Reliability and Fault Tolerance Modelling of Multiprocessor Systems

by

Roberto Abraham VALDIVIA Beutelspacher MSc

A thesis presented to **Brunel University** in part fulfillment of the regulations for the degree of **Doctor of Philosophy**.

December 1989

Abstract

Reliability evaluation by analytic modelling constitute an important issue of designing a reliable multiprocessor system. In this thesis, a model for reliability and fault tolerance analysis of the interconnection network is presented, based on graph theory. Reliability and fault tolerance are considered as deterministic and probabilistic measures of connectivity.

Exact techniques for reliability evaluation fail for large multiprocessor systems because of the enormous computational resources required. Therefore, approximation techniques have to be used. Three approaches are proposed, the first by simplifying the symbolic expression of reliability; the other two by applying a hierarchical decomposition to the system. All these methods give results close to those obtained by exact techniques.

Dedication

To my daughters, ***Marissa*** *and* ***Melissa,***
to my wife ***Maria Isabel*** *and*
to my parents, ***Roberto Valdivia P.*** *and*
Marcela B. de Valdivia

Table of Contents

Acknowledgements

First, I am indebted to my supervisor Dr. A. P. Ambler for his invaluable guidance, suggestions and supervision throughout this research project.

The research work reported in this thesis was performed in the Department of Electrical Engineering and Electronics at Brunel University. Thanks to the departmental staff, led by Prof. G. Musgrave for providing the facilities to undertake this project.

Thanks also to my colleagues and staff at Brunel for their helpful suggestions and comments.

The financial support received from the "Consejo Nacional de Ciencia y Tecnología" (National Council for Science and Technology of Mexico) is gratefully acknowledged, as well as the support from the "Instituto de Investigaciones Eléctricas" (Institute for Electrical Research).

Lastly, I would like to thank my wife Maria Isabel who has given me support, encouragement, assistance and two beautiful daughters during this work; thanks to my brother-in-law Miguel Angel for his help and thanks to my family for their support and encouragement.

Chapter 1

Introduction

1.1 THE IMPORTANCE OF RELIABILITY

The reliability of computer systems has been a major concern since the introduction of the first electronic digital computers which used relays, vacuum tubes and another relatively unreliable components. With the second generation of computers, semiconductor components with much greater reliability were introduced. Nevertheless, today there is a growing interest in reliability, because of the increased advances and complexity of microelectronics and computer systems, together with the increased dependence on such systems, thus demanding for safer, more reliable and more available systems. The importance of human safety, mission success, equipment protection and data integrity, together with recent trends like harsher environments, novice users, increasing repair and maintenance costs and the development of larger systems are some of the reasons for the requirement to improve reliability in computer systems.

1.2 RELIABLE SYSTEM DESIGN

In addition to improvements in component reliability and in test methods to avoid the occurrence of failures; redundancy at various levels of system organisation has to be used to increase the probability of correct operation, providing for tolerance to failures. Fault avoidance and fault tolerance are the two major design approaches to increase reliability, that supported by system evaluation constitute the basic reliable system design methodology, as illustrated in Figure 1.1. Analytic modelling and experimental simulation techniques used for the assessment of the reliability requirements constitute a very important issue of designing a reliable system.

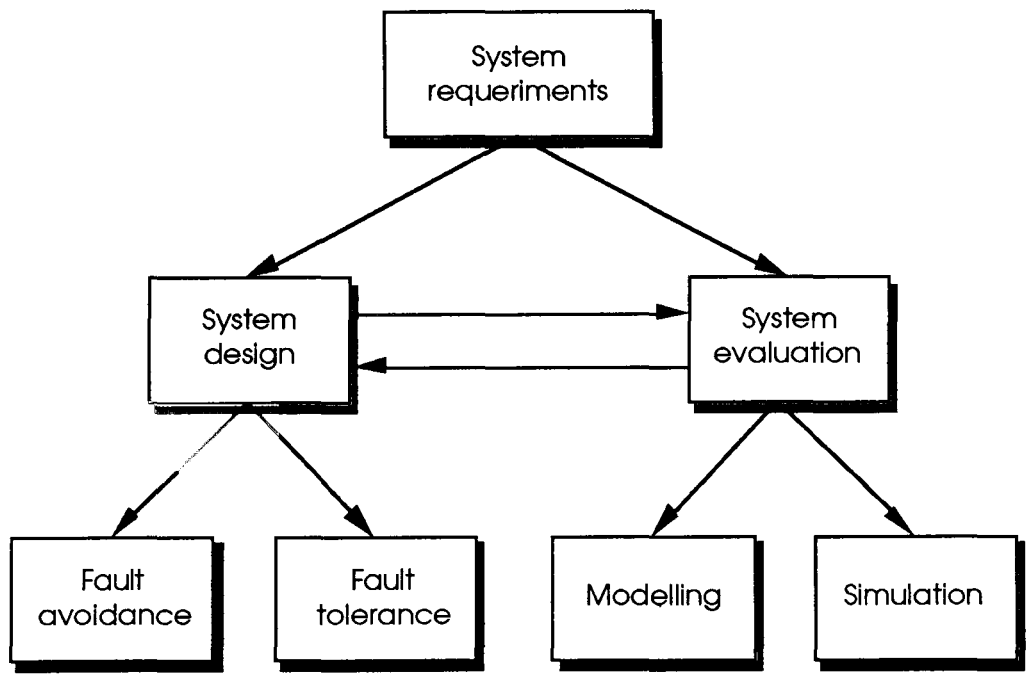


FIGURE 1.1
Reliable system design methodology

1.3 RELIABILITY IN MULTIPROCESSOR SYSTEMS

The rapid expansion of multiprocessor or multicomputer systems has been possible by the continuous decline of hardware costs, the introduction of microprocessors and the development of distributed and parallel systems. Design of computing systems incorporating more processing elements has resulted in a two-sided relationship involving reliability. On one hand, it opened the way to new possibilities of obtaining high reliability and fault tolerance by the use of the inherent redundancy without prohibitive additional costs. On the other hand, as the number of elements increases, the probability of failure existing somewhere in the system at any time also increases.

1.4 PURPOSE OF THIS WORK

The purpose of this work is the study and implementation of models for reliability and fault tolerance analysis of multiprocessor systems. The attention is basically given to the intercommunication structure, i.e. the interconnection network, so models can be based mainly in graph theory. Reliability and fault tolerance are considered as deterministic or probabilistic measures of connectivity, i.e. the successful communication among the nodes (computers) throughout the network in spite of faults in the communication paths (node and/or link failures) for several rooted and unrooted connectivity problems.

The trend towards constructing multiprocessor systems with large number of processors has meant that exact reliability modelling techniques cannot be applied without prohibitive computational overheads. Therefore,

it is proposed to employ approximate techniques for reliability modelling of large multiprocessor systems based in a hierarchical decomposition of the system.

1.5 OUTLINE OF THE THESIS

Chapter 2 provides a general overview and introduces some aspects of fault tolerance and reliability in computer systems, considering basic concepts and definitions of fault tolerance and fault avoidance techniques, characterisation of faults, redundancy and system service where the main reliability measures are introduced. The application areas for fault tolerant systems are described, the need for reliability assessment is highlighted and a general design methodology is suggested for implementing fault tolerance and consequently high reliability in computer systems.

Chapter 3 describes the characteristics of multiprocessor systems followed by the principal considerations and methodology to implement fault tolerance and reliability in such systems. A theoretical model based in graph theory is proposed to study the reliability in the intercommunication network, considering the deterministic or structural as well as the probabilistic, stationary and dynamic, aspects of the network.

Chapter 4 is devoted to the implementation of a deterministic model and a probabilistic model for reliability analysis of multiprocessor systems. An evaluation of some network architectures is also presented.

Chapter 5 presents a description of the hierarchical clustering method and the subsequent hierarchical reliability evaluation of large multiprocessor systems as well as the results obtained when applying this method to some multiprocessor configurations.

Chapter 6 presents a summary, conclusions and recommendations for future work.

Appendix A describes some basic concepts of graph theory related to the graph model for reliability.

Appendix B presents computer implementation details of the reliability model.

Chapter 2

Aspects of Fault Tolerance and Reliability

2.1 INTRODUCTION

In this chapter are presented some aspects of fault tolerance and reliability in computer systems, considering basic concepts and definitions of fault tolerance and fault avoidance techniques, characterisation of faults, redundancy and system service where the main reliability measures are introduced. The application areas for fault tolerant systems are described, the need for reliability assessment is highlighted and a general design methodology is suggested for implementing fault tolerance and consequently high reliability in computer systems.

2.2 BASIC ASPECTS AND TERMINOLOGY

2.2.1 FAULT AVOIDANCE AND FAULT TOLERANCE

There are two major approaches for attempting to improve or maintain normal performance and consequently reliability of a system. These two approaches can be combined and are applicable to all parts of the system.

The first approach is called *fault avoidance* in which the reliability of the system is assured by preventing the cause of unreliability, i.e. of faults. This can be achieved by techniques such as design review, quality control on components and system testing.

The second approach is by *fault tolerance,* which is defined as: "the ability of the system to continue to perform its specified functions regardless of the presence of faults" [AVI 78].

Fault tolerance can be achieved in one of two ways :

(a) *Static*: through the masking or hiding of the effects of faults (*fault masking*), or

(b) *Dynamic*: by identification of sources of failure, followed by undertaking actions to appropriately compensate for the effects of identified failures.

2.2.2 CHARACTERISATION OF FAULTS

A *fault* is defined as any erroneous state of the system. In a computer system there are two types of faults: hardware and software faults. Hardware faults are caused by physical factors resulting from component failures (wear-out or manufacturing defects), external disturbances, and design or implementation mistakes. Software faults result from design or

implementation mistakes. An *error* is the manifestation of a fault in the system. A *failure* or *malfunction* is the effect of an error in the system service or behaviour as it is perceived by the user. An error will lead to the failure of a system unless tolerance to such fault has been provided [JOH 84]. The general effects of faults in a system are illustrated in Figure 2.1.

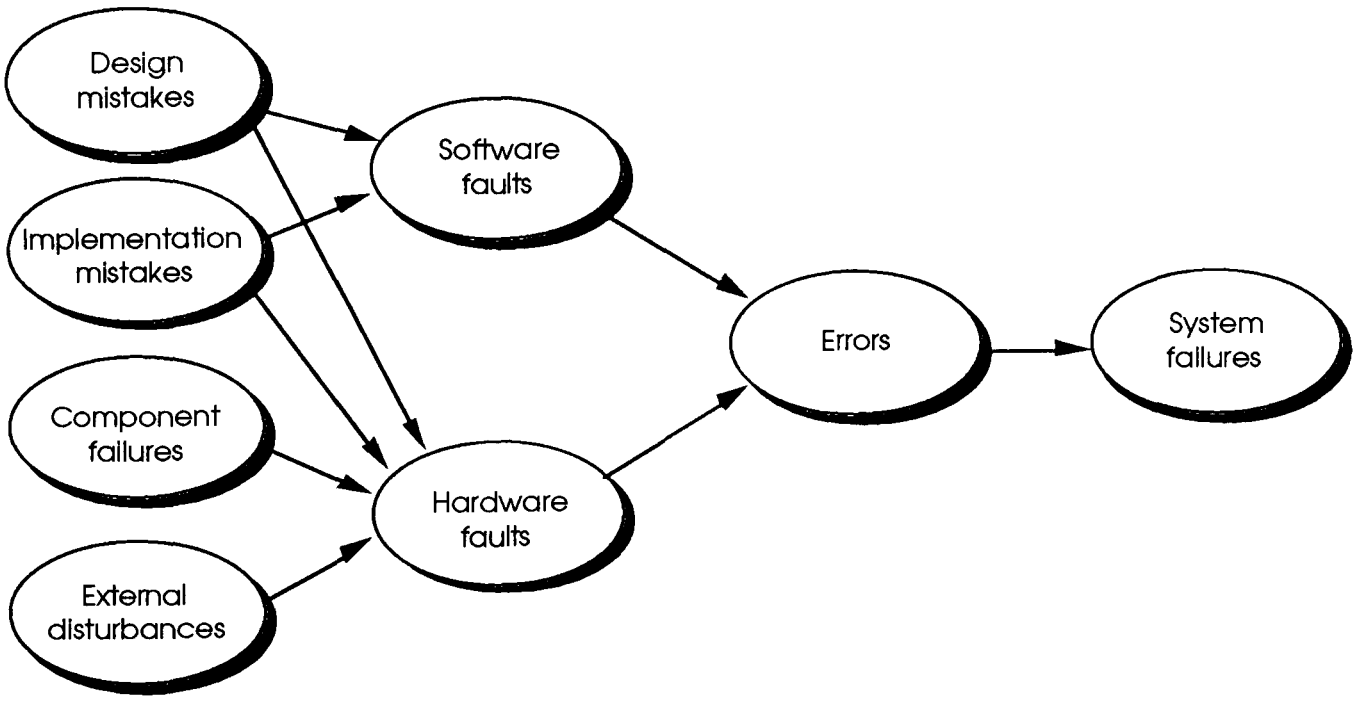


FIGURE 2.1
Cause and effect relationship of faults

Faults may be further characterised by other properties besides their type and cause :

- *value* : determinate (such as stuck-at models) or indeterminate;
- *duration* : permanent, intermittent, transient or latent;
- *level* : fault in a component, module, subsystem, etc.;
- *extent* : local or global.

Figure 2.2 shows the barriers constructed against faults by fault avoidance, static and dynamic fault tolerance.

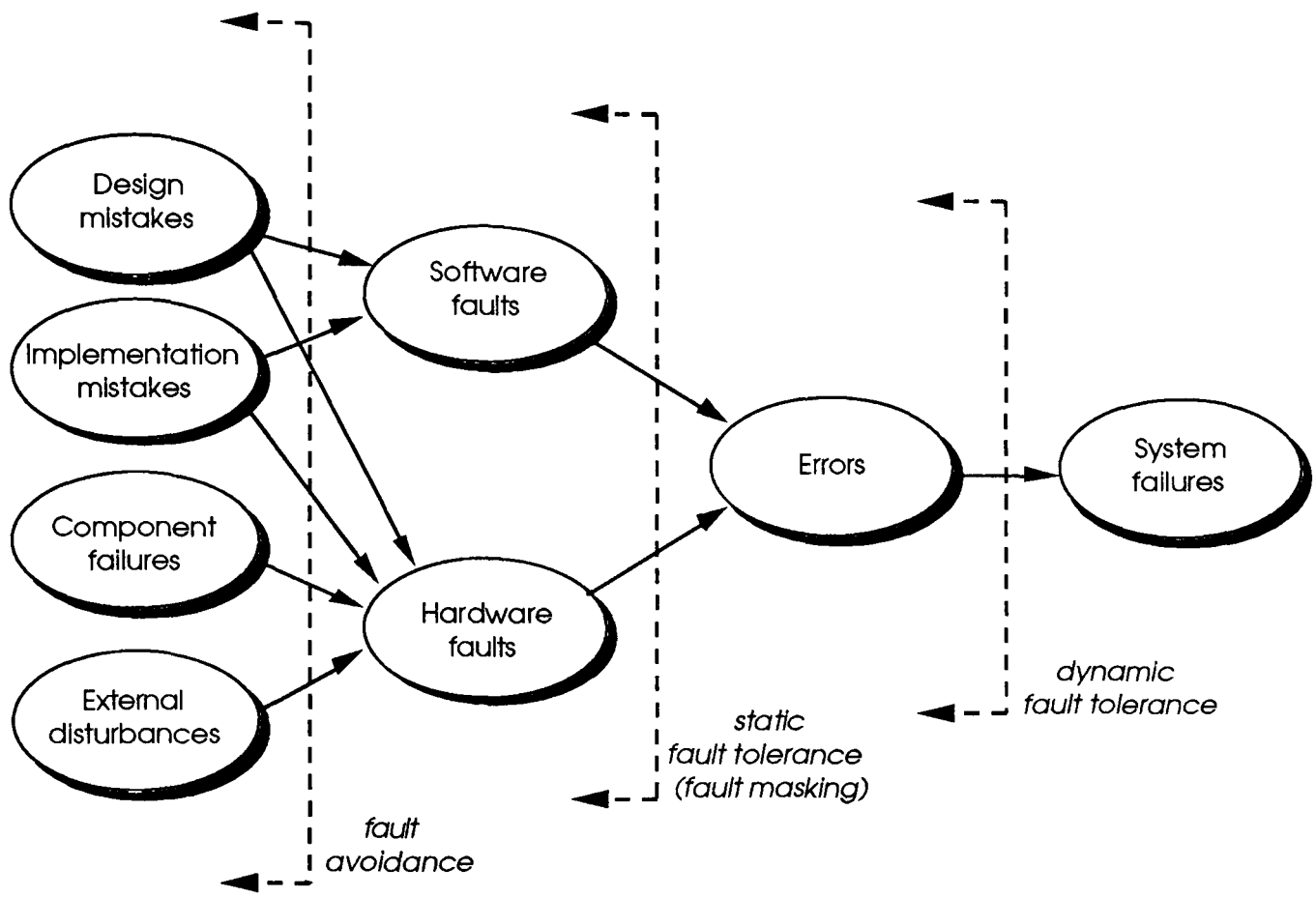


FIGURE 2.2
Barriers against faults

2.2.3 *REDUNDANCY*

Redundancy is the key issue in all fault tolerant systems, it consists in the addition of resources beyond what is needed for normal system operation. Redundancy may take several forms [JOH 84]:

(a) *information redundancy*, e.g. error detecting codes;

(b) *hardware redundancy*, i.e. physical replication of hardware;

(c) *software redundancy*, replication of software or programs to perform validity checks, self-tests, etc.;

(d) *time redundancy*, uses additional time mainly to distinguish between permanent and intermittent failures.

In fault masking systems, generally hardware redundancy is employed in the form of *replication* and *voting* (*n*-modular redundancy), where multiple copies of an entity are utilised with outputs decided by majority vote. A common method is triple modular redundancy or TMR which is illustrated in Figure 2.3 with an ideal voter.

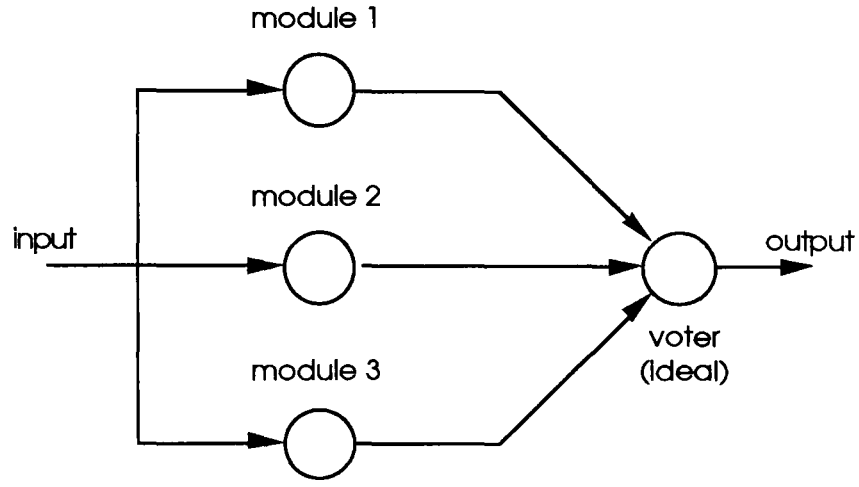


FIGURE 2.3
TMR with ideal voter

In contrast to masking failures which requires a large amount of resources, by using the second (dynamic) approach of fault tolerance, the amount of adittional resources can be minimised. This approach is formally categorised into [KUH 86]:

(1) *Fault detection*: the ability of the system to recognise that a fault has occurred;
(2) *Fault location (diagnosis)*: the process of determining the location of a fault or faults in the system;
(3) *Fault containment*: the process of isolating a fault and preventing its effects from propagating throughout the system;

(4) *System reconfiguration* or *repair*: the logical or physical removal of the failed component, along with rearrangement of the remaining non-faulty elements to compensate for the loss of the failed component.

