ELSEVIER

Information Processing Letters 73 (2000) 199–206

Information Processing Letters

www.elsevier.com/locate/ipl

An improved testing scheme for catastrophic fault patterns ☆

A. Nayak *, J. Ren [1], N. Santoro

School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

Received 3 December 1998; received in revised form 5 December 1999
Communicated by F. Dehne

Keywords: Redundant arrays; Catastrophic faults; Testing schemes; Algorithms

1. Introduction

In a linear array of processors, a single faulty element in any location is sufficient to stop the flow of information from one side to the other. A common approach for achieving fault tolerance in such systems is through the incorporation of redundant links in a regular fashion. These links (called *bypass links*) can be activated in a reconfiguration phase to bypass faulty elements.

There are some inherent limits in this approach. In particular, there are sets of faults occurring in strategic locations which affect the entire system in an unrepairable way, regardless of the amount of redundancy, and cannot be overcome by any clever reconfiguration process, see [7]. These sets of faults are called *Catastrophic Fault Patterns* (CFP) and have been extensively studied in the literature [1,2,4–6,8, 10]. The rather intuitive guess that any cut set is a CFP is unfortunately incorrect; on the contrary, they have a rather interesting structure with non-trivial symmetries and it is known that, to be catastrophic, the number m of faults must be at least as large as the length g of the longest bypass link [7] (by comparison, cut sets need only to be of size k or $2k$, depending on whether the links are uni- or bidirectional).

An important question with regards to CFP is the *Testing Problem*; that is, the problem of determining whether a given set of faults is catastrophic. The complexity of this problem depends on many parameters: the size N of the array, the number k of bypass links at each element and their lengths, and the number m of faults. Any solution to this problem is called a *testing scheme*.

The investigations on the testing problem have been restricted to the particular case of *minimal* fault patterns (i.e., $m = g$), and testing schemes have been presented both for unidirectional and for bidirectional arrays [4,5,7,9].

The general case $m \geqslant g$ has been recently considered in [1]. Based on a graph-theoretic interpretation, different bounds have been established depending on whether the links are uni- or bidirectional. Namely, in the case of bidirectional links they show that the problem has a simple $O(mk)$ solution; on the other hand, if the links of the array are unidirectional, the proposed testing scheme requires time $O(mk \log k)$. Notice that neither bounds depend on the size N of the array.

☆ This work was supported in part by Natural Sciences and Engineering Research Council of Canada under Operating Grant A2415. A preliminary version of the paper has appeared in the 6th International Symposium on Algorithms and Computation.

* Corresponding author. Email: nayak@scs.carleton.ca. Currently with Nortel Networks, Ottawa, Ontario, Canada.

[1] Currently with Microsoft Corp., Seattle, USA.

0020-0190/00/$ – see front matter © 2000 Published by Elsevier Science B.V. All rights reserved.
PII: S0020-0190(00)00012-0

The difference in bounds is not significant from a practical viewpoint; however, it raises the interesting theoretical question of whether the Testing Problem is computationally more difficult in the case of unidirectional links. We provide evidence for a negative answer by abolishing the existing gap between the two upper bounds. In fact, we prove that testing can be done in time $O(mk)$ also for arrays with unidirectional links improving the existing $O(mk \log k)$ bound. The testing scheme achieving the bound is based on a novel "geometric" approach.

We actually solve a more general version of the classical problem of finding a obstacle-avoiding path in a two-dimensional grid (e.g., [11,12]); in our case, the mesh has a rather complex link structure in addition to its own links.

2. Terminology and definitions

A unidimensional linear array A of size N is composed of a set $P = \{p_1, p_2, \ldots, p_N\}$ of processing elements and two special processors, called I (for Input) and O (for Output), responsible for the I/O functions of the system; each p_i is connected to p_{i+1} $(1 \leqslant i < N)$, I is connected to p_1, and O to p_N. In the following, for simplicity, we will denote p_i simply by i; hence $P = \{1, 2, \ldots, N\}$.

