A Graph Model for Fault-Tolerant Computing Systems

JOHN P. HAYES, MEMBER, IEEE

***Abstract*—An approach to fault-tolerant design is described in which a computing system S and an algorithm A to be executed by S are both defined by graphs whose nodes represent computing facilities. A is executable by S if A is isomorphic to a subgraph of S. A k-fault is the removal of k nodes (facilities) from S. S is a k-fault tolerant (k-*FT*) realization of A if A can be executed by S with any k-fault present in S. The problem of designing optimal k-*FT* systems is considered where A is equated to a 0-*FT* system. Techniques are described for designing optimal k-*FT* realizations of single-loop systems; these techniques are related to results in Hamiltonian graph theory. The design of optimal k-*FT* realizations of certain types of tree systems is also examined. The advantages and disadvantages of the graph model are discussed.**

***Index Terms*—Computer architecture, fault-tolerant computing, fault-tolerant design, graph theory, Hamiltonian graphs, single-loop systems, tree systems.**

I. Introduction

FAULT TOLERANCE in computers has been succinctly defined by Avizienis as "the ability to execute specified algorithms correctly regardless of hardware failures and program errors" [1]. This implies that in order to analyze a computer system S with respect to fault tolerance two basic questions must be answered.

1) What are the algorithms that must be executed correctly by S?

2) What types of faults can occur in S?

When S is a logic network, the algorithms in question are usually well-defined combinational or sequential Boolean functions. A few simple fault models, e.g., the line stuck at 0/1 model, suffice for fault analysis at the logic level [2]. Many techniques are known for fault-tolerant logic design [3]. These techniques generally employ special information codes and redundant circuitry.

When S is an entire computer system whose components may be complex processors, neither the faults that may occur, nor the algorithms to be executed can easily be classified. Many systems are "general purpose," which implies that any algorithm if appropriately encoded can be executed, subject to limitations on the available time and memory space. Another major source of difficulty lies in determining the effect of physical faults on algorithms which are usually specified by programs or flowcharts. Thus the definition of fault tolerance quoted above is by no means easy to apply to computer systems viewed as single entities.

A central concept in fault-tolerant design is the use of redundancy [4]. If a fault in a particular component is to be tolerated, then the system must have redundant or spare facilities to take over the duties of the faulty component. Most previous studies of fault-tolerant systems employ reliability modeling techniques, in which a probabilistic measure of reliability is associated with each component [3], [5]. The "specified algorithms" are generally implicit in the structure of the system, in that the designer's goal is usually to maintain some minimum system configuration in the presence of faults.

In this paper a graph model for fault-tolerant systems is presented. Both computing systems and the algorithms to be executed are explicitly represented by graphs. These graphs display the computing facilities (including spare facilities) involved in a particular computation, as well as the interconnections among them. The model permits a precise nonprobabilistic measure of fault tolerance which is in accord with Avizienis' definition. It is directly applicable to the fundamental problem of determining the minimum configuration or structure required to achieve a specified degree of fault tolerance. This problem is solved for two specific cases: single-loop systems and a class of tree-like systems.

II. The Model

A computing system will be viewed here as a set of interconnected facilities or resources. A *facility* is any hardware or software component of the system that can fail independently of the remaining facilities. The hardware facilities of a system might include control units, arithmetic processors, storage units and input/output equipment. Software facilities might include executive programs (operating systems), compilers, library routines, and applications programs.

Each facility can have access to certain other facilities. For example, a control unit can have access to a memory unit, or a master program can have access to one of its subroutines. Thus we can in a natural way, represent a system by a graph [6], [7] which we term a facility graph.

Definition 1: A *facility graph* is a graph G whose nodes $\{x_i\}$ represent system facilities, and whose edges represent access links between facilities. G is a *directed* facility graph if the edges of G are directed. G is a *labeled* facility graph if a number t_i is associated with each node x_i of G; t_i is interpreted as the facility *type* of x_i.

Fig. 1 shows 3 labeled directed facility graphs where the

Manuscript received January 24, 1975; revised July 1975 and December 1975. This work was supported by the National Science Foundation under Grant GK-37455 and by the Joint Services Electronics Program through the Air Force Office of Scientific Research/AFSC under Contract F44620-71-C0067.

The author is with the Department of Electrical Engineering and the Computer Science Program, University of Southern California, Los Angeles, CA 90007.

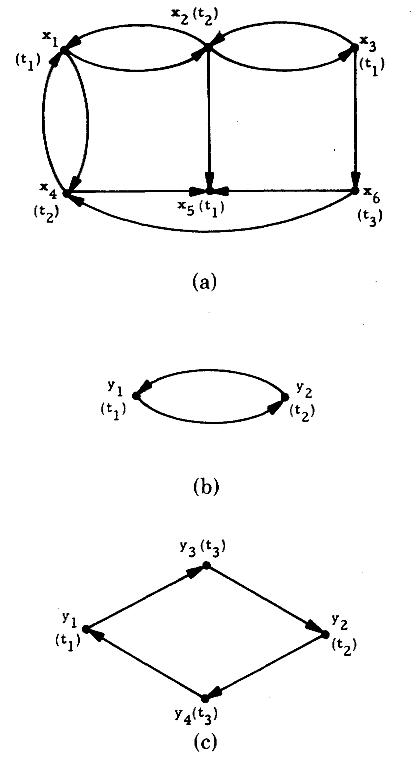


Fig. 1. Three labeled directed facility graphs (a) G_1, (b) G_2, and (c) G_3.

facility types are indicated by numbers in parentheses. Facility graphs will be used to represent both computing systems and algorithms. A *computing system* will be defined by a facility graph whose nodes are the facilities, including spare facilities, present in the system, and whose edges are access links between the facilities. An *algorithm* will be defined by a facility graph whose nodes represent the facilities required to execute the algorithm, and whose edges represent the links required among these facilities.

Definition 2: An algorithm A is *executable* by a computing system S if A is isomorphic to a subgraph of S, i.e., there is a one-to-one mapping from the nodes of A into the nodes of S that preserves node labels and adjacencies between nodes. We will also say that *S contains A*.

Intuitively, this definition means that S contains all the facilities and connections between facilities required by A, so that A can be embedded in S. If G_1 and G_2 shown in Fig. 1 represent a system and an algorithm, respectively, then G_2 is executable by G_1. There are several isomorphisms from G_2 into G_1, e.g., $(y_1,y_2) \rightarrow (x_1,x_2)$ and $(y_1,y_2) \rightarrow (x_3,x_2)$.

Definition 3: A *k-fault* F in a system S is the removal of any k nodes $\{x_1,x_2, \cdots ,x_k\}$ from S. All edges connected to these nodes are also removed. The resultant graph will be denoted by S^F. We will also write $F = \{x_1,x_2, \cdots ,x_k\}$.

We can now define the basic concepts relating to fault tolerance.

Definition 4: A system S is *fault tolerant* with respect to algorithm A and fault F if A is executable by S^F; in other words, A is isomorphic to a subgraph of S^F, or S^F contains A.

Definition 5: S is fault tolerant with respect to a set of algorithms $\{A_1,A_2, \cdots ,A_p\}$ and a set of faults $\{F_1,F_2, \cdots ,F_q\}$ if A_i is executable by S^{F_j} for all i and j where $1 \leqq i \leqq p$ and $1 \leqq j \leqq q$.

Definition 6: S is *k-fault tolerant* (*k-FT*) with respect to algorithms $\{A_1,A_2, \cdots ,A_p\}$ if for every k-fault F in S, A_i is executable by S^F when $1 \leqq i \leqq p$.

If S is *k-FT* with respect to A, then S is *j-FT* with respect to A, for all j where $0 \leqq j \leqq k$. Consider again the graphs of Fig. 1. G_1 is *1-FT* with respect to G_2. It is not *2-FT* because if the fault $F = \{x_2,x_4\}$ is present, G_2 cannot be executed by G_1^F.