(5) *System recovery*: the restoring of data and computations to a consistent operational state. This may involve rolling back computations to a pre-failure state and then restoring them.

2.2.4 *SYSTEM SERVICE*

The life of a system is perceived by the user as an alternation between two states of the delivered service with respect to the specified service [AVI 86].

- *proper service* where the service is delivered as specified;
- *improper service* where the delivered service is different from the specified.

The events which constitute the transitions between these two states are the failure and the restoration of service or repair. Quantifying the alternation between delivery of proper and improper service leads to the two main measures of system reliability.

- *reliability*: a measure of the continuous delivery of proper service from a reference initial instant.
- *availability*: a measure of the delivery of proper service with respect to the alternation of delivery of proper and improper service.

Reliability and availability are formally described in chapter 3.

2.3 APPLICATION AREAS FOR FAULT TOLERANT SYSTEMS

The application area determines the requirements placed upon a system. To employ fault tolerance in a computer system involves trading off the cost of failure against the cost of implementation. Based in this criteria there have been defined five primary application areas [REN 80] (ordered by the most to the less stringent fault tolerance requirements and cost).

(a) *Critical applications* : systems on which failure can place human lives in danger. They require high reliability and short reconfiguration time, such as real time control systems. Examples are: passenger transport, patient monitoring, control of nuclear power plants, etc.

(b) *Long life control systems*: systems in environments that do not allow access for manual maintenance such as spacecrafts, satellites, underwater stations, etc..

(c) *High availability general purpose applications*: the main characteristic of these systems is that they can allow frequent outages as long as the duration of each outage is small. Examples of these systems are large resource sharing systems like telephone switching, book-keeping systems, etc.

(d) *High performance computing*: systems where expected performance cannot be achieved without the use of fault tolerance.

(e) *Maintenance postponement* is required when maintenance is very costly or difficult to perform, such as remote processing stations. The main goals are to postpone maintenance until convenient times and still have a system that can perform at least a subset of its service.

In addition to the above areas, fault tolerance offers significant psychological support for human users who depend on or interact with a computer system.

2.4 RELIABILITY EVALUATION

The choice of fault tolerant functions and redundancy techniques needs to be supported by a quantitative or qualitative assessment whether the system possesses the expected reliability. There are two approaches to reliability evaluation [AVI 78]:

(a) *Analytic approach,* in which fault tolerant and reliability measures are obtained from a mathematical or graph model of the system.
(b) *Experimental approach,* in which faults are inserted either into a simulated model of the system or into a prototype, and fault tolerance and reliability measures are estimated from statistical data.

A variety of models have been created for analytical studies of fault tolerance and reliability, that can be broadly divided into two classes :

(a) *Deterministic models.* For the investigation of problems to describe the architecture, connectivity, diagnosability, robustness, reconfigurability and other aspects related with fault tolerance, reliability and performance.
(b) *Probabilistic models* allowing the computation of reliability and performance parameters such as the probability of success, reliability, availability, MTTF, MTBF, survivability, etc.

2.5 FAULT TOLERANCE AND RELIABILITY DESIGN ISSUES

Fault tolerance can be introduced into the system architecture through a systematic sequence of design activities [AVI 78], [DEP 77]. A general methodology can be summarised as follows :

(1) Specification of the computational task and description of system requirements (I/O interfaces, etc.).

(2) Determination of the basic system architecture.

(3) Specification of the reliability goals according with the application area.

   (a) Identification of classes of faults to be tolerated: implementation errors, component failures or external disturbances.

   (b) Quantitative reliability requirements

   (c) Postulation of the methods for evaluation.

(4) Fault detection mechanisms: initial testing, concurrent detection (on-line) or scheduled detection (off-line), as well as redundant testing.

(5) System reconfiguration and recovery algorithms: manually controlled or automatic; full recovery, degraded recovery (graceful degradation or soft fail operation) or safe shutdown (fail-safe operation). A special case of recovery results from fault masking.

(6) Evaluation of the fault tolerance and reliability of the design by means of analytic modelling, experimental simulation or both. Physical, structural and reliability parameters are used in generating the reliability prediction.

(7) Design refinement. The goal is to balance the protection provided to each subsystem in such a way that reliability goals are obtained without a single dominating contributor of unreliability and at the lowest cost of additional hardware and software.

Chapter 3

Fault Tolerance and Reliability in Multiprocessor Systems

3.1 INTRODUCTION

A key issue for successful operation of a multiprocessor system is the exchange of information between the processing nodes. Therefore, one of the critical problems in designing multiprocessor systems is to provide an appropriate, highly reliable and fault tolerant communication subsystem, so that all the processing nodes are able to communicate at all times.

In this chapter are described the main characteristics of multiprocessor systems, followed by the considerations and methodology to implement fault tolerance and reliability in such systems. A theoretical model based in graph theory is proposed to study the reliability in the intercommunication network, considering the deterministic or structural as well as the probabilistic, stationary and dynamic, aspects of the network.

3.2 PROPERTIES OF MULTIPROCESSOR SYSTEMS

The term multiprocessor systems is used here to represent systems which are known with different names, such as: computer networks, multicomputers, distributed processing systems, parallel processors, etc. Multiprocessor systems extend from geographically distributed networks up to VLSI systems which interconnect a large number of simple processing cells in a single chip.

Multiprocessor architectures can be categorised by their degree of integration and processor granularity [PRA 86] as it is shown in Table 3.1.

TABLE 3.1
Network structures

Degree of integration	Processor granularity	Network examples
LOW	LARGE	Long-haul networks
MEDIUM	MEDIUM	Local area networks
MEDIUM	MEDIUM	Multiprocessor systems
HIGH	SMALL	VLSI based systems

Despite the different names, degree of integration and granularity, multiprocessor systems have the following basic properties : [KUH 86]

(a) *Autonomy*: A number of autonomous, cooperating processing elements (PEs) interconnected between them. At the system level, these PEs and their interconnection links are viewed as the basic components of the system. Each PE has its own local memory and there is no shared memory between PEs. The interconnection schemes

allow high bandwidth communication between the PEs generally through message passing and can be classified into three categories:

- Link oriented
- Bus oriented
- Connection network based.

(b) *Modularity*: A high degree of distribution of control or operating system functions among the PEs (resources distribution).

(c) *Parallelism*: Highly parallel computations, on the classes of SIMD and/or MIMD.

These properties make the system inherently redundant, thus allowing the implementation of fault tolerance capabilities in multiprocessor systems, minimising the need for additional redundancy.

3.3 METHODOLOGY AND CONSIDERATIONS FOR FAULT TOLERANCE AND RELIABILITY

Most of the same design issues described in section 2.5 apply also to multiprocessor systems, but in order to extend this methodology specifically for such systems, the following considerations must be taken in account : [REN 80], [KUH 86]

3.3.1 GENERAL

(a) The design methodology can be applied *locally* (within each processor) and/or *globally* (across the collection of processors and their interconnections).

(b) *Redundant partitioning*. Whole processor partitioning or sub-modules partitioning. In general, for multiprocessor systems, the appropriate level to consider is at the processor level and communication paths in the interconnection structure.

(c) Protection of *hard core* items: Clocks, common control, power supplies, recovery mechanisms, etc.

3.3.2 *REPLICATION AND MASKING*

(d) *Dynamic (selective) redundancy*. In contrasting with traditional static redundancy, selective redundancy is implemented according to the needs and requirements of a specific application and can be adjusted to protect critical computations with higher levels of redundancy compared with less important computations.

3.3.3 *FAULT TOLERANCE THROUGH DIAGNOSIS, REPAIR AND RECOVERY*

(e) *Fault detection*. At processor level can be distinguished in two ways: *external* (generally neighbouring processors) and *internal* detection.

(f) *Fault diagnosis*. Traditional system level diagnosis can be employed, but extended to consider diagnosis of failures in interconnection paths.

(g) *Reconfiguration* and *recovery*. Preferable *logical* to physical hardware reconfiguration due to the non-scarce redundancy in PEs, and the cost and reliability involved in hardware reconfiguration to switch-in spare modules, redirect communication paths, etc. Two important situations can be distinguished related with reconfiguration: configurations with spare nodes in which there is no degraded performance and *graceful degradation*.

(h) Effectiveness of fault detection and recovery: *Coverage.*

3.3.4 *COMMUNICATION FACILITIES*

(i) *Intercommunication structure* and redundancy: If several processors are required to work cooperatively on a task, a frequent exchange of data among them is expected. The amount of data, the frequency with which they are transmitted, the speed of their transmission and the route that they take are all significant in affecting the intercommunication and its reliability.

The key structural consideration in the design of fault tolerant and high performance multiprocessor systems is the system interconnection. Ideally if one processor wants to communicate with another, then it should do it over a channel that directly connects the two. Such a system would be prohibitively expensive. A channel between every pair of processors would require $O(n^2)$ channels for n processors. So it is necessary to trade cost for speed and reliability. The compromise that is made involves routing data from one processor to another via intermediate processors so creating communication paths. A redundant connection that is made to increase reliability, must allow for fault tolerance so that any node can be reached by a different path if one path should fail (*robustness* and *reconfigurability*).

Broadly speaking, a viable interconection strategy must have a small number of channels and easy routing rules, should provide for fault tolerance, re-routing and gracefully recover in case of failures.

3.3.5 *OTHER CONSIDERATIONS*

(j) *Type* and importance of modules, capability, I/O, peripherals connected, etc. The functions that depend on the connected hardware, in case of reconfiguration, can only be delegated to predeterminate modules of the same type.

(k) *Performance*. The structure of the system also affects other factors, such as interprocessor distance, delays, message routing, expansion capability, etc. In degradable systems there is also a degradation in performance (mode of operation or service rate), which is of considerable importance.

3.4 RELIABILITY MODELLING

The operation of a multiprocessor system is a function of the success of many factors; our goals in reliability modelling or assessment are to obtain a measure of a system utility which contributes to its overall performance.

For this work we have concentrated basically on reliability from the point of view of the intercommunication structure of the system, i.e. the interconnection network. Communication network reliability is defined as "the ability of a network to carry out a desired operation" [COL 87]. Necessary network operations have been identified so as to continue to afford communication routes between some target nodes when other nodes or link fail.

The measures of network reliability fall into two classes:

(a) *Deterministic* : depend only on the structure of the network, that is, on the number of nodes and links and the way they are connected.

(b) *Probabilistic* : depend not only on the structure but also on the probabilities of failure of nodes and links.

3.4.1 GRAPH MODEL

An important approach to fault tolerant design and reliability modelling is the utilisation of models based in graph theory [HAY 76]. Graph models have been utilised within the field of fault tolerance for the design of algorithms for fault detection, diagnosis [PRE 67], [MEY 85], [MAE 86], reconfiguration [MAE 86], recovery [YAN 86] and replication [CHE 85] among others.

The basic concepts of graph theory related to the reliability model can be found in Appendix A.

Graph representation

A multiprocessor system can be viewed as a directed or undirected graph $G = (N, E)$ in which the set of nodes or vertices N represents the set of n processors, $N = \{x_1, x_2, ..., x_n\}$ and the set of links or edges E represents the unidirectional or bidirectional interconnection channels between the PEs, $E = \{e_1, e_2, ...\}$; an example of an undirected graph is shown in Figure 3.1 and a directed graph in Figure 3.2.

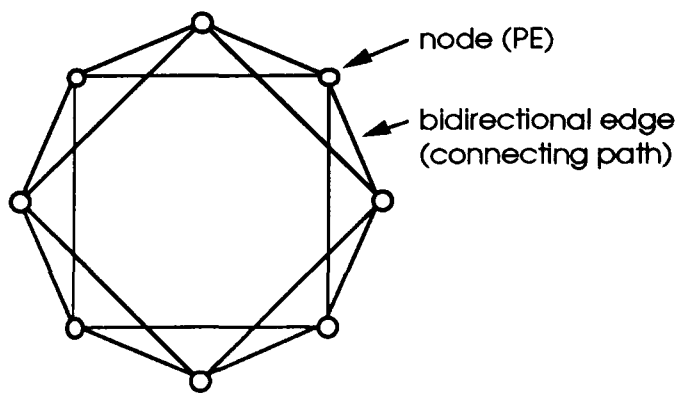


FIGURE 3.1
Undirected Graph

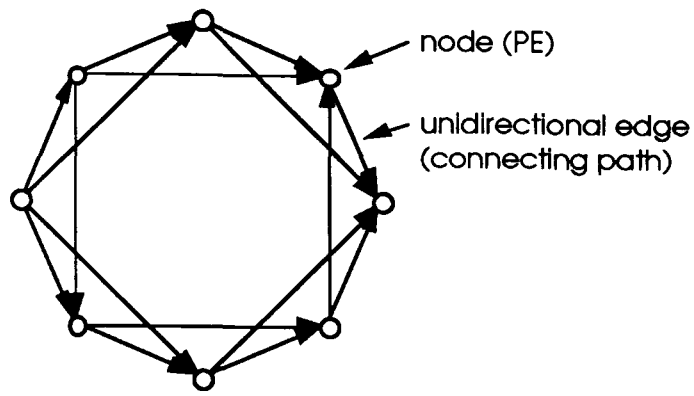


FIGURE 3.2
Directed Graph

In a graph model, the representation of faults in nodes and faults in edges is shown in Figures 3.3a and 3.3b respectively. A node or edge failure has the effect of modifying the graph topology creating a subgraph (G_s) of the graph G when faulty nodes and/or edges are removed from the system graph; it is assumed that removing a node includes removing all its incident edges.

Successive failures can, eventually, result in a disconnection of the system, and therefore prevent some processors from communicating to some other processors.

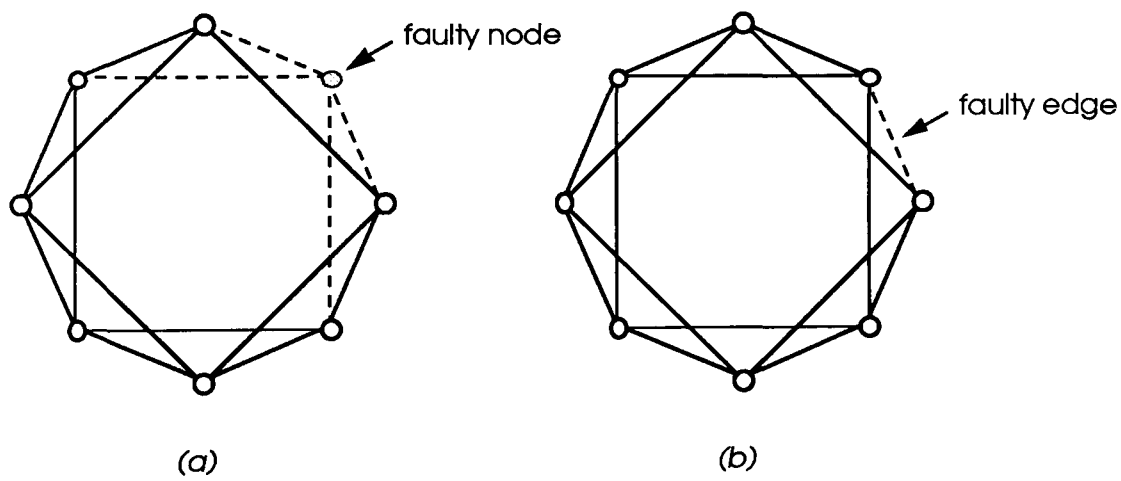


FIGURE 3.3
Representation of faults: (a) fault in a node; (b) fault in an edge.

Assumptions

The following is generally assumed :

(a) Information is directly relating to the topology.
(b) Elements (nodes and edges) have two states: operational and failed.
(c) If the system cannot maintain a specified level of service then is failed.
(d) There is no correlation between the failure of elements (statistically independent failures).
(e) A situation where the graph topology is disconnected is equivalent to a state of total system failure.

Based on these assumptions the reliability goal is then to determine the effect of the topology on the operational states of the network represented as a deterministic or probabilistic graph.

3.4.2 RELIABILITY PROBLEMS

In a graph model of the interconection network it is assumed that any two nodes can communicate if they are both operative and if there is a path of operative nodes and edges between them. Reliability calculation is based not only on the operation of a path but also on the total number of communications of such paths. Based in this criterion, reliability is a measure of *connectivity*.

Reliability problems in a probabilistic communication network are identified and classified in [SAT 82] and [COL 87] as either *unrooted* or *rooted problems*. Rooted problems represent tree connectivity problems which are useful, for example, in studying the reliability of successful

broadcasting of information originated by a central controller (source node) to a set of target nodes in a network. For our model it is proposed to extend this classification to be used also to characterise the deterministic reliability model. For a graph *G*, the reliability problems considered for deterministic and probabilistic models include :

Unrooted problems

(a) *Two-terminal reliability (TT)* : a specified node pair in *G* can communicate each other. TT connectivity is useful because many applications of multiprocessing require connection between two nodes over a period of time, for example in remote interactive computing.

(b) *Overall reliability (AT)* : all node pairs in *G* can communicate.

(c) *K-terminal reliability (KT)* : among a set K of specified nodes in *G*, all node pairs can communicate. It is useful for example in distributed computing.

Rooted problems

(d) *Source to terminal reliability (ST)* : a specified node (*S*) in *G* can communicate to another specified node (*T*).

(e) *Source to all terminal reliability (SAT)* : a specified node (*S*) in *G* can communicate to all other nodes.

(f) *Source to K-terminal reliability (SKT)* : a specified node (*S*) in *G* can communicate to a set *K* of specified nodes.

(g) *K-source to K-terminal reliability (KSKT)* : a set (*Ks*) of specified source nodes in *G* can communicate to a set (*Kt*) of specified terminal nodes.

For undirected graphs, TT and ST can be viewed as equivalent problems since each link can communicate in both ways. Likewise AT and SAT are equivalent, and KT with SKT are equivalent as well. For a graph *G*

with n nodes, TT and AT are special cases of KT with $K=2$ and $K=n$ respectively.

Another reliability problem that has been considered for the probabilistic model due to its importance as the general model of redundant systems is:

(h) *K-out-of-N system reliability (KON)* : probability that K out of N components in G must work for system success.

The general mechanism to define a reliability problem is as follows [COL 87] :

For any graph $G = (N, E)$ it is defined a *state* of G to be a subset S of G; this is interpreted to mean that all elements (edges and nodes) in S are operational and all elements in $G - S$ are failed.

The universe of possible states is the power set $U(G) = 2^{ne}$, where ne is the total number of elements ($ne = n + e$). A network operation is specified by defining the set $OP(G)$ subset of 2^{ne} ; here $OP(G)$ is the set of states considered to be operational. Equivalently, network operation can be defined in terms of $FA(G) = U(G) - OP(G)$ the set of failed states.

3.4.3 DETERMINISTIC MODEL

The graph model is utilised for the deterministic reliability model to analyse the characteristics, in terms of reliability, fault tolerance and structural performance, of the interconnection structure. The most important deterministic measures, related to reliability, taken from the graph theory domain are:

Degree of node. Is the number of neighbours nodes, or equivalently the number of edges incident on a node, it represents the number of communication ports. The largest degree of all nodes is denoted by d_{max} and the smallest by d_{min}, if $d_{max} = d_{min}$ then the graph is regular of degree d.

Distance. Distance or length between two nodes $l(i, j)$ is the number of edges in the shortest path between node i and node j. **Average distance** (l_{av}) is the internode distance averaged over all the node pairs; it is a measure of the average delay. **Diameter** (l_{max}) is the maximum internode distance.

Size (e). Is the total number of edges. **Denseness** (s). Is a measure of how well connected the graph is. Formally, $s = \frac{e}{n}$. Usually, $s = \log_2 n$ is considered a fairly dense graph, $s = O(1)$ is sparse while $s = O(n)$ is a very dense graph.

Node connectivity (K_n). Is the minimum number of nodes which when removed will disconnect the graph. **Edge connectivity** (K_e). Is the minimum number of edges whose removal will disconnect the graph.

The **degree of fault tolerance** (K) has been defined as the maximum number of elements (nodes and/or edges) which can become faulty without disconnecting the graph, i.e. $K = K_n - 1$.