Fault tolerance is achieved by symmetric addition of links. Given an integer $g \in (1, N]$, A has *link redundancy* g, if every $i \in P$ with $i \leqslant N - g$ is connected to $i + g$, I is connected to $1, \ldots, g$, and $N - g + 1, \ldots, N$ are connected to O. The array has link redundancy $G = \{g_1, g_2, \ldots, g_k\}$ where $g_j < g_{j+1}$ and $g_j \in (1, N]$, if it has link redundancy $g_1, g_2, \ldots, g_k$.

A *fault pattern* for P is just a subset $F \subseteq P$ of the processors. The *width* ω_F of a fault pattern $F = \{f_1, f_2, \ldots, f_m\}$ is the number of processors between and including the first and the last faults: $\omega_F = f_m - f_1 + 1$.

Example 1. A fault pattern is shown in Fig. 1, where black dots represent faulty elements.

Fig. 1. A fault pattern for an array with link redundancy {4}.

Consider a linear array P with a link redundancy $G = \{g_1, g_2, \ldots, g_k\}$ and a fault pattern $F = \{f_1, f_2, \ldots, f_m\}$. Without loss of generality, we will always assume $f_1 = 1$ for convenience. The fault pattern F can be uniquely represented by a Boolean matrix W of size $(\omega_F^+ \times g_k)$, where $\omega_F^+ = \lceil \omega_F / g_k \rceil$, defined as follows:

$$W[i, j] = \begin{cases} 1 & \text{if } (ig_k + j + 1) \in F, \\ 0 & \text{otherwise.} \end{cases}$$

In the following, where no ambiguity arises, we will use the coordinate pair (x_l, y_l) to denote $W[x_l, y_l]$.

Example 2. The Boolean matrix representation of the fault pattern $F = \{f_1, f_2, f_3, f_4, f_5\} = \{(0, 0), (0, 3), (1, 1), (1, 3), (2, 2)\}$ for $G = \{4\}$ is given in Fig. 2.

Notice that, since we assume $f_1 = 1$, W represents the "status" (faulty or not faulty) of the first ω_F elements of P.

A fault pattern $F \subseteq P$ represents a set of initially faulty elements. Depending on the initial assignment of faults, some non-faulty elements can become unreachable from I (and thus, unable to participate in the computation), or O becomes unreachable from them (and thus their participation in the computation in irrelevant); these elements are thus "functionally" faulty. A fault pattern is said to be *catastrophic* if it causes all elements to become functionally faulty and, thus, I and O to become disconnected. To describe the impact that the initial fault pattern has on the system, we use the notion of *dead* elements. Consider a linear array P with a link redundancy $G = \{g_1, g_2, \ldots, g_k\}$ and a fault pattern $F = \{f_1, f_2, \ldots, f_m\}$. An element $p \in P$ is *dead* if $p \in F$, or if all the elements in $(p + G) \cap P$ are dead, or all elements in $p - G \cap P$ are dead. An element $p \in P$ which is not dead is said to be *alive*.

Example 3. Consider the matrix shown on Fig. 3 where $G = \{3, 5, 7\}$ and $F = \{(0, 0), (0, 2), (0, 3), (0, 4), (0, 6), (1, 1), (1, 5)\}$. It is not difficult to verify that elements $(1, 0)$, $(1, 2)$, and $(1, 6)$ are dead, while elements $(0, 1)$, $(0, 5)$, $(1, 3)$ and $(1, 4)$ are alive.

The fault pattern F is *catastrophic* if all elements of P are dead. Thus, the pattern of Example 3 is not catastrophic for $G = \{3, 5, 7\}$; on the other hand,

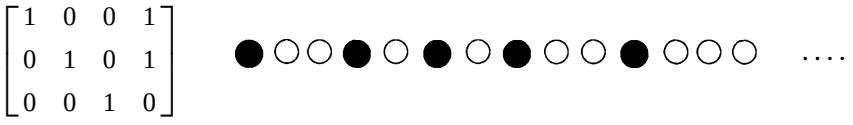

Fig. 2. A Boolean matrix of a catastrophic fault pattern of 5 faults with link redundancy {4}.

