100 1 / 011 010 / 100 0 — I_7	1 / 101 — I_5	0 / 010 — I_4'	0 / 100 — I_8'	0 / 011 — I_{14}'	1 / 100 — I_9	0 / 001 — I_{11}	1 / 110 — I_{15}	0 / 001 — I_{10}'
BG_7	110 1 / 001 011 / 110 0 — I_8	1 / 100 — I_{14}	0 / 011 — I_9'	1 / 110 — I_{11}'	0 / 001 — I_{15}'	1 / 110 — I_{10}	1 / 001 — I_7'	0 / 010 — I_5'	1 / 101 — I_4
BG_8	001 1 / 001 100 / 110 0 — I_9	0 / 100 — I_8'	0 / 011 — I_{14}'	1 / 001 — I_{10}'	0 / 001 — I_{11}	0 / 110 — I_{15}	0 / 010 — I_4'	1 / 010 — I_7	1 / 101 — I_5
B_9	101 0 / 101 110 / 010 1 — I_{10}	1 / 110 — I_{11}'	1 / 001 — I_{15}'	1 / 101 — I_4	0 / 101 — I_7'	0 / 010 — I_5'	0 / 011 — I_9'	1 / 011 — I_8	1 / 000 — I_{14}
BG_{10}	010 0 / 101 001 / 010 1 — I_{11}	0 / 110 — I_{15}	1 / 001 — I_{10}'	1 / 010 — I_7	0 / 101 — I_5	0 / 010 — I_4'	0 / 100 — I_8'	0 / 011 — I_{14}'	1 / 000 — I_9
BG_{11}	010 0 / 010 001 / 010 0 — I_{12}	0 / 001 — I_{12}	0 / 001 — I_{12}	1 / 010 — I_6	0 / 101 — I_6	0 / 101 — I_6	1 / 000 — I_{13}'	0 / 100 — I_{13}'	0 / 100 — I_{13}'
BG_{12}	110 1 / 110 011 / 110 1 — I_{13}	1 / 011 — I_{13}	1 / 011 — I_{13}	1 / 110 — I_{12}'	1 / 110 — I_{12}'	1 / 110 — I_{12}'	0 / 101 — I_6'	0 / 101 — I_6'	0 / 101 — I_6'
BG_{13}	001 1 / 110 100 / 110 1 — I_{14}	0 / 011 — I_9'	1 / 011 — I_8	1 / 001 — I_{15}'	0 / 110 — I_{10}	1 / 110 — I_{11}'	0 / 010 — I_5'	1 / 101 — I_4	0 / 101 — I_7'
BG_{14}	101 0 / 010 110 / 010 0 — I_{15}	1 / 001 — I_{10}'	0 / 001 — I_{11}	1 / 101 — I_5	0 / 010 — I_4'	0 / 010 — I_7	0 / 011 — I_{14}'	1 / 000 — I_9	0 / 000 — I_8'
BG_{15}	111 0 / 000 111 / 000 0 — I_{16}	1 / 000 — I_3'	0 / 000 — I_2	0 / 111 — I_{16}	1 / 000 — I_3'	0 / 000 — I_2	0 / 111 — I_{16}	1 / 000 — I_3'	0 / 000 — I_2

Proof:

A. March-76N is able to detect all simple NPSFs.

When the memory is initialized with a certain background, this causes the appearance of different bit sequences in neighborhoods that have the base cell in different subsets. For example, when the memory is initialized with BG_2, if the base cell belongs to S_1, its neighborhood contains the bit sequence presented in Fig. 4a, and if the base cell belongs to S_5, its neighborhood consists of the bit sequence presented in Fig. 4b.

Table 2 presents for each background, the bit sequences that appear in the neighborhood of the base cell when it belongs to each subset. Note that, whichever subset the base cell belongs to, the initialization of the memory cells with all backgrounds will determine the appearance of all 16 possible bit sequences or their complements (denoted by I_1, I_2,...,I_{16}, and I_1', I_2', ... I_{16}', respectively, as shown in Table 3) in a different order.

On the other hand, another simulation study has been made in order to determine which faults are detected by applying each March element of the memory test on each of the 16 bit sequences.

Table 3 shows these faults grouped by the test sequence that detects them. Each of the sixteen series of our test is composed of three test sequences (two March elements and a test sequence for changing the background) identified with a superscript (x), where $x \in \{1, 2, 3\}$, as follows:

$$[\Uparrow(ra, wb)^{(1)}; \Uparrow(rb, wa)^{(2)}; \mathrm{CBG}_i^{(3)}],\ i = 1, 2, \ldots, 16. \quad (3)$$

To simplify the writing in Table 3, the '<' and '>' symbols usually used to denote a fault primitive have been neglected. Moreover, a bit sequence from a neighborhood is written as NWBES and it contains, respectively, the values in the north, west, base, east, and south cell.

Table 3 includes all 192 fault primitives of this NPSF model. Note that, by applying the two March elements (1) and (2) on a bit sequence or on their complement, the same operations are carried out in the memory, and consequently, the same memory faults are detected. Therefore, when the three test sequences in March-76N are applied to all of the 16 bit sequences I_1, I_2, …, I_{16}, or to their complements, all NPSF simple faults are detected.

Note that Table 2 shows that whichever subset the base cell belongs to, applying all of the 16 backgrounds to the memory will determine the appearance in the neighborhood of the base cell of all 16 bit sequences or their complements in a different order. In conclusion, wherever the base cell is located, March-76N covers the whole model of simple NPSFs.

B. March-76N is able to detect all linked NPSFs.

Let us consider the pairs of simple faults that can mask their effects while running the March-76N memory test. The simple faults that belong to such pairs have opposite effect upon the base cell and are both activated between two consecutive read operations of the base cell. For example, when running the test on bit sequence I_1, the simple faults pairs shown in Table 4 can mask their effects, hiding an incorrect behaviour of the memory. All the pairs composed of two active and/or passive simple faults that can be activated between two consecutive reading operations have been considered.

Table 3. Faults detected by each bit sequence within March-76N

Bit sequence	Notation	NPSF faults (grouped by the test sequences that detect them)			
		(1)	*(2)*		*(3)*
00000	$\mathbf{I_1}$	↑000;0/1 1↑00;0/1	1100;↑/0 1100; -/0 11↑0;1/0 111↑;1/0	↓111;1/0 0↓11;1/0	0011;↓/1 0011; -/1 00↓1;0/1 000↓;0/1
00001	$\mathbf{I_2}$	↑001;0/1 1↑01;0/1	1101;↑/0 1101; -/0 11↑1;1/0 111↓;1/0	↓110;1/0 0↓10;1/0	0010;↓/1 0010; -/1 00↓0;0/1 000↑;0/1
01111	$\mathbf{I_3}$	↑111;1/0 1↓11;1/0	1011;↓/1 1011; -/1 10↓1;0/1 100↓;0/1	↓000;0/1 0↑00;0/1	0100;↑/0 0100; -/0 01↑0;1/0 011↑;1/0
11010	$\mathbf{I_4}$	↓110;0/1 0↓10;0/1	0010;↑/0 0010; -/0 00↓0;1/0 000↑;1/0	↑001;1/0 1↑01;1/0	1101;↓/1 1101; -/1 11↑1;0/1 111↓;0/1
11011	$\mathbf{I_5}$	↓111;0/1 0↓11;0/1	0011;↑/0 0011; -/0 00↓1;1/0 000↓;1/0	↑000;1/0 1↑00;1/0	1100;↓/1 1100; -/1 11↑0;0/1 111↑;0/1
10101	$\mathbf{I_6}$	↓001;1/0 0↑01;1/0	0101;↓/1 0101; -/1 01↑1;0/1 011↓;0/1	↑110;0/1 1↓10;0/1	1010;↑/0 0101; -/0 10↓0;1/0 100↑;1/0
10100	$\mathbf{I_7}$	↓000;1/0 0↑00;1/0	0100;↓/1 0100; -/1 01↑0;0/1 011↑;0/1	↑111;0/1 1↓11;0/1	1011;↑/0 1011; -/0 10↓1;1/0 100↓;1/0
10110	$\mathbf{I_8}$	↓010;1/0 0↑10;1/0	0110;↓/1 0110; -/1 01↓0;0/1 010↑;0/1	↑101;0/1 1↓01;0/1	1001;↑/0 1001; -/0 10↑0;1/0 101↓;1/0
11000	$\mathbf{I_9}$	↓100;0/1 0↓00;0/1	0000;↑/0 0000; -/0 00↑0;1/0 001↑;1/0	↑011;1/0 1↑11;1/0	1111;↓/1 1111; -/1 11↓1;0/1 110↓;0/1
01101	$\mathbf{I_{10}}$	↑101;1/0 1↓01;1/0	1001;↓/1 1001; -/1 10↑1;0/1 101↓;0/1	↓010;0/1 0↑10;0/1	0110;↑/0 0110; -/0 01↑0;1/0 010↑;1/0
00011	$\mathbf{I_{11}}$	↑011;0/1 1↑11;0/1	1111;↑/0 1111; -/0 11↓1;1/0 110↓;1/0	↓100;1/0 0↓00;1/0	0000;↓/1 0000; -/1 00↑0;0/1 001↑;0/1
00010	$\mathbf{I_{12}}$	↑010;0/1 1↑10;0/1	1110;↑/0 1110; -/0 11↓0;1/0 110↑;1/0	↓101;1/0 0↓01;1/0	0001;↓/1 0001; -/1 00↑1;0/1 001↓;0/1
10111	$\mathbf{I_{13}}$	↓011;1/0 0↑11;1/0	0111;↓/1 0111; -/1 01↓1;0/1 010↓;0/1	↑100;0/1 1↓00;0/1	1000;↑/0 1000; -/0 10↑0;1/0 101↑;1/0
11001	$\mathbf{I_{14}}$	↓101;0/1 0↓01;0/1	0001;↑/0 0001; -/0 00↑1;1/0 001↓;1/0	↑010;1/0 1↑10;1/0	1110;↓/1 1110; -/1 11↓0;0/1 110↑;0/1
01100	$\mathbf{I_{15}}$	↑100;1/0 1↓00;1/0	1000;↓/1 1000; -/1 10↑0;0/1 101↑;0/1	↓011;0/1 0↑11;0/1	0111;↑/0 0111; -/0 01↓1;1/0 010↓;1/0
01110	$\mathbf{I_{16}}$	↑110;1/0 1↓10;1/0	1010;↓/1 1010;-/1 10↓0;0/1 100↑;0/1	↓001;0/1 0↑01;0/1	0101;↑/0 0101;-/0 01↑1;1/0 011↓;1/0

Table 4. Simple fault pairs that can mask their effect when applying March-76N on bit sequence I_1

↑000;0/1 1↑00;1/0	1100;↑/0 11↑0;0/1	1100;↑/0 111↑;0/1	1100;↑/0 ↓111;0/1	1100;↑/0 0↓11;0/1
11↑0;1/0 111↑;0/1	11↑0;1/0 ↓111;0/1	11↑0;1/0 0↓11;0/1	111↑;1/0 ↓111;0/1	111↑;1/0 0↓11;0/1
↓111;1/0 0↓11;0/1	0011;↓/1 00↓1;1/0	0011;↓/1 000↓;1/0	00↓1;0/1 000↓;1/0	

Note that the second fault in each pair in Table 4 is detected by applying the test on bit sequence I_5 (see Table 3). I_1 and I_5 have the same value for the base cell and complementary values for the aggressor cells (I_1 is 00000, and I_5 is 11011). Thus, for each of the pairs in Table 4, by applying March-76N to the bit sequence I_5, only the second fault in each of the pairs is activated, such that the masking will not occur.

In the same way, the linked NPSFs that can appear while applying March-76N on a certain bit sequence will

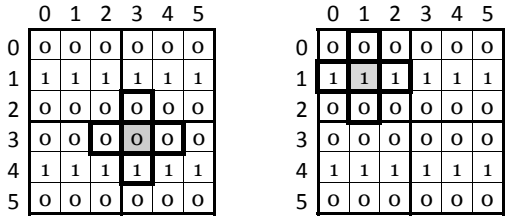


Fig. 4. Bits sequences that appear in BG_2: a – Base cell belongs to S_1; b – Base cell belongs to S_5

be detected by applying the test on another bit sequence that has the same value for the base cell and opposite values for the aggressor cells.

Therefore, every linked NPSF consisting of a pair of simple NPSFs that mask their effect while applying the memory test on a certain background (denoted by BG_m) will be detected when applying the test on another background (denoted by BG_d). Let I_m be the bit sequence that appears in the neighborhood of the base cell when the memory is initialized with BG_m. Then, BG_d is the background that causes the appearance of the pair of bit sequence I_m (as can be found in Table 5) in the neighborhood of the base cell.

Conclusions

The test we propose improves the performance of the test March-76N given by Cheng, Tsai, and Wu [7], which is the shortest published memory test dedicated to this NPSF model. In comparison with CM-79N, March-76N is shorter with 3.33*N*. Moreover, March-76N is able to detect all linked faults whereas, as presented in [16], March-76N cannot cover entirely the set of linked NPSFs.

References

1. **Adams R. D.** High performance memory testing: design principles, fault modelling and self–test – USA: Kluwer Academic Publishers, 2003 – 247 p.
2. **Hamdioui S.** Testing static random access memories: defects, fault models and test patterns – The Netherlands: Kluwer Academic Publishers, 2004. – 240 p.
3. **Hayes J. P**. Testing memories for single–cell pattern–sensitive fault // IEEE Trans. Comput., 1980. – Vol. 29. – P. 249–254.
4. **Kang D. C., Cho S. B.** An efficient build–in self–test algorithm for neighborhood pattern sensitive faults in high–density memories" // Proc. 4th Korea–Russia Int. Symp. Science and Technology, 2000. – Vol. 2. – P. 218–223.
5. **Cockburn B. F.** Deterministic tests for detecting scrambled pattern–sensitive faults in RAMs // Proc. IEEE Int. Workshop Memory Technology, Design and Testing (MTDT), 1995. – P. 117–122.
6. **Suk D. S., Reddy M.** Test procedures for a class of pattern–sensitive faults in semiconductor random–access memories // IEEE Trans. Comput., 1980. – Vol. 29. – P. 419–429.
7. **Cheng K. L., Tsai M. F., Wu C. W.** Neighborhood pattern–sensitive fault testing and diagnostics for random–access memories // IEEE Trans. On CAD, 2002. – Vol. 21. – No. 11. – P. 1328–1336.
8. **Julie R. R., Wan Zuha W. H., Sidek R. M.** 12N test procedure for NPSF testing and diagnosis for SRAMs // Proc. IEEE Int. Conf. on Semiconductor Electronics, 2008. – P. 430–435. DOI: 10.1109/SMELEC.2008.4770357.
9. **Hamdioui S., van de Goor A. J., Rodgers M.** March SS: A test for all static simple RAM faults // Proc. of IEEE Int'l Workshop on Memory Technology, Design and Testing, 2002. – P. 95–100.
10. **Van de Goor A. J.** Testing semiconductor memories: theory and practice – John Wiley & Sons Inc, 1991. – 512 p.
11. **Benso A., Bosio A., Di Carlo S., Di Natale G., Prinetto P**. A 22n March Test for Realistic Static Linked Faults in SRAMs // Proceedings of the Eleventh IEEE European Test Symposium, 2006. – P. 49–54.
12. **Yarmolik V., Klimets Y., Demidenko S.** March PS(23N) test for DRAM pattern–sensitive faults // Proc. Seventh IEEE Asian Test Symp.(ATS), 1998. – P. 354–357.
13. **Cheng K. L., Tsai M. F., Wu C. W.** Efficient neighborhood pattern–sensitive fault test algorithms for semiconductor memories // Proc. IEEE VLSI Test Symp. (VTS), 2001. – P. 225–237.
14. **Caşcaval P., Bennett S., Hutanu C**. Efficient March Tests for a Reduced 3–Coupling and 4–Coupling Faults in Random–Access Memories // J. Electronic Testing: Theory and Applications, 2004. – Vol. 20(3) – P. 227–243.
15. **Cockburn B. F.** Deterministic tests for detecting single V–coupling faults in RAMs // J. Electronic Testing. Theory and Applications, 1994. – Vol. 5(1). – P. 91–113.
16. **Huzum C., Caşcaval P.** Linked Neighborhood Pattern–Sensitive Faults in Random–Access Memories. A Fault Coverage Evaluation // Proc. of 14th ICSTC, 2010. – P. 241–245.

Received 2011 03 10
Accepted after revision 2011 09 15

C. Huzum, P. Cascaval. A Multibackground March Test for Static Neighborhood Pattern-Sensitive Faults in Random-Access Memories // Electronics and Electrical Engineering. – Kaunas: Technologija, 2012. – No. 3(119). – P. 81–86.

A multibackground March test (March-76N) for a model of static neighborhood pattern sensitive faults (NPSFs) in $N \times 1$ random-access memories is presented. March-76N is able to cover both simple and linked NPSFs. As any other test dedicated to the NPSFs, March-76N assumes that the storage cells are arranged in a rectangular grid and the mapping from logical addresses to physical cell locations is known completely. With a length of 76*N*, this March test is more efficient than other published tests dedicated to this model. Ill. 4, bibl. 16, tabl. 4 (in English; abstracts in English and Lithuanian).

C. Huzum, P. Cascaval. Daugiafonis operatyviosios atminties statinių kaimyninių struktūrų klaidų testas // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2012. – Nr. 3(119). – P. 81–86.

Pateikiamas daugiafonis March testas (March-76N), skirtas statinių kaimyninių struktūrų klaidų (SKSK) aptikimo operatyviojoje atmintyje modeliui. March-76N gali aptikti tiek paprastas, tiek susijusias minėto tipo klaidas. Kaip ir kituose testuose, skirtuose SKSK aptikti, March-76N daroma prielaida, kad elementai yra išrikiuoti stačiakampiame tinklelyje ir kreipimasis iš loginių adresų į fizinę elemento vietą yra visiškai žinomas. March testas yra efektyvesnis už kitus skelbtus šiam modeliui skirtus testus. Il. 4, bibl. 16, lent. 4 (anglų kalba; santraukos anglų ir lietuvių k.).

Microelectronics Journal 41 (2010) 212–218

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

March SR3C: A Test for a reduced model of all static simple three-cell coupling faults in random-access memories

Petru Caşcaval [a,*], Doina Caşcaval [b,1]

[a] Department of Computer Engineering, "Gheorghe Asachi" Technical University of Iaşi, Bd. D. Mangeron, 53A, Iaşi, 700050, Romania
[b] Department of Industrial Engineering, "Gheorghe Asachi" Technical University of Iaşi, Bd. D. Mangeron, 53A, Iaşi, 700050, Romania

ARTICLE INFO

Article history:
Received 14 December 2008
Received in revised form
11 February 2010
Accepted 19 February 2010
Available online 9 March 2010

Keywords:
Memory testing
Static fault model
Three-cell coupling fault
Fault primitive
March test

ABSTRACT

A fault primitive-based analysis of all static simple (i.e., not linked) three-cell coupling faults in $n \times 1$ random-access memories (RAMs) is discussed. All realistic static coupling faults that have been shown to exist in real designs are considered: state coupling faults, transition coupling faults, write disturb coupling faults, read destructive coupling faults, deceptive read destructive coupling faults, and incorrect read coupling faults. A new March test with $66n$ operations able to detect all static simple three-cell coupling faults is proposed. To compare this test with other industrial March tests, simulation results are also presented in this paper.

1. Introduction

Semiconductor memories are an integral part of modern ULSI circuits. With each new generation of ULSIs, the memory share of the chip area increases and is expected to be about 94% in 2014 [1]. New memory technologies and processes introduce new defects that significantly impact on the defect-per-million (DPM) level and yield [2]. With increasing densities, certain types of faults, such as multi-cell coupling faults or pattern-sensitive faults, which are harder to detect, are becoming more important. In addition, more time is required to test memories because of their increasing size thus it is necessary to identify more efficient tests, with the ability to detect complex faults, tests that require test time on the order of n.

Many *Functional fault models* (FFMs) for memories have been introduced in the past; some well-known FFMs which date back to before 1999 are: stuck-at fault, decoder fault, data retention fault, stuck open fault, read destructive fault, deceptive read destructive fault, and coupling fault [3]. After 1999, experimental results by applying a large number of tests to a large number of chips indicated that many functional tests do detect faults in memories, which cannot be explained using the well-known set of FFMs [4,5]. This has led to the introduction of new FFMs, based on defect injection and circuit simulation: write disturb fault, incorrect read fault, transition coupling fault, read destructive coupling fault etc. [6]. Taking into consideration all these new FFMs, Hamdioui, van de Goor, and Rodgers [7] have defined a large model of all static simple two-cell coupling faults and have proposed the memory test *March SS* to cover it. Based on this model, in this article we extend the analysis to the more complex model of all static simple three-cell coupling faults.

Many test algorithms dedicated to the model of three-cell coupling faults have been reported. Thus, a memory test that requires $n+32n\log_2 n$ operations is given by Nair, Thatte, and Abraham (*Algorithm B*) [8]. A new test of length $n+24n\log_2 n$ is proposed by Papachristou and Sahgal [9]. Two more efficient test algorithms, *S3CTEST* and *S3CTEST2*, are given by Cockburn [10]. These are tests of approximate length $5n\log_2 n+22.5n$ and $5n\log_2 n+5n[\log_2(1+\log_2 n)]+11n$, respectively. For the memory chips currently available, all these tests take a long time to perform because the authors have assumed that the coupled cells can be anywhere in the memory. For example, to test a 64 Mb memory chip assuming a cycle time of 60 ns, *S3CTEST* takes about 10 min 54 s.

To reduce the length of the test, Caşcaval and Bennett have restricted the model to the more realistic coupling faults that affect only the physically adjacent memory cells [11]. Under this hypothesis and for the cases in which the mapping from logical addresses to physical cell locations is known completely, they have devised a March test of length $38n$, which covers entirely this reduced model of three-cell coupling faults. A new more

* Corresponding author. Tel./fax: +40 232 232430.
E-mail addresses: cascaval@cs.tuiasi.ro, pcascaval@yahoo.com (P. Caşcaval), doina.cascaval@yahoo.com (D. Caşcaval).
[1] Tel./fax: +40 232 230491.

0026-2692/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mejo.2010.02.004

efficient March test with 30*n* operations (*MT-R3CF*) dedicated to a reduced model of three-cell coupling faults is reported by Caşcaval, Bennett, and Huţanu [12].

All these test algorithms assume that a memory fault can be sensitized only by a transition write operation into a cell. In this paper, the model of three-cell coupling faults is extended by considering other classes of faults, such as the faults sensitized by a read or a non-transition write operation, namely: disturb coupling fault, read destructive coupling fault, deceptive read destructive coupling fault, and incorrect read coupling fault. For this large fault model, a new efficient March test is proposed.

The remainder of this paper is organized as follows. Section 2 introduces notations and preliminary considerations, and section 3 defines a reduced model of all static simple three-cell coupling faults. Section 4 presents a new memory test, *March SR3C*, able to cover this fault model. Section 5 contains simulation results and compares the new March test with other published tests. Finally, some conclusions are drawn regarding this work.

2. Notations and memory fault classification

Following the notations and definitions as given in [6], we give first the concept of a fault primitive that will be used to define the targeted FFMs. Second, a classification of memory faults will be given and the scope of the paper will be shown.

2.1. Fault primitive concept

An operation sequence that results in a difference between the observed and the expected memory behavior is called a *sensitizing operation sequence* (*S*). The observed memory behavior that deviates from the expected one is called *faulty behavior* (*F*). In order to specify a certain fault, one has to specify the *S* together with the corresponding faulty behavior *F* and the read result (*R*) of *S*, in case *S* is a read operation or a sequence of operations with a read as last one in the sequence. The combination of *S*, *F* and *R* for a given memory failure is called a *Fault Primitive* (FP).

The following notations are usually used to describe operations on RAMs:

- $r0$ ($r1$) denotes a read 0 (1) operation from a cell;
- r (r^i) denotes a read operation from a cell (cell i) when the expected value is not specified;
- $w0$ ($w1$) denotes a write 0 (1) operation into a cell;
- $0w0$ ($1w1$) denotes a write 0 (1) operation to a cell which contains a 0 (1)—a non-transition write operation when the logical value of the cell does not change;
- w (w^i) denotes a non-transition write operation into a cell (cell i) when the logical value of the cell is not specified;
- $0w1$ ($1w0$) denotes an up (down) transition write operation;
- w_c (w_c^i) denotes a transition write operation into a cell (cell i) when the old logical value of the cell and its complement are not specified;
- $\uparrow$ ($\downarrow$) denotes an up (down) transition due to a certain sensitizing operation.

A FP is a mathematical notation describing a single fault [6] and is usually represented as $<S/F/R>$ where

- S describes the value/operation sensitizing the fault, $S \in \{0, 1, r0, r1, 0w0, 1w1, 0w1, 1w0\}$ (e.g., write 1 to a cell containing 0 (i.e., $0w1$));
- F describes the value of the faulty (*victim*) cell (v-cell), $F \in \{0, 1, \uparrow, \downarrow\}$ (e.g., the cell flips from 0 to 1 (i.e., $\uparrow$));
- R describes the logical value which appears at the output of the memory if the sensitizing operation applied to the v-cell is a read operation or a sequence of operations with a read as last one in the sequence, $R \in \{0, 1, -\}$. The symbol '$-$' in R means that the output data are not applicable; for example, if $S=0w1$, then no data will not appear at the memory output, and therefore R is replaced by a '$-$'; this FP can be written as '$<0w1/\uparrow/->$'.

The concept of FPs allows for establishing a complete framework of all memory faults, since for all allowed operation sequences in the memory, one can derive all possible types of faulty behavior. In addition, the concept of FPs makes it possible to give a precise definition of a FFM as it has to be understood for memory devices [6], namely: a FFM is a non-empty set of fault primitives.

2.2. Classification

Some classifications of FPs can be made based on different and independent factors of *S*.

a) Depending on the number of simultaneous operations required in the *S*, FPs are classified into *single-port* and *multi-port* faults.
   - *Single-ports faults:* These are FPs that require at the most one port in order to sensitize a fault. Note that single-port faults can be sensitized in single-port as well as in multi-port memories.
   - *Multi-port faults:* These are FPs that can only sensitize a fault by performing two or more simultaneous operations via the different ports.

b) Depending on the number of sequential operations required in the *S*, FPs are classified into *static* and *dynamic* faults. Let #O be the number of different operations carried out sequentially in a *S*.
   - *Static faults:* These are FPs which sensitize a fault by performing at the most one operation in the memory (#O=0 or #O=1). For example, the state of the cell is always stuck-at 0 (#O=0), a read operation to a certain cell causes the cell to flip (#O=1), etc.
   - *Dynamic faults:* These are FPs that perform more than one operation sequentially in order to sensitize a fault (#O>1). For example, two successive read operations cause the cell to flip; however, if only one operation is performed, the cell will not flip.

c) Any fault of the FFM can be modeled as a set of distinct FPs simultaneously present in the memory. Depending on the way FPs manifest themselves, they can be divided into *simple faults* and *linked faults*.
   - *Simple faults:* These are faults which cannot be influenced by another fault. That means that the behavior of a simple fault cannot change the behavior of another one; therefore masking cannot occur. For example, a cell j is a victim cell of two aggressor cells i and k. A transition write operation into cell i (i.e., w_c^i) or cell k (i.e., w_c^k) changes the state of cell j from 0 to 1. This is a simple fault that can be modeled by two FPs,

     $FP_1 = <w_c^i/\uparrow/->$, and $FP_2 = <w_c^k/\uparrow/->$.

   - *Linked faults:* These are faults that do influence the behavior of each other. That means that the behavior of a certain fault can change the behavior of another one such that masking can occur. A linked fault consists of two ore more FPs with contrary effects on the same victim cell. For example, take a two-cell coupling fault with cell i the

aggressor cell and cell *j* the *victim* cell. The transition 0*w*1 into cell *i* changes the state of cell *j* from 0 to 1, whereas the transition 1*w*0 into cell *i* changes the state of cell *j* from 1 to 0. This is a linked fault that can be modeled by two FPs,

$FP_1 = <0w1/\uparrow/->$, and $FP_2 = <1w0/\downarrow/->$.

Remark 1. If a test procedure detects all the FPs that define a FFM, it also detects all the simple faults of this model. In other words, the set of simple faults dominates the set of FPs. However, for the linked faults the combined effects of some simple faults may cancel each other out before the victim cell is read again. In this way a linked fault can escape even if the test procedure detects all the simple faults.

In this work, single-port, static, simple faults are considered. From here on, the term 'fault' refers to a single-port, static, simple fault.