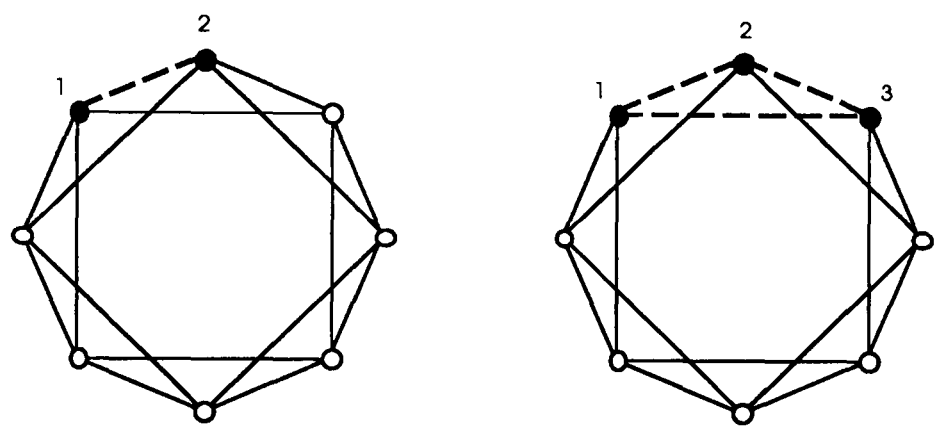
These parameters can also can be used for:

(a) Analysing the *diagnosability* of different configurations, which is a direct function of connectivity.

(b) Analysing the suitability of various configurations for a desired application from the point of view of fault tolerance, diagnosability, *reconfigurability* (number of possible configuration states for a given

application without degradation) and structural performance (such as distance) in order to determine an appropriate (optimal or near optimal) configuration in terms of minimum hardware investment, i.e. minimum size and number of nodes.

(c) Selective redundancy can be incorporated in the model, allowing a critical task to be replicated for two PEs (mutual monitoring) (Fig. 3.4a) or three PEs (2-out-of-3 decision) (Fig. 3.4b), if the configuration allows direct connection between the processors.



(a) (b)

FIGURE 3.4
Replication of modules: (a) two nodes (1 and 2); (b) three nodes (1, 2 and 3).

(d) In a gracefully degrading system it is possible to reconfigure the system (reassign or reduce the computational tasks from the faulty processor(s) to the remaining operational ones) for different degraded configurations down to a minimum configuration allowable or until the graph becomes disconnected, being also possible to analyse the parameters mentioned (connectivity, diameter, distance, etc.) for each degraded configuration in order to obtain a measure of *survivability* (how gracefully the system degrades).

3.4.4 *PROBABILISTIC MODEL*

The probabilistic model is concerned with the probability that the interconnection network is able to perform a desired operation in an environment of random component failures.

The reliability of a system can be derived in terms of the individual reliabilities of the components used to build it. The various reliability modelling techniques that have been developed tend to fall into one of two classes [STI 86]:

(a) *Combinatorial models*: attempt to categorise the set of operational states (or conversely the number of unoperational states) of a system in terms of the functional states of its components in such a way that the probabilities of each of these states can be determined by combinatorial means.

(b) *Continuous-time discrete-state Markov models*: concentrate on the transition rates between the possible states of the system (state probability) and then use this information to determine the probabilities that the system is in each of these states at any given time. Markov models are applicable when the system states are dependent on parameters such as reconfiguration, degradation, repair, coverage, etc.

Markov models have been widely used in the modelling of reliability and behaviour of simple multiprocessor systems since they have the characteristics above explained. Several models have been developed for specific applications. Some of them present a model which also includes performance analysis [BEA 78] (performance & reliability = *performability*).

Others also have considered parameters such as intermittent and transient faults [MAL 81].

A considerable effort has been expended for several researchers to develop a complete model based on Markov methods which deal with the problem of reliability prediction of complex fault tolerant computer systems, mainly for critical applications where ultrahigh reliability is required (e.g. in the order of $1\text{-}10^{-9}$). The most representative Markov models are reviewed and criticized in [GEI 83] : ARIES, SURF, CAST, and CARE-III, where is concluded that all these models suffer from multiple limitations, and therefore they propose a new model: HARP.

The main disadvantage of all Markov methods is that they require to enumerate all possible states of the system, which is impractical for systems of medium to large size. For each probabilistic event considered, the number of states is directly proportional to the branching factor, existence of cross links and the depth of the network. Also, when availability is needed the state diagram has to be expanded to account for the non-homogeneity when the failure and repair rates are different for the different components [MAK 83].

On the other hand, an equivalent analysis of interconnection network reliability is obtained by combinatorial techniques as demonstrated in [MAK 83]. By using a combinatorial Boolean algebraic approach it is possible to achieve efficiency and functionality of the model, as it is described in the following subsection.

3.4.4.1 Combinatorial approach

Several combinatorial methods for system reliability are given in [HWA 81]; these methods are classified as :

(a) State enumeration

(b) Reduction to series-parallel networks

(c) Path enumeration

(d) Cutset enumeration

(e) Others

Type (a) methods present the same disadvantages as Markov models because of the large number of states to be enumerated. Type (b) methods are not applicable when both nodes and links are unreliable and since most of the networks cannot be reduced to series-parallel subnetworks. In methods of type (c), the reliability expression is obtained by finding the set of possible paths for the reliability problem to solve, and then applying Boolean algebra and probability theory to modify the set of paths to an equivalent set of mutually exclusive (disjoint) paths. Cutset enumeration methods (type (d)) are equivalent to path enumeration methods to obtain the unreliability instead of the reliability. The disadvantage is that it is more difficult to implement algorithms for cutsets than for paths.

For the reliability analysis, it is desirable to use a symbolic expression because it presents several advantages [HAR 86] :

(a) when the network has a fixed topology the reliability of its elements can change with time, reliability can be calculated by simply substituting the values of the element reliabilities in the symbolic expression and the effects of their changes can be estimated.

(b) In some applications it is desired to improve reliability of a network under a given cost constraint. The symbolic expression can be used to identify the critical elements to optimise the reliability.

Probabilistic graph

For the probabilistic model, in addition to the graph model of a multiprocessor system, a probabilistic graph having a probability of operation associated with each node and edge, is also required.

Assumptions

First, it is assumed that the system is coherent, i.e. :

(a) when the system has failed, no failure will restore the system to a successful state,
(b) when the system is operating successfully, no repair will cause the system to fail,
(c) failure of components causes the system to fail,
(d) when all components are working the system is successful.

It is also generally assumed that the probability of failures of the elements are statistically independent, i.e. there is no correlation between failures of different nodes and links.

3.4.4.2 Stationary reliability

In the static or stationary reliability analysis, the processing nodes and the communication links are associated with probabilities of being operational, i.e. reliabilities. It is assumed that these reliabilities are constant during the time interval in which the system is being analysed.

The reliability of the i^{th} component (node and/or edge) is given by :

$$p_i = \mathbf{Pr}\{\ i^{th} \text{ component is working }\} \quad \ldots (3.1)$$

and the unreliability is given by :

$$q_i = 1 - p_i \quad \ldots (3.2)$$

3.4.4.3 Dynamic reliability

In practice the parameters that are associated with reliability evaluation are described by probability distributions [BIL 83]. The times-to-failure describe the probability that a given component fail within or survive beyond a certain specified time. To study dynamic or time dependent analysis of the various connectivity problems, there are considered two different operating environments, namely, *closed* or non repairable, i.e. no repair of failed elements (nodes and links) is possible during the time interval of interest, and *repairable* when the failed elements are repaired and made operational. Dynamic reliability analysis has several advantages [MAK 83], such as:

(a) the provision for incorporation of different probability distributions for failure and recovery times,
(b) the computation of task and mission related measures such as MTTF and MTBF (as explained below),
(c) system design is based on the dynamic behaviour of the individual network elements, where a single probability of success p_i is inadequate.

The most important dynamic reliability measures for the design and evaluation of the intercommunication network are the following [BIL 83], [RAG 86]:

For **closed (non repairable) systems** :

Reliability R(t) : Is the probability that the network has not failed by time t, given that it was fully operational at time zero (all components operating). There may be many failures of components but the network remains operational throughout the interval [0, t].

Mean time to failure (MTTF) : Is the average time it takes for the network to enter the failed state for the first time, given that it was fully operational at time zero. Is the average time to first failure or expected life of the system.

For **repairable systems** :

Availability A(t) : Is the probability that the network is operational at time t, given that it was fully operational at time zero. The network might have been failed and repaired one or more times during the interval [0, t] but it was made operational again by repairing or replacing the failed elements.

Mean time to repair (MTTR) : Is the average time it takes to repair the network. Usually this time is very small compared to MTTF.

Mean time between failures (MTBF) : Is the average cycle time between successive failures for repairable networks.

Steady-state availability (SA) : Is the probability of the system being operational once it has reached a steady-state ($t = \infty$). It is a measure of the fraction of time the communication system is operational.

Dynamic reliability evaluation for individual system components.

Failure rate (λ_i) : Is the average measure of the rate at which failures occur. It is generally assumed to be constant for the normal operating period (useful life) of the system, it is characterised by the exponential distribution.

Repair rate (μ_i) : Is the average measure of the rate at which repair occur. It is generally assumed to be constant (exponential distribution).

Figure 3.5 shows the typical bath-tub curve for failure rate of a component. Region I is known as the infant mortality phase; region II is the useful life period or normal operating phase in which the failure rate is constant; and region III represents the wear-out phase.

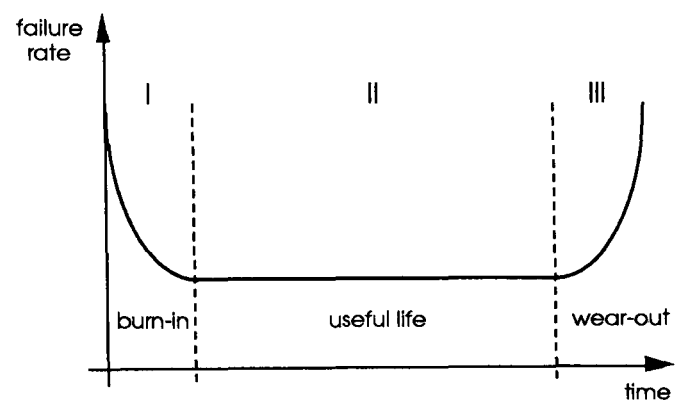


FIGURE 3.5
Bath-tub curve

Under this assumption, the time dependent measures of element x_i in the useful period of the system are :

For closed systems :

The reliability at time t :

$$R\ (x_i\ ,t\ ) = \exp\ [\ -\lambda_i t\ ] \qquad \ldots (3.3)$$

Mean time to failure :

$$MTTF\ (x_i\ ) = \int_0^{\infty} R\ (x_i\ , t\ )\ \mathrm{dt} = \frac{1}{\lambda_i} \qquad \ldots (3.4)$$

where λ_i is the failure rate of element x_i

For repairable systems :

The availability at time t is obtained with Markov modelling for a single repairable component : [BIL 83]

$$A\ (x_i\ ,t\ ) = \frac{\mu_i}{\lambda_i + \mu_i} + \frac{\lambda_i}{\lambda_i + \mu_i} \exp\ [\ -\ (\lambda_i + \mu_i)\ t\ ] \qquad \ldots (3.5)$$

Mean time to repair and mean time between failures are given by:

$$MTTR\ (x_i\ ) = \frac{1}{\mu_i} \qquad \ldots (3.6)$$

$$MTBF\ (x_i\ ) = MTTF\ (x_i\ ) + MTTR\ (x_i\ ) = \frac{1}{\lambda_i} + \frac{1}{\mu_i} \qquad \ldots (3.7)$$

where λ_i is the failure rate and μ_i is the repair rate of element x_i .

The steady-state availability is the availability at time ∞.

$$SA\ \ (x_i\ ) = A\ \ (x_i\ , \infty) = \frac{\mu_i}{\lambda_i + \mu_i} = \frac{MTTF\ \ (x_i)}{MTTF\ (x_i) + MTTR\ \ (x_i\ )} \qquad \ldots (3.8)$$

Figure 3.6 shows the average cycle time performance for a repairable component.

If the component failures and repairs are described by other general probability distribution functions, it is required to use Laplace transform techniques to solve for the reliability measures of network components.

The symbolic expression for reliability based in the probability of elements p_i is transformed into a time dependent expression by substituting $R\ (x_i\ ,\ t\ )$ or $A\ (x_i\ ,\ t\ )$ for p_i.

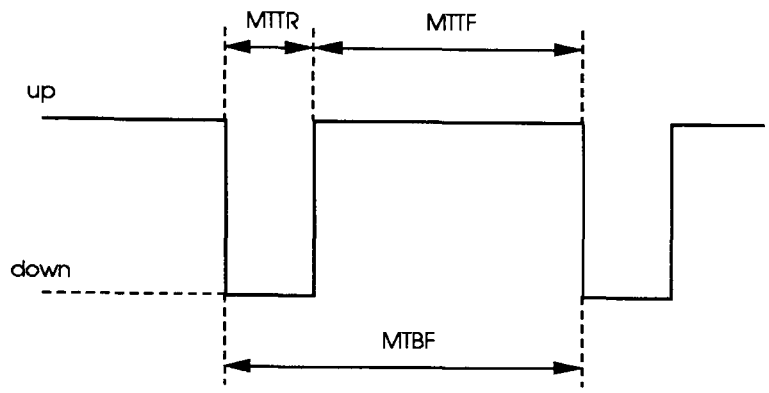


FIGURE 3.6
Average cycle time

3.4.4.4 KON system reliability

A system can be represented as a reliability network for the general model of redundancy, which includes series, parallel and k-out-of-n systems defined as follows:

Series system. A series system represents a non redundant system, where the elements of the system are said to be in series from a reliability point of view if they all must be operational for the system to be operational (R_s) or only one needs to fail for system failure (Q_s).

R_s = **Pr** {all elements are operating} is given by :

$$R_s = \prod_{i=1}^{n} p_i \qquad \ldots (3.9)$$

and $Q_s = 1 - R_s$

where p_i is the probability of element i working

Parallel system. A parallel system represents a fully redundant system, where the elements of the system are said to be in parallel from a reliability point of view if only one needs to be operational for the system to be operational (R_p) or all must fail for system failure (Q_p).

R_p = **Pr** {at least one element is operating} is given by :

$$R_p = 1 - \prod_{i=1}^{n} (1 - p_i) \qquad \ldots (3.10)$$

and $Q_p = 1 - R_p$

where p_i is the probability of element i working

***K*-out-of-*n* system.** In a k-out-of-n system or partially redundant system, at least k elements out of n must be operational for the system to be operational (R_k) or $n - k + 1$ must fail for system failure (Q_k).

A k-out-of-n system is the general model of active redundant systems, where series and parallel systems are particular cases with $k=n$ and $k=1$ respectively. Therefore, the implementation of a reliability model for k-out-of-n systems is sufficient for the modelling of series and parallel systems as well.

In a k-out-of-n system the number of components operating has a binomial distribution with parameters n and p_i. Assuming that the n components have the same probability (p):

R_k = **Pr** {at least k out of n elements are operating} is given by :

$$R_k = \sum_{j=k}^{n} C_n^j \; p^j \; (1 - p)^j \qquad \ldots (3.11)$$

where C_n^j is the number of combinations of j from n elements and is given by :

$$C_n^j = \frac{n!}{j!\,(n - j)!} \qquad \ldots (3.12)$$

and $Q_k = 1 - R_k$

This system can also be analysed for the dynamic (time dependent) environment by substituting the component reliability for the appropriate dynamic parameter.

3.4.5 *COMPLETE NETWORK RELIABILITY MODEL*

After the specification of the deterministic and probabilistic reliability models, we can propose a methodology for the design and analysis of a fault tolerant multiprocessor system incorporating both models for the intercommunication network in order to cover the different aspects described previously in sections 2.3 and 3.2. Broadly speaking, the basic methodology could be as follows:

(1) **Specification** of the initial requirements and constraints:

(a) Suitable system topologies for an application and if applicable the possible degraded configurations.

(b) Structural parameters related with fault tolerance and performance, such as maximum number of elements, degree of node, maximum distance, degree of fault tolerance, diagnosability and reconfigurability, etc.

(c) Parameters for the reliability model: Reliability and performance goals, physical parameters such as failure rates; behavioural parameters, such as repair rate (or no repair), coverage, etc.

(2) **Deterministic evaluation** of these topologies, by studying the results in terms of fault tolerance, diagnosis, reconfiguration, cost, etc. These results are then used as the basis for the structural parameters in the probabilistic model.

(3) **Probabilistic evaluation**: This model utilises the structural parameters (obtained in (2)) and the reliability parameters specified in (1.c) to compute the reliability, availability, MTTF, etc. If the required goals are met, then the most suitable configuration is chosen; If not, it is necessary a refinement of the design, which involves returning to stage (1) to obtain a different configuration.

Chapter 4

Model Implementation

4.1 INTRODUCTION

In this chapter is described the implementation of a deterministic (structural) model and a combinatorial probabilistic model for reliability analysis of multiprocessor systems. Both models are based in concepts of graph theory and the criteria of reliability as a measure of connectivity, i.e. the operation of the communication paths among the different elements in the system which is relative to the number and structure of such paths for specific reliability problems.

In a deterministic model, reliability is dependent of the distance, degree and mainly number of edge and node disjoint paths (connectivity) between the nodes in the graph representing the system. In a probabilistic model it is assumed that the elements (nodes and edges) of the system fail with some known probability, stationary (time invariant) or dynamic (time dependent) in an environment of statistically independent failures.

The computer representation of a graph is described in section 4.2; the deterministic model is presented in section 4.3 and the probabilistic model in section 4.4.

4.2 GRAPH REPRESENTATION

The efficiency of a graph algorithm as well as the ease of implementation depends on the graph representation. For our model two data structures for representing directed and undirected graphs have been used :

- *Adjacency lists*
- *List of edges.*

4.2.1 UNDIRECTED GRAPHS

Adjacency lists

An undirected graph (Figure 4.1) can be described by the list of all neighbours of each node *Adj(i)*. An example of adjacency lists for the graph of Figure 4.1 is shown in Figure 4.2 where the relative order in *Adj(n)* is unimportant. This structure is implemented by an array of n linearly linked lists.

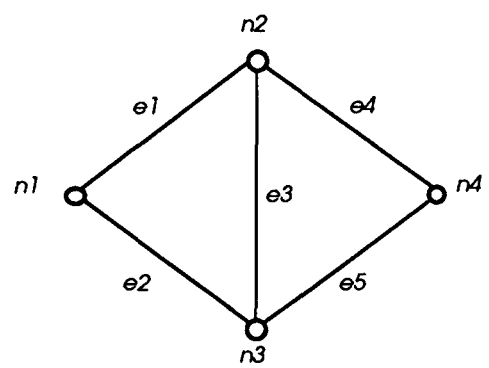


FIGURE 4.1
Undirected Graph

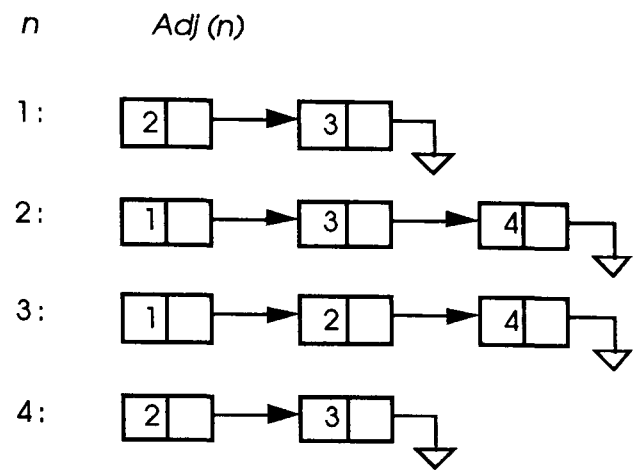


FIGURE 4.2
Adjacency lists for undirected graphs

List of edges

The list of edges in the graph is represented as pair of nodes; it can be implemented by two linear arrays : $g = (g_1, g_2, \ldots, g_e)$ and $h = (h_1, h_2, \ldots, h_e)$. Each entry in these arrays is a node label, the i^{th} edge e_i is between nodes g_i and h_i. For example, the graph in Figure 4.1 would be represented as :

$$g = (1, 1, 2, 2, 3)$$
$$h = (2, 3, 3, 4, 4)$$

4.2.2 *DIRECTED GRAPHS*

Adjacency lists

In a directed graph, the adjacency lists represent the lists of all succesors of each node, as it is shown in Figure 4.4 for the digraph of Figure 4.3.