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Fig. 3. Examples of dead elements in a matrix where $G = \{3, 5, 7\}$.

it is not difficult to verify that the fault pattern in Example 1 is catastrophic for $G = \{4\}$.

3. Segments, gaps and shadows

3.1. Dead segments

Using the introduced notation, we will extend the notion of functionally faulty to sequences of elements in W recursively as follows:

Definition 1. A dead segment is

(1) a set of consecutive dead elements in the same row $S = \{(x, y), (x, y+1), \ldots, (x, y+d)\}$, where $0 \leqslant d \leqslant g_k - 1$; or

(2) if $S_1 = \{(x, y), \ldots, (x, y + d)\}$ and $S_2 = \{(x, y'), \ldots, (x, y' + h)\}$ are dead segments with $y' = y + d + 1$, then $S = \{(x, y), \ldots, (x, y + h')\}$ is a dead segment; or

(3) if $S_1 = \{(x, y), \ldots, (x, g_k)\}$ and $S_2 = \{(x + 1, 0), \ldots, (x + 1, d)\}$ are dead segments where $1 \leqslant d \leqslant g_k$, then $S = \{(x, y), \ldots, (x, g_k), (x + 1, 0), \ldots, (x + 1, d)\}$ is a dead segment.

Using the notion of segment, we now establish a necessary and sufficient condition for a fault pattern, with arbitrary number of faults for an arbitrary link redundancy, to be catastrophic.

A live path is a path on which every element is alive. Given live elements p and q, q is reachable from p if there exists a live path from p to q. The set of entry points of row(i) is: $E_i =$ {live elements in row(i) reachable from some element in E_{i-1}} if $i > 1$, and $E_1 =$ {live elements in row(1)}. We claim that for every row the set of entry points coincides with the set of live elements.

Property 1. *Let L_i be the set of live elements of* row(i)*, then $E_i = L_i$.*

Proof. By induction on i. By definition of E_1, the claim trivially holds for $i = 1$. Let it hold for E_i, $i > 1$. Consider row($i + 1$). By contradiction, define the set of live elements which are not entry points $W_{i+1} = L_{i+1} - E_{i+1} \neq \emptyset$. Let j be the smallest index such that $(i + 1, j) \in W_{i+1}$. Depending on whether $j = 1$ or $j > 1$, we shall consider two cases.

Case 1 ($j = 1$). Since $p = (i + 1, 1) \notin E_{i+1}$, then all $(i, g_k), (i, 1 - g_2 + g_k), \ldots, (i, 1)$ are not in E_i (which is L_i by inductive hypothesis). Then, by Definition 1(2), $p = (i + 1, j)$ is dead. A clear contradiction.

Case 2 ($j > 1$). Since $p = (i + 1, j) \notin E_{i+1}$, then there exists l such that all $(i, j - g_k + g_k), (i, j - g_{k-1} + g_k), \ldots, (i, j - g_l + g_k)$ are not entry points and by inductive hypothesis are dead. Since $p = (i + 1, j)$ is live, then there exists d, $1 \leqslant d \leqslant l - 1$, such that $q = (i + 1, j - g_d) \in L_{i+1}$; otherwise, by Definition 1(2), $p = (i + 1, j)$ is dead. Now, q cannot be in E_{i+1} (otherwise, $p \in E_{i+1}$). This contradicts the fact that j is the smallest index for which $(i + 1, j) \notin W_{i+1}$. Therefore, the claim holds. □

Theorem 1. *A fault pattern is catastrophic if and only if it has at least one dead segment of g_k elements.*