A number of major assumptions are implicit in this model of fault tolerance.

1) There is some mechanism for the detection and isolation of faults in S. This testing capability may be internal to S, in which case S is self-diagnosing, or it may be external to S.

2) When any tolerated fault is detected, S must be capable of reconfiguring itself or being reconfigured so that the execution of an algorithm is shifted from faulty facilities to previously unused fault-free facilities.

Thus there are significant test, reconfiguration, and recovery processes which are not explicitly included in the model, but are necessary to make it meaningful. These processes are in effect being treated as parts of an inherently fault-free "hard core" of the system. When using our model to gauge the tolerance of S with respect to some fault F we are taking into account *only* the following two questions.

1) Does S^F contain all the facilities required to execute A?

2) Is the logical linkage among the facilities of S^F sufficient for the execution of A?

The selection of the facilities to be modeled in a particular system is primarily determined by the possible fault modes. It is implicit in the foregoing definition of a k-fault that the k-fault makes k facilities completely inoperative, i.e., unavailable for execution of algorithms. Partial or temporary failures are not covered by this model.

Faults are associated with the removal of nodes rather than edges. This is not a restrictive assumption, since by inserting a "dummy node" x in an edge e we can equate the removal of e to the removal of x. For example, the graph G_2 in Fig. 1(b) could represent two communicating processors y_1 and y_2. Suppose the physical communication mechanisms, e.g., data and control buses, between y_1 and y_2 are subject to failure. Then we can represent these buses by nodes y_3 and y_4 as shown in the facility graph G_3 of Fig. 1(c). A single fault causing loss of communication from y_1 to y_2 would be represented by the 1-fault $F = \{y_3\}$ in G_3.

Graphs appear to be particularly suitable for the analysis of complex digital systems [8]. Graph models similar to facility graphs have been successfully applied to the study of fault diagnosis at the system level [9], [10]. The PMS descriptions of computer systems introduced by Bell and Newell [11] are facility graphs.

Often the requirements for fault tolerance are specified in terms of a system configuration S_0 that must be maintained under all tolerated fault conditions. This system S_0 is usually minimal in the sense that S_0 itself is 0-*FT* and no edges can be removed from S_0. Hence S_0 can be viewed as a single algorithm to be executed. A fundamental question to which the rest of this paper is devoted follows.

Problem 1: Given a 0-*FT* system S_0, design a system S_k that is k-*FT* with respect to S_0 and has minimal cost.

The design of S_k can be viewed as an extension of the underlying system S_0 by the addition of spare (redundant) nodes and edges. The cost of S_k can be measured in many ways; in general, it will be some function of the number of nodes and edges in S_k. It seems reasonable to assume that the nodes (facilities) cost more than the edges. We will therefore make node minimization the primary design objective and edge minimization the secondary objective. Thus Problem 1 becomes the following.

Problem 2: Given a system S_0, design a system S_k such that:

1) S_k is k-*FT* with respect to S_0;
2) S_k contains as few nodes as possible;
3) no system satisfying conditions 1) and 2) has fewer edges than S_k. S_k will be called an *optimal k-FT realization* of S_0.

It is relatively easy to construct nonoptimal systems that satisfy conditions 1) and 2) of Problem 2. For example, the complete graph [6] with $n_i + k$ nodes of type t_i is such a system. In order to tolerate a k-fault affecting type t_i nodes only, S_k must contain at least $n_i + k$ nodes of this type. Thus every solution S_k to Problem 2 contains exactly $n_i + k$ nodes of type t_i. It is therefore a trivial task to satisfy condition 2) of Problem 2.

Condition 1) is usually much more difficult to satisfy. In order to determine if this condition is met by a particular design S_k, it is necessary to prove that for every k-fault F, S_k^F contains S_0; this is the well-known subgraph isomorphism problem [12]. The amount of computation required to determine if S_k^F contains S_0 can be very great; the problem may not be computationally feasible if the graphs involved are very large or complex.

Condition 3) is also difficult to satisfy in general. The degrees of the various nodes in S_0 can be a useful clue to the number of edges required in S_k. If S_0 contains n_i' nodes of type t_i that have an in (out) degree d_i then S_k must contain at least $n_i' + k$ type t_i nodes having an in (out) degree of d_i or greater.

Fortunately, many of the facility graphs of interest have relatively simple structures. Most of the PMS graphs of computer systems as noted in [11] are either trees or lattices. Bus control and other communications networks frequently take the form of simple closed loops [13]. Thus it is not unrealistic to restrict attention to special graphs such as single-loop and tree graphs, where, as we will show in the following sections, Problem 2 can be solved without excessive computation.

III. Optimal k-*FT* Single-Loop Systems

In this section we consider the design of optimal k-*FT* sytems where the underlying system S_0 consists of n facilities connected in a single loop. As noted already, such systems are frequently encountered in communications networks. The fault diagnosis problem for single-loop systems is studied in [10]. For simplicity it will be assumed that there is only one facility type, and that S_0 is represented by an unlabeled undirected facility graph.

Definition 7: A *single-loop system* C_n is a graph consisting of $n \geqq 3$ nodes each of degree 2, i.e., C_n is a *cycle* of n nodes [6].

The degree of a node x will be denoted by $d(x)$. If r is a real number, $\lfloor r \rfloor$ and $\lceil r \rceil$ will be used to denote the largest integer $\leqq r$ and the smallest integer $\geqq r$, respectively. From the foregoing assumptions, every optimal k-*FT* realization of C_n contains $n + k$ nodes.

Lemma 1: Let C be any optimal k-*FT* realization of the single-loop system C_n. The degree of every node in C is at least $k + 2$.

Proof: Suppose $d(x) \leqq k + 1$ for some node x of C. There exists a k-fault F in C such that F removes all but one of the nodes connected to x, but does not remove x itself. C^F must contain a subgraph G isomorphic to C_n. Since C is optimal, x must be a node of G which is impossible since $d(x) = 1$ in C^F. Hence $d(x) \geqq k + 2$. ■

Theorem 1: If m is the number of edges in any k-*FT* single-loop system, then $m \geqq \lceil (k + 2)(n + k)/2 \rceil$.

Proof: By Lemma 1, there are at least $k + 2$ edges connected to each of the $n + k$ nodes in the system. Since each edge connects two distinct nodes, $m \geqq \lceil (k + 2)(n + k)/2 \rceil$. ■

Corollary 1: C_3 has a unique optimal k-*FT* realization which is K_{k+3}, the complete graph [6] of order $k + 3$.

Let G be a graph with $n + k$ nodes. In order to prove that G is k-*FT* with respect to C_n, we must show that for every k-fault F, G^F contains a cycle passing through all its nodes. This is equivalent to proving that G^F is a Hamiltonian graph [6], [7]. A graph H is *Hamiltonian* if it contains a cycle C passing through all its nodes; C is called a Hamiltonian cycle. Some related concepts from Hamiltonian graph theory are also relevant. Two nodes x_1,x_2 of H are *Hamiltonian connected* [7] if there is a loop-free path from x_1 to x_2 that passes through every node of H; this path is called a Hamiltonian path. H is Hamiltonian connected if every pair of nodes in H are Hamiltonian connected. Every Hamiltonian connected graph is Hamiltonian, but the converse is false. H is *j-Hamiltonian* [14] if the removal of any i nodes from H where $0 \leqq i \leqq j$ results in a Hamiltonian graph. The following theorem is immediate.

Theorem 2: If G is a k-Hamiltonian graph with $n + k$ nodes then G is k-*FT* with respect to C_n.