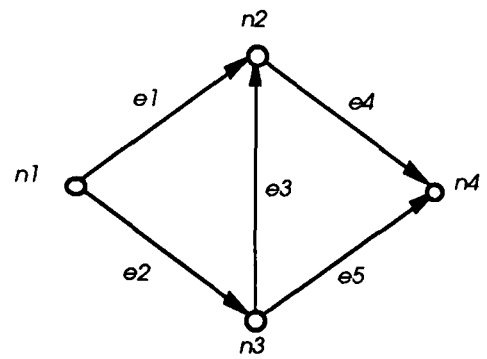


FIGURE 4.3
Directed Graph

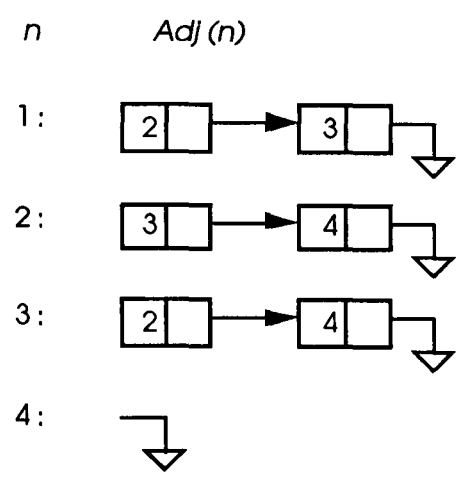


FIGURE 4.4
Adjacency lists for digraph

List of edges

For a digraph, the i^{th} edge e_i is from node g_i (predecessor) in the first array to node h_i (succesor) in the second array. The graph in Figure 4.3 would be represented as :

$$g = (1, 1, 3, 2, 3)$$
$$h = (2, 3, 2, 4, 4)$$

4.3 DETERMINISTIC MODEL

The implementation of the deterministic model consists in the calculation of the different topological parameters affecting reliability: denseness, degree, distance, and edge and node connectivity for the different reliability problems; also, the variation of these parameters is calculated when the graph is degraded by the simulation of faults in one or more nodes and/or edges, which is called *t*-edge and *t*-node deleted denseness, degree, distance and connectivity respectively.

4.3.1 DENSENESS

Denseness is simply obtained by dividing the number of edges by the number of nodes in the system graph. Figure 4.5 illustrates denseness for some graph representations of multiprocessor topologies.

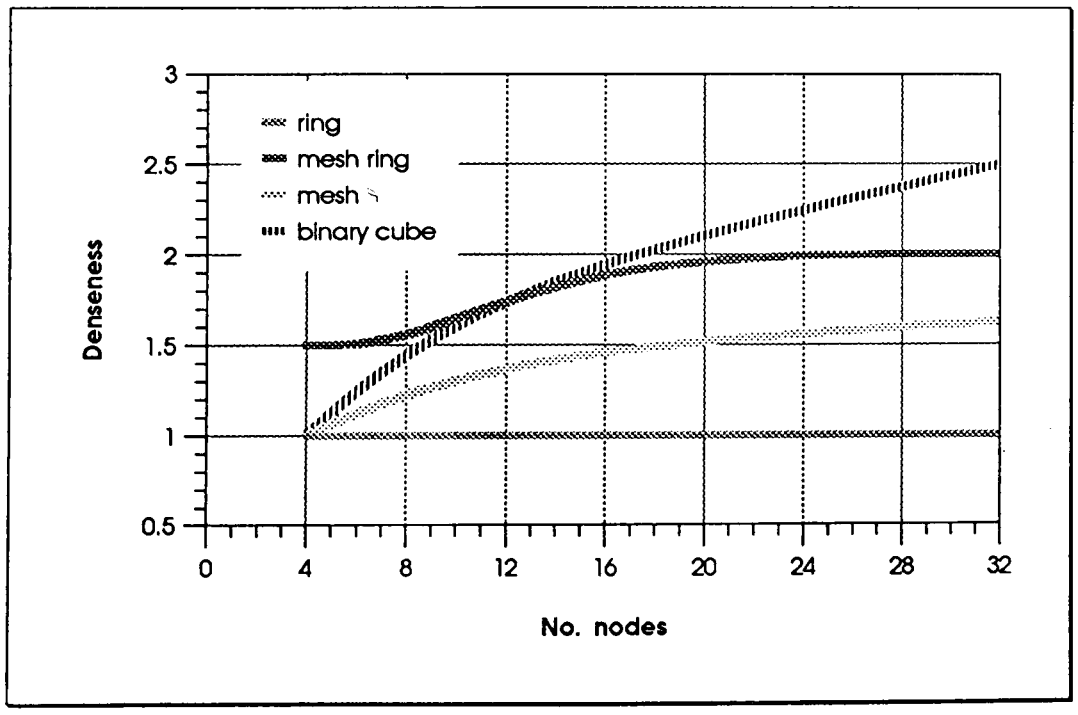


FIGURE 4.5
Denseness (e/n) versus number of nodes

4.3.2 DEGREE

4.3.2.1 Out-degree

Degree for each node of an undirected graph (number of neighbours) and out-degree for each node of a directed graph (number of succesors) are computed in the same way; it is easily done by counting their number from the adjacency lists representation of the graph. The procedure is described as follows (Algorithm 4.1).

```
procedure GetDegree;

    for all i ∈ N do
        degree_out[i] := 0;
        for all j ∈ Adj[i] do
            degree_out[i] := degree_out[i] + 1;
    end;  {for i}
    Obtain maximum, minimum and average degree or out-degree;
    end;  { GetDegree }
```

ALGORITHM 4.1
Degree for undirected graphs and out-degree for directed graphs

Figure 4.6 shows the maximum degree for some undirected configurations.

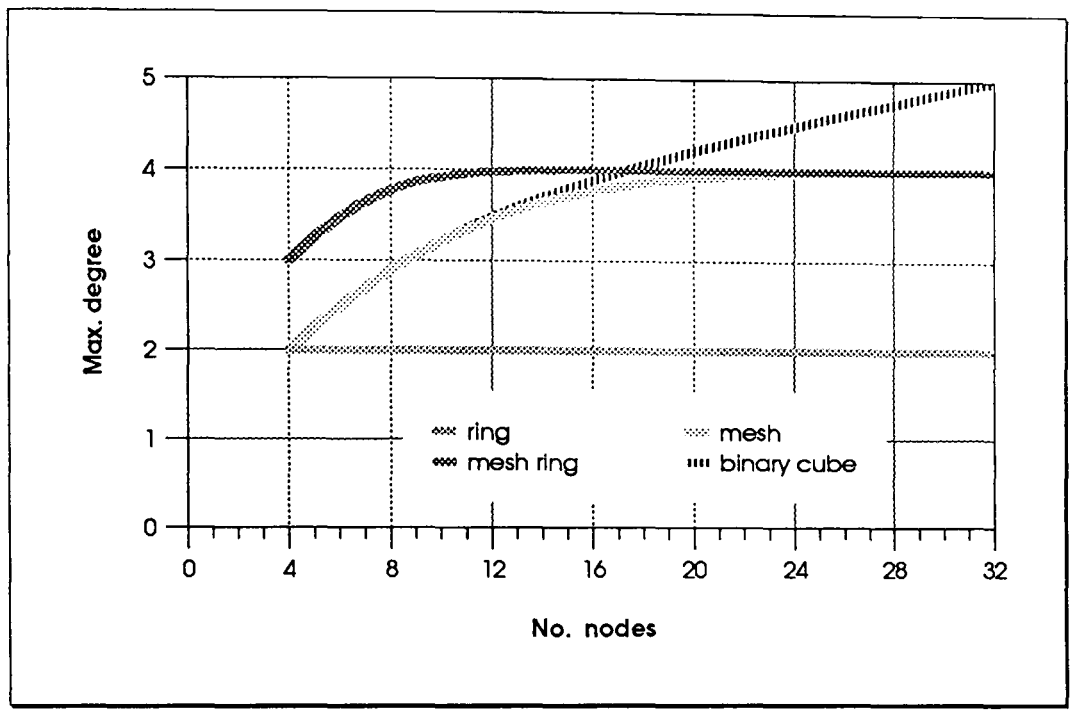


FIGURE 4.6
Max. degree versus number of nodes

4.3.2.2 *In-degree*

Procedure *GetInDegree* obtains the in-degree (number of predecessors) for a directed graph, it also obtains the maximum, minimum and average among all nodes. This can be done by searching the adjacency lists for each node *i* in the graph to get each succesor *Adj[i]*; then by updating the variable *degree_in[Adj[i]]* we obtain the number of predecessors for each node, as it is shown in Algorithm 4.2.

```
procedure GetInDegree;

    for all i ∈ N do
        degree_in[i] := 0;
    for all i ∈ N do
        for all j ∈ Adj[i] do
            degree_in[j] := degree_in[j] + 1;
    Obtain maximum, minimum and average in-degree;
    end;  { GetInDegree }
```

ALGORITHM 4.2
In-degree for a directed graph

4.3.3 *DISTANCE*

The procedure *TotalDistance* obtains the distance (lenght of the shortest path) between pairs of nodes in a way corresponding to the specified reliability problem for a directed or undirected graph. This is done by one or more calls to procedure *BFS* (breadth-first search) which is used to obtain the distance from a specified node to every other node in the graph. *TotalDistance* also obtains the maximum and average distance values among all relevant nodes. This procedure is described in Algorithm 4.3, and *BFS* in subsection 4.3.3.1.

```
procedure TotalDistance;

case problem of
   TT :   BFS (node1, dist_array);                    {get distance between node1 ...}
          distance[1, 1] := dist_array[node2];        {... and node2}
   ST :   BFS (source, dist_array);                   {get distance from source ...}
          distance[1, 1] := dist_array[terminal];     {... to terminal}
   AT :   for i=1 to n do                             {get distance between ...}
             BFS (i, dist_array);                     {... every pair of nodes}
             for j=1 to n do
                distance[i, j] := dist_array[j];
          end;
   SAT :  BFS (source, dist_array);                   {get distance from source ...}
          for j=1 to n do                             {... to every other node}
             distance[source, j] := dist_array[j];
   KT :   for i=1 to k do                             {get distance between nodes ...}
             BFS (k_set[i], dist_array);              {... in k_set}
             for j=1 to k do
                distance[i, j] := dist_array[k_set[j]];
          end;
   SKT :  BFS (source, dist_array);                   {get distance from source ...}
          for j=1 to k do                             {... to every node in k_set}
             distance[source, j] := dist_array[k_set[j]];
   KSKT: for i=1 to k_source do                       {get distance from every ... }
             BFS (k_source_set[i], dist_array);       {... node in source_set to ...}
             for j=1 to k_terminal do                 {... every node in term_set}
                distance[i, j] := dist_array[k_terminal_set[j]];
             end; {for i}
          end; {case}
   Obtain maximum (diameter) and average distance;
end;  { TotalDistance }
```

ALGORITHM 4.3

Distance for each reliability problem

4.3.3.1 *Breadth-first search (BFS)*

An algorithm which finds the distance of the shortest path from a source node (root) to every other node in a directed or undirected unweighted graph is obtained by conducting a breadth-first search [REI 77], as described in Algorithm 4.4. This algorithm uses a queue which is a FIFO data structure, i.e. data is removed in the same order that they are added. The queue used in *BFS* stores progressively the nodes ordered by their distance to the root.

```
procedure BFS (root, dist_array);

(1)   for all i ∈ N do
(2)       dist_array[i] := unlabel;
(3)   Initial empty queue;
(4)   dist := 0;                        {dist = distance to the root}
(5)   dist_array[root] := 0;
(6)   Add root to the queue;
(7)   while the queue is not empty do
(8)       Remove a node from the queue, call it succesor;
(9)       if dist_array[succesor] ≠ dist then
(10)          dist := dist + 1;
(11)      for all i ∈ Adj[succesor] do
(12)          if dist_array[i] = unlabel then
(13)              dist_array[i] := dist + 1;
(14)              Add i to the queue;
                  end; {if}
              end; {for}
          end; {while}
      end; { BFS }
```

ALGORITHM 4.4
BFS algorithm used to find distance

4.3.3.2 *Examples*

Figure 4.7 shows the overall diameter (maximum distance) obtained for some topologies and Figure 4.8 shows the normalised average distance (average degree multiplied by average distance) for the same topologies.

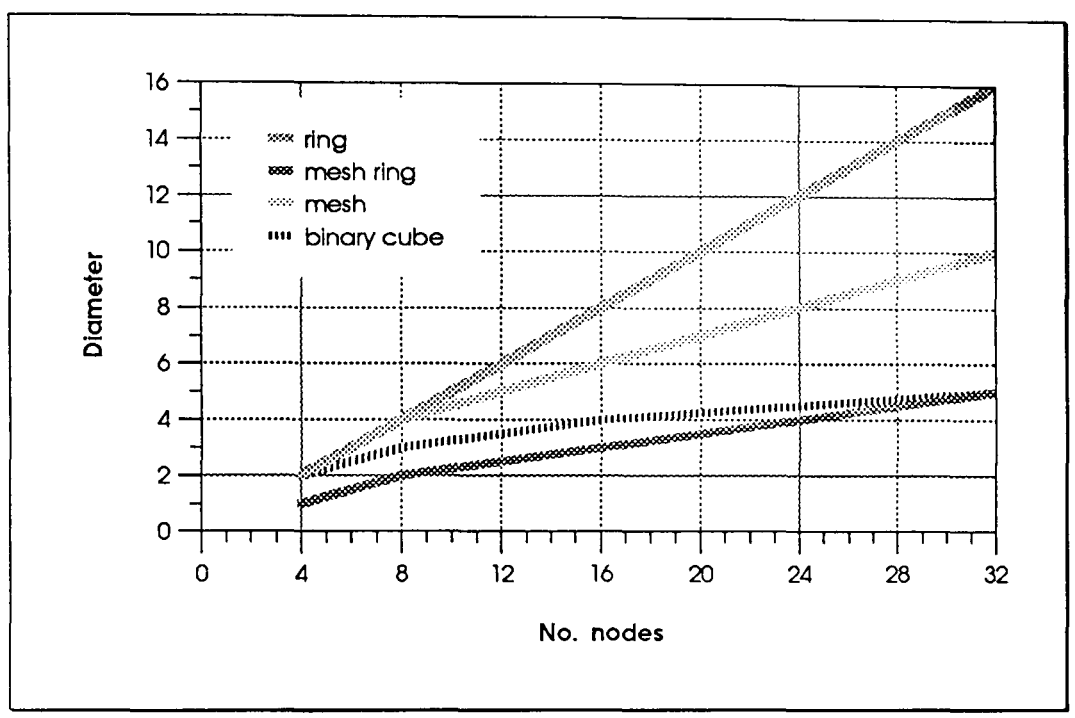


FIGURE 4.7
AT diameter versus number of nodes

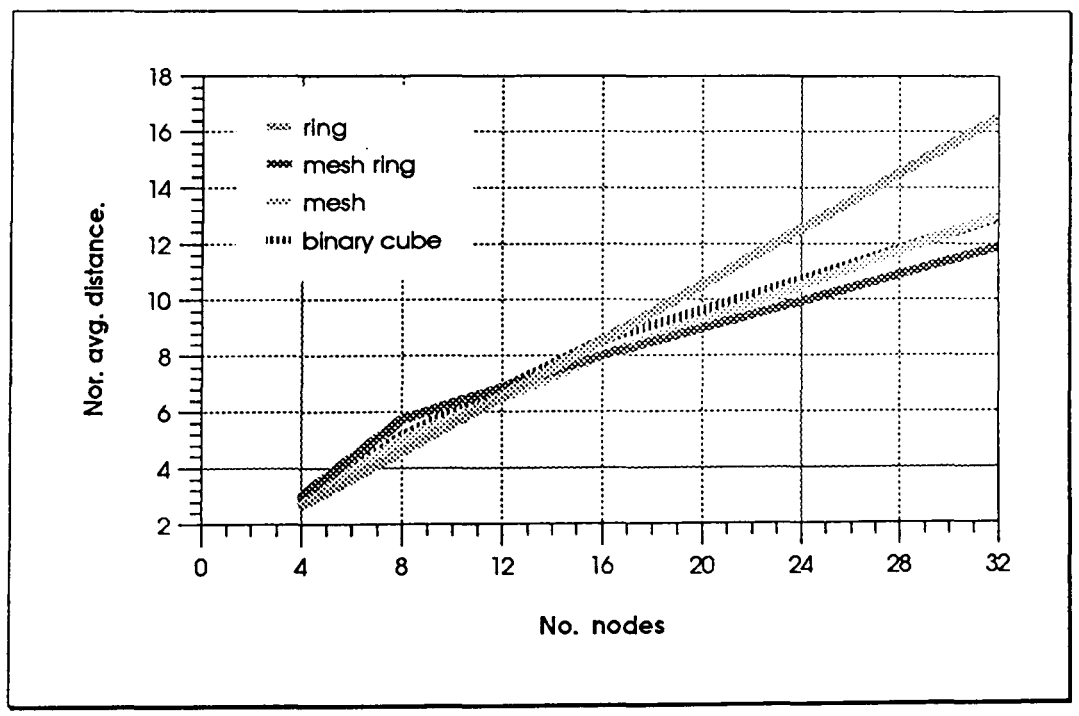


FIGURE 4.8
AT normalised average distance versus number of nodes

4.3.4 *EDGE CONNECTIVITY*

Edge connectivity (K_e) as defined for the different connectivity problems can be found by calling one or more times the maximum flow algorithm (*MaxFlow*). This algorithm (explained in section 4.3.4.2) obtains the maximum flow throughout a directed graph from a source node to a terminal node which is equivalent to the minimum number of disjoint paths between those nodes (Menger's connectivity theorem) [GIB 85].

To calculate edge connectivity for SAT in a directed graph we can solve directly those maximum flow problems for which a particular node is the source. The remaining nodes are then taken as the terminal in turn. For SKT we follow the same procedure taking the k-terminal set of nodes in turn. For KSKT we follow the same procedure as SKT but using a modified graph (described in section 4.3.4.1). ST is obtained directly from *MaxFlow*.

To solve for the unrooted problems (AT, KT and TT) in undirected graphs we follow the same procedure as before taking any node as the source, but before to do so, the graph should be transformed to directed as follows: (1) construct a new graph G' with the same set of nodes as G, and (2) replace each edge of G by two antiparallel edges. each of unit capacity. A practical advantage of the adjacency lists representation of a graph is that to perform this transformation from undirected to directed graph the data structure remains the same. The procedure to obtain edge connectivity is described in Algorithm 4.5.

```
procedure EdgeConnectivity;

   case problem of
      TT, ST :
         flow_max := MaxFlow (source, terminal);
         Ke := flow_max;
         end; {TT...}
      AT, SAT :
         Initialise Ke := |E|;
         for all i ∈ N - {source} do
            flow_max := MaxFlow (source, i);
            if flow_max < Ke then
               Ke := flow_max;
            end; {for}
         end; {AT...}
      KT, SKT, KSKT:
         Initialise Ke := |E|;
         for all i ∈ terminal_set do
            flow_max := MaxFlow (source, i);     {note: for KSKT, source is a new ...}
                                                 {... node S, see section 4.3.4.1)
            if flow_max < Ke then
               Ke := flow_max;
            end; {for}
         end; {KT...}
   end;  {case}
Output Ke;
end;  { EdgeConnectivity }
```

ALGORITHM 4.5
Edge connectivity (Ke) of a graph

4.3.4.1 *KSKT problem*

A generalisation of the SKT problem is to have several source nodes, which is the K-source to K-terminal problem (KSKT).