Proof. ($\Rightarrow$) By contradiction, let the fault pattern be catastrophic, and $\forall i$ $L_i \neq \emptyset$. Consider the element sequence $l_1, l_2, \ldots, L_{Last}, l_i \in L_i, l_i = (x_i, y_i)$, where l_{Last} is an arbitrary element in $L_{Last} \neq \emptyset$; and l_i, $i <$ *Last*, is recursively constructed from l_{i+1} as follows: L_{i+1} (which by Property 1 is E_{i+1}) is the set of live elements in row($i + 1$) reachable from some live element in row(i). Thus for every $a \in L_{i+1}$, there exists at least an element $b \in L_i$ such that a

is reachable from b. Choose l_i be the element in L_i from which l_{i+1} is reachable. Therefore, the sequence of $l_1, l_2, \ldots, l_{Last}$, defines a path of live elements, l_{Last} is reachable from l_1; thus contradicts the fact that the fault pattern is catastrophic.

($\Leftarrow$) By contradiction, let $L_j = \emptyset$ and the fault pattern be not catastrophic. Since the fault pattern is not catastrophic, then there exists $l_1, l_2, \ldots, l_{Last}$ such that $l_i \in L_i$ and $l_i \in \text{row}(i)$, contradicting $L_j = \emptyset$. □

3.2. Gaps and shadows

It is possible for the Boolean matrix to have several contiguous rows (called "gap") which do not contain any faulty element. Should this be the case, the given fault pattern can be partitioned into smaller fault patterns such that the Boolean matrix of each of these fault patterns does not have any gap. The original fault pattern is catastrophic if and only if at least one of these smaller fault patterns is catastrophic; this is due to the following theorem.

Theorem 2. *Given the Boolean matrix representation of a fault pattern* $F = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ *there exists* i *such that* $x_{i+1} - x_i > 2$, *let*

$$F_1 = \big\{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)\big\}$$

and

$$F_2 = \big\{(x_{i+1}, y_{i+1}), (x_{i+2}, y_{i+2}), \ldots, (x_n, y_n)\big\};$$

then F *is catastrophic if and only if* F_1 *or* F_2 *is catastrophic.*

Proof. ($\Rightarrow$) If F_1 or F_2 is catastrophic, then obviously F is catastrophic.

($\Leftarrow$) By contradiction, let F_1 and F_2 be not catastrophic. Then there exists a live path L_1 in F_1 from row(1) to row(x_i), and a live path L_2 in F_2 from row(x_{i+1}) to the last row of F. Since row($x_i + 1$) does not have any faulty element, then (by Definition 1), the only possible dead segment in this row has the form $\{(x_i + 1, 1), \ldots, (x_i + 1, j)\}$ where $j \geqslant 1$, and the rest of the row elements are live. Thus, there is no dead segment in row($x_i + 2$). Therefore L_2 can always be connected with L_1. In other words, there exists a live path from the first row of F to the last row of F, contradicting the assumption that F is catastrophic. □

We now define the "shadow" of a segment.

Definition 2. Let S be a segment, and S' be a segment at distance d from S. Then S is called the shadow of S' at distance d if

(1) $|S| = |S'|$, and

(2) $\forall (x, y) \in S$ either $(x, y - d) \in S'$ or $(x - 1, y - d + g_k) \in S'$.

Given a segment S, let $x(S)$ be the x coordinate of the first element of S, $y_F(S)$ be the y coordinate of the first element of S, $y_L(S)$ be the y coordinate of the last element of S.

Definition 3. Let Ω_i, Ω_j be two segments, $\Omega_i < \Omega_j$ if $y_L(\Omega_i) < y_F(\Omega_j)$ and $x(\Omega_i) = x(\Omega_j)$, or $x(\Omega_i) < x(\Omega_j)$.

Certain segments can be concatenated to form larger segments. The definition for concatenation of segments now follows.