We now present procedures for designing an optimal k-*FT* single-loop system $C_{n,k}$ for all values of n and k. Three separate cases are considered: $k = 1$, k even, k odd and greater than 1. Distinct design methods are given for the three cases. A common characteristic of all the designs presented is that the number of edges employed is exactly $\lceil (k + 2)(n + k)/2 \rceil$ so that optimality follows directly from Theorem 1.

Theorem 3: The graphs $C_{n,1}$ shown in Fig. 2(a) and (b) are optimal 1-*FT* realizations of C_n for n odd and even, respectively.

Proof: Consider the case where n is odd. Suppose one of the "endpoints" x_1 or x_{n+1} of $C_{n,1}$ in Fig. 2(a), say x_1, is removed by a 1-fault F. The resulting graph $C_{n,1}^F$ contains the Hamiltonian cycle shown in Fig. 3(a). If a point x_i is removed from $C_{n,1}$ by F, and $i \neq 1$ or $n + 1$, then $C_{n,1}^F$ contains a Hamiltonian cycle of the form shown in Fig. 3(b). Hence $C_{n,1}$ is 1-*FT*. Every node of $C_{n,1}$ has degree 3, hence $C_{n,1}$ contains $3(n + 1)/2$ edges and is optimal by Theorem 1.

The proof for n even is similar. In this case, every node of $C_{n,1}$ has degree 3 except x_{n+1} which has degree 4. The number of edges is $\lceil 3(n + 1)/2 \rceil$ which is again optimal by Theorem 1. ■

It is interesting to note that the graphs of Fig. 2 are also Hamiltonian-connected graphs of order $n + 1$ having the smallest number of edges (attributed to Moon [7]).

We now define a special graph L_n which will prove useful later.

Definition 8: L_n is the graph with $n \geqq 3$ nodes and $2n - 3$ edges constructed as follows. Let $(x_1, x_2, \cdots, x_n)$ be some ordering of n nodes. For each node x_i where $1 \leqq i \leqq n$, introduce an edge connecting x_i to each x_j satisfying $i < j \leqq i + 2$.

Fig. 4 shows the general form of L_n.

Lemma 2: Two nodes x_i and x_j of L_n are Hamiltonian connected if and only if at least one of the following is true:

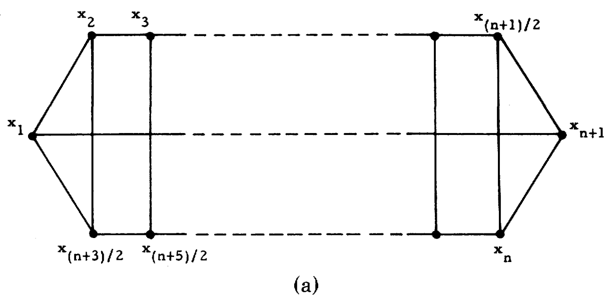
1) $i = 1$ or $i = n$;
2) $|i - j| \geqq 2$.

Proof: (If) There are 3 possibilities:

1) $i = 1$ and $j = n$;
2) $i = 1$ and $j \neq n$;
3) $i,j \notin \{1,n\}$ and $|i - j| \geqq 2$.

Fig. 5(a), (b), and (c) shows Hamiltonian paths joining x_i and x_j in each of these 3 cases.

(Only if) Suppose i and j do not satisfy either condition 1) or 2), then $j = i + 1$ and $i,j \notin \{1,n\}$. Suppose by way of contradiction that there is a Hamiltonian path P joining x_i and x_j. x_i is adjacent in P to either x_{i+2} or x_h where h is $i - 1$ or $i - 2$. If P connects x_i directly to x_h, then P, after passing through nodes to the left of x_i, must eventually bypass x_i to visit nodes to the right of x_i; this can be done only by P passing through x_{i+1} as shown in Fig. 5(c). Since $x_{i+1} = x_j$ and $j \neq n$, x_j cannot be an endpoint of P, which contradicts the fact that P is a Hamiltonian path from x_i to x_j. A similar argument holds when P connects x_i to x_{i+2}. ■



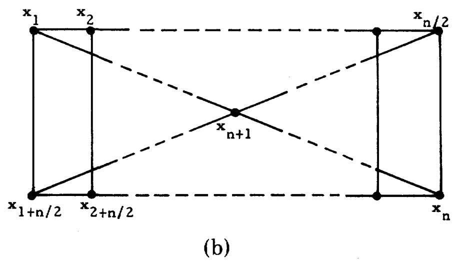
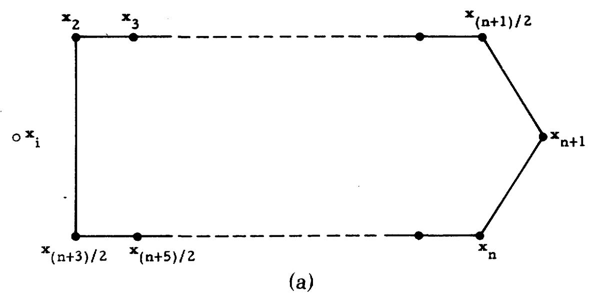


Fig. 2. (a) $C_{n,1}$ for n odd. (b) $C_{n,1}$ for n even.



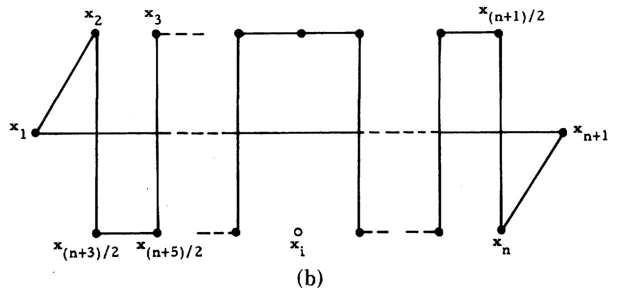


Fig. 3. Hamiltonian cycles in $C_{n,1}^F$ for n odd and $F = \{x_i\}$.

Corollary 2: L_n contains C_n.

We now present a procedure for constructing $C_{n,k}$ when k is even.

Procedure 1: To construct an optimal k-*FT* realization $C_{n,2p}$ of the single-loop system C_n when $k = 2p$ is even.

1) Form the single loop system C_{n+k}.

2) Join every node x_i of C_{n+k} to all nodes at distance j from x_i in C_{n+k}, for all j satisfying $2 \leqq j \leqq p + 1$. The resulting graph is $C_{n,2p}$. ■

Theorem 4: The graph $C_{n,2p}$ generated by Procedure 1 is an optimal $2p$-*FT* realization of C_n.

Proof: It is convenient to represent $C_{n,2p}$ as shown in Fig. 6(a) where the single loop C_{n+k} introduced in step 1)

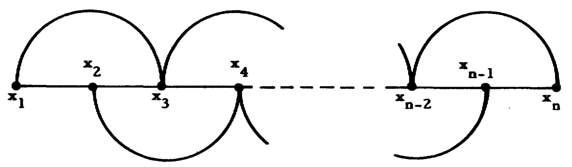


Fig. 4. The graph L_n.