Let $K_s = \{s_1, s_2, \ldots, s_n\}$ be the set of source nodes of a graph G. To solve this problem it is necessary to modify the graph. This is done by adding a

new source node S to each original source s_i, as shown in Figure 4.9. The new node and new edges added, as they do not belong to the original system graph, are considered to be perfectly reliables in order to perform the proper reliability calculations.

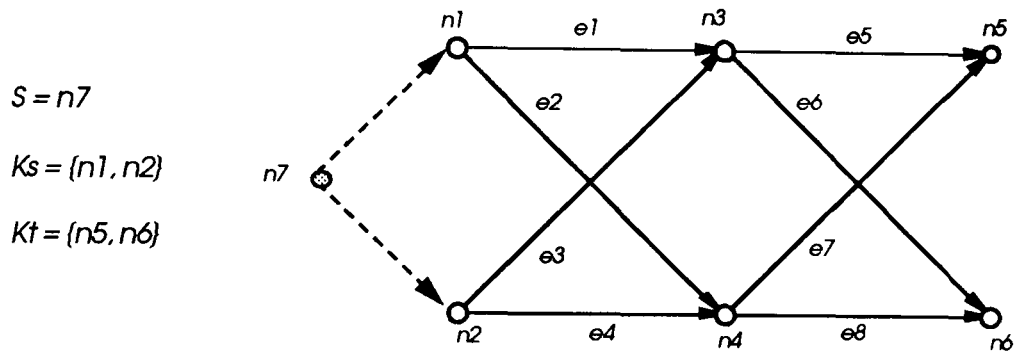


FIGURE 4.9
Modified graph for KSKT problem

4.3.4.2 *Maximum flow algorithm*

To find efficiently the maximum flow throughout a directed graph G, from a source node to a terminal node, it has been used the method of Edmonds & Karp described in [GIB 85] to finding flow augmenting paths in G which is equivalent to finding direct paths in an associate graph G^F. This is the case if G and G^F have the same set of nodes and if for any two nodes i and j, (i, j) is an edge of G^F if and only if either :

$(i, j) \in$ E and $\Delta\,(i, j)$ = capacity (i, j) - flow $(i, j) > 0$ (forward edge)

or $(j, i) \in$ E and $\Delta\,(i, j)$ = flow $(j, i) > 0$ (reverse edge)

MaxFlow algorithm is outlined in Algorithm 4.6 and the procedure to construct the associate graph G^F in Algorithm 4.7.

```
function MaxFlow (source, terminal) : flow;

    for all (i, j) ∈ E do
        capacity (i, j) := 1;                    { unit capacity }
        flow (i, j) := 0;
    path := true;                                { path records whether or not an  ... }
                                                 {... augmentation path exists for G^F }
    while path do
        ConstructAssociateGraph;
        AugmentingPath  (G^F, path, path_list);
        if path then
            Find Δ := min Δ (i, j), among all (i, j) ∈ path_list;
            for all (i, j) ∈ path_list do
                if (i, j) is a forward edge of path_list then
                    flow (i, j) := flow (i, j) + Δ;
            end; {if path}
        end; {while path}
    MaxFlow := Σ flow (source, j), for all j ∈ Adj[source]
    end;  { MaxFlow }
```

ALGORITHM 4.6
Maximum flow algorithm

```
procedure ConstructAssociateGraph;

    for all (i, j) ∈ E do
        Δ (i, j) := capacity (i, j) - flow (i, j);
        if Δ (i, j) > 0 then
            Add node j to Adj[i], recording a forward edge and Δ (i, j);
        if flow (i, j) > 0 then
            Add node i to Adj[j], recording a reverse edge and Δ (j, i) := flow (i, j);
        end;  {for}
    end;  { ConstructAssociateGraph }
```

ALGORITHM 4.7
Construct associate graph G^F

4.3.4.3 *Augmenting path algorithm*

To find the augmenting path in G, i.e. a directed path in the associate graph G^F from a source node (s) to a terminal node (t), the distance from s to t is computed using BFS algorithm as described in Algorithm 4.4, but keeping track of *pre(i)* as the algorithm progresses (that is the node preceding the node i along the shortest path) in order to find the path itself. This is done by editing BFS algorithm (*BFS_Path*) , after line (14) inserting:

(15') *pre[i] := succesor;*

Hence the nodes of the path are :

s, ..., pre(pre(pre(t))), pre(pre(t)), pre(t), t.

The augmenting path algorithm is described in Algorithm 4.8.

```
procedure AugmentingPath (G^F, path, path_list);

  BFS_Path (G^F, dist_array, pre):
  if dist_array[terminal] = unlabel then
     path := false
  else path := true;
  if path then
     for all i ∈ pre do
         Add node pre[i] to path_list;
     end; {if}
  end; { AugmentingPath }
```

ALGORITHM 4.8
Augmenting path

4.3.5 *NODE CONNECTIVITY*

The procedure to obtain node connectivity (K_n) is very similar to that for edge connectivity, but with some modifications. Based on the node

connectivity theorem of Menger, we have also to solve the maximum flow problem but for an auxiliary graph G' derived from G. Such graph is constructed as follows :

For every node $n \in N$ in G, G' contains two nodes n' and n'' and an edge (n', n'') called an *internal edge*. In addition for every edge $(n_i, n_j) \in E$ in G, G' contains two edges (n_i'', n_j') and (n_j'', n_i') which are called *external edges*. The capacity of each internal edge is one, and each external edge has an infinite capacity. Figure 4.10 shows G' for an undirected graph and Figure 4.11 for a directed graph. The maximum flow is obtained from source node s' to terminal node t''.

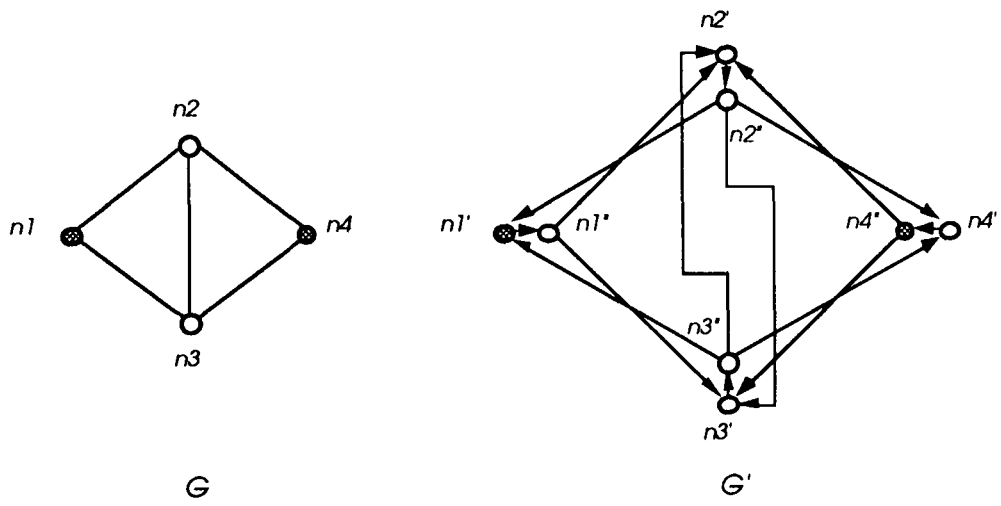


FIGURE 4.10
Auxiliary graph G' derived from undirected graph G, n1 is source and n4 is terminal

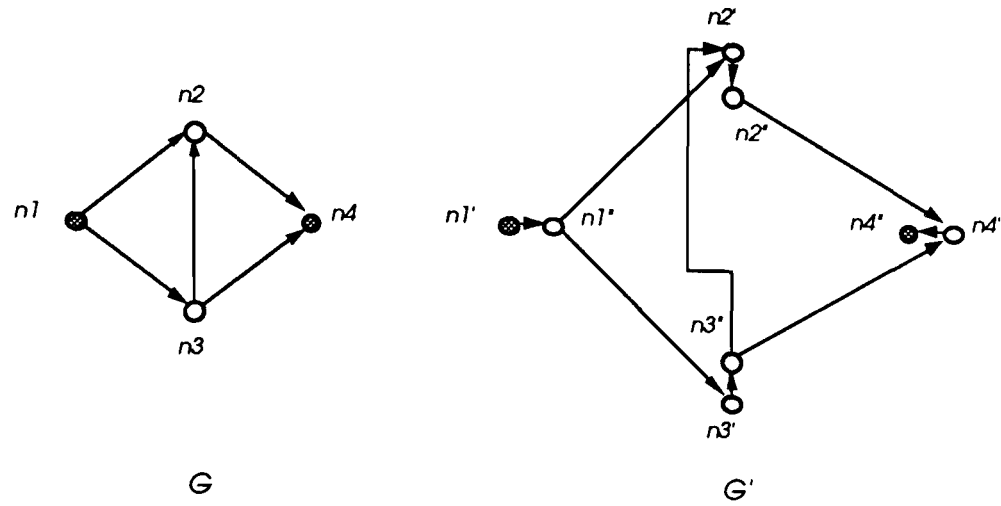


FIGURE 4.11
Auxiliary graph G' derived from directed graph G, n1 is source and n4 is terminal

TT, ST and KSKT node connectivity problems are solved with one call to *MaxFlow* with s'' as source and t' as terminal (for KSKT using the modified graph). SAT and SKT are solved with s'' as source and taking in turn every other node as terminal for SAT and every node in *terminal_set* for SKT.

AT node connectivity is guaranteed to be solved with the following process: First, we solve all those *MaxFlow* problems with n_1 as the source (taking in turn each of n_j, $j = 2, 3, \ldots, n$ as terminal, provided $(n_1, n_j) \notin$ E) then those with n_2 as the source (taking in turn n_j, $j = 3, 4, \ldots, n$ as terminal, provided $(n_2, n_j) \notin$ E) and so on until n_k has taken a turn as the source where $k = K_n(G) + 1$. This process solves all maximisation problems with n_i as source, $n_i \in \{n_1, n_2, \ldots, n_k\}$, to find node connectivity.

A similar process is used for KT, but solving only for the nodes belonging to k-set. Algorithm 4.9 outlines the procedure for node connectivity based in the preceding considerations.

```
procedure NodeConnectivity;

(1)   Generate auxiliary graph G';
(2)   Initialise Kn := n - 1;
(3)   case problem of
(4)      TT, ST, KSKT :
(5)         flow_max := MaxFlow (source', terminal");
(6)         Kn := flow_max;
            end; {TT, ST, KSKT}
(7)      SAT :
(8)         for all i ∈ N - {source} do
(9)            flow_max := MaxFlow (source', i");
(10)           if flow_max < Kn then
(11)              Kn := flow_max;
               end; {for}
            end; {SAT}
(12)     SKT :
(13)        As SAT but substituting line (8) for :  (8')  for all i ∈ terminal_set do
(14)     AT :
(15)        i := 0;
(16)        while i ≤ Kn do
(17)           i := i + 1;
(18)           for j:=i+1 to n do
(19)              if (n_i, n_j) ∉ E then
(20)                 flow_max := MaxFlow (n_i', n_j");
(21)                 if flow_max < Kn then
(22)                    Kn := flow_max;
                  end; {if, for}
               end; {while}
            end; {AT}
(23)     KT :
(24)        As AT but changing n to k in line (18),
(25)        (n_i, n_j) to (n_k-set[i], n_k-set[j]) in line (19), and
(26)        (n_i', n_j") to (n_k-set[i]', n_k-set[j]") in line (20)
      end; {case}
   end; { NodeConnectivity }
```

ALGORITHM 4.9

Node connectivity

4.3.5.1 Examples

The following graph (Figure 4.12) illustrates edge and node connectivity results obtained for the AT problem in some graph configurations.

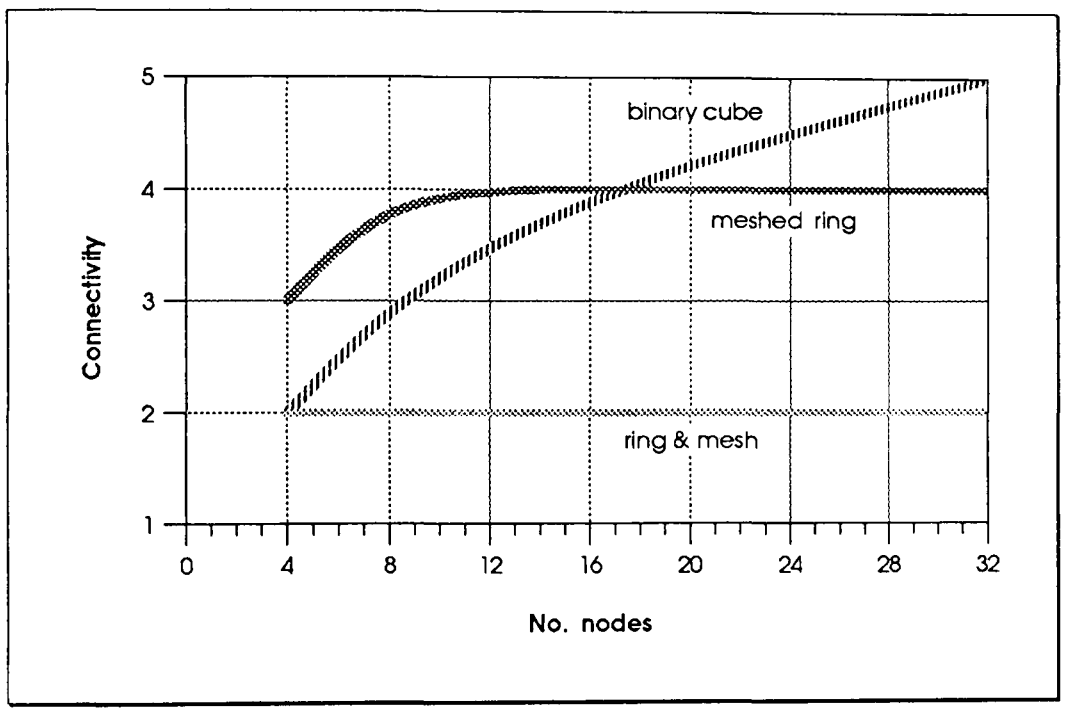


FIGURE 4.12
Edge and node AT connectivity versus number of nodes (note: ring and rectangular mesh have the same connectivity)

4.3.6 FAULT SIMULATION

The removal of edges and/or nodes have been simulated in the deterministic reliability model in such a way that edge or node connectivity is always decreased by one with the removal of a edge or node. All deterministic parameters such as denseness, degree, distance and edge and node connectivity are computed for the degraded configurations, being of particular interest the diameter of the remaining graph, called *t-node (edge) deleted distance.*

Simulation of a fault in an edge is accomplished by selecting any edge (i, j) incident to a node i of minimum degree among all nodes and generating a subgraph G_{er} by removing edge (i, j) from the original system graph G; simulation of a fault in a node is accomplished by selecting any node j, neighbour or predecessor of a node i with minimum degree and creating a subgraph G_{nr} by removing node j from the original graph G as well as its incident edges; proceeding in this way it is guaranteed that the edge or node connectivity is reduced by one when the edge (i, j) or the node j is deleted. An example is shown in Figure 4.13.

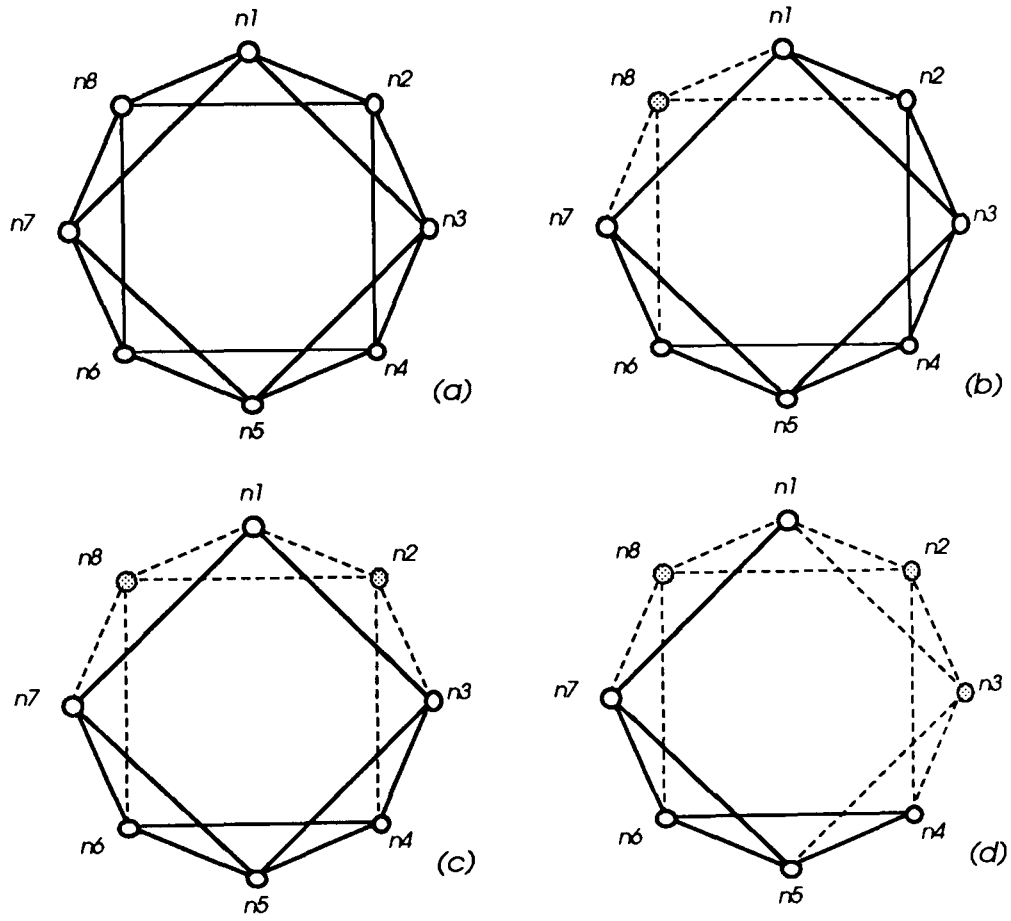


FIGURE 4.13
Example of fault simulation in a 4x2 meshed ring (a), one node removed in (b), two nodes removed in (c) and three nodes removed in (d)

After computing the deterministic parameters of interest, the procedure is repeated succesively until the remaining graph become disconnected. In Algorithm 4.10 is outlined the above procedure.

```
procedure SimulaFaults;

   repeat
      case class of
         edge_fault :
            i := node with minimum degree;
            (i, j) := incident edge;
            G_er := Obtain subgraph (G, (i, j));
            G := G_er;
            end; {edge_fault}
         node_fault :
            i := node with minimum degree;
            j := neighbour or predecessor node;
            G_nr := Obtain subgraph (G, j);
            G := G_nr;
            end; {node_fault}
        end; {case}
      Compute deterministic reliability parameters for G;
   until G is disconnected;
   end; {SimulaFaults}
```

ALGORITHM 4.10
Simulation of faults in edges and nodes

4.3.6.1 *Examples*

Denseness, diameter, normalised average distance and edge and node connectivity have been evaluated for some configurations when faults are simulated as described above. Figures 4.14 to Figure 4.17 show the results obtained.