Definition 4. Let Ω_i, Ω_j be two segments with $y_L(\Omega_i) + 1 = y_F(\Omega_j)$ and $x(\Omega_i) = x(\Omega_j)$, or $y_L(\Omega_i) = g_k$, $y_F(\Omega_j) = 1$, and $x(\Omega_i) + 1 = x(\Omega_j)$. The concatenation of Ω_i and Ω_j, denoted by $\Omega_i @ \Omega_j$, is the segment Ω where $y_F(\Omega) = y_F(\Omega_i)$, $y_L(\Omega) = y_L(\Omega_j)$, and $x(\Omega) = x(\Omega_i)$.

Lemma 1. *Let* S *be a segment and* S_{g_i} *be the segment at distance* g_i *from* S $(1 \leqslant i \leqslant k)$. *The dead segments in* S *are those in the set* {*faulty elements in* S} $\cup$ $\{\prod_{i=1}^{g_k}$ *shadows of dead segments of* S_{g_i} *at distance* $g_i\}$.

Proof. By definition of dead element, the lemma trivially holds. □

Definition 5. Let $DS[i]$ and $DS[i+1]$ be the set of the dead segments of row(i) and row($i + 1$), respectively. The concatenation of $DS[i]$ and $DS[i + 1]$, denoted by $DS[i] \bullet DS[i + 1]$, is the set of dead elements defined as follows: let

$$a = \text{Max}\big\{y_L(\Omega)\colon\ \Omega \in DS[i]\big\},$$

$$b = \text{Min}\big\{y_L(\Omega)\colon\ \Omega \in DS[i + 1]\big\};$$

and let Ω_a and Ω_b be the corresponding segments.

Then

$$DS[i] \bullet DS[i+1] = \begin{cases} DS[i] \cup DS[i+1] \cup \{\Omega_a @ \Omega_b\} - \{\Omega_a, \Omega_b\} & \text{if } a = g_k \text{ and } b = 1, \\ DS[i] \cup DS[i+1] & \text{otherwise.} \end{cases}$$

4. An efficient testing scheme

4.1. The algorithm

The algorithm proceeds as follows. The given fault pattern F is decomposed into patterns which do not contain any gap. By Theorem 2, F is catastrophic if and only if at least one of the pattern is catastrophic. By Theorem 1, any such pattern F' is catastrophic if and only if there exists a dead segment of size g_k. Thus, the algorithm constructs all the dead segments; in particular, it constructs $DS[i]$ (the set of dead segments of row(i)), given $DS[i-1]$ (the set of dead segments of row$(i-1)$). By Definition 1, a segment S is dead if all k segments at distance $g_1, g_2, \ldots, g_k$ from S are dead. An important aspect of the algorithm which is crucial to its efficiency is that the dead segments of row(i) can only be found among the shadows of the dead segments of row$(i-1)$. Therefore, we can disregard all elements of row(i) which are not in the shadows of the distance equal to the longest bypass link.

Let S be a shadow at distance g_k from some segment in $DS[i-1]$. The algorithm determines which parts of the segments in $DS[i-1] \bullet DS[i]$ are at distance $g_1, g_2, \ldots, g_{k-1}$ from S and compute the intersection of all these parts. Any element in S is dead if and only if it belongs to this intersection. Another aspect of the algorithm crucial for its efficiency is that once the above described process has been completed for S, no backtracking is done when considering the next segment in $DS[i-1]$. If, at any time, a dead segment of size g_k is encountered, the fault pattern is catastrophic. If all dead segments have been constructed and no dead segment of size g_k has been found, the pattern is not catastrophic.