RAM faults can also be divided into *single-cell* and *multi-cell* faults. Single-cell faults consist of FPs involving a single cell, while multi-cell faults consists of FPs involving more than one cell. As concerns the multi-cell faults (also called *coupling faults*), we restrict our analysis to the class of three-cell FPs (i.e., three-cell coupling faults).

3. A reduced model of three-cell coupling faults

Based on the notations previously defined, a three-cell FP is presented as $<S/F/R> = <S_{a1};S_{a2};S_v/F/R>_{a1,a2,v}$, where S_{a1}, S_{a2} and S_v are the sensitizing operation (or state) sequences performed on the *a*1-cell and *a*2-cell (*aggressor cells*), and on the *v*-cell (*victim cell*), respectively. The *a*1-cell and *a*2-cell are the cells to which the sensitizing operation (or state) should be applied in order to sensitize the fault, while the *v*-cell is the cell where the fault appears. Note that S_{a1}, $S_v \in \{0, 1, r0, r1, 0w0, 1w1, 0w1, 1w0\}$, whereas $S_{a2} \in \{0, 1\}$. As presented in Table 1, there are 72 FPs compiled into seven FFMs, as defined in [7], namely

a) State coupling faults (*CFst*);
b) Disturb coupling faults (*CFds*);
c) Transition coupling faults (*CFtr*);
d) Write destructive coupling faults (*CFwd*);
e) Read destructive coupling faults (*CFrd*);
f) Deceptive read destructive coupling faults (*CFdrd*);
g) Incorrect read coupling faults (*CFir*).

For example, three cells are said to have a disturb coupling fault if an operation (i.e., read, transition write or non-transition write) performed on the *a*1-cell causes the *v*-cell to flip, when *a*2-cell is in a given state; the FP defined in the row 9 shows that a read 0 operation performed on *a*1-cell causes the *v*-cell to flip from 0 to 1, when the *a*2-cell is in 0 state.

To reduce the length of the test, we assume that only the physically adjacent cells can be three-coupled. For a set of three-coupled cells, six patterns denoted by P_1, P_2, P_3, P_4, P_5 and P_6 are accepted, as shown in Fig. 1.

We call this model of three-cell coupling faults, which comprises only physically adjacent cells, *reduced three-cell coupling*.

In the memory, one or more sets of coupled cells may exist. As in [8] and [9], we assume that the pairs of sets of coupled cells are disjoint.

Remark 2. Because the three-coupled cells are physically adjacent, our model can also be considered a type of neighborhood

Table 1
List of three-cell FPs.

#	Fault primitives	FFM	#	Fault primitives	FFM
1	⟨0; 0; 0; /1/-⟩	*CFst*	37	⟨0; 0; 1*w*0;/1/-⟩	*CFtr*
2	⟨0; 1; 0; /1/-⟩		38	⟨0; 1; 1*w*0;/1/-⟩	
3	⟨1; 0; 0; /1/-⟩		39	⟨1; 0; 1*w*0;/1/-⟩	
4	⟨1; 1; 0; /1/-⟩		40	⟨1; 1; 1*w*0;/1/-⟩	
5	⟨0; 0; 1; /0/-⟩		41	⟨0; 0; 0*w*0; /↑/-⟩	*CFwd*
6	⟨0; 1; 1; /0/-⟩		42	⟨0; 1; 0*w*0; /↑/-⟩	
7	⟨1; 0; 1; /0/-⟩		43	⟨1; 0; 0*w*0; /↑/-⟩	
8	⟨1; 1; 1; /0/-⟩		44	⟨1; 1; 0*w*0; /↑/-⟩	
9	⟨*r*0; 0; 0; /↑/-⟩	*CFds*	45	⟨0; 0; 1*w*1; /↓/-⟩	
10	⟨*r*0; 0; 1; /↓/-⟩		46	⟨0; 1; 1*w*1; /↓/-⟩	
11	⟨*r*0; 1; 0; /↑/-⟩		47	⟨1; 0; 1*w*1; /↓/-⟩	
12	⟨*r*0; 1; 1; /↓/-⟩		48	⟨1; 1; 1*w*1; /↓/-⟩	
13	⟨*r*1; 0; 0; /↑/-⟩		49	⟨0; 0; *r*0; /↑/1⟩	*CFrd*
14	⟨*r*1; 0; 1; /↓/-⟩		50	⟨0; 1; *r*0; /↑/1⟩	
15	⟨*r*1; 1; 0; /↑/-⟩		51	⟨1; 0; *r*0; /↑/1⟩	
16	⟨*r*1; 1; 1; /↓/-⟩		52	⟨1; 1; *r*0; /↑/1⟩	
17	⟨0*w*0; 0; 0; /↑/-⟩		53	⟨0; 0; *r*1; /↓/0⟩	
18	⟨0*w*0; 0; 1; /↓/-⟩		54	⟨0; 1; *r*1; /↓/0⟩	
19	⟨0*w*0; 1; 0; /↑/-⟩		55	⟨1; 0; *r*1; /↓/0⟩	
20	⟨0*w*0; 1; 1; /↓/-⟩		56	⟨1; 1; *r*1; /↓/0⟩	
21	⟨1*w*1; 0; 0; /↑/-⟩		57	⟨0; 0; *r*0; /↑/0⟩	*CFdrd*
22	⟨1*w*1; 0; 1; /↓/-⟩		58	⟨0; 1; *r*0; /↑/0⟩	
23	⟨1*w*1; 1; 0; /↑/-⟩		59	⟨1; 0; *r*0; /↑/0⟩	
24	⟨1*w*1; 1; 1; /↓/-⟩		60	⟨1; 1; *r*0; /↑/0⟩	
25	⟨0*w*1; 0; 0; /↑/-⟩		61	⟨0; 0; *r*1; /↓/1⟩	
26	⟨0*w*1; 0; 1; /↓/-⟩		62	⟨0; 1; *r*1; /↓/1⟩	
27	⟨0*w*1; 1; 0; /↑/-⟩		63	⟨1; 0; *r*1; /↓/1⟩	
28	⟨0*w*1; 1; 1; /↓/-⟩		64	⟨1; 1; *r*1; /↓/1⟩	
29	⟨1*w*0; 0; 0; /↑/-⟩		65	⟨0; 0; *r*0;/0/1⟩	*CFir*
30	⟨1*w*0; 0; 1; /↓/-⟩		66	⟨0; 1; *r*0;/0/1⟩	
31	⟨1*w*0; 1; 0; /↑/-⟩		67	⟨1; 0; *r*0;/0/1⟩	
32	⟨1*w*0; 1; 1; /↓/-⟩		68	⟨1; 1; *r*0;/0/1⟩	
33	⟨0; 0; 0*w*1;/0/-⟩	*CFtr*	69	⟨0; 0; *r*1;/1/0⟩	
34	⟨0; 1; 0*w*1;/0/-⟩		70	⟨0; 0; *r*1;/1/0⟩	
35	⟨1; 0; 0*w*1;/0/-⟩		71	⟨1; 0; *r*1;/1/0⟩	
36	⟨1; 1; 0*w*1;/0/-⟩		72	⟨1; 1; *r*1;/1/0⟩	

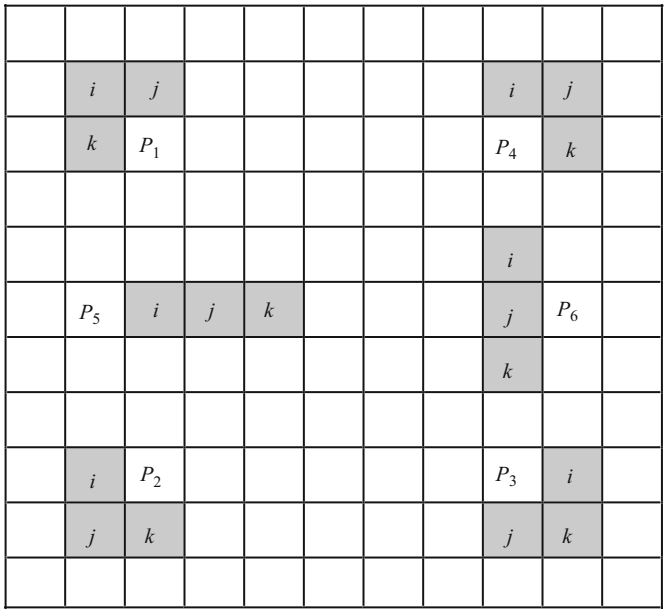


Fig. 1. Patterns for three physically adjacent cells.

pattern-sensitive faults (NPSFs) with two aggressor cells and a victim cell. But in our model any cell in a set of three-coupled cells can be a victim cell, not only the central base cell as is usually considered in the NPSFs model (see for example [13]).

To minimize the silicon area and critical path delay, a RAM's internal address lines are frequently scrambled (called *address scrambling*), so that the physical address is not identical to the logical address. Also, memory cell arrays are invariable repaired by using redundant rows and/or columns, therefore any repair operation changes the mapping from logical addresses to physical cell locations. If we do not have the scramble map, the real neighborhood cannot be identified. As in previous works related to NPSFs, we assume that we have the physical address information (i.e., the scramble map) from the memory designers, so we can run test algorithm using the physical address.

4. The memory test *March SR3C*

The March tests are the most popular and widely accepted deterministic test algorithms because of their low temporal complexity, regular structures and their ability to detect a wide variety of memory faults. Usually, a complete March test is delimited by '{…}' bracket pair, while a March element is delimited by the '(…)' bracket pair. March elements are separated by semicolons, and the operations within a March element are separated by commas. Note that all operations of a March element are performed at a certain address, before proceeding to the next address. The whole memory is checked homogeneously in either one of two orders: ascending address order ($\Uparrow$) or descending address order ($\Downarrow$). When the address order is not relevant, the symbol $\Updownarrow$ is used.

$$\{\Updownarrow(w0)^{(0)};\ \Uparrow(r,r,w,r,\mathbf{w_c})^{(1)};\Uparrow(r,r,w,r,\mathbf{w_c})^{(2)};\Downarrow(r,r,w,r,\mathbf{w_c})^{(3)};\Downarrow(r,r,w,r,\mathbf{w_c})^{(4)};\ I_1{}^{(5)};\Uparrow(r,r,w,r,\mathbf{w_c},r,r,w,r,\mathbf{w_c})^{(6)};I_2{}^{(7)};\Uparrow(r,r,w,r,\mathbf{w_c},r,r,w,r,\mathbf{w_c})^{(8)};\ I_3{}^{(9)};\Uparrow(r,r,w,r,\mathbf{w_c},r,r,w,r,\mathbf{w_c})^{(10)};I_4{}^{(11)};\Uparrow(r,r,w,r,\mathbf{w_c},r,r,w,r,\mathbf{w_c})^{(12)};\ \Updownarrow(r0)^{(13)}\}$$

Fig. 2. Memory test *March SR3C*.

A new March test, *March SR3C*, dedicated to the reduced model of three-cell coupling faults, is presented in Fig. 2, where I_1, I_2, I_3, and I_4 are sequences which initialize the memory as follows: I_1 initializes the odd columns with 0 and the even columns with 1, and I_3 vice versa (*column-stripe data background*); I_2 and I_4 initialize the memory with a *checkerboard data background* and its complement, as illustrated in Fig. 3.

This March test contains fourteen sequences as identified with a superscript (x), where $x\in\{0, 1, \ldots, 13\}$. The test sequences (5)–(12) form an alternating series of background changes and March elements (as Cockburn proposed in [10]). Note that when changing from one background to the next, only the cells that must change states are written. Also, each write operation is preceded by a read operation. We can observe in Fig. 3 that any background change affects only half of the cells. Each test sequence I_1, I_2, I_3 or I_4 performs $n/2$ read operations and $n/2$ write operations. Consequently, *March SR3C* has a length of $66n$. Regarding this March test, symbol $\Uparrow$ denotes an increasing address order, from address 0 to $n-1$, as long as symbol $\Downarrow$ denotes a reverse address order, from address n-1 to 0. Other permutations of the set of memory cell addresses decrease the effectiveness of the March test.

Theorem 1. *March SR3C detects all FPs of this reduced three-cell coupling.*

Proof. To test and find a fault in a memory we need to be able to sensitize the fault by a proper memory operation, and to observe the fault by reading the changed value of the cell affected by the fault.

Proposition 1. The next three conditions are necessary and sufficient for a test to detect any FP that affects a cell in a set S of coupled cells:

Condition 1. The test must perform all the possible cell operations in the set of coupled cells in order to sensitize any fault.

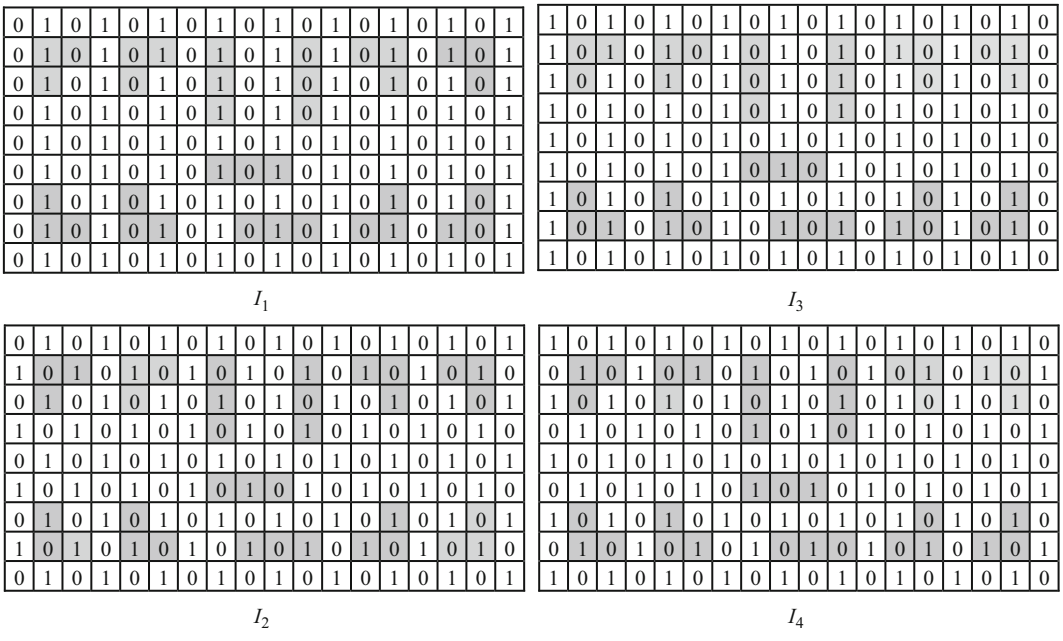


Fig. 3. Memory initialization (data background) for the memory test *March SR3C*.

Condition 2. After any operation into a cell, the test must read the cell to check its state, before another write operation into the cell is allowed to occur.

Condition 3. For each possible victim cell c in S, after one or more operations in other cells in the set, the test must read cell c, prior to a write operation into cell c, to check if the state has been changed by a memory operation in other possible aggressor cell.

Consider an arbitrary triple set of cells $S=\{i, j, k\}$ that corresponds with one of the patterns P_1, P_2, P_3, P_4, P_5 and P_6 as illustrated in Fig. 1. We refer to the cells in S by i, j and k taking into account the order in which these cells are checked during the memory testing. Cells i, j and k are checked in this order when the memory is tested in ascending order ($\Uparrow$), and in reverse order, when the memory is tested in descending order ($\Downarrow$). We must show that *March SR3C* is able to sensitize and to observe any FP that affects the set of cells S.

4.1. March SR3C sensitizes any fault in the set of cells S

According to Condition 1 previously defined in Proposition 1, we must prove that *March SR3C* performs all the possible cell operations in the set of coupled cells S. The proof comprises two steps: in the first step we prove that *March SR3C* performs all possible transition write operations and, based on this result, in the second one we prove that *March SR3C* also performs all possible read and non-transition write operations.

Step 1. Take all the transition writes of *March SR3C* denoted by w_c and marked with bold face in Fig. 2. By applying these operations, *March SR3C* covers the Eulerian graph of states for a set of cells S, regardless of the pattern. Indeed, the initialize sequence $\Updownarrow(w0)$ loads into any triple set of cells the initial state $<0,0,0>$. Applying the March elements (1)–(4), *March SR3C* forces the transitions marked with solid lines in the Eulerian graph presented in Fig. 4, in every set of cells in the memory under test, regardless of the pattern.

On the other hand, the test sequences I_1, I_2, I_3 and I_4 ensure in a set of cells S, depending on the pattern (see Figs. 1 and 3), the initial states presented in Table 2.

As highlighted in Table 2, the test sequences ensure the initial states $<0,1,0>$ and $<1,0,1>$ for all the six patterns. In the Eulerian graph, two adjacent nodes (states) have only one bit

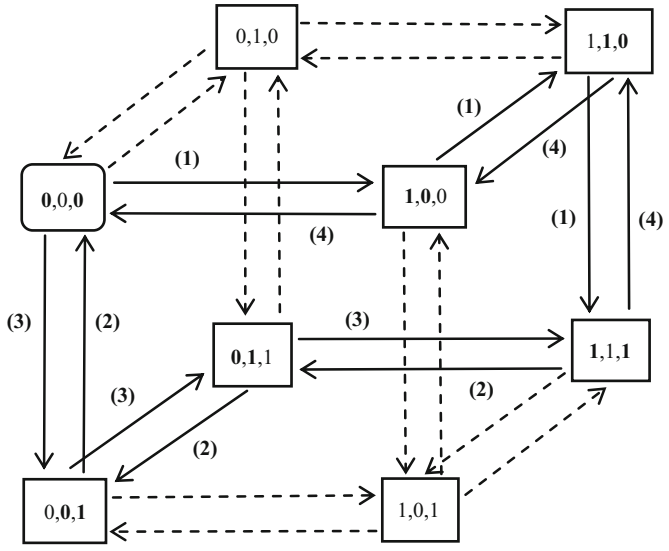


Fig. 4. The Eulerian graph of states for a set of cells $S=\{i, j, k\}$ and the transitions carried out by the March elements (1)–(4).

Table 2
The initial states for a set of cells $S=\{i, j, k\}$.

	I_1 and I_3	I_2 and I_4
P_1	⟨**1,0,1**⟩, ⟨**0,1,0**⟩	⟨0,1,1⟩, ⟨1,0,0⟩
P_2	⟨1,1,0⟩, ⟨0,0,1⟩	⟨**1,0,1**⟩, ⟨**0,1,0**⟩
P_3	⟨**1,0,1**⟩, ⟨**0,1,0**⟩	⟨1,1,0⟩, ⟨0,0,1⟩
P_4	⟨0,1,1⟩, ⟨1,0,0⟩	⟨**1,0,1**⟩, ⟨**0,1,0**⟩
P_5	⟨**1,0,1**⟩, ⟨**0,1,0**⟩	⟨0,1,0⟩, ⟨1,0,1⟩
P_6	⟨0,0,0⟩, ⟨1,1,1⟩	⟨**1,0,1**⟩, ⟨**0,1,0**⟩

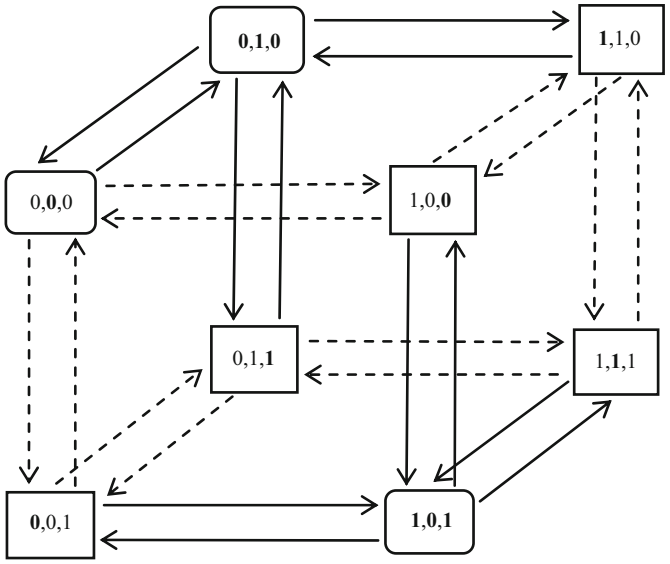


Fig. 5. The transitions carried out in a set of cells $S=\{i, j, k\}$ by the March element $\Uparrow(r, r, w, r, w_c, r, r, w, r, w_c)$ for the initial states $<0,1,0>$ and $<1,0,1>$.

changed and two non-adjacent nodes have at least two bits changed. Consequently, by applying the March element $\Uparrow(r, r, w, r, w_c, r, r, w, r, w_c)$, three adjacent nodes are visited and, finally, the set of cells returns to the initial state (the state that the sequence started with). The transitions forced by this March element in a set of cells S, initialized with $<0,1,0>$ and $<1,0,1>$ respectively, are highlighted (marked with solid line) in Fig. 5.

The transitions marked with solid lines in Figs. 4 and 5 show that the Eulerian graph of states is completely covered in all the six cases.

As follows, we will show that *March SR3C* also performs all possible read operations and non-transition write operations in the set of cells S.

Step 2. For both March elements used by *March SR3C*, (r, r, w, r, w_c) and $(r, r, w, r, w_c, r, r, w, r, w_c)$, every transition write operation (w_c) is preceded by three reads and one non-transition write operation. Thus, before changing the state of a cell, *March SR3C* checks if the read operation and the non-transition write operation applied to this cell are performed properly. For every state in a set of cells S, Table 3 presents the March elements in which the read and the non-transition write operations applied to each cell of S are carried out.

4.2. March SR3C observes any FP activated in the set of cells S

Table 4 presents the operations carried out in a set of cells $S=\{i, j, k\}$ during the memory testing, except on the writes for the first initialization. The cell in which the operation is carried out is identified with a superscript x, where $x\in\{i, j, k\}$. The operations enclosed inside brackets sometimes are made, or sometimes are

Table 3
The March elements in which the read and the non-transition write operations are carried out in S.

State of $S=\{i, j, k\}$	Read and non-transition write operations		
	r^i, w^i	r^j, w^j	r^k, w^k
$\langle 0, 0, 0\rangle$	(1)	(6) or (10)	(3)
$\langle 0, 0, 1\rangle$	(8) or (12)	(3)	(2)
$\langle 0, 1, 0\rangle$	(6) or (10)	(6) or (10)	(6) or (10)
$\langle 0, 1, 1\rangle$	(3)	(2)	(6) or (10)
$\langle 1, 0, 0\rangle$	(4)	(1)	(8) or (12)
$\langle 1, 0, 1\rangle$	(8) or (12)	(8) or (12)	(8) or (12)
$\langle 1, 1, 0\rangle$	(6) or (10)	(4)	(1)
$\langle 1, 1, 1\rangle$	(2)	(8) or (12)	(4)

Table 4
The operations carried out in a set of cells $S=\{i, j, k\}$ by the memory test *March SR3C*.

Operations	Comments
$\dots r^i r^i w^i r^i w_c^i \dots r^j r^j w^j r^j w_c^j \dots r^k r^k w^k r^k w_c^k \dots$	March elements (1) and (2)
$\dots r^i r^i w^i r^i w_c^i \dots r^j r^j w^j r^j w_c^j \dots r^k r^k w^k r^k w_c^k \dots$	
$\dots r^k r^k w^k r^k w_c^k \dots r^j r^j w^j r^j w_c^j \dots r^i r^i w^i r^i w_c^i \dots$	March elements (3) and (4)
$\dots r^k r^k w^k r^k w_c^k \dots r^j r^j w^j r^j w_c^j \dots r^i r^i w^i r^i w_c^i \dots$	
$\dots [r^i w_c^i] \dots [r^j w_c^j] \dots [r^k w_c^k] \dots$	Change to 2nd background (I_1)
$\dots r^i r^i w^i r^i w_c^i r^i r^i w^i r^i w_c^i \dots r^j r^j w^j r^j w_c^j r^j r^j w^j r^j w_c^j \dots$	March element (6)
$\dots r^k r^k w^k r^k w_c^k r^k r^k w^k r^k w_c^k \dots$	
$\dots [r^i w_c^i] \dots [r^j w_c^j] \dots [r^k w_c^k] \dots$	Change to 3rd background(I_2)
$\dots r^i r^i w^i r^i w_c^i r^i r^i w^i r^i w_c^i \dots r^j r^j w^j r^j w_c^j r^j r^j w^j r^j w_c^j \dots$	March element (8)
$\dots r^k r^k w^k r^k w_c^k r^k r^k w^k r^k w_c^k \dots$	
$\dots [r^i w_c^i] \dots [r^j w_c^j] \dots [r^k w_c^k] \dots$	Change to 4th background (I_3)
$\dots r^i r^i w^i r^i w_c^i r^i r^i w^i r^i w_c^i \dots r^j r^j w^j r^j w_c^j r^j r^j w^j r^j w_c^j \dots$	March element (10)
$\dots r^k r^k w^k r^k w_c^k r^k r^k w^k r^k w_c^k \dots$	
$\dots [r^i w_c^i] \dots [r^j w_c^j] \dots [r^k w_c^k] \dots$	Change to 5th background (I_4)
$\dots r^i r^i w^i r^i w_c^i r^i r^i w^i r^i w_c^i \dots r^j r^j w^j r^j w_c^j r^j r^j w^j r^j w_c^j \dots$	March element (12)
$\dots r^k r^k w^k r^k w_c^k r^k r^k w^k r^k w_c^k \dots$	
$\dots r^i \dots r^j \dots r^k \dots$	Final read sequence

not made, depending on the set of cells. We can easily check in Table 4 that Conditions 2 and 3, previously defined in Proposition 1, are also satisfied.

Remark 3. For the patterns P_1, P_3 and P_5, the test sequences (7), (8), (11) and (12) are not necessary (so they can be removed), whereas for the patterns P_2, P_4 and P_6, the sequences (5), (6), (9) and (10) are not necessary (see Fig. 3 and Table 2).

Let us consider the set of the patterns $\{P_1, P_2, P_3, P_4, P_5, P_6\}$ divided into two parts (to say halves), $H_1=\{P_1, P_3, P_5\}$ and $H_2=\{P_2, P_4, P_6\}$. Based on the proof of Theorem 1, we derive a lower bound on the length of any test that covers half of patterns, H_1 or H_2, of reduced three-cell coupling. For the six patterns, the minimum required length of any test that detects all simple faults remains still to be found out.

Corollary. *A memory test needs at least 42n operations to detect all simple faults for half of patterns, H_1 or H_2, of reduced three-cell coupling.*

Proof. Take the March elements (1)–(4), (6) and (10) or (1)–(4), (8) and (12), by case. Because in every set of cells $S=\{i, j, k\}$ all 24 transitions are carried out exactly once, these March elements perform a minimal number of transition write operations. In order to sensitizing all the faults, these March elements also perform a minimal number of reads and non-transition write operations. On the other hand, all reads of these March elements used for observing the faults, as well as the reads of the final sequence, are necessary to satisfy Conditions 2 and 3, previously defined in Proposition 1. Consequently, for half of patterns, H_1 or H_2, a memory test needs at least $42n$ operations (including n operations for memory initialization) to detect all simple three-cell coupling faults.

Table 5
BDS for 8-bit word.

#	Sequence
0	00000000
1	11111111
2	01010101
3	10101010
4	00110011
5	11001100
6	00001111
7	11110000

Remark 4. With $43.5n$ operations, a March test composed of the sequences (0)–(6), (9), (10) and (13) is a near-optimal memory test for the patterns P_1, P_3 and P_5, whereas the March test composed of the sequences (0)–(4), (7), (8), (11)–(13) is a near-optimal one for the patterns P_2, P_4 and P_6. Nevertheless, *March SR3C* is not an optimal test for the model we have considered, so such a test remains still to be worked out.

To cover this reduced model of three-cell coupling, *March SR3C* uses different data backgrounds as presented in Fig. 3. To generate data backgrounds, only the LSB of the row and column addresses are necessary. For example, the I_2 data can be represented as $A_r[0]\oplus A_c[0]$, where $A_r[0]$ is the LSB of the r-bit row address $A_r[r\text{-}1\dots0]$, and $A_c[0]$ is the LSB of the c-bit column address $A_c[c\text{-}1\dots0]$. Consequently, generation of all data backgrounds can be done by a simple circuit. Any typical March-based built-in self-test (BIST) circuit can easily be modified to run the proposed test algorithm.

To make the test even shorter than $66n$ operations, may be data backgrounds based on 3×3 cell patterns should be used, instead of data backgrounds based on 2×2 cell patterns (see Fig. 3). But, such a March test is not so suitable for BIST implementation; if implemented in hardware, the data generator for such a test would be more complicated than that for *March SR3C*.