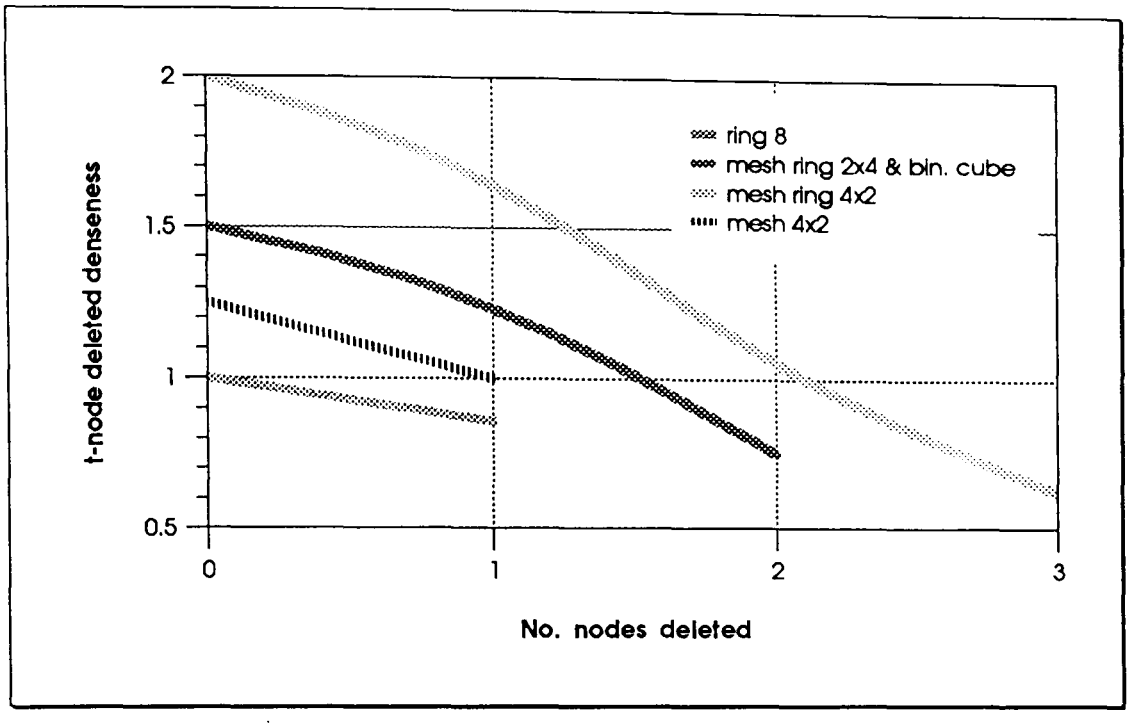


FIGURE 4.14
t-node deleted denseness versus number of nodes deleted (t)

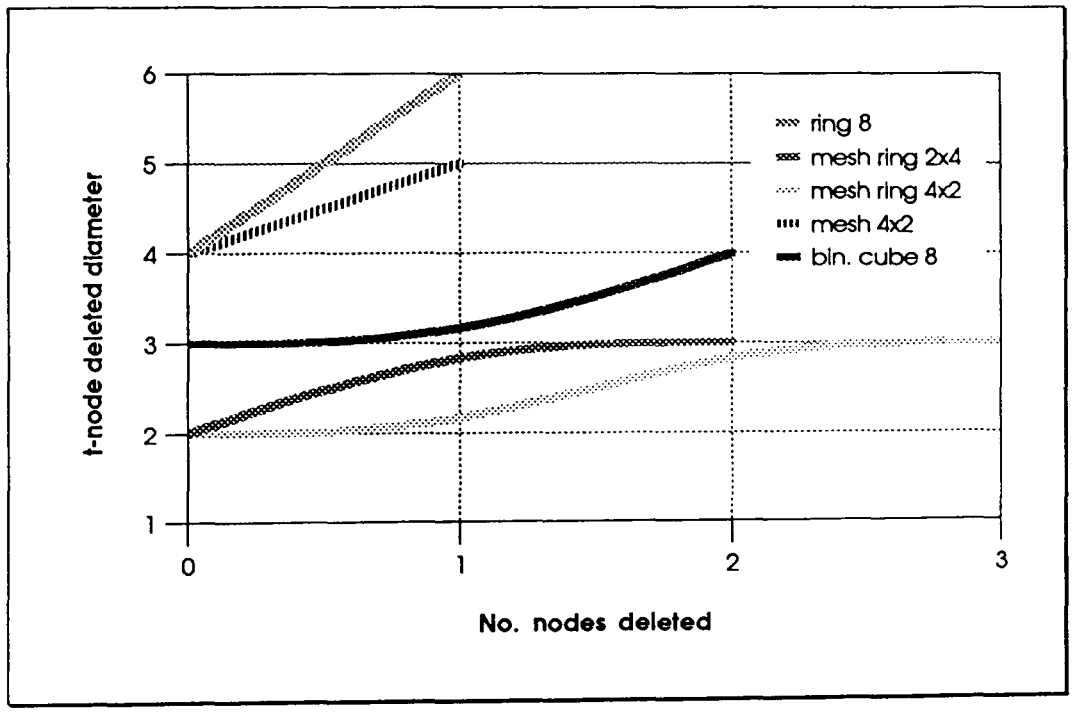


FIGURE 4.15
t-node deleted AT diameter versus number of nodes deleted (t)

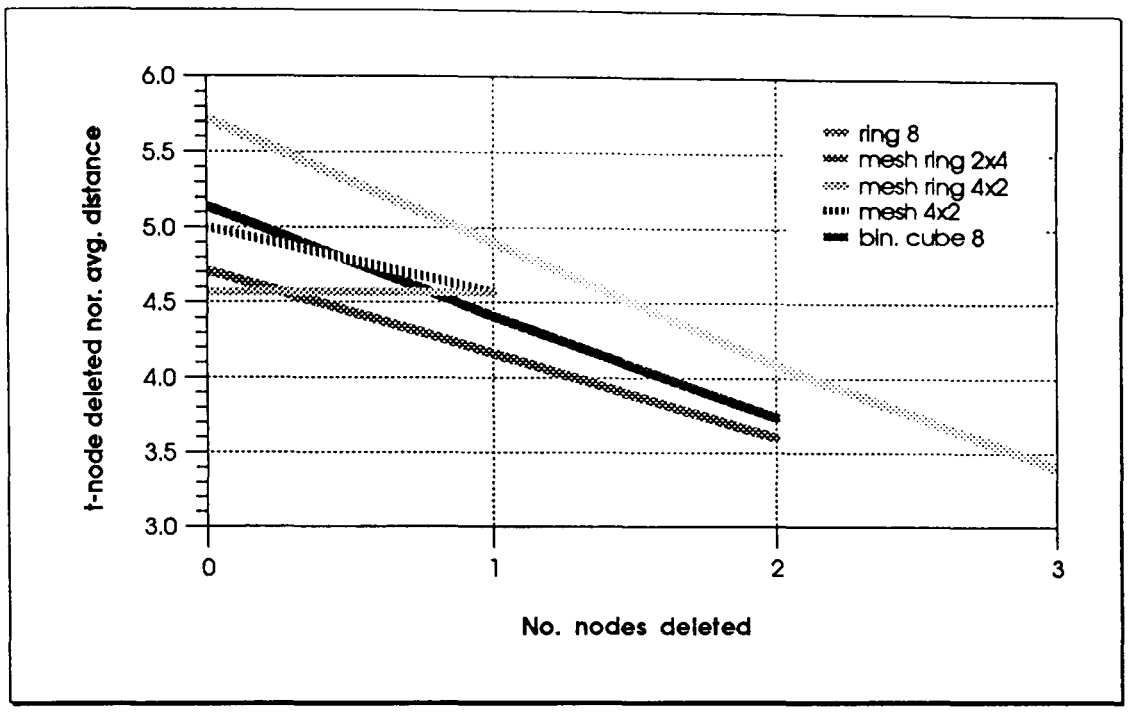


FIGURE 4.16
t-node deleted AT normalised average distance versus number of nodes deleted (t)

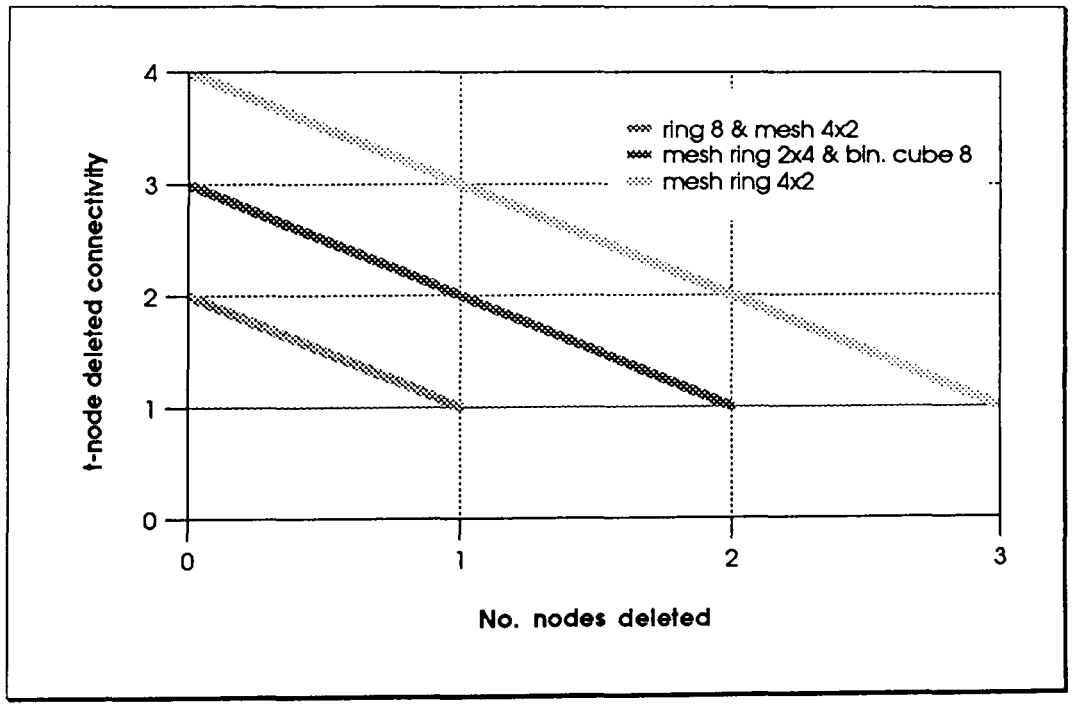


FIGURE 4.17
t-node deleted AT connectivity versus number of nodes deleted (t)

4.4 PROBABILISTIC MODEL

Each of the different rooted and unrooted probabilistic reliability problems for directed and undirected graphs is computationally difficult to solve [COL 87], thus efficiently computable algorithms are of significant interest. In other related work found in the literature, TT and AT problems have been widely studied but treated separately, and very few results apply to KT and to rooted problems in directed graphs[1]. Therefore for this work a simple and efficient methodology has been developed to deal with all reliability problems in a general framework. The general method suggested consists basically of three steps :

(1) Taking either *(i)* all simple paths between a given pair of nodes for TT problem, or *(ii)* all spanning trees for AT problem, or *(iii)* all Steiner trees for KT problem for undirected graphs, or *(iv)* all the directed paths from source to terminal node for ST problem, or *(v)* all the spanning out-trees for SAT problem; or *(vi)* all the Steiner out-trees for SKT, or *(vii)* all the Steiner out-trees of the modified graph for KSKT problem for directed graphs; as the events in the system probability space and represent them by *cubes* as explained in subsection 4.4.1.

(2) Performing some Boolean operations on the cubes to arrive at a Boolean algebraic expression. In this case the *"sharp" operation* among the cubes is applied, as described in subsection 4.4.1.

(3) Interpreting the Boolean expression as a symbolic probability expression in order to obtain the measures for the probabilistic event of

---

[1] For reference to TT problem see [GRN 80], [TOR 83] and [HAR 86] as the most efficient algorithms; for AT problem, see [AGG 81] and [XU 86]; for KT problem see [PAG 88], for rooted problems, particularly SKT, see [SAT 82].

interest, by representing the expression as a disjoint sum. The measures can be stationary probability of success and/or time dependent reliability measures.

Steps (1) and (2) could be applied sequentially, finding first all appropriated trees in the system corresponding to the specified problem, and then obtaining a Boolean expression, but the requirement of generating and storing all trees first makes this approach not practical for large systems since the number of trees grows exponentially with the number of nodes and links.

In our method, based on an algorithm developed for overall reliability by [XU 86], steps (1) and (2) are executed recursively in order to gradually obtain a disjoint sum of terms (Boolean expression); the advantage of this approach is that reduces considerably the storage and computing time since no all trees generated have to be stored. This method is explained in detail in subsection 4.4.2.

4.4.1 *CUBE REPRESENTATION AND "SHARP" OPERATION*

For a graph consisting of n nodes and e edges, a identifier for a tree is defined by the following :

Definition 1

The tree identifier IT_a for the tree T_a is defined as a string of k binary variables

$$IT_a = x_1, x_2, \ldots, x_i, \ldots, x_k$$

where

$$x_i = \begin{cases} 1 & \text{if the } i^{th} \text{ element of the graph is included in the tree} \\ x & \text{otherwise} \end{cases}$$

and k is the number of elements subject to failure, i.e. :

$k = e$ in the case of links subject to failure

$k = n$ in the case of nodes subject to failure

$k = e + n$ in the case of both links and nodes subject to failure

As an example, consider the undirected graph of Fig. 4.1. A simple path from n_1 to n_4 is $T_1 = (n_1, e_1, n_2, e_4, n_4)$ (see Figure 4.18); if only imperfect links are considered, the path is represented by the identifier: $IT_{1(e)}$ = 1xx1x, corresponding to $(e_1, e_2, e_3, e_4, e_5)$; if faults in nodes are considered: $IT_{1(n)}$ = 11x1 corresponding to (n_1, n_2, n_3, n_4); and for faults in nodes and in links: $IT_{1(e+n)}$ = 1xx1x 11x1 corresponding to $(e_1, e_2, e_3, e_4, e_5, n_1, n_2, n_3, n_4)$.

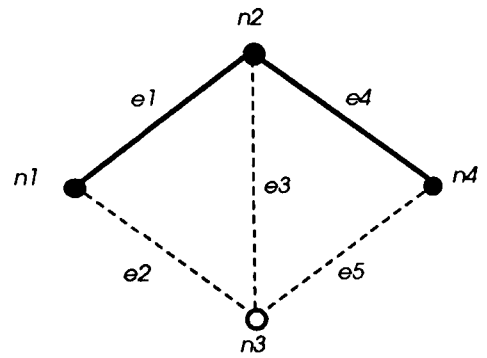


FIGURE 4.18
Simple Path, $IT_{1(e+n)}$ = 1xx1x 11x1

A spanning tree $T_2 = (n_2, e_1, n_1, e_3, n_3, e_4, n_4)$ (shown in Figure 4.19) is represented by the identifiers :

(a) $IT_{2(e)}$ = 1x11x for faults in links.

(b) $IT_{2(n)}$ = 1111 for faults in nodes (obviously, since a spanning tree spans over all nodes).

(c) $IT_{2(e+n)}$ = 1x11x 1111 for faults in links and in nodes.

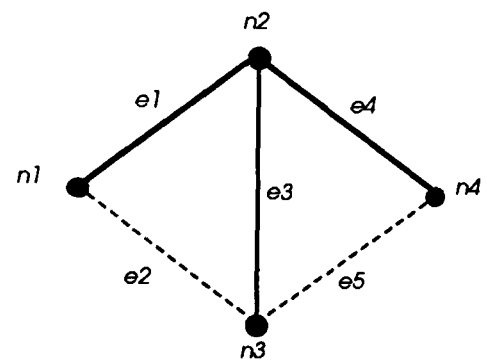


FIGURE 4.19
Spanning Tree, $IT_{2(e+n)}$ = 1x11x 1111

The minimum Steiner tree (T_3) shown in Figure 4.20, which spans over n_1, n_2 and n_3, $T_3 = (n_1, e_1, n_2, e_2, n_3)$ is represented as :

(a) $IT_{3(e)} = 11xxx$ for faults in links.

(b) $IT_{3(n)} = 111x$ for faults in nodes.

(c) $IT_{3(e+n)} = 11xxx\ 111x$ for faults in links and in nodes.

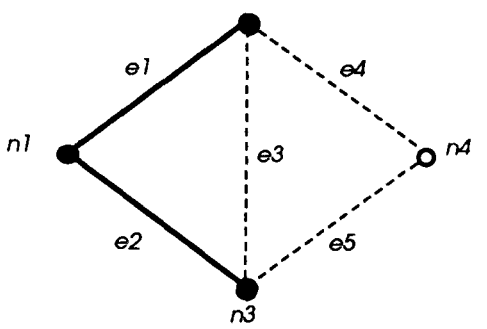


FIGURE 4. 20
Steiner Tree, $IT_{3(e+n)}$ = 11xxx 111x

An example of a digraph was presented in Fig. 4.3; a directed path, spanning out-tree and Steiner out-tree with their corresponding tree identifiers represented as cubes (for edges and nodes) are shown in Figures 4.21, 4.22 and 4.23 respectively.

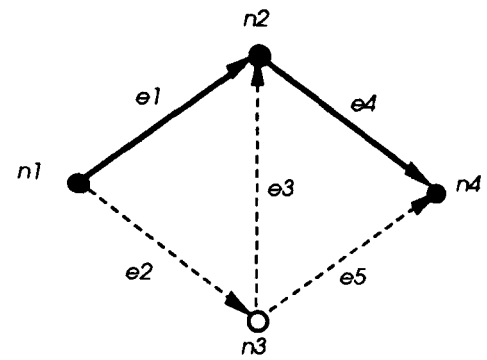


FIGURE 4.21
Directed Path, $IT_{(e+n)}$ = 1xx1x 11x1

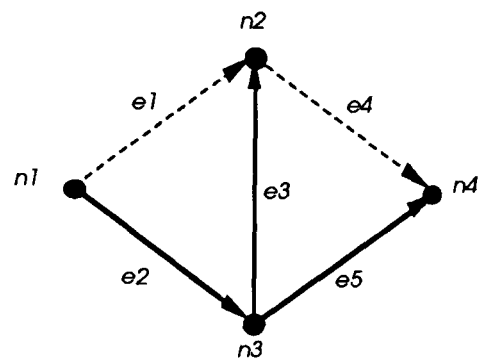


FIGURE 4.22
Spanning Out-tree, $IT_{(e+n)}$ = x11x1 1111

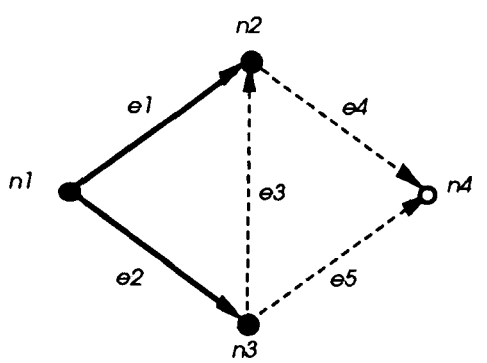


FIGURE 4.23
Steiner Out-tree, $IT_{(e+n)}$ = 11xxx 111x

A *cube* in Boolean algebra is a geometrical representation of a Boolean function by mapping a function of n-variables onto a n-dimensional unit (n-cube) [MIL 65].

From *Definition 1* it can be seen that a tree identifier has the form of a cube, thus a cube will be used to represent a tree in Boolean algebra.

Definition 2

Let s_i be the state of the element x_i of the system graph, where :

$$s_i = \begin{cases} 0 & \text{if } x_i \text{ has a failure} \\ 1 & \text{if } x_i \text{ is good} \\ x & \text{arbitrary} \end{cases}$$

A cube is a string of the type :

$$C = s_1, s_2, \ldots, s_i, \ldots, s_k$$

where k, as before, is the number of elements in the system graph.

A Boolean expression is generated by applying the *"sharp" operation* (#-operation) between two cubes, denoted as $A \# B$, in this way the set of subcubes of A not included in B is obtained, which is the disjoint sum. Definitions 3 and 4 constitute the algebraic description of the #-operation :

Definition 3

The *coordinate* #-operation is defined as given in Table 4.1.