More in detail, given a Boolean matrix representation of a fault pattern F. The algorithm first checks if F is empty. If it is empty, the algorithm stops and reports that the fault pattern is not catastrophic; otherwise, the algorithm scans linearly the fault pattern to detect the existence of gaps (i.e., if $x_{i+1} - x_i > 2$). When the first gap is found, it decomposes F into two fault patterns F_1 and F_2 such that F_1 does not contain any gap. Then the algorithm calls a procedure (**DeadSegment**) to check if F_1 is catastrophic. If F_1 is catastrophic, the algorithm stops; otherwise, it recursively calls itself with F_2 as its input. Note that if F does not contain any gap, then $F_1 = F$ and $F_2 = \emptyset$. The procedure **DeadSegment** is shown in Fig. 4 and the **MergeSegment**() is given in Fig. 5.

Note that $p+1$ is the operation of next(p), and $p-1$ is the operation of previous(p).

4.2. Analysis

Property 2. *Algorithm correctly determines whether or not a fault pattern F is catastrophic.*

Proof. F is decomposed into patterns which do not contain any gap. By Theorem 2, F is catastrophic if and only if at least one of these patterns is catastrophic. Thus, to prove the correctness, it suffices to prove that the algorithm correctly determines whether any such pattern F' is catastrophic. By Theorem 1, F' is catastrophic if and only if there exists a dead segment of size g_k. Thus, it suffices to show that the algorithm correctly constructs all the dead segments; in particular, it suffices to show that $DS[i]$ will be constructed correctly given $DS[i-1]$. By Definition 1, a segment S is dead if all k segments at distance $g_1, g_2, \ldots, g_k$ from S are dead. First of all, observe that it suffices to verify the above condition only for the segments which are shadows at distance g_k from the segments in $DS[i-1]$. In fact, any element of row(i), which is a shadow at distance g_k of a live element, is live; in other words, the only possible candidates for dead elements (and, thus, dead segments) are those which are the shadow at distance g_k of a dead segment in $DS[i-1]$. Let S be a shadow at distance g_k from some segment in $DS[i-1]$. To determine whether an element of S is dead, we must determine whether its shadows at distance $g_1, g_2, \ldots, g_{k-1}$ are dead; that is, we must determine whether its shadows at distance $g_1, g_2, \ldots, g_{k-1}$ are in $DS[i-1] \bullet DS[i]$. The algorithm first determines which parts of the segments in $DS[i-1] \bullet DS[i]$ are at distance $g_1, g_2, \ldots, g_{k-1}$ from S and compute the intersection of all these parts. Any element in S is dead

DeadSegment(F)
Input: A Boolean representation of F which does not contain any gaps.
Output: TRUE if F is catastrophic; FALSE if F is not catastrophic.
Data structure used: $DS[i]$ is an ordered (by the (x, y) coordinates of the starting element) set of dead segments of row(i), and it is implemented as a linked list. For simplicity, let $DS[i](k)$ denote the kth dead segment in $DS[i]$. $DS[i] \bullet DS[j]$ is the concatenation of $DS[i]$ and $DS[j]$, and it is also implemented as a linked list. For simplicity, let $(DS[i] \bullet DS[j])(k)$ denote the kth dead segment in the concatenation of $DS[i]$ and $DS[j]$.
Step 1: Initially, **DeadSegment**() checks if $|F| \geqslant g_k$. If so, it continues; otherwise, it returns FALSE.
Step 2: Then, **DeadSegment**() scans F to form, for each row, the set of dead segments lying on that row. If, at any time, **DeadSegment**() encounters a dead segment encompassing the entire row, it returns TRUE (by Theorem 1, the pattern is catastrophic). If all dead segments in F have been built and no dead segment of size g_k is found, **DeadSegment**() returns FALSE (by Theorem 1, the pattern is not catastrophic). The set of dead segments for row(1), $DS[1]$, is composed of the segments formed by the faulty elements in row(1). Given the set $DS[i-1]$ of dead segments for row($i-1$), the set $DS[i]$ of dead segments for row(i) is constructed as follows:

```
L0: form an initial set DS[i] by merging faulty elements in row(i);
  p = 1;
L1: if DS[i − 1] = ∅ then DS[i] is done;
  α = |DS[i − 1]|;
  for (1 ⩽ d ⩽ k) l_d = 2;
  for (1 ⩽ t ⩽ α) {
    S_k ← DS[i − 1](t);
    Ω ← shadow of S_k at distance g_k;
    for (k − 1 ⩾ j ⩾ 1) {
      l_j = max{l_j, l_{j+1}};
L2:   S_j ← (DS[i − 1] • DS[i])(l_j);
      Ω_j ← shadow of S_j at distance g_j;
      if Ω_j < Ω then
        if l_j = |DS[i − 1] • DS[i]| then goto L3;
          else {l_j = l_j + 1; goto L2;}
      if Ω < Ω_j then goto L3;
      Ω ← Ω ∩ Ω_j;
    }
    MergeSegment(Ω, DS[i], p);
L3:   continue;
  }
```