Modern RAMs are mostly word oriented, i.e., they are accessed word-by-word instead of bit-by-bit. In general, each bit from a different block contributes to a word. In [14], van de Goor and Tlili describe a method for efficiently converting a bit-oriented memory March test to word-oriented memory (WOM) test. WOM March tests can detect inter-word faults (i.e., faults among words) and intra-word faults (i.e., faults within words). In order to sensitize intra-word coupling faults a sequence of bits that is called a background data sequence (BDS) can be used. Table 5 shows a BDS for 8-bit word. This sequence can be applied to all words in an address location. As concerns the inter-word faults, testing such a word-oriented memory can be considered as testing m smaller bit-oriented memories in parallel, where m is the number of blocks. As any March test, the proposed algorithm can be easily converted from a bit-oriented test to a word-oriented memory test.

5. Simulation results

To compare *March SR3C* with other published tests, we present in this section simulation results regarding the ability of the tests to detect simple faults of this model. The following March tests have been considered for the simulation study:

- MT-R3CF [12]: $\Updownarrow(w0)$; $\Uparrow(r0, w1)$; $\Uparrow(r1, w0)$; $\Downarrow(r0, w1)$; $\Downarrow(r1, w0)$; I_1; $\Uparrow(r, w_c, r, w_c)$; I_2; $\Uparrow(r, w_c, r, w_c)$; I_3; $\Uparrow(r, w_c, r, w_c)$;

Table 6
Fault coverage of three-cell coupling faults.

#	Memory test	Test length	Fault coverage (%)
1	March SR3C	66*n*	100
2	MT_R3CF	30*n*	56.73
3	Algorithm A	30*n*	46.30
4	March G	24*n*	44.91
5	March SS	22*n*	53.94
6	March LA	22*n*	42.82
7	March B	17*n*	32.18
8	March LR	14*n*	43.52
9	March SR	14*n*	42.36
10	March U	13*n*	42.59
11	PMOVI	13*n*	39.58
12	March C-	10*n*	36.81

I_4; $\Uparrow(r, w_c, r, w_c)$; $\Updownarrow(r)$}, where I_1, I_2, I_3 and I_4 are sequences that initialize the memory as presented in Fig. 3.

- Algorithm A [8]: {$\Uparrow(w0)$; $\Uparrow(r0,w1)$; $\Uparrow(r1)$; $\Uparrow(r1, w0)$; $\Uparrow(r0)$; $\Downarrow(r0,w1)$; $\Downarrow(r1)$; $\Downarrow(r1,w0)$; $\Downarrow(r0)$; $\Uparrow(r0,w1,w0)$; $\Uparrow(r0)$; $\Downarrow(r0, w1,w0)$; $\Downarrow(r0)$; $\Uparrow(w1)$; $\Uparrow(r1, w0, w1)$; $\Uparrow(r1)$; $\Downarrow(r1, w0, w1)$; $\Downarrow(r1)$ }.
- March G [15]: {$\Updownarrow(w0)$; $\Uparrow(r0, w1, r1, w0, w1)$; $\Uparrow(r1, w0, r0, w1)$; $\Downarrow(r1, w0, w1, w0)$; $\Downarrow(r0, w1, r1, w0)$; $\Uparrow(r0, w1, r1)$; $\Downarrow(r1, w0, r0)$}.
- March SS [7]: {$\Updownarrow(w0)$; $\Uparrow(r0, r0, w0, r0, w1)$; $\Uparrow(r1, r1, w1, r1, w0)$; $\Downarrow(r0, r0, w0, r0, w1)$; $\Downarrow(r1, r1, w1, r1, w0)$; $\Updownarrow(r0)$; }.
- March LA [16] : {$\Updownarrow(w0)$; $\Uparrow(r0, w1, w0, w1, r1)$; $\Uparrow(r1, w0, w1, w0, r0)$; $\Downarrow(r0, w1, w0, w1, r1)$; $\Downarrow(r1, w0, w1, w0, r0)$; $\Downarrow(r0)$}.
- March B [17]: {$\Updownarrow(w0)$; $\Uparrow(r0, w1, r1, w0, r0, w1)$; $\Uparrow(r1, w0, w1)$; $\Downarrow(r1, w0, w1, w0)$; $\Downarrow(r0, w1, w0)$}
- March LR [18]: {$\Updownarrow(w0)$; $\Downarrow(r0, w1)$; $\Uparrow(r1, w0, r0, w1)$; $\Uparrow(r1, w0)$; $\Uparrow(r0, w1, r1, w0)$; $\Uparrow(r\ 0)$}.
- March SR [4]: {$\Downarrow(w0)$; $\Uparrow(r0, w1, r1, w0)$; $\Uparrow(r0, r0)$; $\Uparrow(w1)$; $\Downarrow(r1, w0, r0, w1)$; $\Downarrow(r1, r1)$}.
- March U [19]: {$\Updownarrow(w0)$;$\Uparrow(r0, w1, r1, w0)$; $\Uparrow(r0, w1)$; $\Downarrow(r1, w0, r0, w1)$; $\Downarrow(r1, w0)$}.
- PMOVI [20]: {$\Downarrow(w0)$; $\Uparrow(r0, w1, r1)$; $\Uparrow(r1, w0, r0)$; $\Downarrow(r0, w1, r1)$; $\Downarrow(r1, w0, r0)$ }.
- March C [21,15]: {$\Updownarrow(w0)$;$\Uparrow(r0, w1)$; $\Uparrow(r1, w0)$; $\Downarrow(r0, w1)$; $\Downarrow(r1, w0)$; $\Updownarrow(r0)$}.

All the six patterns illustrated in Fig. 1 and all FPs presented in Table 1 have been considered in this simulation study. Moreover, for each group of coupled cells, composed of two aggressor cells (a_1-cell and a_2-cell) and a victim cell (v-cell), six distinct permutations on the set {a_1, a_2, v} have been considered. Consequently, 2952 FPs ($6 \times 6 \times 72$) have been simulated in order to evaluate the fault coverage of the memory tests. Table 6 summarizes the fault coverage of all three-cell coupling faults in our model.

As shown in Table 6, *March SR3C* is able to detect all three-cell coupling faults.

6. Conclusions

A reduced model of all static simple three-cell coupling faults that includes state coupling fault, transition coupling fault, write disturb coupling fault, read destructive coupling fault, deceptive read destructive coupling fault, and incorrect read coupling fault is discussed, and a new memory test of length 66*n* able to cover it is proposed. To reduce the length of the test, only the coupling faults between physically adjacent memory cells have been considered. The test assumes that the storage cells are arranged in a rectangular grid and that the mapping from logical addresses to physical cell locations is known completely.

Simulation results demonstrate the effectiveness of this March test when compared with other published tests. To cover this reduced model of three-cell coupling, the March test uses different data backgrounds: a solid, column-stripes and checkerboards. These data backgrounds can be generated by using the LSB of the row and column addresses. Consequently, generation of data backgrounds can be done by a simple circuit. Any typical March-based BIST implementation can easily be modified to run the proposed test algorithm.

References

[1] A. Allan, et al., 2001 The international technology roadmap for semiconductors, Computers 35 (1) (2002) 42–53.
[2] S. Hamdioui, Z. Al-Ars, J. Jimenez, J. Calero, PPM Reduction on Embedded Memories in System on Chip, in: IEEE Proceedings of the European Test Symposium, Freiburg, Germany, May 2007, pp. 85–90.
[3] A.J. van de Goor, Testing Semiconductors Memories. Theory and Practice, ComTex Publishing, Gouda, The Netherlands, 1998.
[4] S. Hamdioui and A.J. van de Goor, Experimental Analysis of Spot Defects in SRAM. Realistic Fault Models and Tests, in: Proceeding of the Ninth Asian Test Symposium, Taipei, Taiwan, December 2000, pp. 131–138.
[5] V.K. Kim, T. Chen, On comparing functional fault coverage and defect coverage for memory testing, IEEE Trans. CAD 18 (11) (2000) 1676–1683.
[6] A.J. van de Goor, Z. Al-Ars, Functional Faults Models: A Formal Notation and Taxonomy, in: Proceeding of the 18th IEEE VLSI Test Symposium, Montreal, Canada, May 2000, pp. 281–289.
[7] S. Hamdioui, A.J. van de Goor, M. Rodgers, March SS: A Test for All Static Simple RAM Faults, in: Proceeding of the IEEE International Workshop on Memory Technology, Design and Testing, Isle of Bendor, France, July 2002, pp. 95–100.
[8] R. Nair, S. Thatte, J. Abraham, Efficient algorithms for testing semiconductor random access memories, IEEE Trans. Comput. C-27 (6) (1978) 572–576.
[9] C. Papachristou, N. Sahgal, An improved method for detecting functional faults in semiconductor random access memories, IEEE Trans. Comput. C-34 (2) (1985) 110–116.
[10] B.F. Cockburn, Deterministic tests for detecting single V-coupling faults in RAMs, J. Electron. Testing Theory Appl. 5 (1) (1994) 91–113.
[11] P. Caşcaval, S. Bennett, Efficient march test for 3-coupling faults in random access memories, Microprocessors and Microsystems 24 (10) (2001) 501–509.
[12] P. Caşcaval, S. Bennett, C. Huţanu, Efficient march tests for a reduced 3-coupling and 4-coupling faults in RAMs, J. Electron. Testing. Theory Appl. 20 (2) (2004) 227–243.
[13] K.-L. Cheng, M.-F. Tsai, C.-W. Wu, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 21 (11) (2002) 1328–1336.
[14] A. J. van de Goor, I. B. S Tlili, March Tests for Word-Oriented Memories, in: Proceeding Design, Automation and Test in Europe, Paris, France, February 1998, pp. 501–508.
[15] A.J. van de Goor, Using march tests to test SRAMs, IEEE Des. Test Comput. 10 (1) (1993) 8–14.
[16] A.J. van de Goor, et al., March LA: A Test for Linked Memory Faults, in: Proceedings of the European Design Test Conference, Paris, France, March 1999, pp. 627–634.
[17] D. Suk, S. Reddy, A march test for functional faults in semiconductor random access memories, IEEE Trans. Comput. C-30 (12) (1981) 982–985.
[18] V.N. Yarmolik, A.J. van de Goor, G.N. Gaydadjiev, V.G. Mikitjuk, March LR: A Test for Realistic Linked Faults, in: Proceeding of VLSI Test Symposium, Princeton, USA, March 1996, pp. 272–280.
[19] A.J. van de Goor, G.N. Gaydadjiev, March U: A Test for all unlinked memory faults, IEE Proc. Circuits Devices Syst. 144 (3) (1997) 155–160.
[20] J.H. de Jonge, A.J. Smeulders, Moving inversion test pattern is thorough, yet speedy, Comput. Syst. Des., ACM 19 (5) (1976) 169–173.
[21] M. Marinescu, Simple and Efficient Algorithms for Functional RAM Testing, Digest of Papers, 1982 International Test Conference, Philadelphia, USA, November 1982, pp. 236–239.

ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **12**, Number 4, 2009, 440–454

A BIST Logic Design for $MarchS_3C$ Memory Test BIST Implementation

[1]Petru CAŞCAVAL, [2]Radu SILION, [3]Doina CAŞCAVAL

Gheorghe Asachi Technical University of Iaşi, Romania
E-mail: {[1]cascaval, [2]rsilion}cs.tuiasi.ro,
[3]cascaval@tex.tuiasi.ro

Abstract. A logic design for a possible built-in self-testing implementation of a march test able to detect all static simple three-cell coupling faults in $n \times 1$ random-access memories (RAMs) is presented. Single–array single bit and multiple–array single bit test architectures have been considered. The memory test ($MarchS_3C$ [1]) needs $66n$ operations and is able to detect all realistic simple (i.e. not linked) static three-cell coupling faults that have been shown to exist in real designs, namely: state coupling faults, transition coupling faults, write disturb coupling faults, read destructive coupling faults, deceptive read destructive coupling faults, and incorrect read coupling faults. To reduce the length of the test, only the coupling faults between physically adjacent memory cells have been considered. The test assumes that the storage cells are arranged in a rectangular grid and that the mapping from logical addresses to physical cell locations is known completely.

Key words: Memory testing, static faults, fault primitive, three-cell coupling faults, built-in self-testing.

1. Introduction

This article focuses on high performance memory testing having in view system on chip (SoC) dedicated to critical applications, with high reliability and safety requirements, in which the test confidence degree regarding to normal operation of the embedded memory must be very high. In recent years, embedded memories are the fastest growing segment of SoC. According to the 2001 International Technology Roadmap for Semiconductor [2], today's SoC are moving from logic dominant chips

to memory dominant chips, since future applications require a lot of memory. The same idea is reiterated in [3]. The memory shared on the chip is expected to be about 94% in 2014. For this reason, the importance of memory testing increases.

Rapid developments in semiconductor technology have resulted in continuing growth of larger and denser random-access memories on a single chip. With increasing densities, certain types of faults harder-to-detect such as three-cell or even four-cell coupling faults can not be ignored any more [4, 5]. Consequently, the test algorithms are constrained by two conflicting requirements: to cover a wide variety of memory faults harder-to-detect, and to reduce the number of memory operations in order to allow large memories to be tested in an acceptable period of time. In addition, more time is required to test memories because of their increasing size thus it is necessary to identify more efficient tests, with the ability to detect complex faults, tests that require test time on the order of n, where n denotes the number of locations of the memory. In this work, the class of faults harder-to-detect that includes coupling faults with three adjacent coupled cells has been considered.

BIST is a design-for-testability technique that places test functions physically on chip with the circuit under test. System designers use BIST for periodic testing. This requires incorporating a test process that guarantees the detection of all target faults within a fixed time. Designers also implement on-line BIST with the goals of large fault coverage and low fault latency. For critical or highly available systems, a comprehensive online-testing approach that covers all expected permanent, intermittent, and transient faults is essential [6].

Two kinds of testing methods exist: random testing and deterministic testing. Random testing is based on linear-feedback shift registers (LFSR) for pattern generation. An LSFR can also serve as a response monitor by counting the responses produced by the tests. After receiving a sequence of test responses, an LSFR response monitor forms a fault signature which is compared with a known good signature to determine whether a fault is present. Deterministic testing is especially suited to highly regular chips.

Since the RAM circuit has a regular structure, a deterministic testing is more adequate than a random one. Taking into account the number of simultaneously tested arrays and the number of simultaneously accessed bits within an array, Franklin and Saluja [6] classified all the RAM-BIST test architectures into one of the four test architectures: single-array single bit, single-array multiple bit, multiple-array single bit, and multiple-array multiple bit. In this work only single-array single bit (SASB) test architectures and multiple-array single bit test architectures (MASB) have been considered. SASB test architectures are those in which a single array of the RAM chip is tested at a time and a single bit of the tested array is accessed at a time. MASB test architectures can be used if a memory chip is organized as a number of independent arrays, allowing multiple arrays to be tested simultaneously. Ensuring that fault coverage is sufficiently high and the number of tests is sufficiently low are the main problems with a BIST implementation [5].

In this paper we focus on the model of all static simple three–cell coupling faults, as defined in [1]. This is the largest model of three-cell coupling that includes all the faults that have been shown to exist in real designs, namely: state coupling faults,

transition coupling faults, write disturb coupling faults, read destructive coupling faults, deceptive read destructive coupling faults, and incorrect read coupling faults [4, 7]. For a BIST-RAM logic design, the memory test $MarchS_3C$ that covers entirely this fault model has been considered [1].

The remainder of this paper is organized as follows. Section 2 presents a memory fault classification based on the concept of fault primitives. The fault primitives for the model of all static three-cell coupling are presented in Section 3. The memory test $MarchS_3C$ is presented briefly in Section 4. To compare $MarchS_3C$ with other published memory tests, simulation results are also presented in Section 5. A logic design for a possible implementation of this march test in a BIST-RAM device is discussed in Section 6. Final remarks regarding this work are presented in Section 7.

2. Primitives and a Memory Fault Classification

An operation sequence that results in a difference between the observed and the expected memory behaviour is called a *sensitizing operation sequence* (S). The observed memory behaviour that deviates from the expected one is called *faulty behaviour* (F). In order to specify a certain fault, one has to specify the S, together with the corresponding faulty behaviour F, and the read result (R) of S in case it is a read operation. The combination of S, F and R for a given memory failure is called a *Fault Primitive* (FP). The concept of FPs allows for establishing a complete framework of all memory faults. Some classifications of FPs can be made based on different and independent factors of S.

a) Depending on the number of simultaneous operations required in the S, FPs are classified into *single-port* and *multi-port* faults.

- *Single-ports faults:* These are FPs that require at the most one port in order to sensitize a fault. Note that single-port faults can be sensitized in single-port as well as in multi-port memories.
- *Multi-port faults:* These are FPs that can only sensitize a fault by performing two or more simultaneous operations via the different ports.

b) Depending on the number of sequential operations required in the S, FPs are classified into *static* and *dynamic* faults. Let #O be the number of different operations carried out sequentially in a S.

- *Static faults:* These are FPs which sensitize a fault by performing at the most one operation in the memory (#O=0 or #O=1);
- *Dynamic faults:* These are FPs that perform more than one operation sequentially in order to sensitize a fault (#O > 1).

c) Depending on the way FPs manifest themselves, they can be divided into *simple faults* and *linked faults*.

- *Simple faults:* These are faults which cannot be influenced by another fault. That means that the behaviour of a simple fault cannot change the behaviour of another one; therefore masking cannot occur.
- *Linked faults:* These are faults that do influence the behaviour of each other. That means that the behaviour of a certain fault can change the behaviour of another one such that masking can occur. Note that linked faults consists of two ore more simple faults.

In this work, single-port, static, simple faults are considered. From here on, the term 'fault' refers to a single-port, static, simple fault.

The following notations are usually used to describe operations on RAMs :

- $r0$ ($r1$) denotes a read 0 (1) operation from a cell;
- r denotes a read operation from a cell when the expected value is not specified;
- $w0(w1)$ denotes a write 0 (1) operation into a cell;
- $0w0$ ($1w1$) denotes a write 0 (1) operation to a cell which contains a 0(1) – a non-transition write operation when the logical value of the cell does not change;
- w denotes a non-transition write operation into a cell when the logical value of the cell is not specified;
- $0w1$ ($1w0$) denotes an up (down) transition write operation;
- w_c denotes a transition write operation into a cell when the old logical value of the cell and its complement are not specified;
- $\uparrow$ ($\downarrow$) denotes an up (down) transition due to a certain sensitizing operation.

A FP is usually denoted as $< S/F/R >$ [6], where:

- S describes the value/operation sensitizing the fault, $S \in \{0,\ 1,\ r0,\ r1,\ 0w0,\ 1w1,\ 0w1,\ 1w0\}$;
- F describes the value of the faulty (*victim*) cell (v-cell), $F \in \{0,\ 1,\ \uparrow,\ \downarrow\}$;
- R describes the logical value which appears at the output of the memory if the sensitizing operation applied to the v-cell is a read operation, $R \in \{0,\ 1,\ -\}$. The symbol '$-$' in R means that the output data is not applicable; for example, if S=$0w0$, then not data will appear at the memory output, and therefore R is replaced by a '$-$'.

RAM faults can also be divided into *single-cell* and *multi-cell* faults. Single-cell faults consist of FPs involving a single cell, while multi-cell faults consists of FPs involving more than one cell. As concerns the multi-cell faults (also called *coupling faults*), we restrict our analysis to the class of three-cell FPs (i.e. three-cell coupling faults).

3. All Static Three-Cell Coupling Faults

Based on the notations previously defined, a three-cell FP is presented as $< S/F/R >=< S_{a1}\,;\, S_{a2}\,;S_v/F/R >_{a1,\,a2,\,v}$, where S_{a1}, S_{a2} and S_v are the sensitizing operation (or state) sequences performed on the $a1$-cell and $a2$-cell (*aggressor cells*), and on the v-cell (*victim cell*), respectively. The $a1$-cell and $a2$-cell are the cells to which the sensitizing operation (or state) should be applied in order to sensitize the fault, while the v-cell is the cell where the fault appears. Note that S_{a1}, $S_v \in \{0,\ 1,\ r0,\ r1,\ 0w0,\ 1w1,\ 0w1,\ 1w0\}$, whereas $S_{a2} \in \{0,\ 1\}$. As presented in Table 1, there are 72 FPs compiled into seven FFMs, as defined in [7], namely:

- state coupling faults (*CFst*);
- disturb coupling faults (*CFds*);
- transition coupling faults (*CFtr*);
- write destructive coupling faults (*CFwd*);
- read destructive coupling faults (*CFrd*);
- deceptive read destructive coupling faults (*CFdrd*);
- incorrect read coupling faults (*CFir*).

For example, three cells are said to have a disturb coupling fault if an operation (read, transition or non-transition write) performed on the $a1$-cell causes the v-cell to flip, when $a2$-cell is in a given state; the FP defined in the row 9 shows that a read 0 operation performed on $a1$-cell causes the v-cell to flip from 0 to 1, when the $a2$-cell is in 0 state. Note that, the number of fault primitives is equal to the number of arcs in the graph of states that describes the normal operations for three memory cells (8 states $\times$ 9 arcs for each state). For this reason, the model is called "all static three-cell coupling faults".

Many test algorithms dedicated to such models of three-cell coupling faults have been reported. Thus, a memory test that requires $n + 32n \log_2 n$ operations is given by Nair, Thatte, and Abraham (*Algorithm B*) [8]. A new test of length $n + 24n \log_2 n$ is proposed by Papachristou and Sahgal [9]. Two more efficient test algorithms, *S3CTEST* and *S3CTEST2*, are given by Cockburn [10]. These are tests of approximate length $5n \log_2 n + 22, 5n$ and $5n \log_2 n + 5n\,[\log_2(1 + \log_2 n)] + 11n$, respectively. But, for the memory chips currently available, all these tests take a long time to perform because the authors have assumed that the coupled cells can be anywhere in the memory. For example, to test a 64 Mb memory chip assuming a cycle time of 60 ns, *S3CTEST* takes about 10 min 54s.

To reduce the length of the test, Caşcaval and Bennett have restricted the model to the more realistic coupling faults that affect only the physically adjacent memory cells [11]. Thus, for a set of three coupled cells $\{i,\ j,\ k\}$, six patterns denoted by P_1, P_2, P_3, P_4, P_5 and P_6 are accepted, as shown in Fig. 1.

Table 1. List of three-cell FPs

#	Fault primitives	FFM
1	< 0; 0; 0; / 1 / - >	*CFst*
2	< 0; 1; 0; / 1 / - >	
3	< 1; 0; 0; / 1 / - >	
4	< 1; 1; 0; / 1 / - >	
5	< 0; 0; 1; / 0 / - >	
6	< 0; 1; 1; / 0 / - >	
7	< 1; 0; 1; / 0 / - >	
8	< 1; 1; 1; / 0 / - >	
9	< *r*0; 0; 0; /↑ / - >	*CFds*
10	< *r*0; 0; 1; /↓ / - >	
11	< *r*0; 1; 0; /↑ / - >	
12	< *r*0; 1; 1; /↓ / - >	
13	< *r*1; 0; 0; /↑ / - >	
14	< *r*1; 0; 1; /↓ / - >	
15	< *r*1; 1; 0; /↑ / - >	
16	< *r*1; 1; 1; /↓ / - >	
17	< 0*w*0; 0; 0; /↑ / - >	
18	< 0*w*0; 0; 1; /↓ / - >	
19	< 0*w*0; 1; 0; /↑ / - >	
20	< 0*w*0; 1; 1; /↓ / - >	
21	< 1*w*1; 0; 0; /↑ / - >	
22	< 1*w*1; 0; 1; /↓ / - >	
23	< 1*w*1; 1; 0; /↑ / - >	
24	< 1*w*1; 1; 1; /↓ / - >	
25	< 0*w*1; 0; 0; /↑ / - >	
26	< 0*w*1; 0; 1; /↓ / - >	
27	< 0*w*1; 1; 0; /↑ / - >	
28	< 0*w*1; 1; 1; /↓ / - >	
29	< 1*w*0; 0; 0; /↑ / - >	
30	< 1*w*0; 0; 1; /↓ / - >	
31	< 1*w*0; 1; 0; /↑ / - >	
32	< 1*w*0; 1; 1; /↓ / - >	
33	< 0; 0; 0*w*1; / 0 / - >	*CFtr*
34	< 0; 1; 0*w*1; / 0 / - >	
35	< 1; 0; 0*w*1; / 0 / - >	
36	< 1; 1; 0*w*1; / 0 / - >	
37	< 0; 0; 1*w*0; / 1 / - >	*CFtr*
38	< 0; 1; 1*w*0; / 1 / - >	
39	< 1; 0; 1*w*0; / 1 / - >	
40	< 1; 1; 1*w*0; / 1 / - >	
41	< 0; 0; 0*w*0; /↑ / - >	*CFwd*
42	< 0; 1; 0*w*0; /↑ / - >	
43	< 1; 0; 0*w*0; /↑ / - >	
44	< 1; 1; 0*w*0; /↑ / - >	
45	< 0; 0; 1*w*1; /↓ / - >	
46	< 0; 1; 1*w*1; /↓ / - >	
47	< 1; 0; 1*w*1; /↓ / - >	
48	< 1; 1; 1*w*1; /↓ / - >	
49	< 0; 0; *r*0; /↑ / 1 >	*CFrd*
50	< 0; 1; *r*0; /↑ / 1 >	
51	< 1; 0; *r*0; /↑ / 1 >	
52	< 1; 1; *r*0; /↑ / 1 >	
53	< 0; 0; *r*1; /↓ / 0 >	
54	< 0; 1; *r*1; /↓ / 0 >	
55	< 1; 0; *r*1; /↓ / 0 >	
56	< 1; 1; *r*1; /↓ / 0 >	
57	< 0; 0; *r*0; /↑ / 0 >	*CFdrd*
58	< 0; 1; *r*0; /↑ / 0 >	
59	< 1; 0; *r*0; /↑ / 0 >	
60	< 1; 1; *r*0; /↑ / 0 >	
61	< 0; 0; *r*1; /↓ / 1 >	
62	< 0; 1; *r*1; /↓ / 1 >	
63	< 1; 0; *r*1; /↓ / 1 >	
64	< 1; 1; *r*1; /↓ / 1 >	
65	< 0; 0; *r*0; / 0 / 1 >	*CFir*
66	< 0; 1; *r*0; / 0 / 1 >	
67	< 1; 0; *r*0; / 0 / 1 >	
68	< 1; 1; *r*0; / 0 / 1 >	
69	< 0; 0; *r*1; / 1 / 0 >	
70	< 0; 0; *r*1; / 1 / 0 >	
71	< 1; 0; *r*1; / 1 / 0 >	
72	< 1; 1; *r*1; / 1 / 0 >	

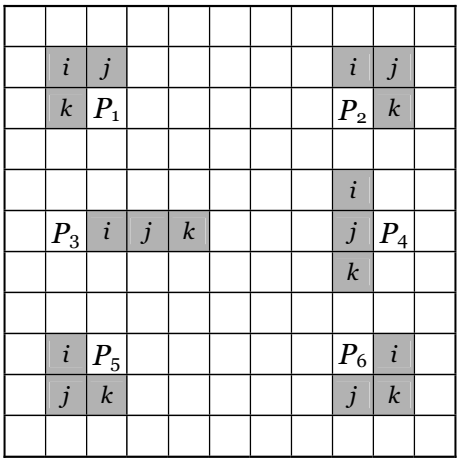


Fig. 1. Patterns for three physically adjacent cells.

This model of three-cell coupling, which comprises only physically adjacent memory cells, is called *reduced three-cell coupling*. Under this hypothesis and for the cases in which the mapping from logical addresses to physical cell locations is known completely, they have devised a march test of length $38n$, which covers this reduced model of three-cell coupling faults. A new more efficient march test with $30n$ operations (*MT-R3CF*) dedicated to the same reduced model of three-cell coupling faults is reported by Caşcaval, Bennett, and Huţanu [12]. But, all these test algorithms assume that a memory fault can be sensitized only by a transition write operation into a cell. Based on the model of all static two-cell coupling faults defined by Hamdioui, van de Goor and Rodgers in [7], this model of three-cell coupling faults has been extended by considering other classes of faults, such as the faults sensitized by a read or a non-transition write operation [1]. To cover this large fault model, called *all static reduced three-cell coupling faults*, the memory test $MarchS_3C$ has been proposed. This test is presented briefly in the following section.