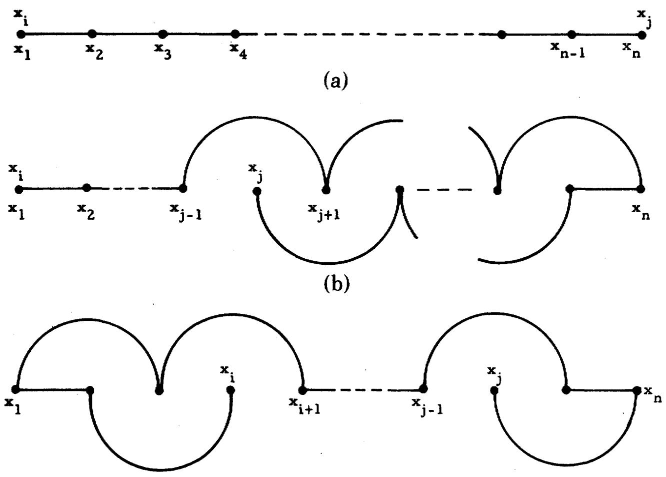
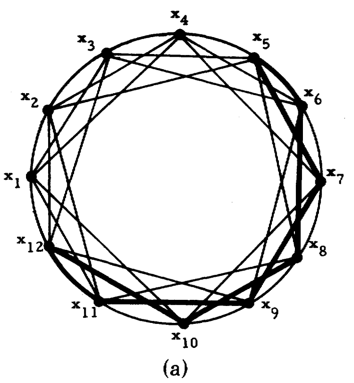
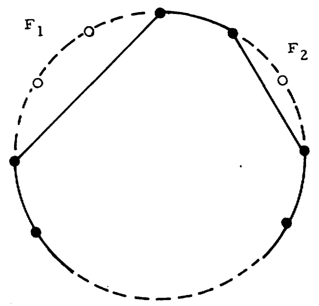


Fig. 5. Hamiltonian paths in L_n.





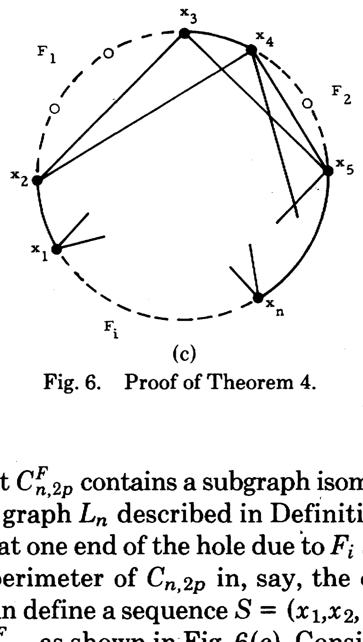


Fig. 6. Proof of Theorem 4.

of Procedure 1 forms the perimeter of $C_{n,2p}$, while the edges added in step 2) are drawn as internal chords.

Now $d(x_i) = 2$ for every node x_i in C_{n+k}. Step 2) connects x_i to $2p$ additional nodes, hence $d(x_i) = 2p + 2 = k + 2$ for every node x_i in $C_{n,2p}$. If $C_{n,2p}$ is k-FT with respect to C_n, then by Theorem 1 it is also optimal. We now show that $C_{n,2p}$ is indeed k-FT.

Let F be any k-fault in $C_{n,2p}$. F consists of $q \leqq 2p$ subfaults $F_1,F_2, \cdots ,F_q$ with the following properties.

1) All nodes in each F_i, where $1 \leqq i \leqq q$, form a sequence of connected nodes along the perimeter C_{n+k} of $C_{n,2p}$.

2) No node of F_i is adjacent in the perimeter subgraph C_{n+k} to any node of $F_j \neq F_i$.

In other words, the q subfaults $F_1,F_2, \cdots ,F_q$ form q disjoint "holes" in the perimeter of $C_{n,2p}$ as illustrated in Fig. 6(b). There are two cases to consider depending on whether or not any of the subfaults F_i contains more than p nodes.

Case 1: $|F_i| \leqq p$ for all $i = 1,2, \cdots ,q$. Let x_{i1} and x_{i2} denote the two nodes of C_{n+k} that are adjacent to nodes of F_i, but are not in F_i and therefore not in F. The distance between x_{i1} and x_{i2} in C_{n+k} is, at most, $p + 1$, hence x_{i1} and x_{i2} are joined by a "chord" of $C_{n,2p}$ that was added in step 2) of Procedure 1. Intuitively speaking, the "hole" in C_{n+k} created by F_i is "spanned" by this edge as shown for F_1 and F_2 in Fig. 6(b). The q edges that span $F_1,F_2, \cdots ,F_q$ and the edges of C_{n+k} that remain along the perimeter of $C^F_{n,2p}$ form a closed loop through all n nodes of $C^F_{n,2p}$. Hence $C_{n,2p}$ tolerates the k-fault F.

Case 2: For some i, $|F_i| \geqq p + 1$. This implies that the hole due to F_i is not spanned by any edge of $C^F_{n,2p}$. We will now show that $C^F_{n,2p}$ contains a subgraph isomorphic to the Hamiltonian graph L_n described in Definition 8.

Beginning at one end of the hole due to F_i and traveling around the perimeter of $C_{n,2p}$ in, say, the clockwise direction, we can define a sequence $S = (x_1,x_2, \cdots ,x_n)$ of all n nodes in $C^F_{n,2p}$ as shown in Fig. 6(c). Consider any node x_j in S where $j \leqq n - 1$. There are at most $p - 1$ nodes in $F' = F - F_i$ since $|F| = 2p$ and $|F_i| \geqq p + 1$. Hence the distance from x_j to x_{j+1} in the perimeter subgraph C_{n+k} of $C_{n,2p}$ is at most p, since at most $|F'|$ nodes of F can separate x_j and x_{j+1}. It follows that x_j and x_{j+1} are joined by step 2) of Procedure 1. Suppose $j \leqq n - 2$. Then the distance from x_j to x_{j+2} in the perimeter subgraph C_{n+k} is, at most, $p + 1$, since, at most, $|F'|$ nodes of F and the node x_{j+1} can separate x_j and x_{j+2}. Again x_j and x_{j+2} must be joined by step 2) of Procedure 1. Hence by Definition 8, $C^F_{n,2p}$contains L_n, and so by Corollary 2 it also contains C_n. Thus in all cases $C^F_{n,2p}$ contains C_n which proves the theorem. ■

Fig. 6(a) shows $C_{8,4}$. It also shows how the 4-fault $F = \{x_1,x_2,x_3,x_4\}$ is tolerated; the subgraph in heavy lines is isomorphic to C_8. We now modify Procedure 1 to handle odd values of k.

Procedure 2: To construct an optimal k-*FT* realization $C_{n,2p+1}$ of the single-loop system C_n when $k = 2p + 1 \geqq 3$.

1) Construct $C_{n+1,2p}$ using Procedure 1. As before, draw this graph with C_{n+k} forming its perimeter, and the remaining edges forming internal chords.

2) Beginning at some node x_1, traverse the perimeter of $C_{n+1,2p}$ in a fixed direction, say clockwise. Let $x_1, x_2, \cdots, x_b$ where $b = \lfloor (n + k + 1)/2 \rfloor$, be the first b nodes encountered. For $1 \leqq i \leqq b$, connect x_i to the node x_i' at distance b from x_i, the distance being measured clockwise along the perimeter. The resulting graph is $C_{n,2p+1}$. ■

Fig. 7 shows $C_{6,3}$ in which case $p = 1$ and $b = 5$. The b edges added in step 2) of Procedure 2 will be referred to as "*bisectors*" since each such edge approximately bisects the perimeter C_{n+k} of $C_{n,2p+1}$. Note that when $n + k$ is odd, x_1 is an endpoint of two bisectors; in all other cases each node of $C_{n,2p+1}$ is an endpoint of exactly one bisector.

Theorem 5: The graph $C_{n,2p+1}$ generated by Procedure 2 is an optimal $(2p + 1)$-*FT* realization of C_n for $(2p + 1) \geqq 3$.

The proof of this theorem is similar to that of Theorem 4; due to its length, however, it is relegated to the Appendix.

In general, Procedure 2 cannot be used to construct 1-*FT* realizations of C_n. It can easily be verified that the system obtained by applying this procedure with $n = 6$ and $k = 1$ is not 1-*FT* with respect to C_6.

IV. Further Examples of Optimal k-*FT* Systems

The results of the preceding section on k-*FT* single-loop systems can be directly extended to the class of systems whose facility graphs are single open paths.

Definition 9: A single-path system is one whose facility graph is a single path. Following Harary [6] we will denote a path with n nodes by P_n.