Fig. 4. Procedure **DeadSegment**(F).

if and only if it belongs to this intersection. Since this is done for every row, the claim follows. □

Property 3. *For any* row(i) *in the Boolean representation of a fault pattern, let* n_i *be the number of faults in that row, and* $|DS(i)|$ *be the number of dead segments in that row, then* $|DS(i)| \leqslant n_i + 1$.

Proof. By contradiction, let $|DS(i)| > n_i + 1$. Then there are at least two dead segments, S_1 and S_2, which do not contain any faults. Without loss of generality, let $S_1 < S_2$; thus $y_F(S_2) > 1$. In other words, $w = (i, y_F(S_2) - 1)$ exists and is live. Since S_2 does not contain any faulty element, all elements in S_2 are reachable by p. This contradicts the fact that S_2 is a dead segment. Thus, the statement holds. □

Property 4. *Algorithm requires* O(kn) *time, where* k *is the number of bypass links, and* n *is the number of faults.*

Proof. The given fault pattern F is decomposed into patterns which do not contain any gap. This is done in time O(n). In the worst case, the procedure

MergeSegment$(\Omega, DS[i], p)$
Input: $\Omega, DS[i], p$;
Output: $DS[i], p$;
Outline: This procedure first scans $DS[i]$ starting from p to find the location for Ω; then inserts Ω into $DS[i]$. The procedure then returns the modified $DS[i]$ and the location of the next segment p.

while $(y_F(\Omega) > y_L(DS[i](p)))$, $p = p + 1$;
$y'_F = \min(y_F(\Omega), y_F(DS[i](p)))$;
$q = p$;
while $(y_L(\Omega) > y_F(DS[i](q)))$, $q = q + 1$;
$y'_L = \max(y_L(\Omega), y_L(DS[i](q-1)))$;
replace all segments from p to $q-1$ with the new segment
$\{(i, y'_F), (i, y'_L)\}$ in $DS[i]$;
$p =$ pointer to this new segment;
return $DS[i]$ and p.

Fig. 5. Procedure **MergeSegment**$(\Omega, DS[i], p)$.

DeadSegment will be applied to each such pattern. For any input F' of **DeadSegment**:

(1) The number of rows in F' is at most $2n'$ where n' is the number of faults in F'. This is because F' does not contain any gap. By Property 3, $|DS(F')|$, the number of dead segments in F', is at most $\sum_i^{2n'}(n'_i + 1) = n' + 2n' = 3n'$. Thus, $|DS(F')|$ is $O(n')$.
(2) **DeadSegment** constructs the overall set DS row by row. It uses $k-1$ pointers (the l_i's). At each step of the execution, each pointer can move forwards ("advance") or not ("stay"); it cannot ever move backwards. The number of "advances" for each pointer is at most $|DS(F')|$, so is the number of "stays". Thus, each pointer requires at most $2|DS(F')| = O(n)$ operations, for a total of $O(kn)$ time for $k-1$ pointers.
(3) As for the cost of the merging operation, the initial merge (executed in step L0 of the algorithm) requires $O(n'_i)$ operations. The total cost of merging new dead segments with $DS[i]$ by calling **MergeSegment** is $O(n'_i)$ since the entire $DS[i]$ is only scanned once. The total cost for F', is $O(n')$.