TABLE 4.1
Coordinate #-operation, $a_i \# b_i$

a_i \ b_i	#	0	1	x
	0	z	y	z
	1	y	z	z
	x	1	0	z

Note that $a_i \# b_i \neq b_i \# a_i$

Definition 4

The #-operation between two cubes $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_n$ is defined as :

$$A \# B = \begin{cases} A & \text{if } a_i \# b_i = \text{y for any } i \\ \varnothing & \text{if } a_i \# b_i = \text{z for all } i \\ \bigcup_{i=1}^{n} C_i & \text{otherwise} \end{cases}$$

where $C_i = ((a_1 b_1), \ldots, (a_{i-1} b_{i-1}), \alpha_i, a_{i+1}, \ldots, a_n)$,

$a_i \# b_i = \alpha_i = 0$ or 1

and $a_i b_i$ is the coordinate intersection as defined in Table 4.2

TABLE 4.2
Coordinate intersection operation, $a_i b_i$

a_i \ b_i	**0**	**1**	**x**
0	0	∅	0
1	∅	1	1
x	0	1	x

The intersection between two cubes is defined as:

Definition 5

$$A \cap B = \begin{cases} \varnothing & \text{if } a_i b_i = \varnothing \\ (a_1 b_1, a_2 b_2, \ldots, a_n b_n) & \text{otherwise} \end{cases}$$

The following are the properties of #-operation :

(a) $A \# B = A$ if $A \cap B = \varnothing$

(b) $A \# B = \varnothing$ if $A \cap B = A$

(c) if $A \# B = \cup C_i$ then $C_j \cap C_k = \varnothing\ (j \neq k)$, $B \cap C_i = \varnothing$, namely all cubes C_i in $\cup C_i$ are pair-disjoint. Therefore $\cup$ can be replaced by Σ, i.e.

$$A \# B = \sum_{i=1}^{n} C_i$$

4.4.2 *ALGORITHM FOR BOOLEAN EXPRESSION*

The basic recursive algorithm for the derivation of the Boolean expression (generation of the total set of pair-disjoint cubes) of a graph G can now be described in pseudo-code by Algorithm 4.11. The variable *BooleanExpression*, which represents the symbolic boolean expression, is stored on disc in a sequential file to be used later to calculate the different numerical reliability measures.

The initial conditions for the procedure are: Y is the universal of the sample space: Y = (x,...,x) and *BooleanExpression* is empty, before calling *GetBooleanExpression*.

---

procedure *GetBooleanExpression (Y, G);*

(1) ***case*** *problem* ***of***

TT : Find a shortest simple path T of the graph G ;

AT : Find a minimum spanning tree T of the graph G ;

KT : Find a minimum Steiner tree T of the graph G ;

ST : Find a shortest directed path of the graph G ;

SAT : Find a minimum spanning out-tree of the graph G ;

SKT : Find a minimum Steiner out-tree of the graph G ;

KSKT : Find a minumum Steiner out-tree of a modified graph G' ;

end; *{case}*

(2) *Represent T as a cube A' ;*

(3) $A = Y \cap A'$; *{Intersection operation to get the real cube representation}*

(4) *BooleanExpression := BooleanExpression + A ;*

(5) *Find* $Y \# A = \sum_{i=1}^{r} B_i$ $(0 < r \leq n)$ *to get a set r of pair-disjoint cubes ;*

B_i *corresponds to a subgraph* G_i *of G, the correspondence is :*

$x_j \notin G_i$ *if* $b_j = 0$

$x_j \in G_i$ *otherwise, (i.e.* $b_j = 1$ *or x)*

(6) *Apply this procedure (GetBooleanExpression) recursively to every connected subgraph* G_i *until all the resulting subgraphs are disconnected :*

for *i := 1* ***to*** *r* ***do***

begin

Find the corresponding subgraph G_i *of* B_i

if G_i *is connected* ***then***

GetBooleanExpression (B_i, G_i);

end *; {for}*

(7) ***end;*** *{ GetBooleanExpression }*

ALGORITHM 4.11
Get Boolean expression

4.4.2.1 *Computer analysis*

The whole recursive computation of the algorithm can be described by a computation tree; the root of the tree indicates the first time the algorithm is called (when the first cube or subgraph is generated). Each subset of cubes

(subgraphs) generated from it is represented by each branch of this node in the computation tree. Subsequent subgraphs, recursively generated, are represented by successive branching of the tree.

In Figure 4.24 is shown an example for the computation of TT problem from n_1 to n_4 for the graph in Fig. 4.1, considering only faults in edges. A terminal node (square) denotes a disconnected subgraph, an internal node (circle) denotes a connected subgraph and the labels in the circles denote the order of path generation (preorder traversal of the tree).

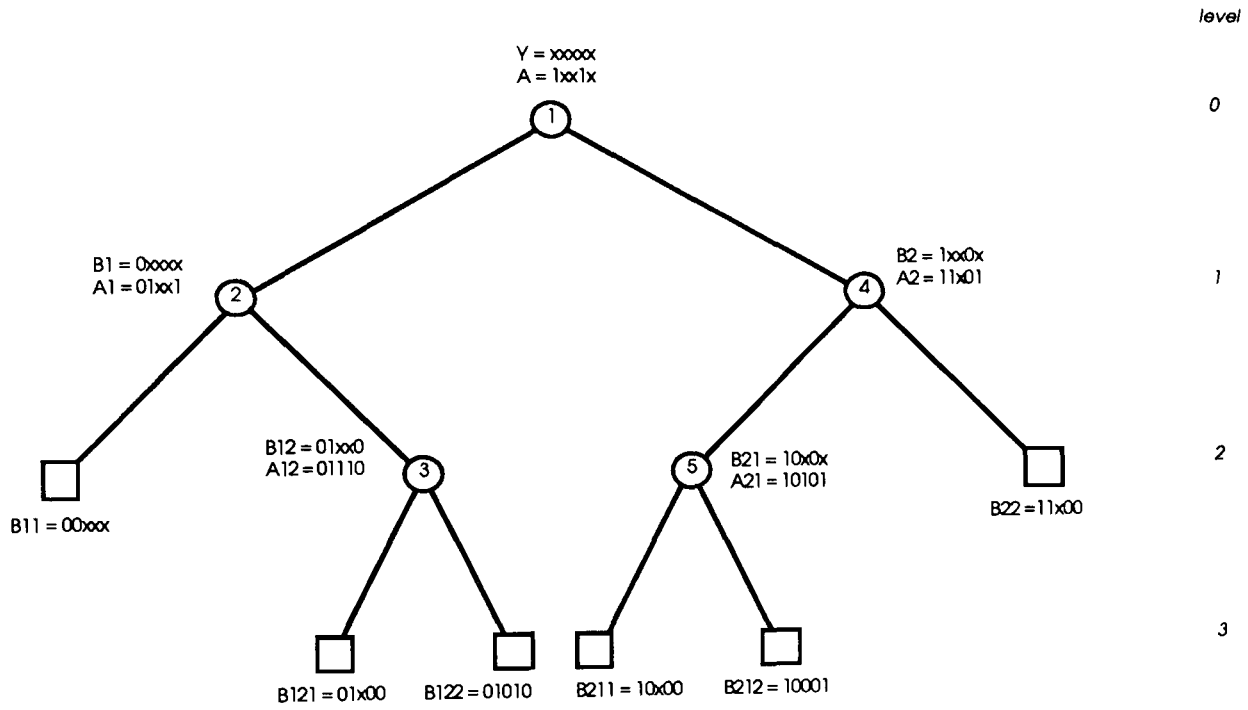


FIGURE 4.24
Computation Tree of graph G (TT problem)

As the recursive algorithm goes deeper (the level of the computation tree is increased), there are more zeros in the cube, i.e. there are fewer edges in the corresponding subgraph (each time a #-operation is done, there is one more zero in the cubes generated). When a subgraph has less than n-1 edges it is disconnected, therefore the depth of the computation tree cannot be higher than e-n+1.

4.4.3 *APPROXIMATION METHOD*

In the deeper levels of the computation tree, as the number of zeros is large, the contribution of a cube to the symbolic expression for the reliability measures of interest can be very small depending of their reliability values.

If a tolerant error is given, then a level L can be decided such if a small contribution is obtained in all levels deeper than L, the algorithm will not go beyond it, i.e. only part (the most significant) of the paths or trees are obtained for the graph; thus saving storage and computation time which can be significantly.

4.4.4 *UNROOTED PROBLEMS*

In section 3.4.2 it was mentioned that TT and AT problems are special cases of KT with k =2 and k =n respectively. So it would be possible to use only one algorithm to generate Steiner trees and generalise it for shortest paths and spanning trees. Unfortunately this approach was not followed since the construction of a minimum Steiner tree is the most difficult and time consuming problem and this generalisation would affect considerably the efficiency of the algorithm. Thus, a different algorithm has been implemented for each of the problems: spanning tree for AT problem, shortest path for TT and Steiner tree for KT.

To represent computationally the graph, it has been used the adjacency lists and list of edges as explained in section 4.2.1. The later representation is very useful for this model because the indices in arrays g and h correspond to the indices in a cube representation of edges, which allow an easy identification of the state of graph elements.

4.4.4.1 *Spanning tree*

Given the adjacency lists and the list of edges representation of a undirected graph, by conducting on it a *BFS* (*breadth-first search*) (see section 4.3.3.1) taking any node as source, a breadth-first spanning tree is constructed, which is a minimum spanning tree. The set of edges obtained which belong to the tree are represented by a cube which is obtained by editing *BFS* (Algorithm 4.4) as follows :

(a) Initialising a cube array,

```
for all i ∈ E do
        cube[i] := x;
```

(b) Inserting after line (11):

```
(12')        cube[index[i]] := G;
```

4.4.4.2 *Shortest path*

An algorithm to find the shortest distance between two nodes was described in section 4.3.3.1 using *BFS*, which is modified as explained in the previous section. Both edges and nodes belonging to the path are represented by a cube.

4.4.4.3 *Steiner tree*

A minimum spanning tree can be obtained with an algorithm like *BFS* or *DFS* (*depth-first search*). However for a problem which appears to be closely related: the minimum Steiner tree problem, there is not a polynomial-bounded solution [LAW 76]. This difficulty can be largely overcome by using

heuristic algorithms, as the one developed for this model which is described in the following (Algorithm 4.12):

---

procedure *FindSteinerTree;*

(1) *Considering the subset K of nodes, the distance among them is calculated applying the BFS procedure (K - 1) times.*

(2) *The pair of nodes in K with minimum distance between them is selected if at least one of the nodes has not been selected before. The shortest path between them is obtained.*

(3) *The Steiner tree is constructed by adding to it the path obtained.*

(4) *Repeat steps (2) and (3) with the next shortest distance between two nodes until all K nodes are selected and the Steiner tree is completed.*

(5) ***end.*** *{ FindSteinerTree }*

ALGORITHM 4.12
Find Steiner tree

With this algorithm it is possible to construct a near-minimal Steiner tree for the majority of graph configurations in which is applied.

4.4.5 ROOTED PROBLEMS

Two of the algorithms utilised to implement the model for undirected graphs can be used for directed graphs: *(i)* to find the directed paths for ST problem (same as shortest path for TT) and *(ii)* to find the spanning out-trees for SAT (same as spanning tree for AT). But for SKT problem a new algorithm was implemented to find a Steiner out-tree, which is also used in the modified graph for KSKT problem.

4.4.5.1 *Steiner out-tree*

The algorithm developed to find a Steiner out-tree is described in the following (Algorithm 4.13):

---

procedure *FindSteinerOutTree;*

(1) *Obtain the distance from root node R to the nodes belonging to the K-terminal set (K_t) using BFS algorithm;*
(2) *Initially nodes in K_t have not been visited yet;*
repeat
(3) *Find node t with longest distance from R, which has not been visited;*
(4) *Obtain shortest path from R to t;*
(5) *Visit all nodes along the path which belong also to K_t ;*
(6) ***until*** *all nodes in K_t have been visited;*
(7) ***end.*** *{ FindSteinerOutTree }*

ALGORITHM 4.13
Find Steiner out-tree

4.4.6 *RELIABILITY MEASURES*

Once a Boolean expression has been obtained, which consists of a disjoint sum of cubes, it can be transformed into a symbolic or numerical reliability expression by substituting the cube values for the different stationary and time dependent (for closed and repairable systems) reliability measures described in section 3.4.4.

4.4.6.1 *Stationary reliability*

The stationary probability of success, corresponding to a cube C_j in the Boolean expression can be calculated as :

$$\mathbf{Pr}\{C_j\} = P * Q \qquad \ldots (4.1)$$

where

$$P = \prod p_i \qquad \text{for all } i \text{ satisfying } s_i = 1$$

$$Q = \prod (1 - p_i) \qquad \text{for all } i \text{ satisfying } s_i = 0$$

p_i = **Pr** {element i is working}

The symbolic reliability expression $R(e)$ is then :

$$R(e) = \sum_{j=1}^{r} \mathbf{Pr}\{C_j\} \qquad \ldots (4.2)$$

where

r is the total number of cubes,

e is the reliability problem (TT, AT, KT, etc.)

The respective unreliability $U(e)$ is :

$$U(e) = 1 - R(e) \qquad \ldots (4.3)$$

In order to test the algorithms that have been implemented, TT, AT and KT stationary reliability were computed for the undirected graph of Fig. 4.1; ST, SAT and SKT for the directed graph of Fig. 4.3 and KSKT for the directed graph of Fig. 4.5. Figure 4.25 to Figure 4.27 show the results obtained.

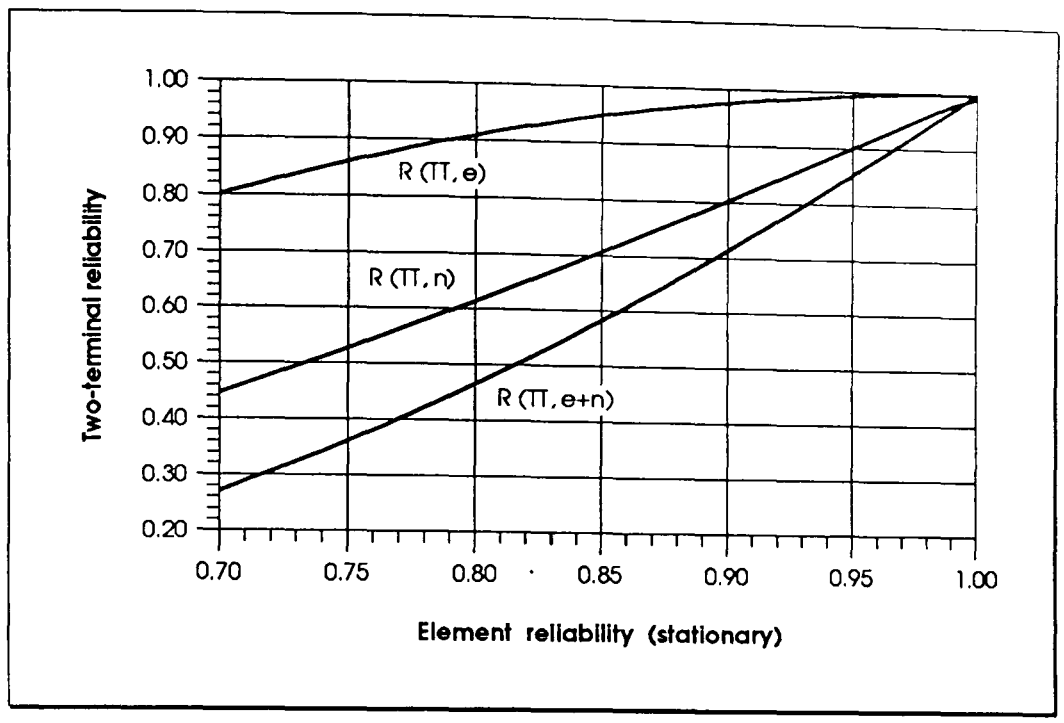


FIGURE 4.25
Two-terminal stationary reliability of Fig. 4.1, considering faults in edges R(e), in nodes R(n) and in both R(e+n), $t_1 = n_1$, $t_2 = n_4$

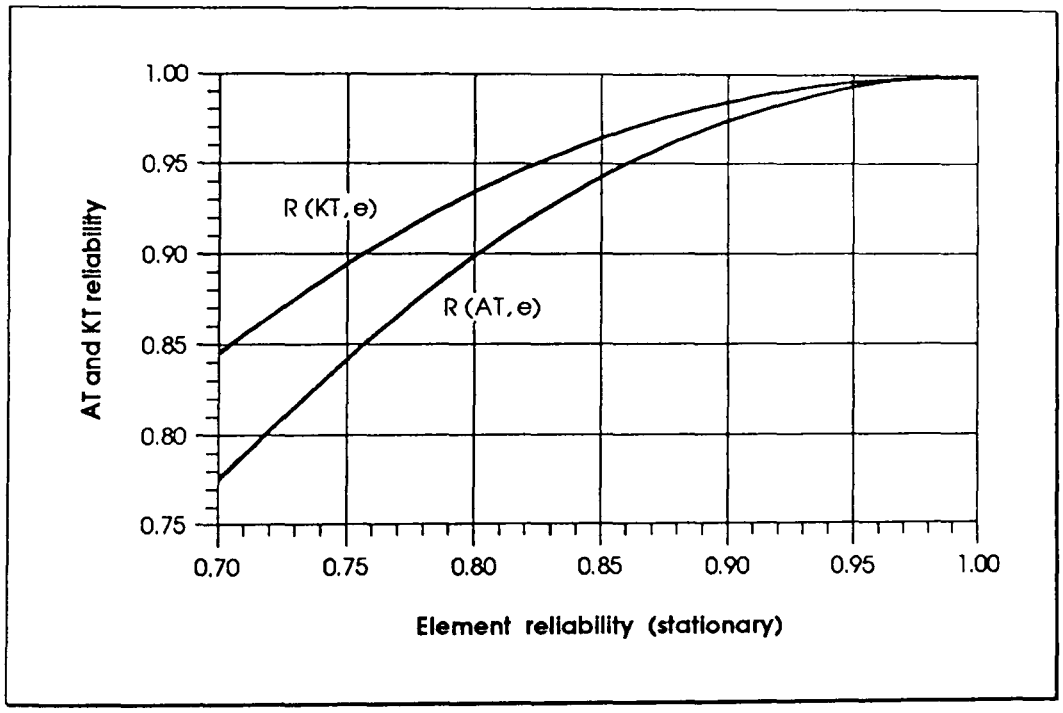


FIGURE 4.26
Overall (AT) and k-terminal (KT) with $K = \{n_1, n_2, n_3\}$ stationary reliability of Fig. 4.1, considering only faults in edges

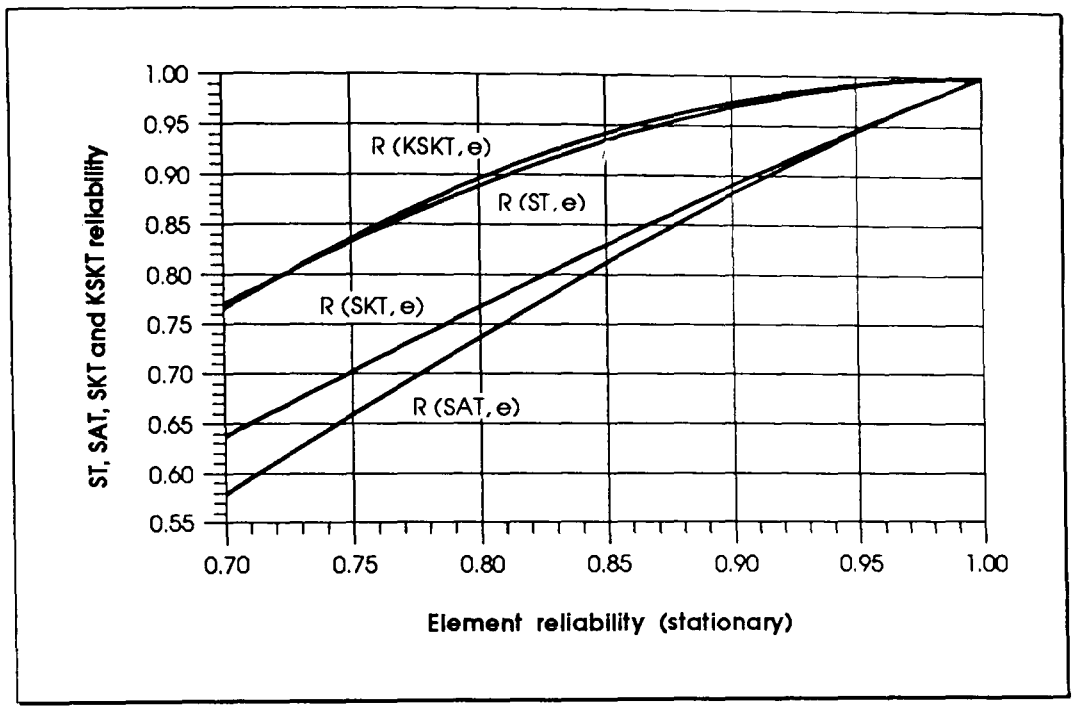


FIGURE 4.27
Source to terminal (source = n_1, terminal = n_4), source to all terminal (source = n_1), source to K-terminal (source = n_1, Kt = {n_2, n_3}) of Fig. 4.3 and K-source to K-terminal (Ks = {n_1, n_2}, Kt = {n_5, n_6}) reliability of Fig. 4.5 considering faults in edges.