The following results are analogous to Lemma 1 and Theorem 1 and they are proved in a similar way.

Lemma 3: Let $P_{n,k}$ be any optimal k-*FT* realization of the single-path system P_n. The degree of every node in $P_{n,k}$ is at least $k + 1$.

Theorem 6: If m is the number of edges in any k-*FT* single-path system, then $m \geqq \lceil (k + 1)(n + k)/2 \rceil$.

Theorem 7: Let $C_{n+1,k-1}$ be an optimal $(k - 1)$-*FT* realization of the single-loop system C_{n+1}. $C_{n+1,k-1}$ is also an optimal k-*FT* realization of P_n.

Proof: By Theorem 1, the number of edges in $C_{n+1,k-1}$ is $\lceil (k + 1)(n + k)/2 \rceil$, hence by Theorem 6, if $C_{n+1,k-1}$ is k-*FT* with respect to P_n, it is also optimal. Now a $(k - 1)$-fault F in $C_{n+1,k-1}$ is tolerated, so $C^F_{n+1,k-1}$ contains a subgraph G isomorphic to C_{n+1}. An additional 1-fault in $C^F_{n+1,k-1}$ reduces the single-loop G to a single path with n nodes, hence P_n is executable by $C_{n+1,k-1}$ with any k-fault present. ■

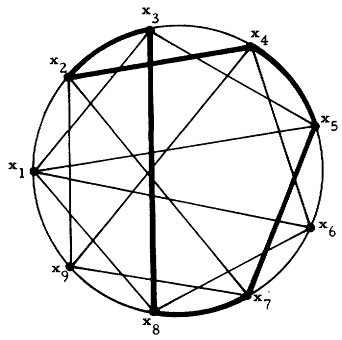


Fig. 7. $C_{6,3}$ generated by Procedure 2.

Thus, for example, the graph $C_{6,3}$ of Fig. 7 is a 4-*FT* realization of P_5. P_n is a simple type of tree graph. Trees are important structures in computing hardware systems; they are also considered by Knuth to be "the most important nonlinear structures arising in computer algorithms" [15]. The solution of Problem 2 for arbitrary trees appears to be very difficult. Although S_0 is a tree, S_k may have very little resemblance to a tree, cf. Theorem 7.

Trees in computer systems often have the form of rooted hierarchical trees where each level of the tree represents a different level of control within a control (or command) hierarchy. In such cases it may be reasonable to associate a unique facility type with each level. Fig. 8 shows a 3-level tree structure which is typical of many computer hardware configurations, where the facility types associated with levels 0, 1, and 2 are CPU's, I/O processors, and I/O devices, respectively.

We now consider the design of optimal 1-*FT* realizations of hierarchical tree systems. To simplify the problem, we restrict attention to a class of symmetric trees characterized by 2 parameters: the degree of the nonterminal nodes d, and the number of levels p.

Definition 10: A symmetric hierarchical tree or *SH-tree* $T_{d,p}$ is a rooted tree with the following properties.

1) Every nonterminal node has degree d, i.e., $T_{d,p}$ is a d-ary tree [15].

2) The level of node x is the distance of x from the root node x_0. With each level i of $T_{d,p}$ is associated a facility type t_i such that all nodes on level i are of type t_i, and $t_i \neq t_j$ for all $i \neq j$.

3) Every terminal node is at distance $p - 1$ from x_0; hence there are d^i nodes in each level i of $T_{d,p}$.

Fig. 8 shows the *SH*-tree $T_{3,3}$.

Procedure 3: To construct an optimal 1-*FT* realization $T_{d,p,1}$ of the *SH*-tree $T_{d,p}$.

1) Define a set $X_i = \{x_{i,0}, x_{i,1}, \cdots, x_{i,d^i}\}$ of $d^i + 1$ type t_i nodes for each i where $0 \leqq i \leqq p - 1$. Partition X_{i+1} into $d^i + 1$ disjoint subsets $Y_0, Y_1, Y_2, \cdots, Y_{d^i}$ such that $Y_0 = \{x_{i+1,0}\}$ and $Y_j = \{x_{i+1,j,1}, x_{i+1,j,2}, \cdots, x_{i+1,j,d}\}$ for $1 \leqq j \leqq d^i$.

2) Join each node $x_{i,j} \in X_i$ to all members of $Y_j \subseteq X_{i+1}$ as shown in Fig. 9(a).

3) For $0 \leqq j \leqq d^i - 1$, join each $x_{i,j} \in X_i$ to $x_{i+1,j+1,2}$ and join x_{i,d^i} to $x_{i+1,0}$ as shown in Fig. 9(b).

4) Join $x_{i,0}$ to $x_{i+1,j,k}$ for all j and k such that $1 \leqq j \leqq d^i$

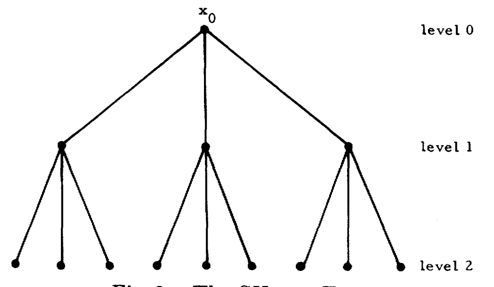


Fig. 8. The *SH*-tree $T_{3,3}$.

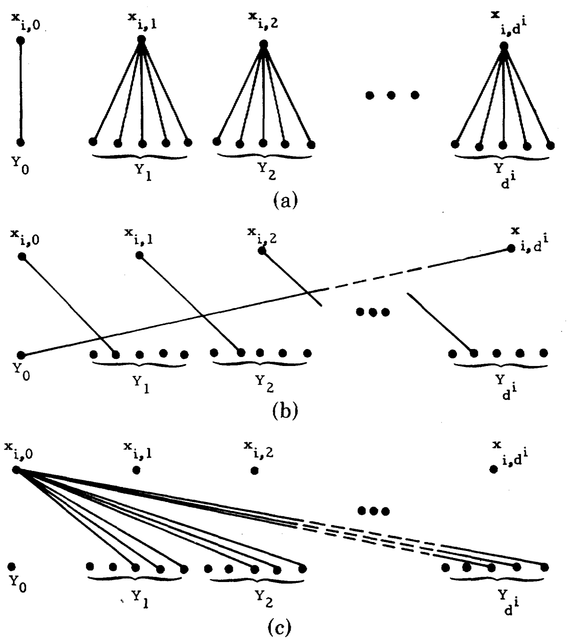


Fig. 9. Illustration of Procedure 3.

and $3 \leqq k \leqq d$, as shown in Fig. 9(c). The resulting graph is $T_{d,p,1}$. ■

Fig. 10 shows the graph $T_{3,3,1}$ generated by Procedure 3. Step 2) constructs a copy of $T_{d,p}$ containing all nodes except $x_{0,0}, x_{1,0}, \cdots ;x_{p-1,0}$; these p nodes can be viewed as spares. It is also convenient to regard the set X_i of type t_i nodes as forming level i of $T_{d,p,1}$.

Lemma 4: Let T be any optimal 1-*FT* realization of $T_{d,p}$ with $d \geqq 2$. If m_i is the number of edges in T connecting nodes in level i to nodes in level $i + 1$, then $m_i \geqq 2(d^{i+1} + 1) - d^i$.

Proof: Let X_i denote the level i nodes of T. Every node $x_{i,j} \in X_i$ of T is adjacent to a set $Y_j \subset X_{i+1}$. First we show that there can be, at most, one node of Y_j that is not adjacent to some node $x_{i,k} \neq x_{i,j}$ in X_i.