4. The Memory Test $MarchS_3C$

The march tests are the most popular and widely accepted deterministic test algorithms because of their low temporal complexity, regular structures and their ability to detect a wide variety of memory faults. Usually, a complete march test is delimited by '$\{\ldots\}$' bracket pair, while a march element is delimited by the '$(\ldots)$' bracket pair. March elements are separated by semicolons, and the operations within a march element are separated by commas. Note that all operations of a march element are performed at a certain address, before proceeding to the next address. The whole memory is checked homogeneously in either one of two orders: ascending address order ($\Uparrow$) or descending address order ($\Downarrow$). When the address order is not relevant, the symbol $\Updownarrow$ is used.

The memory test we have considered for a possible BIST implementation is $MarchS_3C$. This memory test dedicated to the reduced model of all static three-cell coupling is presented in Fig. 2, where I_1, I_2, I_3, and I_4 are sequences which initialize the memory as follows: I_1 initializes the odd columns with 0 and the even columns with 1, and I_3 vice versa (*column-stripe data background*); I_2 and I_4 initialize the memory with a *checkerboard data background* and its complement, as illustrated in Fig. 3.

$$
\begin{aligned}
&\{\ \Updownarrow (w0)^{(0)};\\
&\ \Uparrow (r,r,w,r,w_c)^{(1)};\Uparrow (r,r,w,r,w_c)^{(2)};\ \Downarrow (r,r,w,r,w_c)^{(3)};\Downarrow (r,r,w,r,w_c)^{(4)};\\
&\ I_1{}^{(5)};\Uparrow (r,r,w,r,w_c,r,r,w,r,w_c)^{(6)};I_2{}^{(7)};\ \Uparrow (r,r,w,r,w_c,r,r,w,r,w_c)^{(8)};\\
&\ I_3{}^{(9)};\Uparrow (r,r,w,r,w_c,r,r,w,r,w_c)^{(10)};I_4{}^{(11)};\Uparrow (r,r,w,r,w_c,r,r,w,r,w_c)^{(12)};\\
&\ \Updownarrow (r0)^{(13)}\ \}
\end{aligned}
$$

Fig. 2. The memory test $MarchS_3C$.

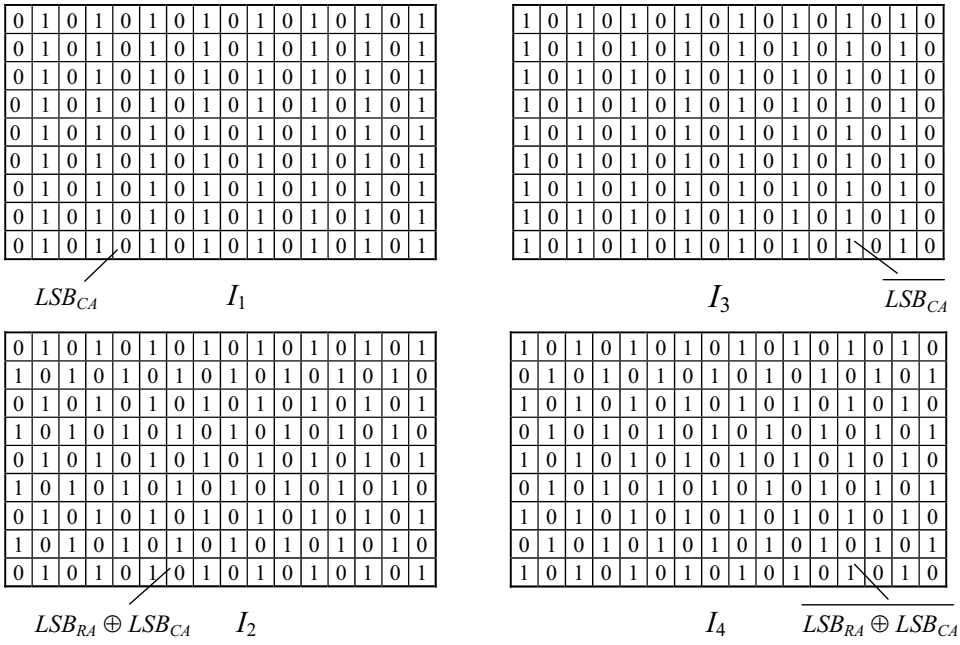


Fig. 3. Memory initialization for the memory test $MarchS_3C$.

This march test contains fourteen sequences as identified with a superscript (x), where $x \in \{0, 1, \ldots, 13\}$. The test sequences (5)-(12) form an alternating series of

background changes and march elements (as Cockburn proposed in [10]). Note that when changing from one background to the next, only the cells that must change states are written. Also, each write operation is preceded by a read operation. We can observe in Fig. 4 that any background change affects only a half of the cells. Each test sequence I_1, I_2, I_3 or I_4 performs $\frac{n}{2}$ read operations and $\frac{n}{2}$ write operations. Consequently, $March S_3C$ needs $66n$ operations. This march memory test is able to detect all the 72 FPs presented in Table 1 for all the six patterns illustrated in Fig. 1.

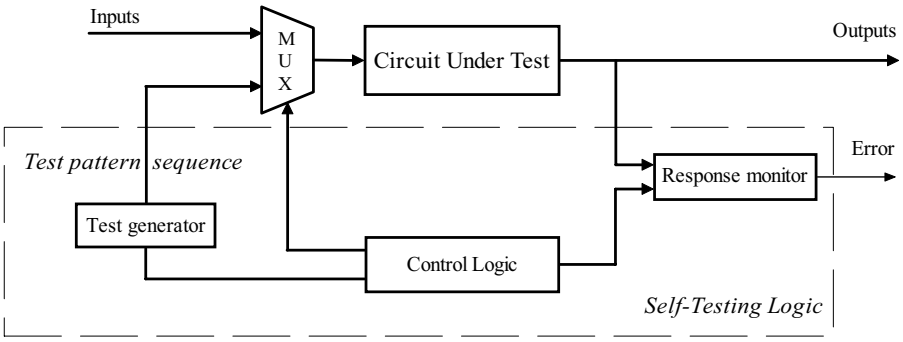


Fig. 4. Generic BIST scheme.

Regarding this march test, note that symbol $\Uparrow$ denotes an increasing address order, from address 0 to n-1, as long as symbol $\Downarrow$ denotes a reverse address order, from address n-1 to 0. Other permutations of the set of memory cell addresses decrease the effectiveness of the march test.

5. Simulation Results

To compare $March S_3C$ with other published tests, simulation results are presented in this section. The following march tests have been considered for the simulation study:

a) MT-R3CF [12] : $\{\Updownarrow(w0);\Uparrow(r,w1);\Uparrow(r,w0);\Downarrow(r,w1);\Downarrow(r,w0);\ I_1;$
$\Uparrow(r,wc,r,wc);\ I_2;\ \Uparrow(r,wc,r,wc);\ I_3;\ \Uparrow(r,wc,r,wc);\ I_4;\ \Uparrow(r,wc,r,wc);\ \Updownarrow(r)\}$, where I_1, I_2, I_3 and I_4 are sequences that initialize the memory as illustrated in Fig. 3.

b) Algorithm A [8] : $\{\Uparrow(w0);\ \Uparrow(r0,w1);\ \Uparrow(r1);\ \Uparrow(r1,w0);\ \Uparrow(r0);\ \Downarrow(r0,w1);\ \Downarrow(r1);$
$\Downarrow(r1,w0);\ \Downarrow(r0);\ \Uparrow(r0,w1,w0);\ \Uparrow(r0);\ \Downarrow(r0,w1,w0);\ \Downarrow(r0);\ \Uparrow(w1);\ \Uparrow(r1,w0,w1);$
$\Uparrow(r1);\ \Downarrow(r1,w0,w1);\ \Downarrow(r1)\ \}$.

c) March G [13] : $\{\Updownarrow(w0);\ \Uparrow(r0,w1,r1,w0,w1);\ \Uparrow(r1,w0,r0,w1);$
$\Downarrow(r1,w0,w1,\ w0);\ \Downarrow(r0,w1,r1,w0);\ \Uparrow(r0,w1,r1);\ \Downarrow(r1,w0,r0)\}$.

d) March S2C [14] : $\{\Updownarrow(w0);\Uparrow(r0,w1,r1,r1,w1);\ \Uparrow(r1,w0,r0,r0,w0);$ $\Downarrow(r0,w1,r1,r1,w1);\ \Downarrow(r1,w0,r0,r0,w0);\ \Updownarrow(r0);\ \}$.

e) March LA [15] : $\{\Updownarrow(w0);\ \Uparrow(r0,w1,w0,w1,r1);\ \Uparrow(r1,w0,w1,w0,r0);$ $\Downarrow(r0,w1,w0,w1,r1);\ \Downarrow(r1,w0,w1,w0,r0);\ \Downarrow(r0)\}$.

f) March B [16] : $\{\Updownarrow(w0);\ \Uparrow(r0,w1,r1,w0,r0,w1);\ \Uparrow(r1,w0,w1);$ $\Downarrow(r1,w0,w1,w0);\ \Downarrow(r0,w1,w0)\}$.

g) March LR [17] : $\{\Updownarrow(w0);\ \Downarrow(r0,w1);\ \Uparrow(r1,w0,r0,w1);\ \Uparrow(r1,w0);$ $\Uparrow(r0,w1,r1,w0);\ \Uparrow(r0)\}$.

h) March U [18] : $\{\Updownarrow(w0);\Uparrow(r0,w1,r1,w0);\Uparrow(r0,w1);\ \Downarrow(r1,w0,r0,w1);$ $\Downarrow(r1,w0)\}$.

All the six patterns P_1, P_2, ..., P_6 illustrated in Fig. 1, and all FPs presented in Table 1 have been considered in this simulation study. Moreover, for each group of coupled cells, composed of two aggressor cells ($a1$-cell and $a2$-cell) and a victim cell (v-cell), six distinct combinations on the set $\{a1,\ a2,\ v\}$ have been considered. Consequently, 2952 (6×6×72) FPs have been simulated in order to evaluate the fault coverage of this reduced model of three-cell coupling faults. The simulation results are presented in Table 2.

Table 2. Fault coverage of this reduced model of three-cell coupling faults

#	Memory test	Test length	Fault coverage (%)
1	MarchS3C	66*n*	100
2	MT_R3CF	30*n*	56.73
3	Algorithm A	30*n*	46.30
4	March G	24*n*	44.91
5	MarchS2C	22*n*	53.40
6	March LA	22*n*	42.82
7	March B	17*n*	32.18
8	March LR	14*n*	43.52
9	March U	13*n*	42.59

As shown in Table 2, only $MarchS_3C$ is able to detect all FPs of simple three-cell coupling faults.

6. A BIST Logic Design for $MarchS_3C$

Generally, a circuit with BIST facilities has two operation modes: normal operation and test. In normal operation, the circuit receives its inputs from other modules

and performs the function for which it was designed. In the test mode, a test generator applies a sequence of test patterns to the memory, and a response monitor evaluates the test responses as illustrated in Fig. 4.

For the memory test $MarchS_3C$, a BIST logic with the block diagram presented in Fig. 5 is proposed.

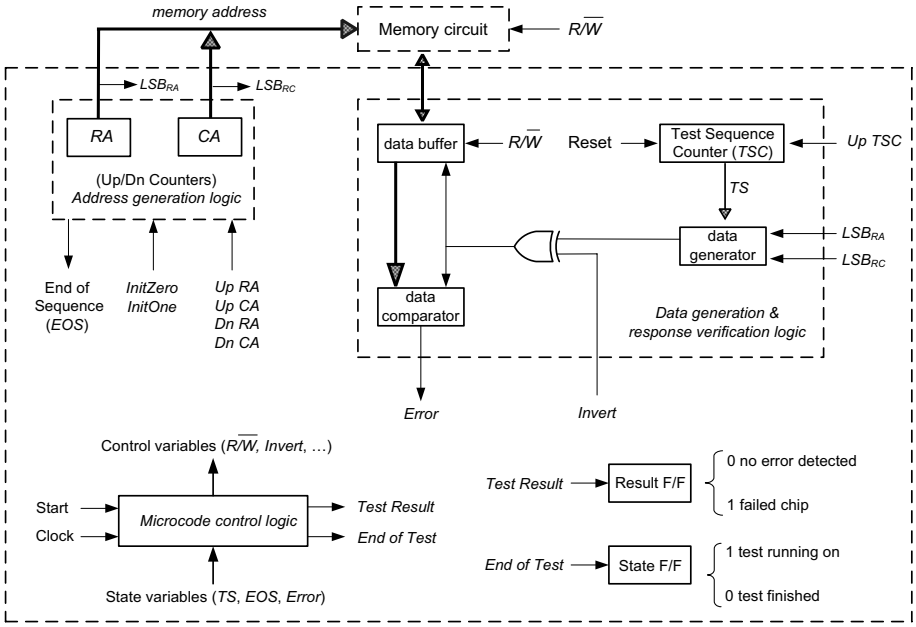


Fig. 5. The block diagram of BIST logic for the memory test $MarchS_3C$.

The BIST logic is composed of three parts: address–generation logic, data–generation and response–verification logic, and microcode control logic.

A. Address–generation logic

The address generation logic is composed of two up/down counters, for row address (RA) and column address (CA), respectively. Each address counter can be initialised with 0 ($InitZero$) or 1 ($InitOne$) in all bit locations. The memory is checked in ascending or descending order, so that, depending on the test sequence (TS), the control logic increments (Up) or decrements (Dn) one of the address counters, RA or CA.

B. Data generation and response-verification logic

The data-generation logic supplies data to be written in the memory and the expected data for response monitoring. An unique logic to generate both data for writing operations and expected data for response monitoring can be used. Note that, except on the first initialisation ($w0$) and the final checking of the memory ($r0$), the test algorithm is composed of two kinds of march elements, (r, r, w, r, w_c) and (r, r, w,

r, w_c, r, r, w, r, w_c). Every write operation into a cell is preceded by a read operation. The expected data in the first read operation of a cell is presented in Table 3, where *LSBCA* and *LSBRA* denote the less significant bit of the column address of the cell, and the less significant bit of the row address of the cell, respectively (Fig. 3).

Table. 3. Expected data in the first read operation of a cell

Test sequence (TS)	*Expected data*
(1)	0
(2)	1
(3)	0
(4)	1
(5)	0
(6)	LSB_{CA}
(7)	LSB_{CA}
(8)	$LSB_{CA} \oplus LSB_{RA}$
(9)	$LSB_{CA} \oplus LSB_{RA}$
(10)	$\overline{LSB_{CA}}$
(11)	$\overline{LSB_{CA}}$
(12)	$\overline{LSB_{CA} \oplus LSB_{RA}}$
(13)	$\overline{LSB_{CA} \oplus LSB_{RA}}$

Data generator is basically composed of a multiplexor with input variables in accordance with the expected data presented in Table 3, and a test sequence counter (*TSC*) that supplies the selection inputs. The logic variable *Invert* is used to command the XOR gate for changing the data bit generated.

C. Microcode control logic

The control logic initiates and controls the testing process. The control flow of the test algorithm $MarchS_3C$ is presented in Fig. 6.

In Fig. 6 the following notations are used to denote specific operations performed on the current memory cell:

- O ($w0$) − for the first initialization;
- O (*shortME*) − for the march element (r, r, w, r, w_c);
- O (*longME*) − for the march element (r, r, w, r, w_c, r, r, w, r, w_c);
- O ($r0$) − for the final checking.

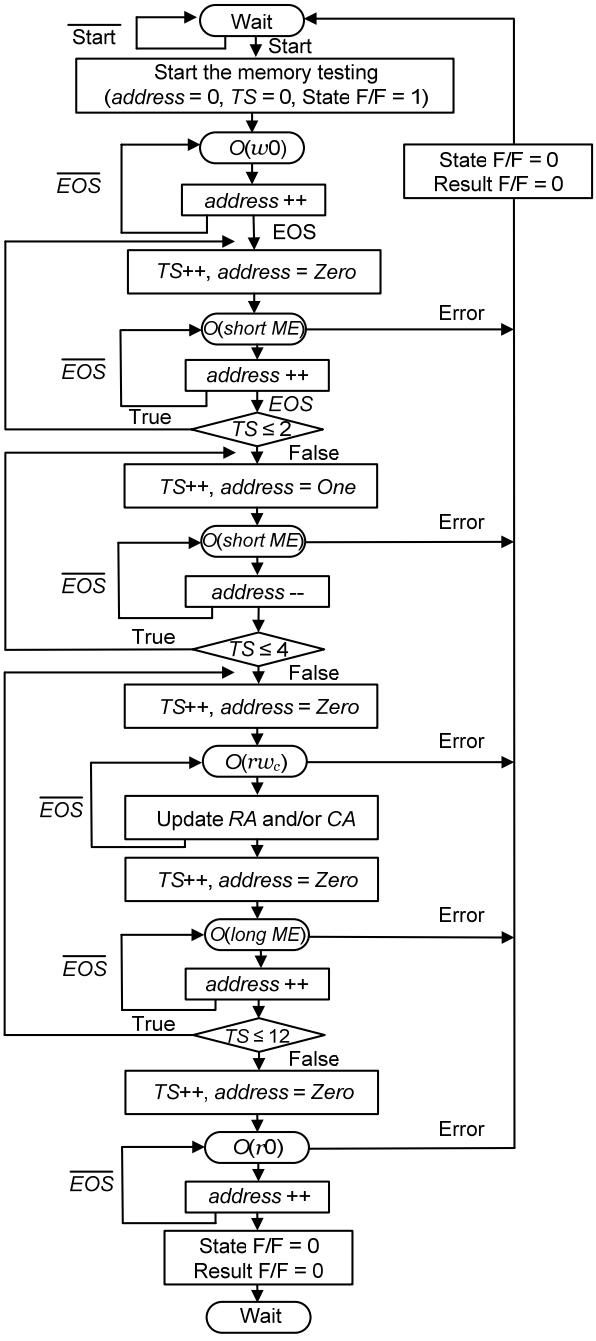


Fig. 6. Control flow for the memory test $MarchS_3C$.

7. Final Remarks

We are unaware of other memory test able to detect all FPs of this large model of three-cell coupling faults. To cover these harder-to-detect faults, $MarchS_3C$ uses multiple data backgrounds: a solid, a column−stripe, and a checkerboard. Although $MarchS_3C$ uses different patterns for memory initialization, the logic design we have proposed for a possible built-in self-testing implementation is not so complicated.

Regarding the BIST logic design for a MASB architecture, another response verification method is comparing the outputs of symmetrically placed bits in the tested arrays. An advantage of the parallel comparison method is that the expected values need not be generated.

References

[1] CAŞCAVAL P., CAŞCAVAL D., *March Test for a Reduced Model of All RAM Static 3-Cell Coupling Faults*, Bul. Inst. Polit. Iaşi, Tom **LIII (LVII)**, Autom. and Comput., pp. 87–96, 2007.

[2] ALLAN A. *et al.*, *2001 International Technology Roadmap for Semiconductors*, Computers, **35** (1), pp. 42–53, 2002.

[3] HAMDIOUI S., AL-ARS Z., JIMENEZ J., CALERO J., *PPM Reduction on Embedded Memories in System on Chip*, *Proc. IEEE European Test Symp.* (ETS'07), Freiburg, Germany, May 2007, pp. 85–90.

[4] HAMDIOUI S., *Testing Static Random Access Memories: Defects, Fault Models and Test Patterns*, *Kluwer Academic Publishers*, Norwell, USA, 2004.

[5] ADAMS R.D., *High Performance Memory Testing*, *Kluwer Academic Publishers*, Norwell, USA, 2003.

[6] FRANCKLIN M., SALUJA K., *Built-in Self Testing of RAMs*, IEEE Computer, **23** (10), pp. 45–56, 1990.

[7] HAMDIOUI S., VAN DE GOOR A.J., RODGERS M., *March SS: A Test for All Static Simple RAM Faults*, *Proc. of 10th IEEE Int. Workshop on Memory Technology, Design and Testing*, pp. 95–100, Isle of Bendor, France, July, 2002.

bibitemnair NAIR R., THATTE S., ABRAHAM J., *Efficient Algorithms for Testing Semiconductor Random-Access Memories*, IEEE Trans. on Computers, **C-27** (6), pp. 572–576, 1978.

[8] PAPACHRISTOU C., SAHGAL N., *An Improved Method for Detecting Functional Faults in Semiconductor Random-Access Memories*, IEEE Trans. on Computers, **C-34** (2), pp. 110–116, 1985.

[9] COCKBURN B.F., *Deterministic Testing for Detecting Single V-Coupling Faults in RAMs*, Journal of Electronic Testing, Theory and Applications, **5** (1), pp. 91–113, 1994.

[10] CAŞCAVAL P., BENNETT S., *Efficient March Test for 3-Coupling Faults in Random-Access Memories*, *Microprocessors and Microsystems*, **24** (10), pp. 501–509, 2001.

[11] CAŞCAVAL P., BENNETT S., HUŢANU C., *Efficient March Tests for a Reduced 3-Coupling and 4-Coupling Faults in Random-Access Memories*, Journal of Electronic Testing, Theory and Applications, **20** (3), pp. 227–243, 2004.

[12] VAN DE GOOR A.J., *Using March Tests to Test SRAMs*, IEEE Design and Test of Computers, **10** (1) pp. 8–14, 1993.

[13] CAŞCAVAL P., SILION R., STAN A., *MarchS2C: A Test for All Static 2-Cell RAM Coupling Faults*, Bul. Inst. Polit. Iaşi, Tom. **LII (LVI)**, Fasc. 1–4, Autom. and Comput., pp. 79–86, 2006.

[14] VAN DE GOOR A.J. *et al.*, *March LA: A Test for Linked Memory Faults*, *Proc. of European Design and Test Conference*, pp. 627–634, Paris, France, 1999.

[15] SUK D., REDDY S., *Test Procedures for a Class of Pattern-Sensitive Faults in Semiconductor Random-Access Memories*, IEEE Trans. on Computers, **C-29** (6), pp. 419–429, 1980.

[16] YARMOLIK V.N., VAN DE GOOR A.J., GAYDADJIEV G.N., MIKITJUK V.G., *March LR: A test for realistic linked faults*, *Proc. 14th IEEE VLSI Test Symp.* (VTS'96), pp. 272–280, 1996.

[17] VAN DE GOOR A.J., GAYDADJIEV G.N., *March U: A Test for All Unlinked Memory Faults*, *IEE Proc. of Circuits Devices and Systems*, **144** (3), pp. 155–160, 1997.


JOURNAL OF ELECTRONIC TESTING: Theory and Applications 20, 227–243, 2004

Efficient March Tests for a Reduced 3-Coupling and 4-Coupling Faults in Random-Access Memories

PETRU CAŞCAVAL
Department of Computer Science,"Gh. Asachi" Technical University of Iaşi, Bd. D. Mangeron, nr.53A, 6600, Iaşi, Romania
cascaval@cs.tuiasi.ro

STUART BENNETT
Department of Automatic Control and Systems Engineering, The University of Sheffield, Mappin Street, Sheffield S13JD, UK
S.Bennett@sheffield.ac.uk

CORNELIU HUŢANU
Department of Automatic Control, "Gh. Asachi" Technical University of Iaşi, Bd. D. Mangeron, nr.53A, 6600, Iaşi, Romania
chutanu@ac.tuiasi.ro

Received July 17, 2000; Revised August 19, 2002

Editor: B.F. Cockburn

Abstract. This paper presents two new march test algorithms, *MT-R3CF* and *MT-R4CF*, for detecting reduced 3-coupling and 4-coupling faults, respectively, in $n \times 1$ random-access memories (RAMs). To reduce the length of the tests, only the coupling faults between physically adjacent memory cells have been considered. The tests assume that the storage cells are arranged in a rectangular grid and that the mapping from logical addresses to physical cell locations is known completely. The march tests need $30n$ and $41n$ operations, respectively. In this paper any memory fault is modelled by a set of primitive memory faults called simple faults. We prove, using an Eulerian graph model, the ability of the test algorithms to detect all simple coupling faults. This paper also includes a study regarding the ability of the test *MT-R3CF* to detect interacting linked 3-coupling faults. This work improves the results presented in [1] where a similar model of reduced 3-coupling faults has been considered and a march test with $38n$ operations has been proposed. To compare these new march tests with other published tests, simulation results are presented in this paper.

Keywords: memory testing, functional faults, coupling faults, march test, fault simulation

1. Introduction

A memory module can be said to be running properly if it is possible to read and correctly change the state of any memory cell independently of the other cell states. But it is impossible to check any cell for all the states of the other cells because the test length would grow by 2^n, where n is the memory size [5]. Therefore any practical memory test must focus on a fault model which covers only a limited set of physical faults, for which

the occurrence probability cannot be ignored. Test procedures are constrained by the following conflicting requirements:

(a) to detect a wide variety of memory faults;
(b) to reduce the number of memory operations in order to allow large memories to be tested in an acceptable period of time.

Rapid developments in semiconductor technology have resulted in continuing growth of larger and denser random-access memories on a single chip (now 256 Mb and more). With increasing densities, certain types of faults, such as coupling faults or pattern sensitive faults, which are harder to detect are becoming more important [10]. In addition, more time is required to test memories because of their increasing size thus it is necessary to identify more efficient tests, with the ability to detect complex faults, tests that require test time on the order of n.

We report new efficient march test algorithms for difficult-to-detect faults such as 3-coupling and 4-coupling faults. For the 3-coupling fault model, a memory test that requires $n + 32n \log_2 n$ operations is given by Nair et al. [5] (Algorithm B, *NTA*(B) in this article). In [6] a new test of length $n + 24n \log_2 n$ is proposed by Papachristou and Sahgal (*PS*(B) in this article). Two more efficient test algorithms, *S3CTEST* and *S3CTEST2*, are proposed by Cockburn in [2]. For 4-coupling faults Cockburn also proposed the test *S4CTEST*. Taking into account the fault model considered in this paper and based on the specification given by Cockburn in [3], we consider the Cockburn tests with the sequence of operations $(r_u w_{\bar{u}} r_{\bar{u}} w_u)$ instead of the reduced sequence $(r_u w_{\bar{u}} w_u)$. Consequently, *S3CTEST*, *S3CTEST2*, and *S4CTEST* are tests of approximate length $5n \log_2 n + 22.5n$, $5n \log_2 n + 5n\,[\log_2(1 + \log_2 n)] + 11n$, and $10.75n(\log_2 n)^{1.585}$, respectively. For the memory chips currently available, these tests take a long time to perform. For example, to test a 64 Mb memory chip assuming a cycle time of 60 ns, *PS*(B) and *S3CTEST* take about 44 min 24 s and 10 min 54 s, respectively. These memory tests are long because the authors have assumed that the coupled cells can be anywhere in the memory.

We restrict ourselves to coupling faults that affect only physically adjacent memory cells. Under this hypothesis and for the cases in which the mapping from logical addresses to physical cell locations is known completely, we have devised a march test with $30n$ operations for 3-coupling faults and a march test with $41n$ operations for 4-coupling faults. This is an improvement on the results presented in [1] where a march test with $38n$ operations is proposed.

The remainder of this paper is organised as follows. Section 2 defines the fault model and Section 3 introduces notations, definitions and preliminary considerations. Section 4 presents a new march test *MT-R3CF* for a reduced model of 3-coupling and analyses the ability of the test to detect simple coupling faults and interacting linked coupling faults. Section 5 presents and analyses the test *MT-R4CF* for a reduced model of 4-coupling. Section 6 contains simulation results and compares the new march tests with other published tests. Finally, some conclusions are drawn regarding this work.

2. Memory Fault Model

This paper treats the problem of coupling faults in random-access memories. Because the address decoders, sense amplifiers and write drivers are easier to test, we assume that these modules are fault-free and we focus only on the functional faults in the memory cell array where difficult-to-detect faults may exist.

Many different faults can occur in a memory cell array. These can be classified as faults which involve only a single cell (such as stuck-at, stuck-open, transition and data retention faults) and faults where a cell or set of cells influences the behaviour of another cell (such as coupling faults and pattern sensitive faults) [10].

Coupling faults involve v cells ($v \geq 2$). In a set of coupled cells an active and/or a passive influence on a victim cell may exist [10]. Accordingly, coupling faults can be divided into transition and state coupling faults.

(1) *Transition coupling faults (TCFs).* Consider a set of v-coupled cells, $v \geq 2$. The transition coupling fault is used to represent the situation when write operations addressed to one memory cell of the set, say cell j, cause the state of another cell in the set, say cell i, to change from 0 to 1 or from 1 to 0, while the $v - 2$ remaining cells hold a specific pattern. Cell i is called the coupled (victim) cell and cell j is called the coupling (aggressor) cell. If $v > 2$ then the $v - 2$ remaining cells are called the enabling cells. We write down this fault as $j \rightarrow i$ TCF.

For the case in which two cells are coupled ($v = 2$), two types of TCFs are usually considered: inversion

coupling fault (CFin) and idempotent coupling fault (CFid) as defined in [10].