Therefore, the total cost for F' is $O(kn')$. Thus, the total cost for each pattern F' into which F has been decomposed is $O(kn')$. Since these patterns are disjoint, the total cost of algorithm is $O(kn)$, where k is the number of bypass links and n is the number of faults. □

Theorem 3. *Algorithm correctly determines whether or not a fault pattern is catastrophic in* $O(kn)$ *time, where* k *is the number of bypass links at each element,* n *is the number of faults.*

Proof. The theorem follows from Properties 2 and 4. □

5. Concluding remarks

Let W^* be W with two extra rows of all 0's, one at the top and the other at the bottom. We can view the Boolean matrix W^* as a directed mesh with an additional link structure: $\forall g \in G$, every mesh element (p, q) is also connected to (r, s) where $r = p + 1$ if $q + g \geqslant g_k$, otherwise p, and $s = (q + g) \bmod g_k$.

In this rather complex mesh, elements corresponding to 1 entries in W are *obstacles*. It is not difficult to verify that the pattern corresponding to W is not catastrophic for G if and only if there is an obstacle-avoiding path from any element in the first row to any element in the last row of W^*. Hence, our testing scheme actually solves a more general version of the classical problem of finding a obstacle-avoiding path in a two-dimensional grid (e.g., [11, 12]).

References

[1] R. De Prisco, A. Monti, L. Pagli, Testing and reconfiguration of VLSI linear arrays, Theoret. Comput. Sci. 197 (1998) 171–188.

[2] R. De Prisco, A. De Santis, Catastrophic faults in reconfigurable systolic arrays, Discrete Appl. Math. 75 (1997) 105–129.

[3] A. Nayak, V. Acciaro, P. Gissi, A note on isomorphic chordal rings, Inform. Process. Lett. 55 (1995) 339–341.

[4] A. Nayak, L. Pagli, N. Santoro, On testing of catastrophic faults in reconfigurable arrays with arbitrary link redundancy, Integration, the VLSI J. 20 (1996) 327–342.

[5] A. Nayak, L. Pagli, N. Santoro, Efficient construction of catastrophic patterns for VLSI reconfigurable arrays, Integration, the VLSI J. 15 (2) (1993) 133–150.

[6] A. Nayak, L. Pagli, N. Santoro, Combinatorial and graph problems arising in the analysis of catastrophic fault patterns, in: Proc. 23rd Southeastern Conf. on Combinatorics, Graph Theory, and Computing, Congressus Numerantium 88 (Utilitas Mathematica), 1992, pp. 7–20.

[7] A. Nayak, N. Santoro, R. Tan, Fault-intolerance of reconfigurable systolic arrays, in: Proc. 20th Internat. Symp. on Fault-Tolerant Computers, Newcastle upon Tyne, 1990, pp. 202–209.

[8] L. Pagli, G. Pucci, Reliability analysis of redundant VLSI arrays, Inform. Process. Lett. 50 (1994) 337–342.

[9] J. Ren, Geometric characterization of fault patterns in linear systolic arrays, M.C.S. Thesis, School of Computer Science, Carleton University, Ottawa, Canada, 1994.

[10] P. Sipala, Faults in linear arrays with multiple bypass links, Research Report No. 18, Dipartimento di Informatica, Università Degli Studi di Trieste, Italy, 1993.

[11] P. Widmayer, On graphs preserving rectilinear shortest paths in the presence of obstacles, in: P.L. Hammer (Ed.), Annals of Operations Research, Topological Network Design, Vol. 33, Baltzer, 1991, pp. 557–575.

[12] Y.F. Wu, P. Widmayer, M.D.F. Schlag, C.K. Wong, Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles, IEEE Trans. Comput. C-36 (1987) 321–331.