Suppose by way of contradiction that two members of Y_j, say y_1 and y_2, are not adjacent to any level i node besides $x_{i,j}$. If the fault $F = \{x_{i,j}\}$ occurs, both y_1 and y_2 are effectively disconnected from X_i. Since T is an optimal 1-*FT* realization, there is only one extra node in X_{i+1}, hence at least one of the nodes y_1 and y_2 is necessary to reconstruct $T_{d,p}$ from the faulty system. This is impossible if both are disconnected from X_i. Hence we conclude that either y_1 or y_2 is adjacent to at least two nodes of X_i. Thus each of the $d^i + 1$ subsets Y_j of X_{i+1} contains, at most, one node that is adjacent to only one node, $x_{i,j}$, on level i. We now show that at most d^i nodes of X_{i+1} can have this property.

Suppose by way of contradiction that for every node $x_{i,j} \in X_i$, there is a node $y_j \in X_{i+1}$ such that y_j is not adjacent to any node in X_i besides $x_{i,j}$. Let $F = \{z\}$ be any 1-fault where $z \in X_{i+1}$, and z is adjacent to at least two nodes in X_i. Since T is 1-*FT*, T^F must contain a subgraph S that is isomorphic to $T_{d,p}$. S must have d^i and d^{i+1} nodes on levels i and $i + 1$, respectively. Hence there is some level i node $x_{i,j}$ that is not included in S. This implies that the node $y_j \in X_{i+1}$ is also not included in S, since $x_{i,j}$ is the only level i node to which it is joined. It follows that neither of the level $i + 1$ nodes y_j and z can be included in S. This is impossible since level $i + 1$ of S must contain d^{i+1} of the $d^{i+1} + 1$ nodes of X_{i+1}.

Thus we conclude that there at most d^i nodes in X_{i+1} that are each adjacent to exactly one node in X_i. There are, therefore, at least $d^{i+1} + 1 - d^i$ nodes in X_{i+1} that are adjacent to two or more nodes in X_i. Hence $m_i \geqq 2(d^{i+1} + 1 - d^i) + d^i = 2(d^{i+1} + 1) - d^i$. ■

Theorem 8: The graph $T_{d,p,1}$ generated by Procedure 3 is an optimal 1-*FT* realization of $T_{d,p}$.

Proof: When $d = 1$, the theorem is trivially true; we will assume, therefore, that $d \geqq 2$. Consider any two adjacent levels of $T_{d,p,1}$ involving the node sets X_i and X_{i+1}. We will show that on removing any $x \in X_i \cup X_{i+1}$, we can always find d^i disjoint copies of $T_{d,1}$ between levels i and $i + 1$. It follows immediately that $T_{d,p,1}$ is 1-*FT* with respect to $T_{d,p}$. There are 3 cases to consider.

Case 1: $x = x_{i,0}$ or $x_{i+1,0}$, i.e., one of the "spare" nodes is removed. Step 2) of Procedure 3 generates d^i disjoint copies of $T_{d,1}$ between levels i and $i + 1$ as shown in Fig. 9(a), none of which uses a spare node.

Let $U = U_1, U_2, \cdots, U_{d^i}$ be the d^i copies of $T_{d,1}$ introduced in step 2) of Procedure 3 where U_j has root node $x_{i,j}$. In the remaining cases, the members of U are modified slightly to reconfigure around the missing node x.

Case 2: $x = x_{i+1,j,k}$ where $1 \leqq j \leqq d^i$ and $1 \leqq k \leqq d$. The edge $x_{i,j}x_{i+1,j,k}$ in U_j is replaced by $x_{i,j}x_{i+1,j+1,2}$ (which is generated by step 2) of Procedure 3) to form a new $T_{d,1}$ tree U'_j. Similarly, for all $j < h < d^i$, $x_{i,h}x_{i+1,h+1,2}$ is substituted for $x_{i,h}x_{i+1,h,2}$ in U_h to form U'_h. Finally, $x_{i,d^i}x_{i+1,0}$ is substituted for $x_{i,d^i}x_{i+1,d^i,2}$ in U_{d^i} to form U'_{d^i}. The trees $U_1, \cdots, U_{j-1}, U'_j, U'_{j+1}, \cdots, U'_{d^i}$ form the desired d^i disjoint copies of $T_{d,1}$ connecting levels i and $i + 1$.

Case 3: $x = x_{i,j}$ where $1 \leqq j \leqq d^i$. Substitute $x_{i,h}x_{i+1,h+1,2}$ for $x_{i,h}x_{i+1,h,2}$ in U_h to form U'_h for all h such that $1 \leqq h < j$. U_j is replaced by U'_0 which has root node $x_{i,0}$ and edges $x_{i,0}x_{i+1,0}, x_{i,0}x_{i+1,1,2}$ and $x_{i,0}x_{i+1,j,k}$ for $k = 3,4, \cdots, d$. The trees $U'_0, U'_1, \cdots, U'_{j-1}, U_{j+1}, \cdots, U_{d^i}$ constitute the desired d^i disjoint copies of $T_{d,1}$.

Hence on removing any node from X_i or X_{i+1}, we can find d^i disjoint copies of $T_{d,1}$ between levels i and $i + 1$, therefore $T_{d,p,1}$ is 1-*FT* with respect to $T_{d,p}$.

The number of edges between levels i and $i + 1$ introduced by steps 2), 3), and 4) of Procedure 3 are $d^{i+1} + 1$,

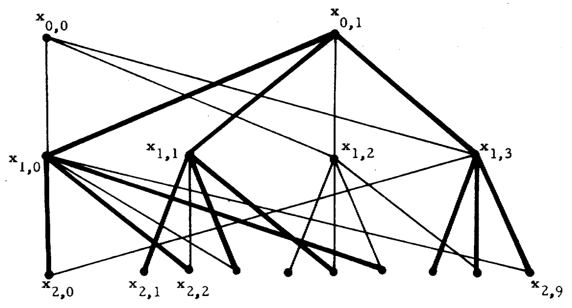


Fig. 10. $T_{3,3,1}$ generated by Procedure 3.

$d^i + 1$, and $(d - 2)d^i$, respectively. Hence the total number of edges between these levels is $2(d^{i+1} + 1) - d^i$, so by Lemma 4, $T_{d,p,1}$ is optimal. ∎

Fig. 10 shows how the 1-fault $F = \{x_{1,2}\}$ in level 1 is tolerated by $T_{3,3,1}$; the heavy lines form a subgraph that is isomorphic to $T_{3,3}$ and does not include $x_{1,2}$.

The extension of Procedure 3 to more general hierarchical trees appears to be difficult. When $d = 1$, $T_{1,p}$ is a "hierarchical path" with p nodes, and $T_{1,p,1}$ consists of 2 disjoint copies of $T_{1,p}$. Note that only steps 1) and 2) of Procedure 3 contribute edges to $T_{1,p,1}$. This example is a special case of the following easily proven theorem.

Theorem 9: Let S_0 be a labeled (directed) facility graph in which no two nodes have the same facility type. Let S_k be the graph consisting of $k + 1$ disjoint copies of S_0. S_k is an optimal k-FT realization of S_0.

V. Discussion

The central idea behind the approach to fault tolerance presented here is the use of facility graphs to represent both computing systems and the algorithms they may execute. This graph model permits a precise definition of the basic concept of a k-fault tolerant system. The model presented here is structural rather than behavioral; it does not take into account computing performance or timing considerations as do, for example, Petri nets, or the "computation graphs" of Karp and Miller [16]. Indeed, there is no explicit representation of the information being processed. Nevertheless, structural models of this type have proven very valuable in the analysis of complex systems [8]–[11], [17].

As noted in Section II, there are several practical difficulties associated with the application of this model. It may not be easy to define the facilities of an actual system in such a way that all 1-faults are independent. Furthermore, the computation required to determine if a given system is k-FT or optimal may be considerable. We have shown that the optimal design problem (Problem 2) can be solved without undue difficulty for certain types of systems. These systems are highly specialized; nevertheless, they reflect, to some extent, the structure of real systems; cf., the fault-tolerant multiprocessor system designed at the C.S. Draper Laboratory [18]. When dealing with systems whose facility graphs are relatively complex, it may not be feasible to design strictly optimal k-FT systems. It appears to be reasonable, however, to attempt to design near-optimal k-FT systems in such cases. The optimal solutions obtained for the special cases can provide useful bounds on the number of edges required.