(2) State coupling faults (SCFs). The state coupling fault is used to represent the situation when a cell in a set of $v-$coupled cells ($v \geq 2$), say cell i, fails to undergo a $0 \rightarrow 1$ or a $1 \rightarrow 0$ transition when the complement of the contents of the memory cell is written into the cell, while the remaining $v-1$ cells in the set (enabling cells) hold a specific pattern. In such a case, we say that the enabling cells have a passive influence on the coupled (victim) cell and call this fault an i-state coupling fault (i-SCF). Note that a SCF is activated by a transition in the victim cell, whereas a TCF is activated by a transition in other aggressor cell.

In this work we have limited ourselves to the model with at most four coupled cells and we assume that only physically adjacent cells can be 3-coupled or 4-coupled. For a set of 3-coupled cells, six patterns P_1, P_2, P_3, P_4, P_5 and P_6 are accepted, as shown in Fig. 1. For a set of 4-coupled cells our model is limited to the square pattern. We call these models of 3-coupling and 4-coupling faults, which comprise only physically adjacent cells, reduced 3-coupling and 4-coupling, respectively.

Remark 1. Because in our model the 3-coupled cells and 4-coupled cells are physically adjacent, the transition coupling faults and the state coupling faults can also be considered as active neighbourhood pattern sensitive faults (ANPSFs) and passive neighbourhood pattern sensitive faults (PNPSFs), respectively [8]. But in our model any cell in a set of v-coupled cells can be a victim cell, not only the central base cell as is usually considered in the NPSFs model (see for example [8]).

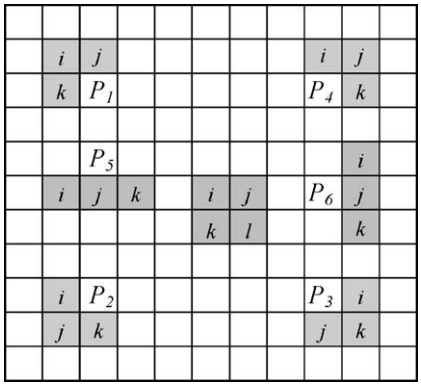


Fig. 1. Patterns for three and four physically adjacent cells.

In the memory, one or more sets of coupled cells may exist. As in [5, 6] we assume that the pairs of sets of coupled cells are disjoint.

Definition 1. A *triggering transition* is defined as one that is initiated by the testing algorithm by writing into a cell the complement of the previous logic value of the cell. With respect to a given fault, the triggering transition which may activate the fault is called an activating transition.

We assume that a memory fault can be activated only by a triggering transition into a cell. In other words, we do not consider in this work disturb coupling faults which can be activated by either read or write operations (as is possible for dynamic coupling faults).

3. Notations, Definitions and Preliminaries

We use the following notations to describe operations on RAMs:

- x denotes that a cell is in a logical state x; $x \in \{0, 1\}$.
- $r(r^i)$—the read operation on a cell (specifically to cell i).
- $w_x(w_x^i)$—the operation of writing x into a cell (cell i), $x \in \{0,1\}$.
- $w_c(w_c^i)$—the operation of writing the complement of the previous state of a cell (cell i).
- $\uparrow$ ($\uparrow i$)—the operation of writing 1 into a cell (cell i) when the previous state of the cell was 0.
- $\downarrow$ ($\downarrow i$)—the operation of writing 0 into a cell (cell i) when the previous state of the cell was 1.

Consider a set S of v-coupled cells. A state of set S is given by the logical state of each cell in S. In order to describe a failed write operation in S, we use a vector F with $2v$ elements grouped in two parts, separated by ":". The first part shows the initial state of set S and the triggering transition which activates the fault and the second part shows the state of set S after the triggering transition is carried out. Vector F is composed by the symbols $0, 1, \downarrow$ and $\uparrow$. Only one symbol in vector F can be $\uparrow$ or $\downarrow$. For example, for a set of cells $S = \{i, j, k\}$,

Table 1. Simple 2-coupling faults in a set of cells $S = \{i, j\}$.

Nr.	Vector F	Fault type	Nr.	Vector F	Fault type
1	$\langle\uparrow, 0 : D, 0\rangle$	i-SCFs	9	$\langle 0, \uparrow: 0, D\rangle$	j-SCFs
2	$\langle\uparrow, 1 : D, 1\rangle$		10	$\langle 1, \uparrow: 1, D\rangle$	
3	$\langle\downarrow, 0 : \bar{D}, 0\rangle$		11	$\langle 0, \downarrow: 0, \bar{D}\rangle$	
4	$\langle\downarrow, 1 : \bar{D}, 1\rangle$		12	$\langle 1, \downarrow: 1, \bar{D}\rangle$	
5	$\langle\uparrow, 0 : 1, \bar{D}\rangle$	$i \rightarrow j$ TCFs	13	$\langle 0, \uparrow: \bar{D}, 1\rangle$	$j \rightarrow i$ TCFs
6	$\langle\uparrow, 1 : 1, D\rangle$		14	$\langle 1, \uparrow: D, 1\rangle$	
7	$\langle\downarrow, 0 : 0, \bar{D}\rangle$		15	$\langle 0, \downarrow: \bar{D}, 1\rangle$	
8	$\langle\downarrow, 1 : 0, D\rangle$		16	$\langle 1, \downarrow: D, 1\rangle$	

- vector $F_1 = \langle 0, \uparrow, 0 : 0, 0, 0\rangle$ describes a SCF in which the triggering transition $\uparrow j$ fails to write 1 into cell j while cells i and k are in the state 0, and
- vector $F_2 = \langle\uparrow, 0, 0 : 1, 0, 1\rangle$ describes a TCF in which the triggering transition $\uparrow i$ changes the state of cell k from 0 to 1 if cell j is in the state 0.

To emphasis the state of cell affected by the fault (the victim cell) we use a logic variable D (the well known Roth's notation [7]) as follows:

$$D = \begin{cases} 1 & \text{the cell is fault-free} \\ 0 & \text{the coupling fault has been activated} \end{cases}$$

Thus, the faults previously defined become: $F_1 = \langle 0, \uparrow, 0 : 0, D, 0\rangle$ and $F_2 = \langle\uparrow, 0, 0 : 1, 0, \bar{D}\rangle$, where $\bar{D} = \text{NOT } D$.

Definition 2. A memory fault that affects a cell in a set of cells S is called a *simple fault* if and only if it is activated by a single cell transition in set S. Only one vector F is necessary to describe a simple fault. If at least two vectors are necessary for a complete description, the fault is called a *complex fault*. Any complex fault can be modelled as a set of distinct simple faults simultaneously present in the memory.

Table 1 presents all possible simple faults for a set $S = \{i, j\}$ of coupled cells and Table 2 presents all simple 3-coupling faults which may affect cell i in a set of cells $S = \{i, j, k\}$. As shown in Tables 1 and 2, for each victim cell in a set S of v-coupled cells, $v \cdot 2^v$ simple faults must be considered.

Definition 3. An *interacting linked fault* denotes a complex fault comprising two simple faults with contrary effects on the same victim cell. Thus, an interacting linked fault is described by two vectors F_1 and F_2 which contain in the same cell position the symbols D and $\bar{D}$, respectively, or vice-versa.

Examples of complex fault modelling are:

- Take 2-coupled cells with cell i the coupling cell and cell j the coupled cell. The transition $\uparrow i$ changes the state of cell j from 0 to 1 and the transition $\downarrow i$ changes the state of cell j from 1 to 0. This interacting linked fault can be modelled by two vectors,

$$F_1 = \langle\uparrow, 0 : 1, \bar{D}\rangle \quad \text{and} \quad F_2 = \langle\downarrow, 1 : 0, D\rangle.$$

- Consider a cell j that is a victim cell of two aggressor cells i and k. When cells i, j and k are in the state 0, the transition $\uparrow i$ or $\uparrow k$ changes the state of cell j from 0 to 1. This non-interacting linked fault can be modelled by two vectors,

$$F_1 = \langle\uparrow, 0, 0 : 1, \bar{D}, 0\rangle \quad \text{and}$$
$$F_2 = \langle 0, 0, \uparrow: 0, \bar{D}, 1\rangle.$$

Table 2. Simple 3-coupling faults which affect cell i in set of cells $S = \{i, j, k\}$.

Nr.	Vector F (i-SCFs)	Nr.	Vector F ($j \rightarrow i$ TCFs)	Nr.	Vector F ($k \rightarrow i$ TCFs)
1	$\langle\uparrow, 0, 0 : D, 0, 0\rangle$	9	$\langle 0, \uparrow, 0 : \bar{D}, 1, 0\rangle$	17	$\langle 0, 0, \uparrow: \bar{D}, 0, 1\rangle$
2	$\langle\uparrow, 0, 1 : D, 0, 1\rangle$	10	$\langle 0, \uparrow, 1 : \bar{D}, 1, 1\rangle$	18	$\langle 0, 1, \uparrow: \bar{D}, 1, 1\rangle$
3	$\langle\uparrow, 1, 0 : D, 1, 0\rangle$	11	$\langle 1, \uparrow, 0 : D, 1, 0\rangle$	19	$\langle 1, 0, \uparrow: D, 0, 1\rangle$
4	$\langle\uparrow, 1, 1 : D, 1, 1\rangle$	12	$\langle 1, \uparrow, 1 : D, 1, 1\rangle$	20	$\langle 1, 1, \uparrow: D, 1, 1\rangle$
5	$\langle\downarrow, 0, 0 : \bar{D}, 0, 0\rangle$	13	$\langle 0, \downarrow, 0 : \bar{D}, 0, 0\rangle$	21	$\langle 0, 0, \downarrow: \bar{D}, 0, 0\rangle$
6	$\langle\downarrow, 0, 1 : \bar{D}, 0, 1\rangle$	14	$\langle 0, \downarrow, 1 : \bar{D}, 0, 1\rangle$	22	$\langle 0, 1, \downarrow: \bar{D}, 1, 0\rangle$
7	$\langle\downarrow, 1, 0 : \bar{D}, 1, 0\rangle$	15	$\langle 1, \downarrow, 0 : D, 0, 0\rangle$	23	$\langle 1, 0, \downarrow: D, 0, 0\rangle$
8	$\langle\downarrow, 1, 1 : \bar{D}, 1, 1\rangle$	16	$\langle 1, \downarrow, 1 : D, 0, 1\rangle$	24	$\langle 1, 1, \downarrow: D, 1, 0\rangle$

To test and find a fault in a memory we need to be able to:

- activate (to sensitise) the fault by a proper triggering transition, and,
- observe the fault by reading the changed value of the cell affected by the fault.

Proposition 1. *Assume that in a set of cells at most one simple fault may exist. The next three conditions are necessary and sufficient for a test to detect any simple fault that affects a cell in a set S of coupled cells*:

- *Condition* 1. *The test must force all the possible cell transitions in the set of coupled cells in order to activate any fault.*
- *Condition* 2. *After a triggering transition into a cell in set S, the test must read the cell to check if the state has changed before another triggering transition into the cell is allowed to occur. This condition is required to detect any simple SCF activated in set S.*
- *Condition* 3. *For each possible coupled cell c in S, after one or more triggering transition in other cells in the set, the test must read cell c, prior to a triggering transition into cell c, to check if the state has been changed by a triggering transition in other possible coupling cell. This condition is required to detect any simple TCF activated in set S.*

Proposition 2. *A memory test must force at least* $v \cdot 2^v$ *cell transitions in a set of v cells to be able to activate any fault that may affect the set of cells.*

Proof: Consider the state transition diagram describing the triggering transitions in a set of v cells without faults (for $v = 2$, see for example the graph in Fig. 3). The memory test must force all the transitions in this graph. Because the number of nodes in this graph of states is 2^v and from each node v arcs go to other adjacent nodes, the memory test must force $v \cdot 2^v$ different transitions to cover the graph of states. □

Remark 2. Generally, the complex non-interacting linked faults are easier to detect than the simple faults because there are more situations in which a complex fault is activated. Thus, if a test procedure detects all the simple faults, it also detects all the complex non-interacting linked faults. In other words, the set of complex non-interacting linked faults dominates the set of simple faults. However, for the interacting linked faults the combined effects of some simple faults may cancel each other out before the affected cell is read again. In this way an interacting linked fault can escape even if the test procedure detects all the simple faults.

Proposition 3. *A memory test is able to detect an interacting linked fault comprising two simple faults if, between two consecutive read operations of the coupled cell, one simple fault is activated and another one is not.*

The march tests are the most popular and widely accepted deterministic test algorithms because of their low temporal complexity, regular structures and their ability to detect a wide variety of memory faults.

Definition 4. A march element (M) consists of a sequence of operations applied to each cell in the memory before proceeding to the next cell. The whole memory is checked homogeneously in either one of two orders: ascending address order ($\Uparrow$) or descending address order ($\Downarrow$). Symbol $\Updownarrow$ denotes either $\Uparrow$ or $\Downarrow$ address order [11].

Definition 5. A march test consists of a sequence of m march elements, $\langle M^{(0)}; M^{(1)}; \ldots; M^{(m-1)} \rangle$.

4. March Test for Reduced 3-Coupling

In this section we propose the following new march test *MT-R3CF* for the reduced model of 3-coupling faults.

$$\begin{aligned} \textit{MT-R3CF} = \langle & \Updownarrow (w_0)^{(0)}; \Uparrow (rw_1)^{(1)}; \Uparrow (rw_0)^{(2)}; \\ & \Downarrow (rw_1)^{(3)}; \Downarrow (rw_0)^{(4)}; I_1^{(5)}; \\ & \Uparrow (rw_c rw_c)^{(6)}; I_2^{(7)}; \Uparrow (rw_c rw_c)^{(8)}; \\ & I_3^{(9)}; \Uparrow (rw_c rw_c)^{(10)}; I_4^{(11)}; \\ & \Uparrow (rw_c rw_c)^{(12)}; \Updownarrow (r)^{(13)} \rangle \end{aligned} \quad (1)$$

where I_1, I_2, I_3, and I_4 are sequences which initialise the memory as follows: I_1 initialises the odd columns with 0 and the even columns with 1, and I_3 vice versa (column-stripe data background); I_2 and I_4 initialise the memory with a checkerboard data background and its complement (Fig. 2).

This march test contains fourteen sequences as identified with a superscript (x) where $x \in \{0, \ldots, 13\}$. The test sequences (5)–(12) form an alternating series of background changes and march elements (as Cockburn proposed in [2]). Note that when changing from one background to the next, only the cells that

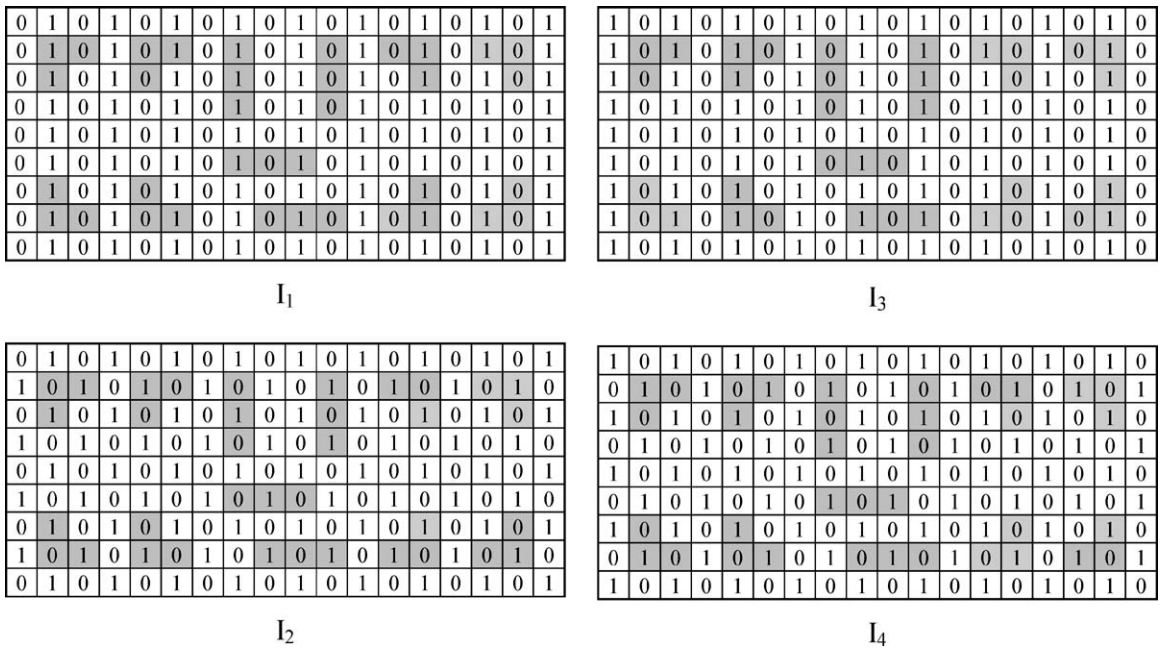


Fig. 2. Data background used by *MT-R3CF* and the possible initial states for all six distinct patterns.

must change states are written. Also, each write operation is preceded by a read operation. We can observe in Fig. 2 that any background change affects only a half of the cells. Each test sequence I_1, I_2, I_3, or I_4 performs $\frac{n}{2}$ read operations and $\frac{n}{2}$ writes operations. Consequently, *MT-R3CF* has a length of $30n$. Regarding the march test *MT-R3CF*, symbol $\Uparrow$ denotes an increasing address order, from address 0 to $n-1$, as long as symbol $\Downarrow$ denotes a reversed address order, from address $n-1$ to 0. Other permutations of the set of memory cell addresses decrease the effectiveness of the march test.

Proposition 4. *March elements* (1)–(4) *in MT-R3CF are able to activate all simple* 2*-coupling faults.*

Proof: Take an arbitrary set of two cells $S = \{i, j\}$. Fig. 3 shows the triggering transition diagram for this set of cells. We represent a state of the set by the contents of cells i and j. As shown in Fig. 3 march elements (1)–(4) force all the possible transitions in the set of cells (Condition 1 previously defined in Proposition 1 is satisfied). Note that these four march elements perform a minimal number of write operations because each arc in the graph has been traversed only once. □

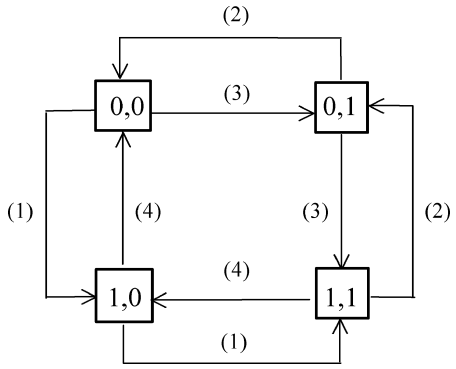


Fig. 3. The Eulerian graph of states for a set of cells $S = \{i, j\}$.

Remark 3. Except for the final read sequence $\Updownarrow (r)$, the first part of *MT-R3CF*, $\langle \Updownarrow (w_0)^{(0)}; \Uparrow (rw_1)^{(1)}; \Uparrow (rw_0)^{(2)}; \Downarrow (rw_1)^{(3)}; \Downarrow (rw_0)^{(4)} \rangle$, is a test sequence identical with the test March C- [11], that is an optimal march test for simple 2-coupling faults. Without a final read sequence, this test sequence does not detect all simple 2-coupling faults, but the next test sequences (5) and (6) detect all simple 2-coupling faults activated but undetected by march elements (1)–(4).

In the following section we analyse the ability of this new march test to detect reduced 3-coupling faults.

First, we show that *MT-R3CF* detects all simple faults and then, based on this result, we analyse the ability of the test to detect the interacting linked coupling faults.

I. Simple 3-Coupling Faults

Theorem 1. *The march test MT-R3CF detects all simple reduced 3-coupling faults.*

Proof: Consider an arbitrary triple set of cells $S = \{i, j, k\}$ that corresponds with one of the patterns P_1, P_2, P_3, P_4, P_5 and P_6 (see Fig. 1). As shown in Fig. 1, we refer to the cells in S by i, j and k taking into account the order in which these cells are checked during the memory testing. Cells i, j and k are checked in this order when the memory is tested in ascending order ($\Uparrow$), and in reverse order, when the memory is tested in descending order ($\Downarrow$). We must show that *MT-R3CF* activates and observes any simple fault that affects the set of cells S.

I.1 *MT-R3CF* activates any fault which affects any triple set of cells S

According to Condition 1 previously defined in Proposition 1, we must prove that *MT-R3CF* covers the Eulerian graph of states for a set of cells S, in all the six cases.

First, note that the sequence $\Updownarrow$ (w_0) loads into any triple set of cells the initial state $\langle 0, 0, 0\rangle$. Applying the march elements (1)–(4) *MT-R3CF* forces the transitions marked with solid lines in the Eulerian graph presented in Fig. 4, in every set of cells in the memory under test regardless of the pattern.

The test sequences I_1, I_2, I_3 and I_4 ensure in a set of cells S, depending on the pattern (see Figs. 1 and 2), the initial states presented in Table 3. As highlighted in Table 3, the test sequences ensure the initial states $\langle 0, 1, 0\rangle$ and $\langle 1, 0, 1\rangle$ for all the six patterns.

Table 3. The initial states for a set of cells $S = \{i, j, k\}$.

	I_1 and I_3	I_2 and I_4
P_1	$\langle \mathbf{1}, \mathbf{0}, \mathbf{1}\rangle$, $\langle \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle$	$\langle 0, 1, 1\rangle$, $\langle 1, 0, 0\rangle$
P_2	$\langle 1, 1, 0\rangle$, $\langle 0, 0, 1\rangle$	$\langle \mathbf{1}, \mathbf{0}, \mathbf{1}\rangle$, $\langle \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle$
P_3	$\langle \mathbf{1}, \mathbf{0}, \mathbf{1}\rangle$, $\langle \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle$	$\langle 1, 1, 0\rangle$, $\langle 0, 0, 1\rangle$
P_4	$\langle 0, 1, 1\rangle$, $\langle 1, 0, 0\rangle$	$\langle \mathbf{1}, \mathbf{0}, \mathbf{1}\rangle$, $\langle \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle$
P_5	$\langle \mathbf{1}, \mathbf{0}, \mathbf{1}\rangle$, $\langle \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle$	$\langle 0, 1, 0\rangle$, $\langle 1, 0, 1\rangle$
P_6	$\langle 0, 0, 0\rangle$, $\langle 1, 1, 1\rangle$	$\langle \mathbf{1}, \mathbf{0}, \mathbf{1}\rangle$, $\langle \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle$

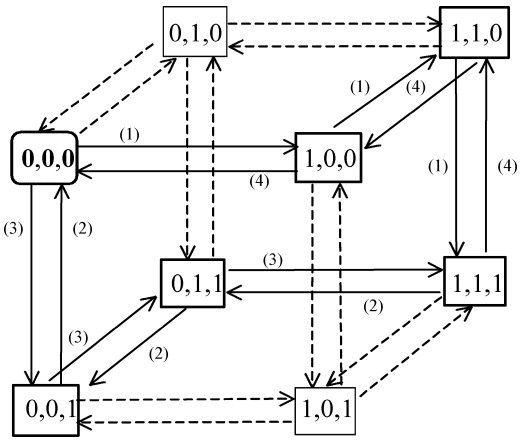


Fig. 4. The Eulerian graph of states for a set of cells $S = \{i, j, k\}$ and the transitions carried out by the march elements (1)–(4).

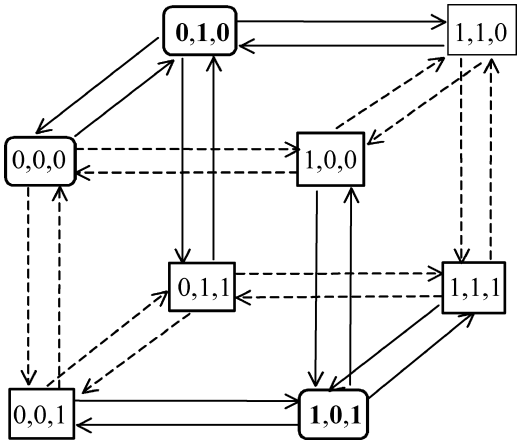


Fig. 5. The transitions carried out in a set of cells $S = \{i, j, k\}$ by $\Uparrow$ (rw_crw_c) for the initial states $\langle 0, 1, 0\rangle$ and $\langle 1, 0, 1\rangle$.

In the Eulerian graph two adjacent nodes (states) have only one bit changed and two non-adjacent nodes have at least two bits changed. Applying $\Uparrow$ (rw_crw_c) three adjacent nodes are visited and, finally, the set of cells returns to the initial state (the state that the sequence started with). The transitions forced by $\Uparrow$ (rw_crw_c) in a set of cells S initialised with $\langle 0,1,0\rangle$ and $\langle 1,0,1\rangle$, respectively, are highlighted (marked with solid line) in Fig. 5. The transitions marked with solid lines in Figs. 4 and 5 show that the Eulerian graph of states is completely covered in all the six cases.

Table 4. The operations carried out in a set of cells $S = \{i, j, k\}$ by the march test *MT-R3CF*.

Operations	Comments
$\ldots r^i w_1^i \ldots r^j w_1^j \ldots r^k w_1^k \ldots r^i w_0^i \ldots r^j w_0^j \ldots r^k w_0^k \ldots$	March elements (1) and (2)
$\ldots r^k w_1^k \ldots r^j w_1^j \ldots r^i w_1^i \ldots r^k w_0^k \ldots r^j w_0^j \ldots r^i w_0^i \ldots$	March elements (3) and (4)
$\ldots [r^i w_c^i] \ldots [r^j w_c^j] \ldots [r^k w_c^k] \ldots$	Change to 2nd background (I_1)
$\ldots r^i w_c^i r^i w_c^i \ldots r^j w_c^j r^j w_c^j \ldots r^k w_c^k r^k w_c^k \ldots$	March element (6)
$\ldots [r^i w_c^i] \ldots [r^j w_c^j] \ldots [r^k w_c^k] \ldots$	Change to 3rd background (I_2)
$\ldots r^i w_c^i r^i w_c^i \ldots r^j w_c^j r^j w_c^j \ldots r^k w_c^k r^k w_c^k \ldots$	March element (8)
$\ldots [r^i w_c^i] \ldots [r^j w_c^j] \ldots [r^k w_c^k] \ldots$	Change to 4th background (I_3)
$\ldots r^i w_c^i r^i w_c^i \ldots r^j w_c^j r^j w_c^j \ldots r^k w_c^k r^k w_c^k \ldots$	March element (10)
$\ldots [r^i w_c^i] \ldots [r^j w_c^j] \ldots [r^k w_c^k] \ldots$	Change to 5th background (I_4)
$\ldots r^i w_c^i r^i w_c^i \ldots r^j w_c^j r^j w_c^j \ldots r^k w_c^k r^k w_c^k \ldots$	March element (12)
$\ldots r^i \ldots r^j \ldots r^k \ldots$	Final read sequence

I.2 *MT-R3CF* observes any simple fault activated in set S

Table 4 presents the operations carried out in a set of cells $S = \{i, j, k\}$ during the memory testing, except on the writes for the first initialisation. The operations enclosed inside brackets sometimes are made, or sometimes are not made, depending on the set of cells. We can easily check in Table 4 that Conditions 2 and 3, previously defined in Proposition 1, are also satisfied. □

II. Interacting Linked 3-Coupling Faults

In this section we show that *MT-R3CF* is able to detect nearly all of the interacting linked faults of reduced 3-coupling. Some interacting linked faults can not be detected even if *MT-R3CF* detects all simple faults.

We proved in the previous section that *MT-R3CF* is able to activate any reduced 3-coupling fault. Now, we must check the ability of the test to observe, in time, the activated faults. This analysis takes into consideration only the interacting linked faults modelled by two simple 3-coupling faults. For the interacting linked faults modelled by one simple 2-coupling fault and one simple 3-coupling fault the approach is similar and simpler.

Consider a victim cell k. For other two cases when cell i or cell j is a victim cell the analysis is similar.

Generally, a fault which affects the initialisation of a cell is detected by the next march element applied to this cell because a march element starts with a read operation. A fault undetected at that moment will be definitely reactivated later. For example, take a test sequence that changes the background and must bring the set $S = \{i, j, k\}$ into the state (x, y, z). An arbitrary fault prevents the change of state in cell k from $\bar{z}$ to z. Because cell k is the last cell written in set S only a k-SCF may cause the initialisation of cell k. Fig. 6 shows that even if this k-SCF is not detected by the first read operation of cell k, because of another $i \rightarrow k$ or $j \rightarrow k$ TCF, it is activated again by the operations $w_c^k r^k w_c^k$.

Based on these considerations we assume that all the memory initialisations are successfully performed in order to simplify the analysis of the interacting linked faults.

Because the number of possible interacting linked faults in a set of 3-coupled cells is very large, we must limit the analysis to faults liable to remain undetected by *MT-R3CF*. Thus, the analysis requires first

- to identify and model the interacting linked faults in a set of 3-coupled cells, that can possibly remain undetected by *MT-R3CF*, and then,
- to check if *MT-R3CF* is able to detect these interacting linked faults.

II.1. Identification and modelling of the interacting linked faults which affect cell k

We concentrate on the situations in which two or more write operations are carried out in a set of cells $S = \{i, j, k\}$ between two consecutive read operations

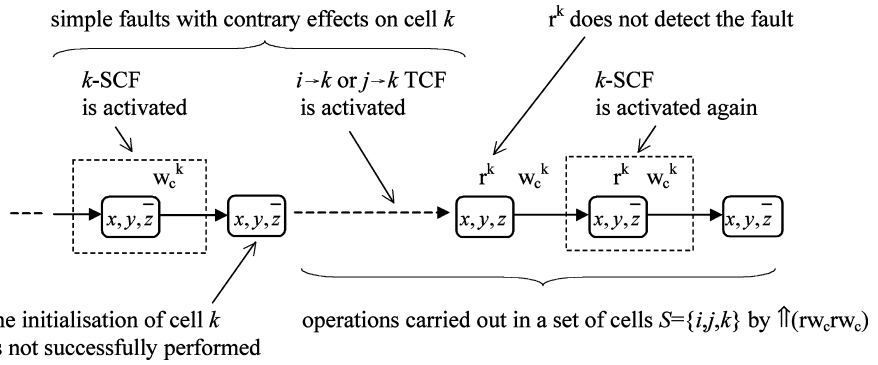


Fig. 6. The fault k-SCF which affects the initialisation is reactivated by the next march element.

of cell k. As shown in Table 4, five types of interacting linked faults which affect cell k must be considered. These interacting linked faults are modelled by two simple faults, as follows:

(a) one simple k-SCF and one simple $i \rightarrow k$ TCF;
(b) one simple k-SCF and one simple $j \rightarrow k$ TCF;
(c) one simple $i \rightarrow k$ TCF and one simple $j \rightarrow k$ TCF;
(d) two simple $i \rightarrow k$ TCFs;
(e) two simple $j \rightarrow k$ TCFs.

Case (a) We must determine all pairs of simple faults k-SCF and $i \rightarrow k$ TCF with contrary effects on cell k. Tables 5 and 6 present all simple k-SCFs and $i \rightarrow k$ TCFs, respectively. Consequently, we must determine all combinations between one simple fault in Table 5 (F_1) and other one in Table 6 (F_2), in which F_1 contains symbol D and F_2 symbol $\bar{D}$, or vice-versa. For example,

$$F_1 = \langle 0, 0, \uparrow: 0, 0, D\rangle \quad \text{and}$$
$$F_2 = \langle \uparrow, 0, 0 : 1, 0, \bar{D}\rangle, \langle \downarrow, 0, 0 : 0, 0, \bar{D}\rangle,$$
$$\langle \uparrow, 1, 0 : 1, 1, \bar{D}\rangle, \text{or } \langle \downarrow, 1, 0 : 0, 1, \bar{D}\rangle.$$

Regarding the k-SCFs we note that some faults are detected by the test sequence $w_c^k r^k$ when the cell k is checked immediately after the triggering transition. Table 5 highlights two simple k-SCFs, $\langle 0, 1, \uparrow: 0, 1, D\rangle$ and $\langle 1, 0, \downarrow: 1, 0, \bar{D}\rangle$, that are detected by the operations $w_c^k r^k$ for all the six patterns P_1–P_6 . These two simple k-SCFs can be ignored in this analysis. All the interacting linked faults we must consider in this case are presented in Table 8, rows 1–24.

Case (b) This is similar to (Case a). We must consider the combinations between one simple fault in Table 5 (F_1) and other one in Table 7 (F_2), in which F_1 contains symbol D and F_2 symbol $\bar{D}$, or vice-versa. These interacting linked faults are presented in Table 8, rows 25–48.

Table 5. Simple k-state coupling faults (k-SCFs).

	Initial states and patterns for which $w_c^k r^k$ activates and observes the fault	
Vector F	Initial state	Pattern
$\langle 0, 0, \uparrow: 0, 0, D\rangle$	$\langle 0, 0, 0\rangle$	P_6
$\langle 0, 0, \downarrow: 0, 0, \bar{D}\rangle$	$\langle 0, 0, 1\rangle$	P_2, P_3
$\langle \mathbf{0}, \mathbf{1}, \uparrow : \mathbf{0}, \mathbf{1}, \boldsymbol{D}\rangle$	$\langle \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle$	$\boldsymbol{P_1, P_2, P_3, P_4, P_5, P_6}$
$\langle 0, 1, \downarrow: 0, 1, \bar{D}\rangle$	$\langle 0, 1, 1\rangle$	P_1, P_4
$\langle 1, 0, \uparrow: 1, 0, D\rangle$	$\langle 1, 0, 0\rangle$	P_1, P_4
$\langle \mathbf{1}, \mathbf{0}, \downarrow: \mathbf{1}, \mathbf{0}, \bar{\boldsymbol{D}}\rangle$	$\langle \mathbf{1}, \mathbf{0}, \mathbf{1}\rangle$	$\boldsymbol{P_1, P_2, P_3, P_4, P_5, P_6}$
$\langle 1, 1, \uparrow: 1, 1, D\rangle$	$\langle 1, 1, 0\rangle$	P_2, P_3
$\langle 1, 1, \downarrow: 1, 1, \bar{D}\rangle$	$\langle 1, 1, 1\rangle$	P_6

Table 6. Simple $i \rightarrow k$ TCFs.

Nr.	Vector F
1	$\langle \uparrow, 0, 0 : 1, 0, \bar{D}\rangle$
2	$\langle \downarrow, 0, 0 : 0, 0, \bar{D}\rangle$
3	$\langle \uparrow, 0, 1 : 1, 0, D\rangle$
4	$\langle \downarrow, 0, 1 : 0, 0, D\rangle$
5	$\langle \uparrow, 1, 0 : 1, 1, \bar{D}\rangle$
6	$\langle \downarrow, 1, 0 : 0, 1, \bar{D}\rangle$
7	$\langle \uparrow, 1, 1 : 1, 1, D\rangle$
8	$\langle \downarrow, 1, 1 : 0, 1, D\rangle$

Table 7. Simple $j \rightarrow k$ TCFs.

Nr.	Vector F
1	$\langle 0, \uparrow, 0 : 0, 1, \bar{D} \rangle$
2	$\langle 0, \downarrow, 0 : 0, 0, \bar{D} \rangle$
3	$\langle 0, \uparrow, 1 : 0, 1, D \rangle$
4	$\langle 0, \downarrow, 1 : 0, 0, D \rangle$
5	$\langle 1, \uparrow, 0 : 1, 1, \bar{D} \rangle$
6	$\langle 1, \downarrow, 0 : 1, 0, \bar{D} \rangle$
7	$\langle 1, \uparrow, 1 : 1, 1, D \rangle$
8	$\langle 1, \downarrow, 1 : 1, 0, D \rangle$

Case (c) We must have in view all combinations between one simple fault in Table 6 (F_1) and another one in Table 7 (F_2) in which F_1 contains symbol D and F_2 symbol $\bar{D}$, or vice-versa. These interacting linked faults are presented in Table 8, rows 49–80.

Case (d) We must consider all combinations of simple $i \rightarrow k$ TCFs with contrary effects on cell k which can be activated by the test sequence $r^i w_c^i r^i w_c^i$. Thus, two vectors F_1 and F_2 in Table 6 must satisfy three conditions:

- F_1 contains symbol $\uparrow$ and F_2 symbol $\downarrow$, or vice-versa;
- F_1 contains symbol D and F_2 symbol $\bar{D}$, or vice-versa;
- F_1 and F_2 contain the same value regarding cell j.

The interacting linked faults we must consider for this case are given in Table 8, rows 81–84.

Case (e) This is similar to (Case d). We must consider all pairs of vectors F_1 and F_2 in Table 7 that satisfy the conditions previously defined. These interacting linked faults are given in Table 8, rows 85–88.

II.2. The analysis of the interacting linked faults previously defined

Table 8 presents the whole set of interacting linked faults which affect cell k that could possibly remain undetected by *MT-R3CF*. To check if the march test *MT-R3CF* is able to detect an interacting linked fault, according to Proposition 3 we identified first an initial state which allows, in the next test sequence, one simple fault to be activated and the other one not to be. Table 8 gives the test sequence which detects an interacting linked fault, for each pattern of coupled cells (in the hypothesis in which the fault is not activated by $\Updownarrow (w_0)$). For example, independent of the set of coupled cells, the first interacting linked fault is detected by the test sequence $\Uparrow(\mathrm{rw}_1)$, when F_2 is activated and F_1 is not. The second interacting linked fault is detected by sequence (6) if the set of coupled cells corresponds with pattern P_1, P_2 or P_3, by sequence (8) for pattern P_4, and by sequence (10) for pattern P_5 or P_6.

As shown in Table 8, *MT-R3CF* does not detect the following three interacting linked faults which affect cell k:

- Fault 60, modelled by $F_1 = \langle \uparrow, 0, 1 : 1, 0, D \rangle$ and $F_2 = \langle 1, \downarrow, 0 : 1, 0, \bar{D} \rangle$, for P_5 and P_6,
- Fault 65, modelled by $F_1 = \langle \uparrow, 1, 0 : 1, 1, \bar{D} \rangle$ and $F_2 = \langle 0, \uparrow, 1 : 0, 1, D \rangle$, for P_5,
- Fault 88, modelled by $F_1 = \langle 1, \downarrow, 0 : 1, 0, \bar{D} \rangle$ and $F_2 = \langle 1, \uparrow, 1 : 1, 1, D \rangle$, for P_2.

Finally we present other two interacting linked faults undetected by *MT-R3CF*, when cell i is a victim cell:

- Fault modelled by $F_1 = \langle 0, \downarrow, 0 : \bar{D}, 0, 0 \rangle$ and $F_2 = \langle 1, 1, \downarrow : D, 1, 0 \rangle$, for P_1, P_4 and P_6,
- Fault modelled by $F_1 = \langle 1, \uparrow, 1 : D, 1, 1 \rangle$ and $F_2 = \langle 0, 0, \uparrow : \bar{D}, 0, 1 \rangle$, for P_1, P_4 and P_6.

5. March Test for Reduced 4-Coupling Faults

In this section we propose the following new march test of length $41n$ for the reduced model of 4-coupling faults considered in this paper.

MT-R4CF

$$\begin{aligned} = \langle & I_1; \Uparrow (r w_c r w_c); I_2; \Uparrow (r w_c r w_c); \\ & I_3; \Uparrow (r w_c r w_c); I_4; \Uparrow (r w_c r w_c); I_5; \Uparrow (r w_c r w_c); \\ & I_6; \Uparrow (r w_c r w_c); I_7; \Uparrow (r w_c r w_c); I_8; \Uparrow (r w_c r w_c); \\ & \Updownarrow (R) \rangle \end{aligned} \tag{2}$$

where I_1, I_2, I_3, I_4, I_5, I_6, I_7 and I_8 are test sequences which initialise the memory as follows: I_1 and I_3 initialise all the cells with 0 and 1, respectively (solid data background); I_2 initialises the odd columns with 0 and the even columns with 1, and I_4 vice versa (column-stripe data background); I_5 initialises the odd rows with 0 and the even rows with 1, and I_7 vice versa (row-stripe data background); I_6 and I_8 initialise the memory with a checkerboard data background and its complement (Fig. 7).

Table 8. The interacting linked coupling faults which affect cell k and the test sequences which detect them.

Nr.	Case	Interacting linked faults F_1	Interacting linked faults F_2	Test sequence that detects the fault for $P_1, P_2, P_3, P_4, P_5, P_6$
1	a	$\langle 0, 0, \uparrow : 0, 0, D\rangle$	$\langle \uparrow, 0, 0 : 1, 0, \bar{D}\rangle$	1
2		”	$\langle \uparrow, 1, 0 : 1, 1, \bar{D}\rangle$	6, 6, 6, 8, 10, 10
3		”	$\langle \downarrow, 0, 0 : 0, 0, \bar{D}\rangle$	4
4		”	$\langle \downarrow, 1, 0 : 0, 1, \bar{D}\rangle$	4
5		$\langle 0, 0, \downarrow : 0, 0, \bar{D}\rangle$	$\langle \uparrow, 0, 1 : 1, 0, D\rangle$	3
6		”	$\langle \uparrow, 1, 1 : 1, 1, D\rangle$	3
7		”	$\langle \downarrow, 0, 1 : 0, 0, D\rangle$	3
8		”	$\langle \downarrow, 1, 1 : 0, 1, D\rangle$	2
9		$\langle 0, 1, \downarrow : 0, 1, \bar{D}\rangle$	$\langle \uparrow, 0, 1 : 1, 0, D\rangle$	10, 9, 7, 9, 8, 6
10		”	$\langle \uparrow, 1, 1 : 1, 1, D\rangle$	4
11		”	$\langle \downarrow, 0, 1 : 0, 0, D\rangle$	10, 9, 7, 9, 8, 6
12		”	$\langle \downarrow, 1, 1 : 0, 1, D\rangle$	2
13		$\langle 1, 0, \uparrow : 1, 0, D\rangle$	$\langle \uparrow, 0, 0 : 1, 0, \bar{D}\rangle$	1
14		”	$\langle \uparrow, 1, 0 : 1, 1, \bar{D}\rangle$	6, 6, 6, 6, 9, 6
15		”	$\langle \downarrow, 0, 0 : 0, 0, \bar{D}\rangle$	6, 6, 6, 6, 5, 5
16		”	$\langle \downarrow, 1, 0 : 0, 1, \bar{D}\rangle$	6, 6, 6, 6, 9, 6
17		$\langle 1, 1, \uparrow : 1, 1, D\rangle$	$\langle \uparrow, 0, 0 : 1, 0, \bar{D}\rangle$	1
18		”	$\langle \uparrow, 1, 0 : 1, 1, \bar{D}\rangle$	2
19		”	$\langle \downarrow, 0, 0 : 0, 0, \bar{D}\rangle$	2
20		”	$\langle \downarrow, 1, 0 : 0, 1, \bar{D}\rangle$	6, 6, 6, 8, 8, 10
21		$\langle 1, 1, \downarrow : 1, 1, \bar{D}\rangle$	$\langle \uparrow, 0, 1 : 1, 0, D\rangle$	6, 8, 6, 12, 6, 6
22		”	$\langle \uparrow, 1, 1 : 1, 1, D\rangle$	4
23		”	$\langle \downarrow, 0, 1 : 0, 0, D\rangle$	10, 8, 8, 12, 6, 6
24		”	$\langle \downarrow, 1, 1 : 0, 1, D\rangle$	2
25	b	$\langle 0, 0, \uparrow : 0, 0, D\rangle$	$\langle 0, \uparrow, 0 : 0, 1, \bar{D}\rangle$	6, 8, 6, 8, 10, 10
26		”	$\langle 1, \uparrow, 0 : 1, 1, \bar{D}\rangle$	1
27		”	$\langle 0, \downarrow, 0 : 0, 0, \bar{D}\rangle$	4
28		”	$\langle 1, \downarrow, 0 : 1, 0, \bar{D}\rangle$	4
29		$\langle 0, 0, \downarrow : 0, 0, \bar{D}\rangle$	$\langle 0, \uparrow, 1 : 0, 1, D\rangle$	3
30		”	$\langle 1 \uparrow, 1 : 1, 1, D\rangle$	3
31		”	$\langle 0, \downarrow, 1 : 0, 0, D\rangle$	2
32		”	$\langle 1, \downarrow, 1 : 1, 0, D\rangle$	3
33		$\langle 0, 1, \downarrow : 0, 1, \bar{D}\rangle$	$\langle 0, \uparrow, 1 : 0, 1, D\rangle$	4
34		”	$\langle 1 \uparrow, 1 : 1, 1, D\rangle$	10, 9, 7, 9, 6, 6
35		”	$\langle 0, \downarrow, 1 : 0, 0, D\rangle$	2
36		”	$\langle 1, \downarrow, 1 : 1, 0, D\rangle$	10, 9, 7, 9, 6, 6
37		$\langle 1, 0, \uparrow : 1, 0, D\rangle$	$\langle 0, \uparrow, 0 : 0, 1, \bar{D}\rangle$	6, 8, 6, 6, 9, 6
38		”	$\langle 1, \uparrow, 0 : 1, 1, \bar{D}\rangle$	1
39		”	$\langle 0, \downarrow, 0 : 0, 0, \bar{D}\rangle$	6, 8, 6, 6, 9, 6
40		”	$\langle 1, \downarrow, 0 : 1, 0, \bar{D}\rangle$	6, 6, 6, 6, 5, 5
41		$\langle 1, 1, \uparrow : 1, 1, D\rangle$	$\langle 0, \uparrow, 0 : 0, 1, \bar{D}\rangle$	2
42		”	$\langle 1, \uparrow, 0 : 1, 1, \bar{D}\rangle$	1
43		”	$\langle 0, \downarrow, 0 : 0, 0, \bar{D}\rangle$	6, 6, 6, 8, 6, 10
44		”	$\langle 1, \downarrow, 0 : 1, 0, \bar{D}\rangle$	2

(*Continued on next page.*)

Table 8. (*Continued*).

Nr.	Interacting linked faults			Test sequence that detects the fault for P_1, P_2, P_3, P_4, P_5, P_6
	Case	F_1	F_2	
45		$\langle 1, 1, \downarrow : 1, 1, \bar{D}\rangle$	$\langle 0, \uparrow, 1 : 0, 1, D\rangle$	4
46		”	$\langle 1 \uparrow, 1 : 1, 1, D\rangle$	6, 8, 6, 12, 6, 5
47		”	$\langle 0, \downarrow, 1 : 0, 0, D\rangle$	2
48		”	$\langle 1, \downarrow, 1 : 1, 0, D\rangle$	10, 8, 10, 12, 6, 6
49	c	$\langle \uparrow, 0, 0 : 1, 0, \bar{D}\rangle$	$\langle 0, \uparrow, 1 : 0, 1, D\rangle$	1
50		”	$\langle 0, \downarrow, 1 : 0, 0, D\rangle$	1
51		”	$\langle 1, \uparrow, 1 : 1, 1, D\rangle$	10, 12, 10, 12, 6, 5
52		”	$\langle 1, \downarrow, 1 : 1, 0, D\rangle$	1
53		$\langle \downarrow, 0, 0 : 0, 0, \bar{D}\rangle$	$\langle 0, \uparrow, 1 : 0, 1, D\rangle$	4
54		”	$\langle 0, \downarrow, 1 : 0, 0, D\rangle$	2
55		”	$\langle 1, \uparrow, 1 : 1, 1, D\rangle$	6, 12, 6, 8, 5, 5
56		”	$\langle 1, \downarrow, 1 : 1, 0, D\rangle$	6, 12, 6, 8, 5, 5
57		$\langle \uparrow, 0, 1 : 1, 0, D\rangle$	$\langle 0, \uparrow, 0 : 0, 1, \bar{D}\rangle$	6, 8, 6, 8, 8, 6
58		”	$\langle 0, \downarrow, 0 : 0, 0, \bar{D}\rangle$	6, 8, 6, 8, 8, 6
59		”	$\langle 1, \uparrow, 0 : 1, 1, \bar{D}\rangle$	1
60		”	$\langle 1, \downarrow, 0 : 1, 0, \bar{D}\rangle$	6, 6, 6, 6, −, −
61		$\langle \downarrow, 0, 1 : 0, 0, D\rangle$	$\langle 0, \uparrow, 0 : 0, 1, \bar{D}\rangle$	6, 8, 6, 8, 8, 6
62		”	$\langle 0, \downarrow, 0 : 0, 0, \bar{D}\rangle$	6, 8, 6, 8, 8, 6
63		”	$\langle 1, \uparrow, 0 : 1, 1, \bar{D}\rangle$	1
64		”	$\langle 1, \downarrow, 0 : 1, 0, \bar{D}\rangle$	8, 6, 8, 6, 9, 9
65		$\langle \uparrow, 1, 0 : 1, 1, \bar{D}\rangle$	$\langle 0, \uparrow, 1 : 0, 1, D\rangle$	8, 6, 8, 10, −, 9
66		”	$\langle 0, \downarrow, 1 : 0, 0, D\rangle$	2
67		”	$\langle 1, \uparrow, 1 : 1, 1, D\rangle$	6, 8, 6, 8, 6, 6
68		”	$\langle 1, \downarrow, 1 : 1, 0, D\rangle$	6, 8, 6, 8, 6, 6
69		$\langle \downarrow, 1, 0 : 0, 1, \bar{D}\rangle$	$\langle 0, \uparrow, 1 : 0, 1, D\rangle$	4
70		”	$\langle 0, \downarrow, 1 : 0, 0, D\rangle$	2
71		”	$\langle 1, \uparrow, 1 : 1, 1, D\rangle$	6, 8, 6, 8, 6, 6
72		”	$\langle 1, \downarrow, 1 : 1, 0, D\rangle$	6, 8, 6, 8, 6, 6
73		$\langle \uparrow, 1, 1 : 1, 1, D\rangle$	$\langle 0, \uparrow, 0 : 0, 1, \bar{D}\rangle$	4
74		”	$\langle 0, \downarrow, 0 : 0, 0, \bar{D}\rangle$	4
75		”	$\langle 1, \uparrow, 0 : 1, 1, \bar{D}\rangle$	1
76		”	$\langle 1, \downarrow, 0 : 1, 0, \bar{D}\rangle$	4
77		$\langle \downarrow, 1, 1 : 0, 1, D\rangle$	$\langle 0, \uparrow, 0 : 0, 1, \bar{D}\rangle$	2
78		”	$\langle 0, \downarrow, 0 : 0, 0, \bar{D}\rangle$	6, 8, 6, 8, 6, 10
79		”	$\langle 1, \uparrow, 0 : 1, 1, \bar{D}\rangle$	1
80		”	$\langle 1, \downarrow, 0 : 1, 0, \bar{D}\rangle$	2
81	d	$\langle \uparrow, 0, 0 : 1, 0, \bar{D}\rangle$	$\langle \downarrow, 0, 1 : 0, 0, D\rangle$	1
82		$\langle \downarrow, 0, 0 : 0, 0, \bar{D}\rangle$	$\langle \uparrow, 0, 1 : 1, 0, D\rangle$	6, 12, 6, 8, 8, 6
83		$\langle \uparrow, 1, 0 : 1, 1, \bar{D}\rangle$	$\langle \downarrow, 1, 1 : 0, 1, D\rangle$	2
84		$\langle \downarrow, 1, 0 : 0, 1, \bar{D}\rangle$	$\langle \uparrow, 1, 1 : 0, 1, D\rangle$	4
85	e	$\langle 0, \uparrow, 0 : 0, 1, \bar{D}\rangle$	$\langle 0, \downarrow, 1 : 0, 0, D\rangle$	2
86		$\langle 0, \downarrow, 0 : 0, 0, \bar{D}\rangle$	$\langle 0, \uparrow, 1 : 0, 1, D\rangle$	4
87		$\langle 1, \uparrow, 0 : 1, 1, \bar{D}\rangle$	$\langle 1, \downarrow, 1 : 1, 0, D\rangle$	1
88		$\langle 1, \downarrow, 0 : 1, 0, \bar{D}\rangle$	$\langle 1, \uparrow, 1 : 1, 1, D\rangle$	6, −, 6, 6, 6, 5

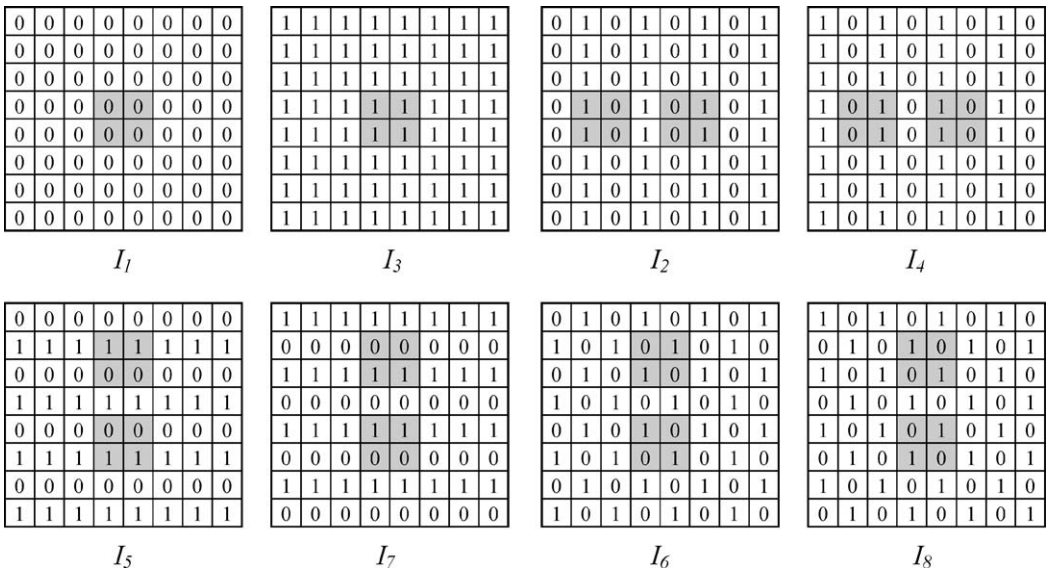


Fig. 7. Data background used by *MT-R4CF* and the initial states for a set of 4-coupled cells.

MT-R4CF contains an alternating series of background changes and march elements. Note that when changing from one background to the next, only the cells that must change states are written. Also, each write operation is preceded by a read operation. We can observe in Fig. 7 that any background change affects only a half of cells. Each test sequence I_1, I_2, I_3, I_4, I_5, I_6, I_7 or I_8 performs $\frac{n}{2}$ read operations and $\frac{n}{2}$ write operations. Consequently, *MT-R4CF* contains $41n$ operations.

Regarding the march test *MT-R4CF*, note that for the $\Uparrow$ address order any sequence may be used, as long as the $\Downarrow$ address order uses the exact inverse address sequence.

Theorem 2. *MT-R4CF detects all simple reduced* 4*-coupling faults.*

Proof: Consider an arbitrary set of four cells $S = \{i, j, k, l\}$ which forms a square pattern.

1. *MT-R4CF* activates all 4-coupling faults in set S

We show that *MT-R4CF* performs all $4 \cdot 2^4$ possible transitions in the set of cells according to Proposition 2 in Section 3. The sequences I_1, I_2, I_3, I_4, I_5, I_6, I_7 and I_8 load into S (see Fig. 7) the eight different initial states presented in Table 9. Every pair of initial states in Table 9 has at least two bits changed. Consequently, in the Eulerian graph of states, the nodes associated with these initial states are not adjacent nodes.

Applying the march element $\Uparrow (rw_crw_c)$ eight different transitions are forced in set S and, finally, the set of cells is left into the initial state (in the Eulerian graph four adjacent nodes are visited, going and coming back). For example, the transitions forced in S starting from the state $\langle 0, 0, 0, 0\rangle$ are shown in Fig. 8.

Because *MT-R4CF* applies the march element $\Uparrow (rw_crw_c)$ eight times starting from different initial states, and because every pair of initial states has at least two bits changed, then 64 distinct transitions are forced in set S. *MT-R4CF* covers the Eulerian graph of states and, consequently, is able to activate any fault in the set of 4-coupled cells.

2. *MT-R4CF* observes any simple fault activated in set S

Table 9. The initial states for a set of cells $S = \{i, j, k, l\}$.

I_1	I_3	I_2 and I_4	I_5 and I_7	I_6 and I_8
$\langle 0, 0, 0, 0\rangle$	$\langle 1, 1, 1, 1\rangle$	$\langle 1, 0, 1, 0\rangle$, $\langle 0, 1, 0, 1\rangle$	$\langle 1, 1, 0, 0\rangle$, $\langle 0, 0, 1, 1\rangle$	$\langle 0, 1, 1, 0\rangle$, $\langle 1, 0, 0, 1\rangle$

Table 10. The operations carried out in a set of cells $S = \{i, j, k, l\}$ by the march test *MT-R4CF*.