The central question studied here, the design of optimal k-FT systems, can be viewed as the problem of optimally assigning connections in a redundant system [10], [17]. This particular connection assignment problem does not include any constraints on the number of connections to any facility. In practice, some "fan-in" and "fan-out" constraints limiting the number of connections to a node could be expected.

An implicit constraint in our graph model is that a faulty facility can only be replaced by a facility of the same type. A less restrictive assumption would permit node x_i of type t_i to replace x_j of type t_j provided x_i is "more powerful" than x_j. Thus we could associate with each facility type t_i a set of facility types that could be replaced by t_i. This would, in general, increase the reconfiguration possibilities and permit "graceful degradation" of the faulty system to be modeled.

Appendix

Proof of Theorem 5

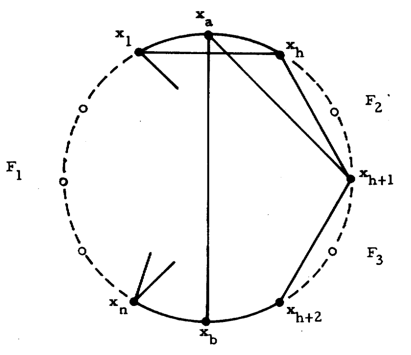
Let $F = \{F_1, F_2, \cdots, F_q\}$ be any k-fault in $C_{n,2p+1}$. As in the proof of Theorem 4, let $F_1, F_2, \cdots, F_q$ correspond to disjoint "holes" along the perimeter of $C_{n,2p+1}$. There are three cases to consider depending on the size of the members of F.

Case 1: $|F_i| \leqq p$ for all $i = 1,2,\cdots,q$. Each hole F_i is spanned by an edge of $C_{n,2p+1}$, implying that $C_{n,2p+1}$ contains C_n, cf., Case 1 in the proof of Theorem 4.

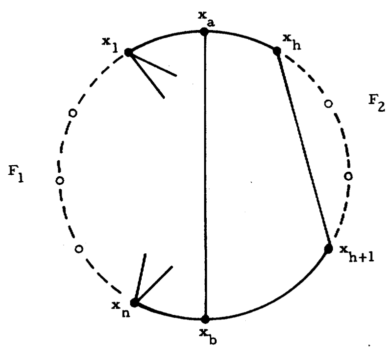
Case 2: $|F_i| > p$ for some i, and $|F_j| \leqq p - 1$ for all $j \neq i$. As in the proof of Theorem 4, let $S = (x_1, x_2, \cdots, x_n)$ be the sequence of nodes obtained by traversing the perimeter of $C^F_{n,2p+1}$ in the clockwise direction beginning at F_i. F_i is not spanned by any nonbisector edge of $C^F_{n,2p+1}$. All the remaining holes are spanned by a nonbisector edge, i.e., for $1 \leqq h \leqq n - 1$, there is an edge in $C^F_{n,2p+1}$ joining x_h and x_{h+1}.

If for $1 \leqq h \leqq n - 2$, x_h is also joined to x_{h+2}, then, as in Case 2 of the proof of Theorem 4, $C^F_{n,2p+1}$ contains L_n and therefore C_n. If, however, some x_h is not joined to x_{h+2}, then the distance between x_h and x_{h+2} as measured along the perimeter is greater than $p + 1$. The number of missing nodes between x_h and x_{h+2} is $\Sigma_{j \neq i} F_j = p$. In addition, x_{h+1} lies between x_h and x_{h+2}; hence the distance from x_h to x_{h+2} measured along the perimeter is exactly $p + 2$. Since all missing nodes not in F_i lie between x_h and x_{h+2}, they are divided into two holes by x_{h+1}. This means that F comprises 3 holes, i.e., $q = 3$, as shown in Fig. 11(a). It remains to show that $C^F_{n,2p+1}$contains a Hamiltonian path in this situation.

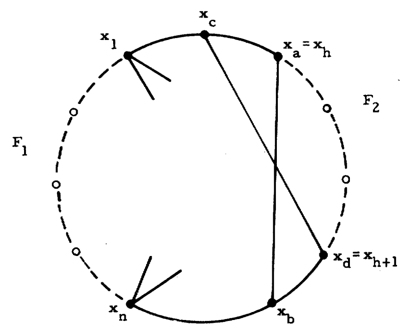
Now $b = \lfloor (n + k + 1)/2 \rfloor = \lfloor n/2 \rfloor + p + 1$. Since $n \geqq 3$ we have $\lfloor n/2 \rfloor \geqq 1$ and $\lfloor n/2 \rfloor \leqq n - 2$, so b must obey the inequality $p + 2 \leqq b \leqq n + p - 1$. The distance (measured clockwise along the perimeter) from x_1 to x_n is $n + p - 1$, and the distance from x_h to x_{h+2} is $p + 2$; therefore $C_{n,2p+1}$



(a)



(b)



(c)

Fig. 11. Proof of Theorem 5.

must contain a bisector $x_a x_b$ as shown in Fig. 11(a) with $1 \leqq a \leqq h$ and $h + 2 \leqq b \leqq n$.

Consider the subsets $N_a = \{x_1, x_2, \cdots, x_{h+1}\}$ and $N_b = \{x_{h+1}, x_{h+2}, \cdots, x_n\}$ of the nodes of $C^F_{n,2p+1}$. $C^F_{n,2p+1}$ contains the subgraphs L_{h+1} and L_{n-h} which involve the nodes N_a and N_b, respectively. By Lemma 2 there is a path P_a from x_a to x_{h+1} that is a Hamiltonian path through all the nodes of N_a. Similarly there is a Hamiltonian path P_b from x_{h+1} to x_b through N_b. Hence the path $P_a P_b$ is a Hamiltonian path from x_a to x_b passing through all n nodes of $C^F_{n,2p+1}$. Since x_a and x_b are adjacent, $C^F_{n,2p+1}$ contains C_n.

Case 3: $F = \{F_1, F_2\}$, $|F_1| = p + 1$ and $|F_2| = p$. The hole due to F_2 is spanned by a nonbisector edge $x_h x_{h+1}$ of $C_{n,2p+1}$ while as in Case 2, there is also a bisector edge $x_a x_b$ as shown in Fig. 11(b) where $1 \leqq a \leqq h$ and $h + 1 \leqq b \leqq n$.

Subcase 3.1: Either $h = 1$ or $h = n - 1$. Suppose $h = 1$, then $x_1 = x_a = x_h$. Now the distance from x_a to x_{h+1} measured along the perimeter is $p + 1$ while the distance from x_a to x_b is at least $p + 2$, therefore $x_b \neq x_{h+1}$. By Lemma 2 there is a Hamiltonian path from x_{h+1} to x_b through the nodes $\{x_{h+1}, x_{h+2}, \cdots, x_n\}$. Since x_1 is adjacent to both x_{h+1} and x_b, we have a Hamiltonian cycle through all the nodes of $C^F_{n,2p+1}$. The argument is identical when $h = n - 1$, in which case $x_{h+1} = x_b = x_n$.

The conditions $h \neq 1$ and $h \neq n - 1$ will be assumed to hold for all the remaining subcases. Let N_1 denote $\{x_1, x_2, \cdots, x_h\}$ and N_2 denote $\{x_{h+1}, x_{h+2}, \cdots, x_n\}$.