Operations	Comments
$\ldots r^i w_c^i r^i w_c^i \ldots r^j w_c^j r^j w_c^j \ldots r^k w_c^k r^k w_c^k \ldots r^l w_c^l r^l w_c^l \ldots$	March element
$\ldots [r^i w_c^i] \ldots [r^j w_c^j] \ldots [r^k w_c^k] \ldots [r^l w_c^l] \ldots$	Change to the next background
$\ldots r^i w_c^i r^i w_c^i \ldots r^j w_c^j r^j w_c^j \ldots r^k w_c^k r^k w_c^k \ldots r^l w_c^l r^l w_c^l \ldots$	March element
$\ldots$	$\ldots$
$\ldots r^i w_c^i r^i w_c^i \ldots r^j w_c^j r^j w_c^j \ldots r^k w_c^k r^k w_c^k \ldots r^l w_c^l r^l w_c^l \ldots$	March element
$\ldots r^i \ldots r^j \ldots r^k \ldots r^l \ldots$	Final read sequence

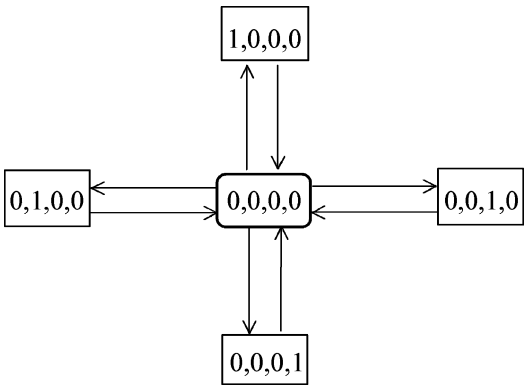


Fig. 8. The transitions carried out in a set of cells $S = \{i, j, k, l\}$ by $\Uparrow (r w_c r w_c)$ starting from the initial state $\langle 0, 0, 0, 0 \rangle$.

Table 10 shows the operations carried out in a set of cells $S = \{i, j, k, l\}$. Analysing these operations we can check easily that Conditions 2 and 3 previously defined in Section 3 are satisfied. □

Based on the proof of Theorem 2 we derive a lower bound on the length of any test that detects simple reduced 4-coupling faults.

Corollary. *A memory test needs at least* $34n$ *operations to detect all simple reduced* 4*-coupling faults.*

Proof: Because in every set $S = \{i, j, k, l\}$ all 64 transitions are carried out exactly once, the eight march elements $\Uparrow (r w_c r w_c)$ in Eq. (2) perform a minimal number of writes to activate all simple reduced 4-coupling faults. On the other hand, all reads of these march elements, as well as the reads of the final sequence, are necessary to satisfy Conditions 2 and 3, previously defined in Section 3. Consequently, a memory test needs at least $34n$ operations (including n operations for memory initialisation) to detect all simple reduced 4-coupling faults. □

Remark 4. With $41n$ operations we can say that *MT-R4CF* is a near-optimal test for this reduced model of 4-coupling.

6. Simulation Results

To compare the effectiveness of the march tests proposed in this paper with other published tests, we present in this section simulation results regarding the ability of the tests to detect v-coupling faults in our models. The following published tests have been considered for the simulation study:

- March test with $38n$ operations given by Caşcaval and Bennett [1] (*CB* in this paper);
- March test with $36n$ operations given by Papachristou and Sahgal [6] (*PS*(*A*) in this paper);
- Algorithm A with $30n$ operations given by Nair et al. [5] (*NTA*(*A*) in this paper);
- Symmetric *March G* algorithm with $24n$ operations given by van de Goor [11];
- *March LR* with $18n$ operations given by Yarmolik, van de Goor, Gaydadjiev and Mikitjuk [12];
- Test B with $16n$ operations given by Suk and Reddy [9] (*March B* in this paper);
- Algorithm *March C-* with $10n$ operations given by van de Goor [11] as an improved version of March C given by Marinescu [4];
- The test procedure given by Suk and Reddy [8] that requires $165n$ operations, obtained by concatenating the procedures TANPSF1 and TLPNPSF1 (*SR* in this paper);

- Memory tests *NTA*(B), *PS*(B), *S3CTEST2* and *S4CTEST* mentioned in Section 1.

We have evaluated by fault simulation the ability of these test algorithms to detect simple 2-coupling, reduced 3-coupling and reduced 4-coupling faults. Thus, for each test we have calculated the fault coverage as a ratio between the number of faults detected by the test and the total number of simulated faults. In our simulation program the RAM memory under test (RUT) is simulated as an array with n locations. For this experiment we have considered a RUT with 64 storage cells. We use the *TRAP* interrupt of Pentium microprocessor to simulate a permanent fault in the RUT.

1. Simple 2-coupling faults

We have selected six groups of 2-coupled cells, as shown in Fig. 9. For each set of 2-coupled cells $S = \{i, j\}$ we have simulated, one-by-one, the simple faults presented in Table 1 and have checked if the test algorithms are able to detect them. The simulation results regarding the fault coverage of the tests we have checked are given in Table 11.

2. Simple reduced 3-coupling faults

Six sets of 3-coupled cells have been considered, one for each pattern $P_i, i \in \{1, 2, 3, 4, 5, 6\}$ as shown in Fig. 10. For each set, all possible simple 3-coupling faults have been simulated: 24 SCFs and 48 TCFs. Table 2 presents the simple faults which affect cell i (8 SCFs and 16 TCFs) in a set of 3-coupled cells $S = \{i, j, k\}$. In total, we have simulated 432 simple faults and have verified the ability of the test algorithms to detect them.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Fig. 9. The sets of 2-coupled cells used in the simulation study.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Fig. 10. The sets of 3-coupled cells used in the simulation study.

Table 11. Fault coverage of simple 2-coupling faults (expressed as %).

Test algorithm	Length*	Fault coverage
March C−	$10n$	100
March B	$17n$	81.25
March LR	$18n$	100
March G	$24n$	100
NTA(A)	$30n$	100
PS(A)	$37n$	100
CB	$38n$	100
MT-R3CF	$30n$	100
MT-R4CF	$41n$	100
NTA(B)	$n + 32n \log_2 n$	100
PS(B)	$n + 24n \log_2 n$	100
S3CTEST2	$5n \log_2 n + 5n[\log_2(1 + \log_2 n)] + 11n$	100
SR	$165n$	79.17

*Including the initialisation sequence $\Uparrow(w_0)$.

Table 12. Fault coverage of simple reduced 3-coupling faults (expressed as %).

Test algorithm	Length	Fault coverage
March C–	$10n$	50
March B	$17n$	47.22
March LR	$18n$	62.5
March G	$24n$	62.5
NTA(A)	$30n$	63.89
PS(A)	$37n$	63.89
CB	$38n$	94.91
MT-R3CF	$30n$	100
MT-R4CF	$41n$	97.22
NTA(B)	$n + 32n \log_2 n$	100
PS(B)	$n + 24n \log_2 n$	100
S3CTEST2	$5n \log_2 n + 5n[\log_2(1 + \log_2 n)] + 11n$	100
SR	$165n$	73.38

Table 13. Fault coverage of simple reduced 4-coupling faults (expressed as %).

Test algorithm	Length	Fault coverage
March C–	$10n$	25.39
March B	$17n$	26.95
March LR	$18n$	35.55
March G	$24n$	35.16
NTA(A)	$30n$	37.11
PS(A)	$37n$	37.11
CB	$38n$	75.00
MT-R3CF	$30n$	74.22
MT-R4CF	$41n$	100
NTA(B)	$n + 32n \log_2 n$	75.78
PS(B)	$n + 24n \log_2 n$	75.78
S3CTEST2	$5n \log_2 n + 5n[\log_2(1 + \log_2 n)] + 11n$	93.36
SR	$165n$	54.3
S4CTEST	$10.75n(\log_2 n)^{1.585}$	100

Simulation results regarding the ability of the test algorithms to detect these simple 3-coupling faults are presented in Table 12.

Remark 5. The non-march test *SR*, which detects all single NPSFs as defined in [8], is able to detect only 73.38% of simple faults in our model. This result demonstrates that the model of reduced 3-coupling is not covered by the model of NPSFs.

3. Simple reduced 4-coupling faults

For the set of four coupled cells {28, 29, 36, 37} all the 256 possible simple faults have been simulated: 64 SCFs and 192 TCFs. The simulation results are presented in Table 13.

7. Conclusion

This paper presents two new efficient march test algorithms, *MT-R3CF* and *MT-R4CF*, for reduced 3-coupling and 4-coupling faults in Random-Access Memories. To reduce the length of the tests only the

Table 14. Test time for a 4 Mb and a 64 Mb memory chip (assuming a cycle time of 60 ns).

Test algorithm	MT-R3CF	MT-R4CF	S3CTEST2	S4CTEST
4 Mb	7.55 s	10.32 s	33.34 s	6 min 3 s
64 Mb	2 min	2 min 45 s	10 min 12 s	2 h 6 min

coupled faults between physically adjacent memory cells have been considered. *MT-R3CF* and *MT-R4CF* are march tests of length $30n$ and $41n$, respectively. The first test detects all simple faults and nearly all the interacting linked reduced 3-coupling faults. The second test detects all simple reduced 4-coupling faults and is a near-optimal test for this model. The simulation results presented in this paper confirm the correctness of the march tests *MT-R3CF* and *MT-R4CF*.

Our tests allow test engineers to obtain greatly reduced test time at the cost of reasonable restrictions on the achievable fault coverage. For example, to compare our tests with Cockburn's tests, *S3CTEST2* and *S4CTEST*, Table 14 presents the test time for a 4 Mb and a 64 Mb memory chip, assuming a cycle time of 60 ns. *MT-R3CF* and *MT-R4CF* are march tests adequate for a BIST implementation in embedded RAM.

Both tests given in this paper require the mapping from logical addresses to physical cell locations. If the row and column addresses for the square grid are scrambled in a way unknown to the tester then the effectiveness of these tests can be affected. The fault coverage of simple faults which affect scrambled coupled cells depends on the addresses of the coupled cells. For example, *MT-R3CF* detects all simple 3-coupling faults which affect the set of cells $\{18, 23, 52\}$ in Fig. 10, whereas, for the set of cells $\{12, 36, 52\}$, *MT-R3CF* detects only 83.33% of simple 3-coupling faults. This is the first limitation of the tests proposed in this paper.

References

1. P. Caşcaval and S. Bennett, "Efficient March Test for 3-Coupling Faults in Random Access Memories," *J. Microprocessors and Microsystems*, vol. 24, no. 10, pp. 501–509, 2001.
2. B.F. Cockburn, "Deterministic Tests for Detecting Single V-Coupling Faults in RAMs," *J. Electronic Testing*, vol. 5, no. 1, pp. 91–113, 1994.
3. B.F. Cockburn, "Deterministic Test for Detecting Scrambled Pattern-Sensitive Faults in RAMs," *IEEE Workshop on Memory Technology, Design and Testing*, 1995, pp. 117–122.
4. M. Marinescu, "Simple and Efficient Algorithms for Functional RAM Testing," in *Digest of Papers, 1982 Int'l Test Conf.*, Philadelphia, PA, Nov. 1982, pp. 236–239.
5. R. Nair, S. Thatte, and J. Abraham, "Efficient Algorithms for Testing Semiconductor Random-Access Memories," *IEEE Trans. Comput.*, vol. C-27, no. 6, pp. 572–576, 1978.
6. C. Papachristou and N. Sahgal, "An Improved Method for Detecting Functional Faults in Semiconductor Random Access Memories," *IEEE Trans. Comput.*, vol. C-34, no. 2, pp. 110–116, 1985.
7. J.P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IMB J. Research and Development*, vol. 10, no. 4, pp. 278–291, 1966.
8. D. Suk and S. Reddy, "Test Procedures for a Class of Pattern-Sensitive Faults in Semiconductor Random-Access Memories," *IEEE Trans. Comput.*, vol. C-29, no. 6, pp. 419–429, 1980.
9. D. Suk and S. Reddy, "A March Test for Functional Faults in Semiconductor Random Access Memories," *IEEE Trans. Comput.*, vol. C-30, no. 12, pp. 982–985, 1981.
10. A.J. van de Goor, *Testing Semiconductor Memories, Theory and Practice*, Chichester, UK: John Wiley & Sons, 1991.
11. A.J. van de Goor, "Using March Tests to Test SRAMs," *IEEE Design and Test of Computers*, pp. 8–14, 1993.
12. V.N. Yarmolik, A.J. van de Goor, G.N. Gaydadjiev, and V.G. Mikitjuk, "March LR: A Test for Realistic Linked Faults," *Proc. VLSI Test Symp.*, March 1996, pp. 272–280.

Petru Caşcaval is Assistant Professor in the Department of Computer Engineering at "Gh.Asachi" Technical University of Iaşi, Romania. He received a PhD in Systems Engineering from "Gh.Asachi" Technical University of Iaşi. His research interests include test generation and testable design of digital systems, modeling and simulation, reliability evaluation, and fault-tolerance. As part of this work he is involved in RAMs testing and test generation and fault simulation for both combinational and sequential circuits.

Stuart Bennett is Reader in Automatic Control and Systems Engineering at the University of Sheffield. His current research interests include the design of dependable systems including the incorporation of smart actuators and voting systems. As part of this work he is involved in reliability modeling of mixed hardware/software systems.

Corneliu Huţanu is Professor in the Department of Automatic Control of the "Gh.Asachi" Technical University of Iaşi. His fields of interest include microcontrollers and their applications, testing and design for testability of digital systems, robotics and distributed systems. Huţanu received a BEng degree and a PhD from "Gh.Asachi" Technical University of Iaşi.

ELSEVIER

Microprocessors and Microsystems 24 (2001) 501–509

MICROPROCESSORS AND MICROSYSTEMS

www.elsevier.nl/locate/micpro

Efficient march test for 3-coupling faults in random access memories

P. Caşcaval[a,*], S. Bennett[b,1]

[a]*Department of Computer Science, "Gh. Asachi" Technical University of Iasi, Bd. D. Mangeron, nr.53A, 6600 Iasi, Romania*
[b]*Department of Automatic Control and Systems Engineering, The University of Sheffield, Mappin Street, Sheffield S13JD, UK*

Received 11 January 2000; revised 1 November 2000; accepted 13 November 2000

Abstract

A new efficient march test algorithm for detecting the 3-coupling faults in Random Access Memories (RAM) is given in this paper. To reduce the length of the test algorithm only the 3-coupling faults between physically adjacent memory cells have been considered. The proposed test algorithm needs *38N* operations. We have proved, using an Eulerian graph model, that the algorithm detects all non-interacting coupling faults. This paper also comprises a study about the ability of the algorithm to cover the interacting coupling faults.

Simulation results with regard to the coupling fault coverage of the march tests, obtained based on a fault injection mechanism, are also presented in this paper.

Keywords: Memory testing; Functional faults; Coupling faults; March test; Fault injection

1. Introduction

Rapid developments in semiconductor technology have resulted in continuing growth of larger and denser random access memories (RAM) on a single chip. More time is required to test memories because of their increasing size. On the other hand, because of the increased cell density the nature of the failure mode becomes more complex and subtle. Therefore it is necessary to identify more efficient tests with the ability to detect more complex and subtle faults into the $O(N)$ class of complexity [4].

Test procedures are constrained by two conflicting requirements:

(a) to detect a wide variety of complex faults;
(b) to reduce the number of memory operations in order to allow the testing of large memory size to be carried out in an acceptable period of time.

Very efficient test algorithms for detecting stuck-at faults and 2-coupling faults have been proposed, see for example Refs. [1–4]. All of these are march algorithms with a reduced number of operations.

For 3-coupling faults, a memory test that requires $N + 36Nlog_2N$ operations is given by Nair, Thatte and Abraham [1] ($NTA(B)$ in this article). Papachristou and Sahgal [2] proposed a new algorithm with the same ability to detect 3-coupling faults but with only $37N + 24Nlog_2N$ operations ($PS(B)$ in this article). Unfortunately, for the memory chips currently available, these tests take a long time to perform. For example, assuming a cycle time of 100 ns, $PS(B)$ takes about 4 min to test a 4 Mb memory chip and 1 h 14 min to test a 64-Mb memory chip. Of course, this time is not acceptable in many cases, such as the on-line testing. Both memory tests, $NTA(B)$ and $PS(B)$, are lengthy because the authors have assumed that the three coupled cells can be anywhere in the memory.

In this paper we propose a new march test for 3-coupling faults with an acceptable compromise between the fault detection ability and the length of the test. We have limited ourselves to the 3-coupling faults that affect only the physically adjacent memory cells. In this hypothesis, and for the cases in which the structure of memory is known (the number of columns), we have devised a test algorithm with $38N$ operations, which detects all restricted 3-coupling faults.

2. Memory fault model

This paper focuses only on the functional faults in RAM and because the address decoders, sense amplifiers and write drivers are easier to test, we consider that these modules are fault free. Consequently, we have concentrated only on the

* Corresponding author. Tel.: +40-32-232430; fax: +40-32-214290.
E-mail addresses: cascaval@cs.tuiasi.ro (P. Caşcaval), s.bennett@sheffield.ac.uk (S. Bennett).
[1] Tel.: +44-0-114-222-5230; fax: +44-0-114-273-1729.

0141-9331/00/$ - see front matter
PII: S0141-9331(00)00103-4

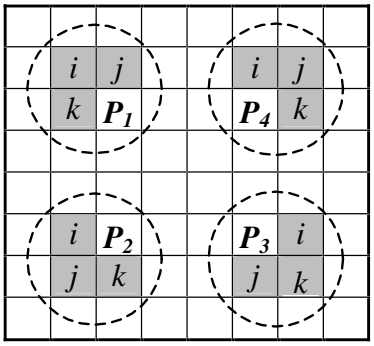


Fig. 1. Patterns for three physically adjacent cells.

functional faults in the memory cell array where difficult to detect faults may exist.

We assume that any cell in the memory can be read (fault free read operations) and a memory fault is activated only by a transition into a cell (fault free non-transition writes).

Generally we have used the fault models formalised by Nair, Thatte and Abraham [1] which were refined later by Papachristou and Sahgal [2]. At the same time we have used some definitions and fault models given by Suk and Reddy [3] and by David, Fuentes and Courtois [5]. We have also adopted some notations given by van de Goor [4]. The fault models are presented as follows.

(1) Stuck-at faults One or more cells are stuck-at *s*, where $s \in \{0,1\}$. If a cell is stuck-at *s* then it will remain in state *s* independent of reads and writes on any cell of the memory.

(2) Coupling faults Coupling faults occur because of the mutual capacitance between physically adjacent cells or because of leakage current from one cell to another in large RAM [2]. Two or more cells can be coupled. In a group of coupled cells, active and/or passive influence on a cell may exist. To model these physically faults we use transition coupling faults and state coupling faults models.

(a) Transition coupling faults

- *2-coupled cells.* A write operation that affects a $0 \rightarrow 1$ or a $1 \rightarrow 0$ state transition into cell *j* changes the state of another memory cell *i* independently of the contents of other cells. This does not necessarily imply that a state transition into cell *i* changes the contents of cell *j* [1]. Cell *i* is called the *coupled cell* and cell *j* is called the *coupling cell*. We say that cell *j* has an *active influence* on cell *i* and we write down this fault $j \rightarrow i$ *coupling fault*.
- *ν-coupled cells.* A set of ν cells **($\nu \pm 3$)** is said to be ν-coupled if a transition into one cell of the set causes the state of another cell in the set to change from 0 to 1 or from 1 to 0, when the $\nu - 2$ remaining cells have a fixed state [1]. In this article we consider only 3-coupled cells.

We assume that only the physically adjacent cells can be 3-coupled. Two cells are physically adjacent if they have a border or even a corner in common. Fig. 1 shows all the four distinct patterns P_1, P_2, P_3 and P_4 for a group of three physically adjacent cells (i,j,k).

(b) State coupling faults. A state-coupling fault occurs when one or more cells in a group of ν-coupled cells **($\nu \pm 2$)** fail to undergo a $0 \rightarrow 1$ or a $1 \rightarrow 0$ transition when the complement of the contents of the memory cell is written into the cell [4,6]. This type of fault depends on the states of the remaining $\nu - 1$ cells in the group and in this case, we say that the remaining $\nu - 1$ cells in the group have a *passive influence* on the coupled cell. If *i* is a coupled cell we name this fault *i-state coupling fault*.

We also accept that in a group with ν coupled cells two or more *interacting faults* may exist. Informally, the significance of interacting faults is that their combined effects may cancel each other [2]. In the memory, one or more groups of coupled cells may exist. When the pairs of groups of coupled cells are disjoint the model is called *restricted coupling faults*. As in Refs. [1,2] we consider only this restricted model.

3. Notations, definitions and preliminaries

The following two definitions are drawn from Ref. [3]:

Definition 1. For every memory cell in a RAM three possible states can be considered:

- *Internal state* is the actual contents of the memory cell.
- *Apparent state* is the result of a read operation of a memory cell.
- *Expected state* is the expected contents of a cell after one or more write operations.

Definition 2. Faults are *detected* if and only if one or more differences between the expected states and the apparent states of the cells occur during the test.

To describe operations on RAMs the following notations are used [3]:

- *R* read operation on a cell;
- W_x the operation of writing *x* into a cell, $x \in \{0,1\}$;
- W_c the operation of writing the complement of the previous apparent or expected state of a cell;
- $\uparrow i$ the operation of writing 1 into cell *i* when the previous apparent or expected state was 0;
- $\downarrow i$ the operation of writing 0 into cell *i* when the previous apparent or expected state was 1.

Definition 3. A *forced transition* is defined in Ref. [1] as one that is initiated by the testing algorithm by writing into a cell (of course, this may cause transitions in other cells because of coupling).

Consider a group S of ν cells. In order to describe a failed operation in S we use a vector F with 2ν elements grouped in two parts. The first part shows the conditions to activate the fault (the initial state of group S and the forced transition) and the second part shows the effect of the fault sensitising (the state of group S after the forced transition). An element in vector F can be one of the symbols 0, 1, ϕ, $\downarrow$ and $\uparrow$ (ϕ is an irrelevant logic value). Only one symbol in F can be $\downarrow$ or $\uparrow$ because it shows the forced transition in S.

For example, for a group of cells $S = (i, j, k)$:

- If j is a coupled cell, vector $F_1 = \langle 0, \uparrow, 0 : 0, 0, 0\rangle$ describes a *state coupling fault* in which the transition $\uparrow j$ has no effect when cell i and k are in the state 0.
- If k is a coupled cell, vector $F_2 = \langle \uparrow, \phi, 0 : 1, \phi, 1\rangle$ describes a *transition coupling fault* in which the forced transition $\uparrow i$ changes the state of cell k from 0 to 1.

To emphasise the cell affected by the fault we use a logic variable D as follows:

$$D = \begin{cases} 0 \text{ the cell is fault free} \\ 1 \text{ the coupling fault has been activated} \end{cases}$$

Thus, the faults previously defined become: $F_1 = \langle 0, \uparrow, 0 : 0, \underline{\mathrm{D}}, 0\rangle$ and $F_2 = \langle 0, \uparrow, 0 : 0, 1, \mathrm{D}\rangle$, where $\underline{\mathrm{D}} = \mathrm{NOT\ D}$.

Definition 4. In this paper a fault is called *simple fault* if only one vector F is necessary for description the fault behaviour. If at least two vectors are necessary for a complete description the fault is called a *complex fault*.

Any complex fault can be modelled as a set of distinct simple faults simultaneously present in the memory. An interacting fault is a complex fault which comprises two or more (even number) simple faults with complementary influence on the same cell. Examples of complex fault modelling are:

- Take an $i \rightarrow j$ coupling fault in which the transition $\uparrow i$ changes the state of cell j from 0 to 1 and the transition $\downarrow i$ from 1 to 0. This interacting fault can be modelled by two vectors

  $F_1 = \langle \uparrow, 0, \phi : 1, \mathrm{D}, \phi\rangle$ and $F_2 = \langle \downarrow, 1, \phi : 0, \underline{\mathrm{D}}, \phi\rangle$.

- Take a linked coupling fault with cell j a coupled cell. Both transition $\uparrow i$ and $\uparrow k$ change the state of cell j from 0 to 1. This non-interacting fault can be modelled by two vectors

  $F_1 = \langle \uparrow, 0, \phi : 1, \mathrm{D}, \phi\rangle$ and $F_2 = \langle \phi, 0, \uparrow\colon \phi, \mathrm{D}, 1\rangle$.

To test and find a fault in a memory we need to be able to:

- activate (to sensitise) the fault by a proper forced transition;
- observe the fault by reading the changed value of the cell affected by the fault.

Remark 1. Generally, the complex non-interacting faults are easier to detect than the simple faults because there are more situations in which a complex fault is activated. Thus, if a test procedure detects all the simple faults furthermore it detects all the complex non-interacting faults. With regard to the interacting faults, we observe that the combined effects of some simple faults may cancel each other out before the affected cell is read again. In this way, an interacting fault can be masked even if the test procedure detects all the simple faults. Consequently, in this article we focus on the simple faults and the interacting complex faults.

Assertion 1. The next three conditions are necessary and sufficient for a test to detect all the simple faults in a group of coupled cells [1]:

- *Condition 1.* For a group of cells S the test must force all the possible cell transitions. Thus, any fault which affects a cell in S is definitely activated.
- *Condition 2.* After a forced transition into a cell the test must read the cell to check if the state has changed before another forced transition into the cell is allowed to occur.
- *Condition 3.* Every cell must be read prior to a forced transition, in order to check if the state has been changed by a transition in a coupled cell.

A *march test* (T) consists of a sequence of m march elements: $T = \langle \mathrm{M}_1; \mathrm{M}_2; \ldots; \mathrm{M}_{m-1}; \mathrm{M}_m\rangle$. A *march element* (M) consists of a sequence of operations applied to each cell in the memory before proceeding to the next cell [4]. During a *march sequence* the whole memory is checked homogeneously in either one of two orders: increasing address order from address 0 ($\Uparrow$) or decreasing address order from address $N - 1$($\Downarrow$).

A march test may also comprise one or more sequences to initialise the memory.

4. March tests currently used

The best-known march test procedures currently used are presented below.

(1) The march test A with *30N* operations given by Nair, Thatte and Abraham [1] (*NTA*(*A*)):

$$NTA(A) = \langle \Uparrow (\mathrm{W}_0); \Uparrow (\mathrm{RW}_1); \Uparrow (\mathrm{R}); \Uparrow (\mathrm{RW}_0); \Uparrow (\mathrm{R});$$

$$\Downarrow (\mathrm{RW}_1); \Downarrow (\mathrm{R}); \Downarrow (\mathrm{RW}_0); \Downarrow (\mathrm{R});$$

$$\Uparrow (\mathrm{RW}_1\mathrm{W}_0); \Uparrow (\mathrm{R}); \Downarrow (\mathrm{RW}_1\mathrm{W}_0); \Downarrow (\mathrm{R}); \Uparrow (\mathrm{W}_1);$$

$$\Uparrow (\mathrm{RW}_0\mathrm{W}_1); \Uparrow (\mathrm{R}); \Downarrow (\mathrm{RW}_0\mathrm{W}_1); \Downarrow (\mathrm{R})\rangle$$

Table 1
Simple 3-coupling fault coverage

March test	*March C*	*March B*	*March G*	*NTA(A)*	*PS(A)*
Fault coverage (%)	41.67	47.22	62.50	63.89	63.89

(2) The march test A with *37N* operations given by Papachristou and Sahgal [2] (*PS*(*A*)):

$$PS(A) = \langle \Uparrow (W_0); \Uparrow (RW_1R); \Uparrow (RW_0R); \Uparrow (RW_1W_0); \Uparrow (RW_1); \Uparrow (RW_0W_1); \Uparrow (RW_0); \Uparrow (RW_1W_0); \Downarrow (RW_1); \Downarrow (RW_0); \Downarrow (RW_1W_0); \Downarrow (RW_1); \Downarrow (RW_0W_1); \Downarrow (RW_0); \Downarrow (RW_1W_0)\rangle$$

(3) The march test *MarchG* with *24N* operations given by van de Goor [4]:

$$MarchG = \langle \Uparrow (W_0); \Uparrow (RW_1RW_0W_1); \Uparrow (RW_0RW_1); \Downarrow (RW_0W_1W_0); \Downarrow (RW_1RW_0); \Uparrow (RW_1R); \Downarrow (RW_0R)\rangle$$

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	**0**	**0**	0	0	0	0	0	0	**0**	**0**	0	0
0	0	**0**	0	0	0	0	0	0	0	0	**0**	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	**0**	0	0	0	0	0	0	0	0	**0**	0	0
0	0	**0**	**0**	0	0	0	0	0	0	**0**	**0**	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ia

0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	**1**	**0**	1	**0**	**1**	0	1	**0**	**1**	0	**1**	**0**	1
0	**1**	0	1	**0**	1	0	1	0	**1**	0	1	**0**	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	**1**	0	1	**0**	1	0	1	0	**1**	0	1	**0**	1
0	**1**	**0**	1	**0**	**1**	0	1	**0**	**1**	0	**1**	**0**	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1

Ib

0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	**0**	**1**	0	**1**	**0**	1	0	**1**	**0**	1	**0**	**1**	0
0	**1**	0	1	**0**	1	0	1	0	**1**	0	1	**0**	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	**0**	1	0	**1**	0	1	0	1	**0**	1	0	**1**	0
0	**1**	**0**	1	**0**	**1**	0	1	**0**	**1**	0	**1**	**0**	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0

Ic

Fig. 2. Patterns used by *MT* to initialise the memory.