Subcase 3.2: $x_a \neq x_h$ and $x_b \neq x_{h+1}$. Then by Lemma 2 there is a Hamiltonian path P_1 from x_a to x_h through N_1, and a Hamiltonian path P_2 from x_b to x_{h+1}, through N_2. P_1, P_2, and the edges $x_a x_b$, $x_h x_{h+1}$ form a Hamiltonian cycle in $C^F_{n,2p+1}$.

Subcase 3.3: $x_a = x_h$. This requires $x_b \neq x_{h+1}$ as noted in Subcase 3.1. Since $h \neq 1$, there must exist a second bisector $x_c x_d$ where $1 \leqq c < a$ and $h + 1 \leqq d < b$. Now $x_c \neq x_h$. If $x_d \neq x_{h+1}$, then this case reduces to Subcase 3.2. Suppose $x_d = x_{h+1}$ as shown in Fig. 11(c). There are paths P_1 and P_2 from x_h to x_c and x_{h+1} to x_b which are Hamiltonian paths for N_1 and N_2, respectively. Combining P_1 and P_2 with the edges $x_a x_b$ and $x_c x_d$ we can construct a Hamiltonian cycle for $C^F_{n,2p+1}$.

Thus in all cases $C^F_{n,2p+1}$ contains C_n, hence $C_{n,2p+1}$ is a $(2p + 1)$-FT realization of C_n. The number of edges in $C_{n,2p+1}$ is $\lfloor (k + 2)(n + k)/2 \rfloor$, so by Theorem 1, $C_{n,2p+1}$ is also optimal. ■

Fig. 7 shows how the 3-fault $F = \{x_1, x_6, x_9\}$ is tolerated by $C_{6,3}$. The subgraph in heavy lines is isomorphic to C_6 and illustrates Subcase 3.2 of the foregoing proof.

REFERENCES

[1] A. Avizienis, "Fault tolerant computing—An overview," *IEEE Computer*, vol. 4, pp. 5–8, Jan./Feb. 1971.
[2] J. P. Hayes, "Modeling faults in digital logic circuits," in *1974 Symp. Rational Fault Analysis* (Lubbock, TX, Aug. 1974), to be published.
[3] W. H. Pierce, *Failure-Tolerant Computer Design.* New York: Academic, 1965.
[4] R. A. Short, "The attainment of reliable digital systems through the use of redundancy—A survey," *IEEE Computer Group News*, pp. 2–17, Mar. 1968.
[5] W. G. Bouricius *et al.*, "Reliability modeling for fault tolerant computers," *IEEE Trans. Comput.*, vol. C-20, pp. 1306–1311, Nov. 1971.
[6] F. Harary, *Graph Theory.* Reading, MA: Addison-Wesley, 1969.
[7] C. Berge, *Graphes et Hypergraphes.* Paris, France: Dunod, 1970.

[8] B. Langefors, *Theoretical Analysis of Information Systems*, 4th ed. Philadelphia, PA: Auerbach, 1973.
[9] C. V. Ramamoorthy, "A structural theory of machine diagnosis," in *1967 Spring Joint Computer Conf. Proc.* Montvale, NJ, pp. 743–756.
[10] F. P. Preparata, G. Metze, and R. Chien, "On the connection assignment problem of diagnosable systems," *IEEE Trans. Electronic Comput.*, vol. EC-14, pp. 848–854, Dec. 1967.
[11] C. G. Bell and A. Newell, *Computer Structures: Readings and Examples*. New York: McGraw-Hill, 1971.
[12] A. T. Berztiss, "A backtrack procedure for isomorphism of directed graphs," *J.ACM*, vol. 20, pp. 365–377, 1973.
[13] K. J. Thurber *et al.*, "A systematic approach to the design of digital bussing structures," in *1972 Fall Joint Computer Conf. Proc.* Montvale, NJ, pp. 719–740.
[14] G. Chartrand, S. F. Kapoor, and D. R. Lick, "n-Hamiltonian graphs," *J. Combinatorial Theory*, vol. 9, pp. 308–312, 1970.
[15] D. Knuth, *The Art of Computer Programming: Fundamental Algorithms*, vol. 1, 2nd ed. Reading, MA: Addison-Wesley, 1973.
[16] R. E. Miller, "A comparison of some theoretical models of parallel computation," *IEEE Trans. Comput.*, vol. C-22, pp. 710–717, Aug. 1973.
[17] R. S. Wilkov, "Analysis and design of reliable computer networks," *IEEE Trans. Communications*, vol. COM-20, pp. 660–678, June 1968.
[18] A. H. Hopkins and T. B. Smith, "The architectural elements of a symmetric fault-tolerant multiprocessor," *IEEE Trans. Comput.*, vol. C-24, pp. 498–505, May 1975.

John P. Hayes (S'67–M'70), for a photograph and biography please see p. 620 of the June 1976 issue of this TRANSACTIONS.

On the Properties of Irredundant Logic Networks

JOHN P. HAYES, MEMBER, IEEE

Abstract—**The constraints imposed by various types of irredundancy on the structure of combinational logic networks are investigated. It is shown that the usual notion of irredundancy, here called a-irredundancy, places bounds on the maximum number of inputs to certain types of network structures. A network is called b-redundant if it contains a cascade of single-input gates that can be reduced to either an inverter or a single line. Let $N_{\bar{a}\bar{b}}(Z)$ denote all realizations of Z that are both a- and b-irredundant. If $N \in N_{\bar{a}\bar{b}}(Z)$, then the number of gates in any fan-out-free subnetwork of N is bounded. It is shown that a solution to some important design optimization problems can be found in $N_{\bar{a}\bar{b}}(Z)$. It is conjectured that $N_{\bar{a}\bar{b}}(Z)$ is finite and some results supporting this conjecture are presented. For example, it is impossible to construct an arbitrarily long cascade of networks that perform the identity transformation without introducing a- or b-redundancy. A more general type of redundancy, c-redundancy, is defined which includes both a- and b-redundancy as special cases. The class of c-irredundant realizations of Z is finite.**

Index Terms—**Combinational networks, irredundant networks, logic circuits, logic design optimization, redundancy, redundancy testing, redundancy types.**

I. Introduction

FROM the beginning, switching theory has been mainly concerned with the study of switching networks that have certain restrictions imposed on their structure or behavior. Of fundamental importance is the constraint that a combinational network be well-formed (wf). The class of wf combinational networks can be defined recursively as follows [1]. A single (AND, OR, NAND, NOR, or NOT) gate is a wf network. Let N_1 and N_2 be disjoint wf networks. The juxtaposition of N_1 and N_2 is wf. A network formed by joining input lines of N_1 is wf. A network formed by connecting an output of N_1 to an input of N_2 is wf.

An additional constraint often imposed on switching networks is that they be irredundant. A network is usually termed redundant if lines or gates can be removed from the network without altering its primary output function. Redundancy may be deliberately employed in order to achieve fault tolerance [2]. In many situations, however, redundancy is unintentional and undesirable, mainly for the following reasons.

1) Redundancy increases network cost.

2) Certain faults in redundant circuits are undetectable. The presence of such faults can significantly increase the difficulty and cost of test generation and fault analysis.

Since well-formedness is a structural constraint, its major implications on network topology are clear. wf networks are acyclic, i.e., they do not contain feedback. In a wf network the outputs of two gates may not be joined together. Thus it is relatively easy to determine if a given network is wf by examining its structure. Irredundancy on the other hand is defined in behavioral terms, and the constraints it imposes on network structure are by no means clear. This paper attempts to identify some of these constraints. The results presented here have some relevance to the problem of determining if a given network

Manuscript received March 1975; revised January 1976. This work was supported by the Office of Naval Research under Contract N00014-67-A-0269-0019 and the Joint Services Electronics Program through the Air Force Office of Scientific Research under Contract F44620-71-C-0067.

The author is with the Department of Electrical Engineering and the Computer Science Program, University of Southern California, Los Angeles, CA 90007.