(4) The march algorithm *MarchB* with *16N* operations given by Suk and Reddy [3]:

$$MarchB = \langle \Uparrow (W_0); \Uparrow (RW_1RW_0W_1); \Uparrow (RW_0W_1); \Downarrow (RW_0W_1W_0); \Downarrow (RW_1W_0)\rangle$$

(5) The march algorithm *MarchC* with 10*N* operations presented in Ref. [4]:

$$MarchC = \langle \Uparrow (W_0); \Uparrow (RW_1); \Uparrow (RW_0); \Downarrow (RW_1); \Downarrow (RW_0); \Downarrow (R)\rangle$$

We have evaluated experimentally the 3-coupling fault coverage of these march tests. In order to simulate a permanent fault into the memory under test we have used a software fault injection mechanism [7,8].

For every possible transition in a group of coupled cells *S* we have checked two kind of faults:

- *state coupling faults* — the forced transition does not change the state of the addressed cell;
- *transition coupling faults* — the forced transition changes the state of the addressed cell but, at the same time, it changes the state of another coupled cell in *S*.

Four groups of three coupled cells have been considered, one for each pattern P_1, P_2, P_3 and P_4 (see Fig. 1). For each group, all possible simple 3-coupling faults have been injected: 24 *state coupling faults* and 48 *transition coupling faults*. In total, for all groups of cells, 288 simple faults have been checked. The simulation results are shown in Table 1.

All these march tests have good fault coverage for *stuck-at faults* and 2-*coupling faults* but, as we can see in Table 1, they have low fault coverage for 3-*coupling faults*. For example, the best of them, *NTA*(*A*) and *PS*(*A*), cover completely 2-coupling faults but only 63.89% of simple 3-coupling faults. This is the reason we have devised a new march test for 3-coupling faults, generically called *MT*. We argue that it is reasonable to consider that only the physically adjacent cells can be coupled. We assume that the number of columns for the memory cell array is known in order to determine which are the physically adjacent cells.

5. A new march test (*MT*)

We propose a new march test *MT* for the restricted 3-coupling faults which affect the physically adjacent cells

$$(6)\ MT = \langle Ia; \Uparrow (RW_c); \Uparrow (R); \Uparrow (RW_c); \Uparrow (R); \Downarrow (RW_c); \Downarrow (R); \Downarrow (RW_c); \Downarrow (R);$$

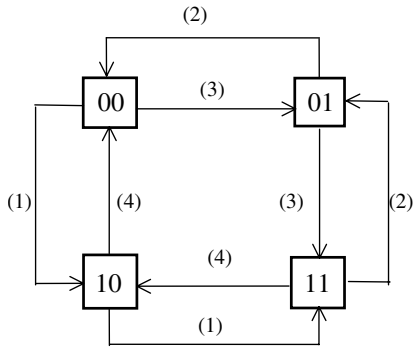


Fig. 3. The Eulerian graph of states for a group of cells $S = (i, j)$.

Ib; ⇑ (RW_c); ⇑ (R); ⇑ (RW_c); ⇑ (R); ⇓ (RW_c);

⇓ (R); ⇓ (RW_c); ⇓ (R);

Ic; ⇑ (RW_c); ⇑ (R); ⇑ (RW_c); ⇑ (R); ⇓ (RW_c);

⇓ (R); ⇓ (RW_c); ⇓ (R)⟩

where *Ia*, *Ib* and *Ic* are three sequences for memory initialisation: *Ia* initialises all the cells with 0 (solid data background), *Ib* initialises the odd columns with 0 and the even columns with 1 (column-stripe data background) and *Ic* initialises the memory by a checkerboard pattern (Fig. 2).

Remark 2. *MT* performs only one transition into each memory cell during a march sequence.

Assertion 2. The march test *MT* covers the restricted 2-coupling faults and the restricted 3-coupling faults which affect the physically adjacent cells.

Demonstration

1. Simple faults

1.1 MT activates any fault in a group of v-coupled cells

We prove that *MT* performs all the possible transitions in a group of coupled cells (*condition 1* previously defined in Section 3 is satisfied). In fact, we prove that the Eulerian graph of states for the considered group of coupled cells is completely covered during the memory testing.

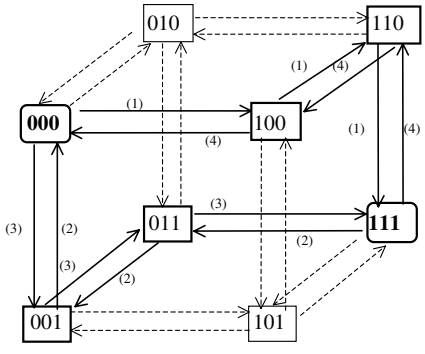


Fig. 4. The Eulerian graph of states for a group of cells $S = (i, j, k)$.

(a) Simple restricted 2-coupling faults

Take a group of two coupled cells $S = (i, j)$, $\forall i, j \in \{0, 1, \ldots, N-1\}$, $i \neq j$. The graph of states for the group of cells *S* is shown in Fig. 3. As shown in Fig. 3, during the following test sequence

(1) (2) (3) (4)

Ia; ⇑ (RW_1); ⇑ (R); ⇑ (RW_0); ⇑ (R); ⇓ (RW_1); ⇓ (R); ⇓ (RW_0); ⇓ (R);

the graph of states is completely covered and, consequently, *MT* performs all the possible transitions in group of cells *S*.

MT performs a minimal number of write operations because each arc in the graph has been traversed only once. This test sequence, identical with the sequence given by Nair, Thatte and Abraham [1] for restricted 2-coupling faults (first part of the test *NTA*(*A*)), is an optimal test sequence for the non-interacting 2-coupling faults.

(b) Simple restricted 3-coupling faults

Take a group of three cells $S = (i, j, k)$. The Eulerian graph of states for the group *S* is shown in Fig. 4. Observe that two adjacent states (nodes) in the graph have only one bit changed and two non-adjacent states have at least two bits changed.

Every group of three physically adjacent coupled cells corresponds with one of the patterns P_1, P_2, P_3 or P_4 (see Fig. 1). Consequently, we have to prove that the graph of states for a group of coupled cells is completely covered by *MT* in all these four cases.

For all groups of cells in the memory under test, regardless of the pattern, the first four march sequences (RW_c) perform the same transitions because every group starts from the initial state 000. These transitions are marked with solid lines in the graph in Fig. 5.

Note that the graph has been traversed from the initial node to the opposite node (the node with all the bits changed). Moreover, we can see that the same arcs will be traversed in the graph if the opposite state 111 becomes initial state. Consequently, it does not matter if the group of cells has been initialised with 010 or 101.

A state of group *S* before a new march sequence is started is called *initial state*. Table 2 shows the initial states for each pattern P_i, $i \in \{1,2,3,4\}$ after the sequences *Ia*, *Ib* and *Ic*.

Table 2 shows that the sequence *Ib* or *Ic* ensures the state 010 or 101 for all the groups of cells, regardless of the pattern. Consequently, the following test sequences

Ib; ⇑ (RW_c); ⇑ (R); ⇑ (RW_c); ⇑ (R); ⇓ (RW_c);

⇓ (R); ⇓ (RW_c); ⇓ (R);

Ic; ⇑ (RW_c); ⇑ (R); ⇑ (RW_c); ⇑ (R); ⇓ (RW_c); ⇓ (R);

⇓ (RW_c); ⇓ (R);

perform at least the transitions marked with solid lines in Fig. 6, for all the groups of three physically adjacent cells in the memory under test.

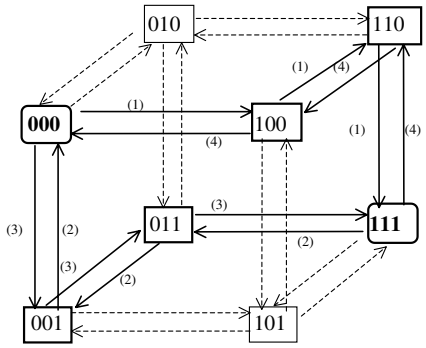


Fig. 5. The transitions performed in group S by the first four march sequences of *MT*.

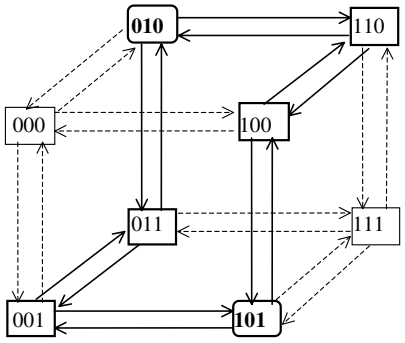


Fig. 6. The transitions between the states *010* and *101* and vice versa.

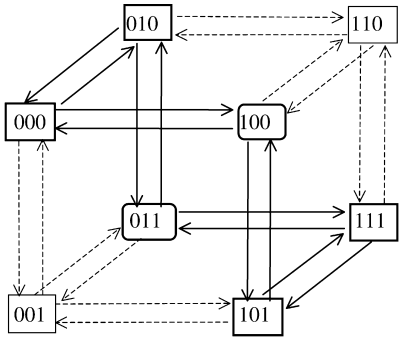


Fig. 7. The transitions between the states *011* and *100* and vice versa.

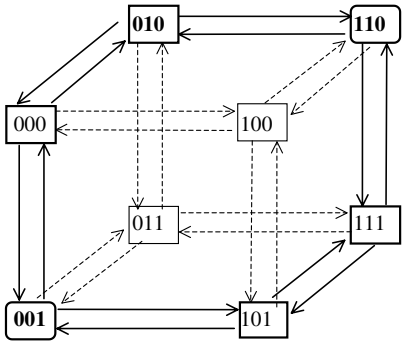


Fig. 8. The transitions between the states *001* and *110* and vice versa.

Table 2
The initial states performed by *Ia*, *Ib*, *Ic*

	Ia	*Ib*	*Ic*
P_1	000	010 or 101	011 or 100
P_2	000	001 or 110	010 or 101
P_3	000	010 or 101	001 or 110
P_4	000	011 or 100	010 or 101

Table 3
Simple $i \rightarrow k$ coupling faults

Nr.	Vector *F*
1	$\langle\uparrow,0,0: 1,0,\mathrm{D}\rangle$
2	$\langle\uparrow,0,1: 1,0,\overline{\mathrm{D}}\rangle$
3	$\langle\uparrow,1,0: 1,1,\mathrm{D}\rangle$
4	$\langle\uparrow,1,1: 1,1,\overline{\mathrm{D}}\rangle$
5	$\langle\downarrow,0,0: 0,0,\mathrm{D}\rangle$
6	$\langle\downarrow,0,1: 0,0,\overline{\mathrm{D}}\rangle$
7	$\langle\downarrow,1,0: 0,1,\mathrm{D}\rangle$
8	$\langle\downarrow,1,1: 0,1,\overline{\mathrm{D}}\rangle$
9	$\langle\uparrow,\phi,0: 1,\phi,\mathrm{D}\rangle$
10	$\langle\uparrow,\phi,1: 1,\phi,\overline{\mathrm{D}}\rangle$
11	$\langle\downarrow,\phi,0: 0,\phi,\mathrm{D}\rangle$
12	$\langle\downarrow,\phi,1: 0,\phi,\overline{\mathrm{D}}\rangle$

Table 4
Simple $j \rightarrow k$ coupling faults

Nr.	Vector *F*
1	$\langle 0,\uparrow,0: 0,1,\mathrm{D}\rangle$
2	$\langle 0,\uparrow,1: 0,1,\overline{\mathrm{D}}\rangle$
3	$\langle 1,\uparrow,0: 1,1,\mathrm{D}\rangle$
4	$\langle 1,\uparrow,1: 1,1,\overline{\mathrm{D}}\rangle$
5	$\langle 0,\downarrow,0: 0,0,\mathrm{D}\rangle$
6	$\langle 0,\downarrow,1: 0,0,\overline{\mathrm{D}}\rangle$
7	$\langle 1,\downarrow,0: 1,0,\mathrm{D}\rangle$
8	$\langle 1,\downarrow,1: 1,0,\overline{\mathrm{D}}\rangle$
9	$\langle\phi,\uparrow,0: \phi,1,\mathrm{D}\rangle$
10	$\langle\phi,\uparrow,1: \phi,1,\overline{\mathrm{D}}\rangle$
11	$\langle\phi,\downarrow,0: \phi,0,\mathrm{D}\rangle$
12	$\langle\phi,\downarrow,1: \phi,0,\overline{\mathrm{D}}\rangle$

Table 5
All the initial states for a group $S = (i, j, k)$

Pattern	Initial states
P_1	*000, 111, 010, 101, 011, 100*
P_2	*000, 111, 010, 101, 001, 110*
P_3	*000, 111, 010, 101, 001, 110*
P_4	*000, 111, 010, 101, 011, 100*

Table 6
The interacting 3-coupling faults and the conditions in which *MT* detects the faults

	Interacting faults		Conditions in which *MT* detects the faults					
Nr.	$i \rightarrow k$ coupling fault F_1	$j \rightarrow k$ coupling fault F_2	Initial state	Test order	Result	Initial state	Test order	Result
1	⟨↑,0,0: 1,0,D⟩	⟨0,↑,1: 0,1,D⟩	*101*	⇑	E_1,F_2			
2		⟨1,↑,1: 1,1,D⟩	*100*	⇓	E_1,F_2	*001*	⇑	E_1,F_2
3		⟨0,↓,1: 0,0,D⟩	*000*	⇑	F_1,E_2			
4		⟨1,↓,1: 1,0,D⟩	*000*	⇑	F_1,E_2			
5		⟨ϕ,↑,1: ϕ,1,D⟩	*101*	⇑	E_1,F_2			
6		⟨ϕ,↓,1: ϕ,0,D⟩	*111*	⇑	E_1,F_2			
7	⟨↑,0,1: 1,0,D⟩	⟨0,↑,0: 0,1,D⟩	*001*	⇑	F_1,E_2	*100*	⇑	E_1,F_2
8		⟨1,↑,0: 1,1,D⟩	*101*	⇓	E_1,F_2			
9		⟨0,↓,0: 0,0,D⟩	*010*	⇓	F_1,E_2			
10		⟨1,↓,0: 1,0,D⟩	*111*	⇓	E_1,F_2			
11		⟨ϕ,↑,0: ϕ,1,D⟩	*010*	⇓	F_1,E_2			
12		⟨ϕ,↓,0: ϕ,0,D⟩	*010*	⇓	F_1,E_2			
13	⟨↑,1,0: 1,1,D⟩	⟨0,↑,1: 0,1,D⟩	*010*	⇑	F_1,E_2			
14		⟨1,↑,1: 1,1,D⟩	*010*	⇑	F_1,E_2			
15		⟨0,↓,1: 0,0,D⟩	*010*	⇑	F_1,E_2			
16		⟨1,↓,1: 1,0,D⟩	*110*	⇓	E_1,F_2	*011*	⇑	E_1,F_2
17		⟨ϕ,↑,1: ϕ,1,D⟩	*101*	⇑	E_1,F_2			
18		⟨ϕ,↓,1: ϕ,0,D⟩	*111*	⇑	E_1,F_2			
19	⟨↑,1,1: 1,1,D⟩	⟨0,↑,0: 0,1,D⟩	*000*	⇓	F_1,E_2			
20		⟨1,↑,0: 1,1,D⟩	*101*	⇓	E_1,F_2			
21		⟨0,↓,0: 0,0,D⟩	*011*	⇑	F_1,E_2	*110*	⇑	E_1,F_2
22		⟨1,↓,0: 1,0,D⟩	*111*	⇓	E_1,F_2			
23		⟨ϕ,↑,0: ϕ,1,D⟩	*101*	⇓	E_1,F_2			
24		⟨ϕ,↓,0: ϕ,0,D⟩	*111*	⇓	E_1,F_2			
25	⟨↓,0,0: 0,0,D⟩	⟨0,↑,1: 0,1,D⟩	*101*	⇑	E_1,F_2			
26		⟨1,↑,1: 1,1,D⟩	*100*	⇑	F_1,E_2	*001*	⇑	E_1,F_2
27		⟨0,↓,1: 0,0,D⟩	*010*	⇓	E_1,F_2			
28		⟨1,↓,1: 1,0,D⟩	*111*	⇓	F_1,E_2			
29		⟨ϕ,↑,1: ϕ,1,D⟩	*000*	⇓	E_1,F_2			
30		⟨ϕ,↓,1: ϕ,0,D⟩	*010*	⇓	E_1,F_2			
31	⟨↓,0,1: 0,0,D⟩	⟨0,↑,0: 0,1,D⟩	*100*	⇑	E_1,F_2	*001*	⇓	E_1,F_2
32		⟨1,↑,0: 1,1,D⟩	*101*	⇑	F_1,E_2			
33		⟨0,↓,0: 0,0,D⟩	*101*	⇑	F_1,E_2			
34		⟨1,↓,0: 1,0,D⟩	*101*	⇑	F_1,E_2			
35		⟨ϕ,↑,0: ϕ,1,D⟩	*000*	⇑	E_1,F_2			
36		⟨ϕ,↓,0: ϕ,0,D⟩	*010*	⇑	E_1,F_2			
37	⟨↓,1,0: 0,1,D⟩	⟨0,↑,1: 0,1,D⟩	*000*	⇓	E_1,F_2			
38		⟨1,↑,1: 1,1,D⟩	*101*	⇓	F_1,E_2			
39		⟨0,↓,1: 0,0,D⟩	*010*	⇓	E_1,F_2			
40		⟨1,↓,1: 1,0,D⟩	*101*	⇓	F_1,E_2			
41		⟨ϕ,↑,1: ϕ,1,D⟩	*000*	⇓	E_1,F_2			
42		⟨ϕ,↓,1: ϕ,0,D⟩	*010*	⇓	E_1,F_2			
43	⟨↓,1,1: 0,1,D⟩	⟨0,↑,0: 0,1,D⟩	*111*	⇑	F_1,E_2			
44		⟨1,↑,0: 1,1,D⟩	*111*	⇑	F_1,E_2			
45		⟨0,↓,0: 0,0,D⟩	*110*	⇑	E_1,F_2	*100*	⇓	F_1,E_2
46		⟨1,↓,0: 1,0,D⟩	*111*	⇑	F_1,E_2			
47		⟨ϕ,↑,0: ϕ,1,D⟩	*111*	⇑	F_1,E_2			
48		⟨ϕ,↓,0: ϕ,0,D⟩	*111*	⇓	E_1,F_2			
49	⟨↑,ϕ,0: 1,ϕ,D⟩	⟨0,↑,1: 0,1,D⟩	*101*	⇑	E_1,F_2			
50		⟨1,↑,1: 1,1,D⟩	*010*	⇑	F_1,E_2			
51		⟨0,↓,1: 0,0,D⟩	*000*	⇑	F_1,E_2			
52		⟨1,↓,1: 1,0,D⟩	*000*	⇑	F_1,E_2			
53		⟨ϕ,↑,1: ϕ,1,D⟩	*010*	⇑	F_1,E_2			
54		⟨ϕ,↓,1: ϕ,0,D⟩	*000*	⇑	F_1,E_2			
55	⟨↑,ϕ,1: 1,ϕ,D⟩	⟨0,↑,0: 0,1,D⟩	*000*	⇓	F_1,E_2			
56		⟨1,↑,0: 1,1,D⟩	*101*	⇓	E_1,F_2			
57		⟨0,↓,0: 0,0,D⟩	*010*	⇓	F_1,E_2			
58		⟨1,↓,0: 1,0,D⟩	*010*	⇓	F_1,E_2			
59		⟨ϕ,↑,0: ϕ,1,D⟩	*010*	⇓	F_1,E_2			
60		⟨ϕ,↓,0: ϕ,0,D⟩	*010*	⇓	F_1,E_2			

Table 6 (*continued*)

	Interacting faults		Conditions in which *MT* detects the faults					
Nr.	$i \rightarrow k$ coupling fault F_1	$j \rightarrow k$ coupling fault F_2	Initial state	Test order	Result	Initial state	Test order	Result
61	$\langle\downarrow,\phi,0{:}\ 1,\phi,D\rangle$	$\langle 0,\uparrow,1{:}\ 0,1,D\rangle$	*000*	$\Downarrow$	E_1,F_2			
62		$\langle 1,\uparrow,1{:}\ 1,1,D\rangle$	*111*	$\Downarrow$	F_1,E_2			
63		$\langle 0,\downarrow,1{:}\ 0,0,D\rangle$	*010*	$\Downarrow$	E_1,F_2			
64		$\langle 1,\downarrow,1{:}\ 1,0,D\rangle$	*111*	$\Downarrow$	F_1,E_2			
65		$\langle \phi,\uparrow,1{:}\ \phi,1,D\rangle$	*000*	$\Downarrow$	E_1,F_2			
66		$\langle \phi,\downarrow,1{:}\ \phi,0,D\rangle$	*010*	$\Downarrow$	E_1,F_2			
67	$\langle\downarrow,\phi,1{:}\ 0,\phi,D\rangle$	$\langle 0,\uparrow,0{:}\ 0,1,D\rangle$	*111*	$\Uparrow$	F_1,E_2			
68		$\langle 1,\uparrow,0{:}\ 1,1,D\rangle$	*111*	$\Uparrow$	F_1,E_2			
69		$\langle 0,\downarrow,0{:}\ 0,0,D\rangle$	*101*	$\Uparrow$	F_1,E_2			
70		$\langle 1,\downarrow,0{:}\ 1,0,D\rangle$	*101*	$\Uparrow$	F_1,E_2			
71		$\langle \phi,\uparrow,0{:}\ \phi,1,D\rangle$	*111*	$\Uparrow$	F_1,E_2			
72		$\langle \phi,\downarrow,0{:}\ \phi,0,D\rangle$	*101*	$\Uparrow$	F_1,E_2			

For a group of coupled cells, regardless of the pattern, the Eulerian graph is covered except on the arcs between 000 and 010 and between 101 and 111 (see both Figs. 5 and 6). But, we can see in Table 2 that the sequence *Ib* or *Ic* ensures the initial state 011 or 100 for a group with pattern P_1 or P_4, and, 001 or 110 for a group with pattern P_2 or P_3. In both cases the arcs between 000 and 010 and between 101 and 111 are traversed (see Figs. 7 and 8) and, consequently, the whole graph is covered for every group of cells in the memory under test.

Finally, we have proved that *MT* activates any fault which affects a group of three physically adjacent cells.

1.2 MT observes any simple fault activated in group of cells S

The march test *MT* repeats the pair of march elements (RW_c) and (R) many times. Because every write operation (W_c) is preceded and succeeded by a read operation (R) *conditions 2 and 3* previously defined in Section 3 are satisfied.*1.2 Interacting 3-coupling faults*

According to Definition 4 defined in Section 3, a complex fault causes two or more transitions in one or more cells and can be modelled as a set of distinct simple faults which are simultaneously present in the memory. As follows, we prove that *MT* detects all the interacting faults modelled by two simple transition coupling faults.

Remark 3. In a group of cells $S = (i, j, k)$, in which cell j is a coupled-cell, all the transition coupling faults are definitely detected because *MT* performs only one forced transition in cell i or cell k between two consecutive reads of cell j.

Remark 4. *MT* performs a march sequence (RW_c) in both orders, increasing ($\Uparrow$) and decreasing ($\Downarrow$), for all the initial states. In other words, *MT* is a symmetrical test. Therefore it is sufficient to analyse only the case in which cell i is a coupled cell or cell k is a coupled cell. We have chosen to prove that *MT* detects the interacting coupling faults which affect cell k.

We must consider all the combinations between one *simple $i \rightarrow k$ coupling fault* and one *simple $j \rightarrow k$ coupling fault*. These interacting faults are presented in Tables 3 and 4, respectively.

Every combination between a vector F in Table 3 with symbol D, and a vector F in Table 4 with symbol $\underline{D}$, or vice versa, describes an interacting coupling fault which affects cell k. All these interacting faults are shown in Table 6.

A march memory test detects an interacting coupling fault if and only if, during a march sequence, one simple fault is activated and another simple fault is masked. Consequently, we must prove that for each interacting fault which affects cell k, the march test *MT* brings the group S into an appropriate initial state which allows, in the next march sequence, one simple fault to be activated and another masked. Of course, this initial state of group S depends on the pattern. Table 5 comprises all the initial states for each pattern.

Table 6 comprises all the 72 interacting transition coupling faults which may affect cell k and the conditions in which *MT* detects these faults. In some cases there are two conditions in which *MT* detects the fault: one condition for the patterns P_1 and P_4 and another for P_2 and P_3. In the

Table 7
The coupling fault coverage of the march tests (expressed as %)

March test	*March C 10N*	*March B 17N*	*March G 25N*	*NTA(A) 30N*	*PS(A) 37N*	*MT 38N*
2-Coupling	81.25	87.5	100	100	100	100
3-Coupling	41.67	47.22	62.50	63.89	63.89	100

column *Result* the pair $F_1, \underline{F}_2$ for example, reflects that the fault F_1 has been activated and F_2 has not. Table 6 shows that *MT* detects all the interacting coupling faults which may affect cell *k*. Because *MT* is a symmetrical test it also detects all the interacting transition coupling faults which may affect cell *i*.

Similarly, we can prove that *MT* detects any interacting faults modelled by one simple state coupling fault and one simple transition coupling fault. □

The simulation result confirms that *MT* detects all simple 3-coupling faults and, also, all the interacting faults presented in Table 6.

6. Conclusions

A new efficient march test (*MT*) for detecting the 3-coupling faults in RAM is given in this paper. Comparing with the most known march tests currently used *MT* is slightly longer but it covers completely the 3-coupling faults which affect the physically adjacent cells. The simulation result presented in Table 7 gives emphasis to the effectiveness of the new march test *MT*.

This new march test has *38N* operations. For a 64 Mb memory chip the test experiment takes about 5 min if we assume a cycle time of 100 ns. Remember that the non-march test *PS*(*B*), especially designed for 3-coupling faults, takes about 1h 14 min.

MT is a homogeneous test algorithm and comprises only two kind of march sequences, (RW_c) and (R). Consequently, *MT* is adequate for BIST implementation in embedded RAM [9,10].

Acknowledgements

This work was done while Petru Caşcaval was an Academic Research Visitor at the Department of Automatic Control and Systems Engineering, University of Sheffield, under the kind supervision of Dr Stuart Bennett.

References

[1] R. Nair, S. Thatte, J. Abraham, Efficient algorithms for testing semiconductor random access memories, IEEE Trans. Comput. C-27 (6) (1978) 572–576.
[2] C. Papachristou, N. Sahgal, An improved method for detecting functional faults in semiconductor random access memories, IEEE Trans. Comput. C-34 (2) (1985) 110–116.
[3] D.S. Suk, M. Reddy, A march test for functional faults in semiconductor random access memories, IEEE Trans. Comput. C-30 (12) (1981) 982–985.
[4] A.J. van de Goor, Using march tests to test SRAMs, IEEE Design Test Comput. March (1993) 8–14.
[5] R. David, A. Fuentes, B. Courtois, Random pattern testing versus deterministic testing of RAM's, IEEE Trans. Comput. 38 (5) (1989) 637–650.
[6] J. Hayes, Testing memories for single-cell pattern-sensitive faults, IEEE Trans. Comput. C-29 (3) (1980) 249–254.
[7] P. Caşcaval, R. Silion, Memory test algorithm study by fault injection mechanism, Sixth International Symposium on Automatic Control and Computer Science, SACCS'98, vol. II, Iasi, Romania, November 1998, pp. 23–28.
[8] P. Caşcaval, C. Hutanu, R. Silion, Memory fault coverage evaluation for march tests, Buletinul Institutului Politehnic din Iasi XLV (IL) (1–4) (1999) 103–110 (Automatica si Calculatoare).
[9] M. Franckil, K. Saluja, Built-in self-testing of random-access memories, IEEE Comput. October (1990) 45–56.
[10] C. Huang, et al., A programmable BIST core for embedded DRAM, IEEE Design Test Comput. January–March (1999) 59–70.