The approximation algorithm was applied to a medium size configuration (4x4 rectangular mesh) as illustrated by the following graph (Figure 4.28); in this graph we can observe that it is not necessary to go beyond level 5 or 6 in the computation tree (see section 4.4.3) to obtain a very accurate reliability value.

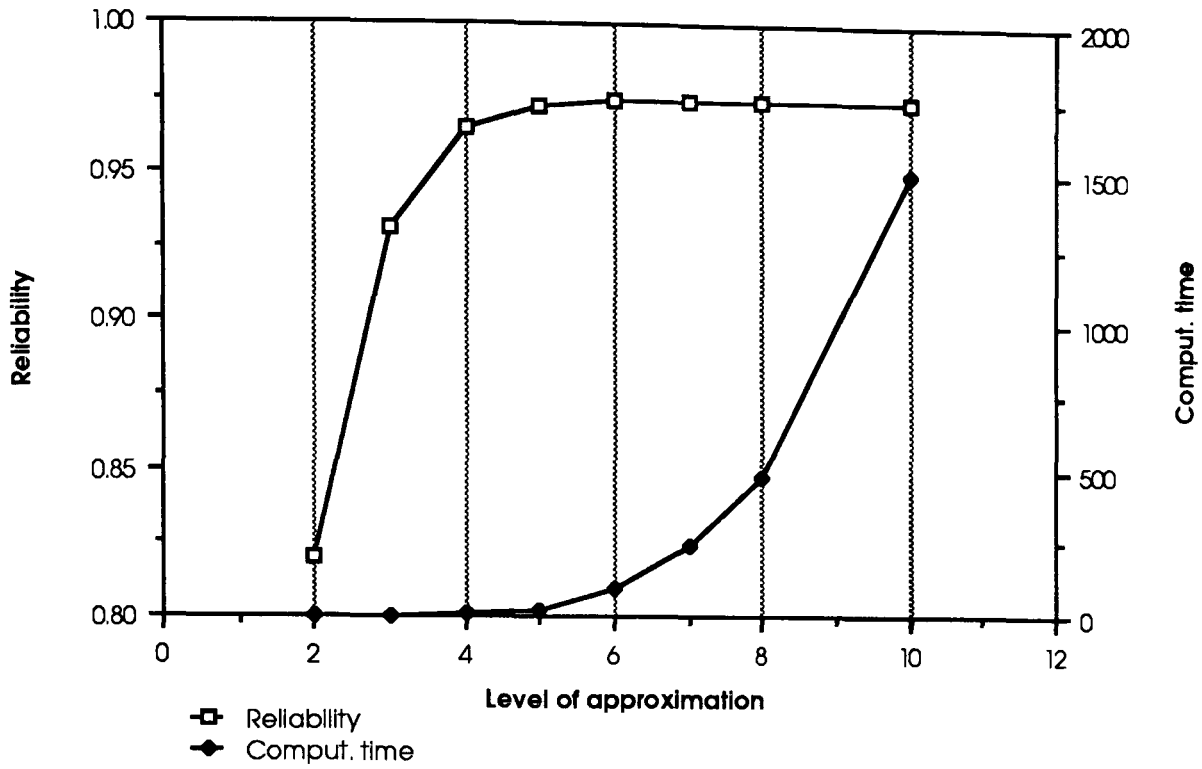


FIGURE 4.28
Double Y graph: Reliability and computation time vs. level of approximation for a 4x4 rectangular mesh graph.

4.4.6.2 Closed systems

Reliability, R (e, t)

As it was seen in section 3.4.4, assuming exponential distribution, the reliability of element i is :

$$R(x_i, t) = \exp[-\lambda_i t] \qquad \ldots (4.4)$$

where λ_i is the failure rate for element i

The time dependent system reliability expression $R(e, t)$ is obtained by substituting $R(x_i, t)$ for p_i in the symbolic expression for $R(e)$ (equations 4.1

and 4.2). Numerical values of $R\ (e,\ t)$ can be obtained by calculating for different values of t in a given interval $[t_1, t_2]$.

Mean time to failure, MTTF (e) was defined as :

$$MTTF\ (e\ ) = \int_0^{\infty} R\ (e,\ t)\ \mathrm{dt} \qquad \ldots (4.5)$$

Since it is not possible in the general case to substitute $MTTF\ (x_i)$ directly from the symbolic expression, it is required to employ numerical integration for this problem. Given the appropriate limits to the integral (for the upper limit, a very high value; and for the lower limit, zero) and enough number of t intervals, a very accurate value of *MTTF* can be derived using Simpson rule for numerical integration [CHU 81].

Given the array of values for reliability at different time :

$R\ (e,\ 0),\ R\ (e,\ t_1\ ),\ \ldots,\ R\ (e,\ t_n\ )$

where n is the number of t intervals,

MTTF is derived using Simpson rule as follows :

$$MTTF\ (e) = \frac{t_n}{3n}\left[R(e,\ 0) + 2\sum_{i=0}^{\frac{n}{2}-1} R(e,\ t_{2i+1}\ ) + 4\sum_{i=1}^{\frac{n}{2}-1} R(e,\ t_{2i}) + R(e,\ t_n\ ) \right] \qquad \ldots (4.6)$$

4.4.6.3 *Repairable systems*

Availability, A (e, t)

The availability of element i, assuming exponential distribution is given by :

$$A\,(x_i, t) = \frac{\mu_i}{\lambda_i + \mu_i} + \frac{\lambda_i}{\lambda_i + \mu_i} \exp\,[-\lambda_i t\,] \qquad \ldots (4.7)$$

The availability expression $A\,(e, t)$ is obtained in the same manner as $R\,(e, t)$ by substituting $A\,(x_i, t)$ for p_i in the symbolic expression.

Steady-state availability, SA (e)

The availability at time ∞ of element i is :

$$SA\,(x_i) = A\,(x_i\,, \infty) = \frac{\mu_i}{\lambda_i + \mu_i} \qquad \ldots (4.8)$$

$SA\,(e)$ is obtained also by substituting $SA\,(x_i)$ for p_i in the symbolic expression, as $R\,(e, t\,)$ and $A\,(e, t\,)$.

Mean time between failures, MTBF (e) is calculated from the equation :

$$MTBF\,(e) = \frac{MTTF\,(e)}{SA\,(e)} \qquad \ldots (4.9)$$

4.4.6.4 Examples

Time dependent measures were computed for the undirected graph of Fig. 4.1, as presented in Figure 4.29 for $R(t)$ and $A(t)$ and in Table 4.3 for *MTTF*, *SA* and *MTBF*.

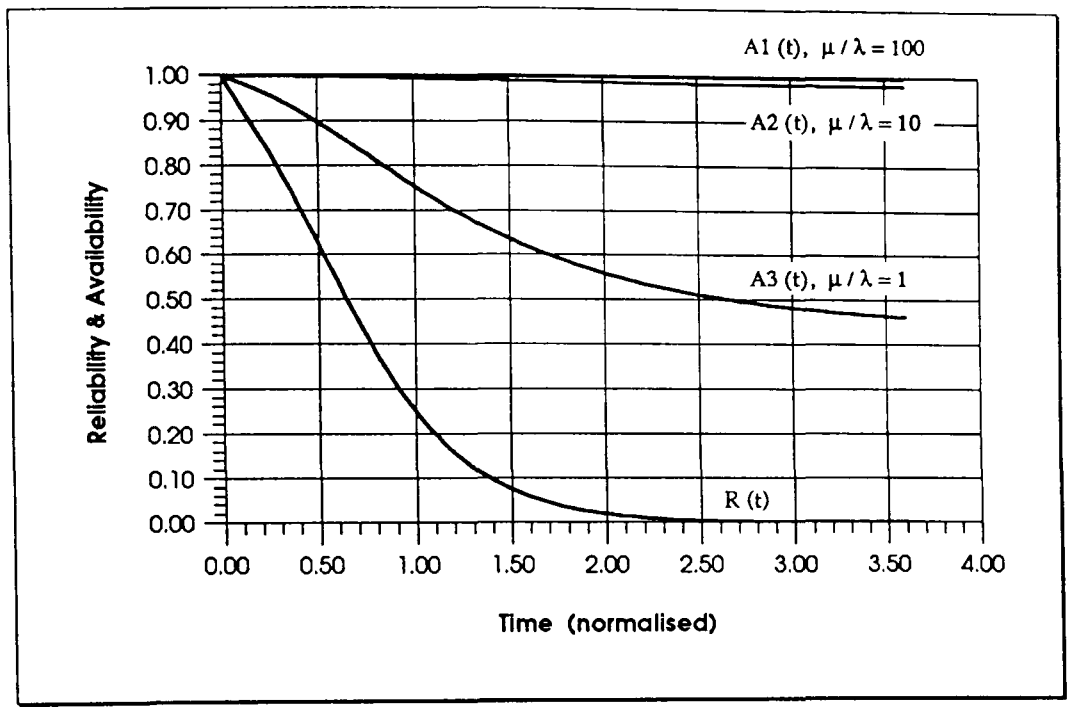


FIGURE 4.29

Time-dependent measures, R(t) and A(t) for Fig.4.1, AT problem; the time units are normalised, i.e. are the product of failure rate (λ) and time; A(t) is obtained for different ratios μ / λ

TABLE 4.3

MTTF, SA and MTBF (graph in Fig.4.1, AT problem) for different ratios μ / λ

μ / λ	MTTF	SA	MTBF
0	716.18	1.0000	716.18
1	--	0.9999	716.18
10	--	0.9998	716.32
100	--	0.9811	730.00
1000	--	0.4375	1636.98

4.4.7 FAULT SIMULATION

As for the deterministic model, faults in nodes and edges have been simulated as described in section 4.3.6. To calculate probabilistic reliability measures for degraded configurations, line (12) in Algorithm 4.10 is replaced by:

(12') Compute probabilistic reliability measures

Figure 4.30 shows an example of stationary reliability when the selected graph configurations have been succesively degraded until they become disconnected.

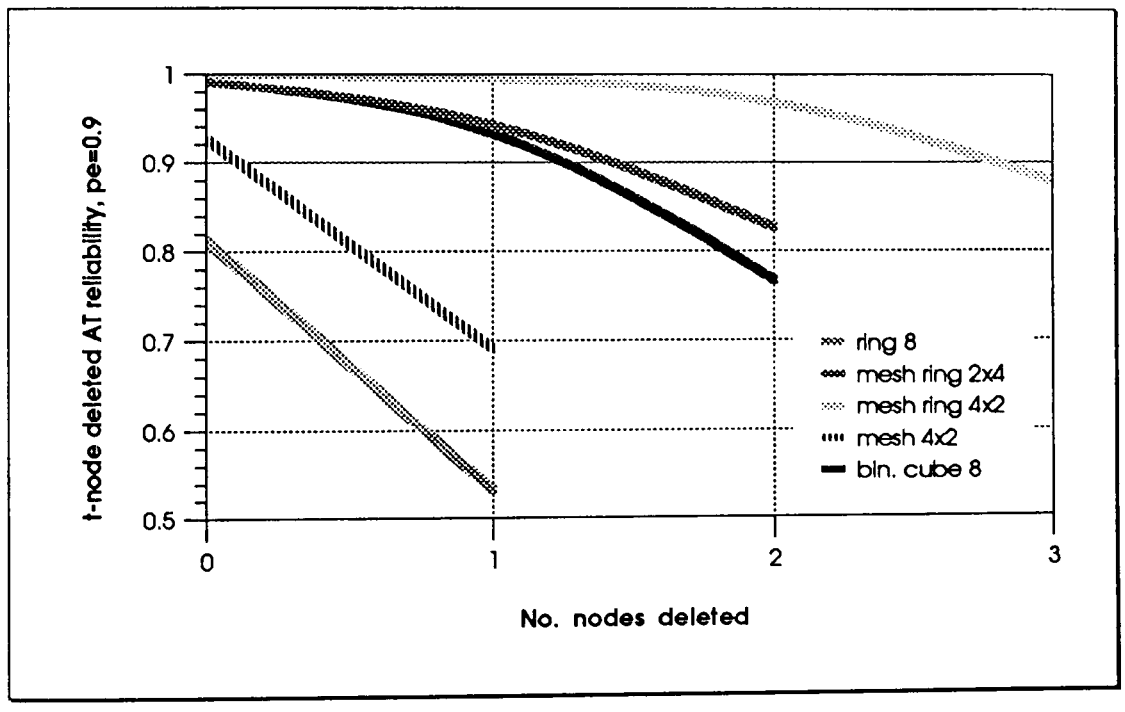


FIGURE 4.30
t-node deleted AT stationary reliability versus number of nodes deleted for edge reliability = 0.9

4.4.8 K-OUT-OF-N PROBLEM

As it was seen in section 3.4.4, a k-out-of-n system is the general model of active redundant systems, where series and parallel systems are particular

cases with k=n and k=1 respectively. Therefore, in the implementation of a reliability model for k-out-of-n systems there are included series and parallel systems.

If equations (3.11) and (3.12) are used directly to calculate R_k, for large n the number of terms obtained is very large and the algorithm is computationally inefficient. Also, the algorithm becomes more complicated when the element reliabilities are different and for calculation of time dependent measures.

Some efficient methods have been presented in the literature for evaluating the reliability of k-out-of-n systems which reduce the number of terms by avoiding the generation of cancelling terms, see for example [LOC 84], [BAR 84], [JAI 85], [RIS 87]. For this model, it has been developed a very efficient algorithm based in the method for network reliability (described in sections 4.4.1 and 4.4.2); the algorithm for k-out-of-n systems uses the same principle of recursive sum of disjoint products where the generation of cancelling terms is avoided and uses the same data structures for cube representation and symbolic expression.

4.4.8.1 Algorithm for Boolean expression

The procedure developed for the derivation of a Boolean expression for evaluation of k-out-of-n system reliability can be summarised as follows (Algorithm 4.14):

procedure *K_out_of_n;*

(1) *Enter and check the initial data for the problem.*
   (a) *Enter problem : k-out-of-n system reliability (KON)*
   (b) *Enter n and k*
   (c) *Check* $1 \leq k \leq n$

(2) *Use symmetry to do the shortest calculation.*
   Because of duality, the probability of success for a k-out-of-n system is the complement of the probability of failure for a (n-k+1)-out-of-n system.
   if $k > \frac{n+1}{2}$ ***then***
   begin
      $k := n - k + 1;$
      $p_i := 1 - p_i$ *for all elements;* *{or a time dependent measure}*
      ct := true *{ct is a Boolean indicator}*
   end
   else *ct := false;*

(3) *Step (1) in proc. GetBooleanExpression (section 4.3.1.2) is modified as follows :*
   case *problem* ***of***
      TT ,AT, KT, ST, SAT, SKT, KSKT : ...
      KON : Find a tree representation (cube) with k working elements (in good state ***(1)*** *or arbitrary state* ***(x)****) from the cube Y of n elements. This cube is obtained by finding the first k elements in state* ***1*** *or* ***x*** *from cube Y and changing those in state* ***x*** *to* ***1****, so a cube of at least k good elements is obtained.*
   end; *{case}*

(4) *Proceed as steps (2) to (5) in algorithm GetBooleanExpression.*

(5) *Step (6) in GetBooleanExpression is modified since it is not required to check for connectedness and it is possible to know beforehand the maximum level of the computation tree for this recursive procedure in order to reduce the number of calculations, which is: n-k.*

(6) *Once a Boolean expression has been obtained, the reliability measure of interest is calculated as for network reliability, but if symmetry was employed to reduce the calculations (ct is true)* R_{sys} *is substituted by* $1 - R_{sys}$ *.* (R_{sys} *is R(e), R(e, t), A(e, t), or SA(e)).*

end; *{ K_out_of_n }*

ALGORITHM 4.14
K-out-of-n system reliability

4.4.8.2 *Computer analysis*

An example of the recursive computation of a 3-out-of-5 system is shown in the following computation tree (Figure 4.31). Each subset of cubes generated ($B's$) at each computation is represented by each of the branches of a previous node in the tree. The $A's$ are the terms of the Boolean expression.

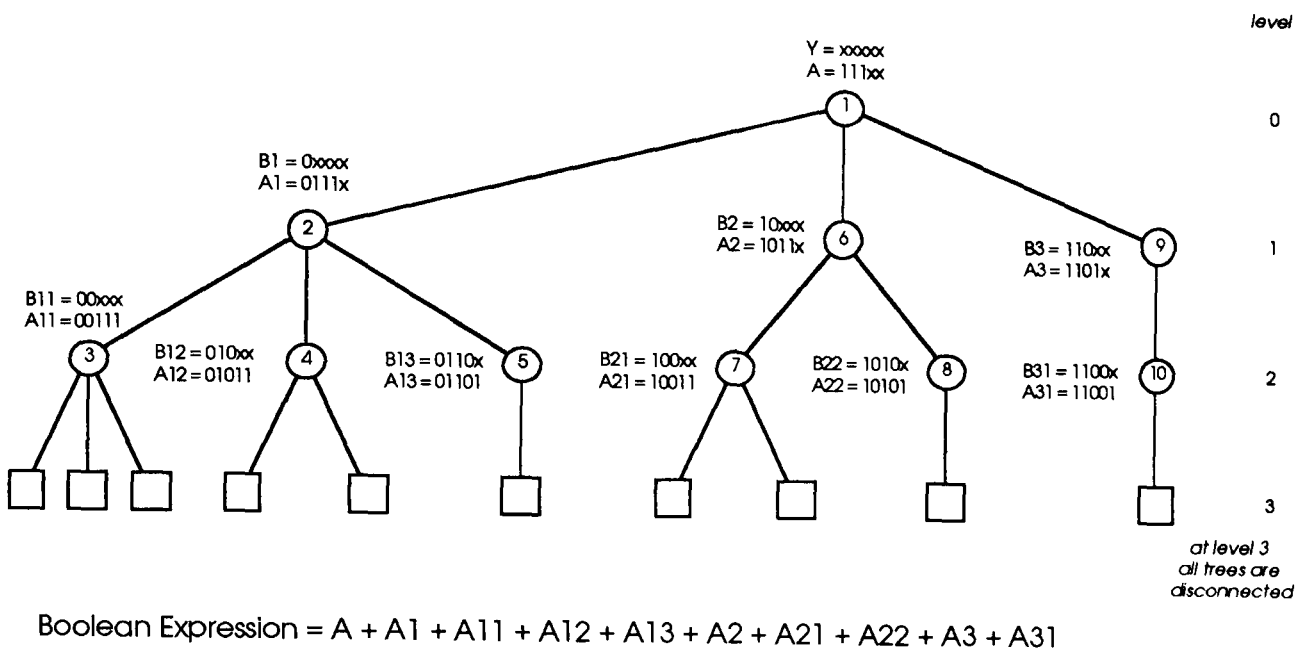


FIGURE 4.31
Computation tree of 3-out-of-5 system

4.4.8.3 *Example*

As an example of partially redundant systems, in a distributed system environment, consider the following problem: k computers are required to execute a given program; to improve its reliability, one, two or more computers (in general r computers) can be added to the set of k computers. This is a k-out-of-n system with $n = k + r$; considering r as the level of redundancy.

For a distributed system with k = 2, 5 and 10, the reliability improvement when increasing the level of redundancy (r) was obtained as it is shown in Figure 4.32 for an element (computer) reliability, p_i = 0.8.

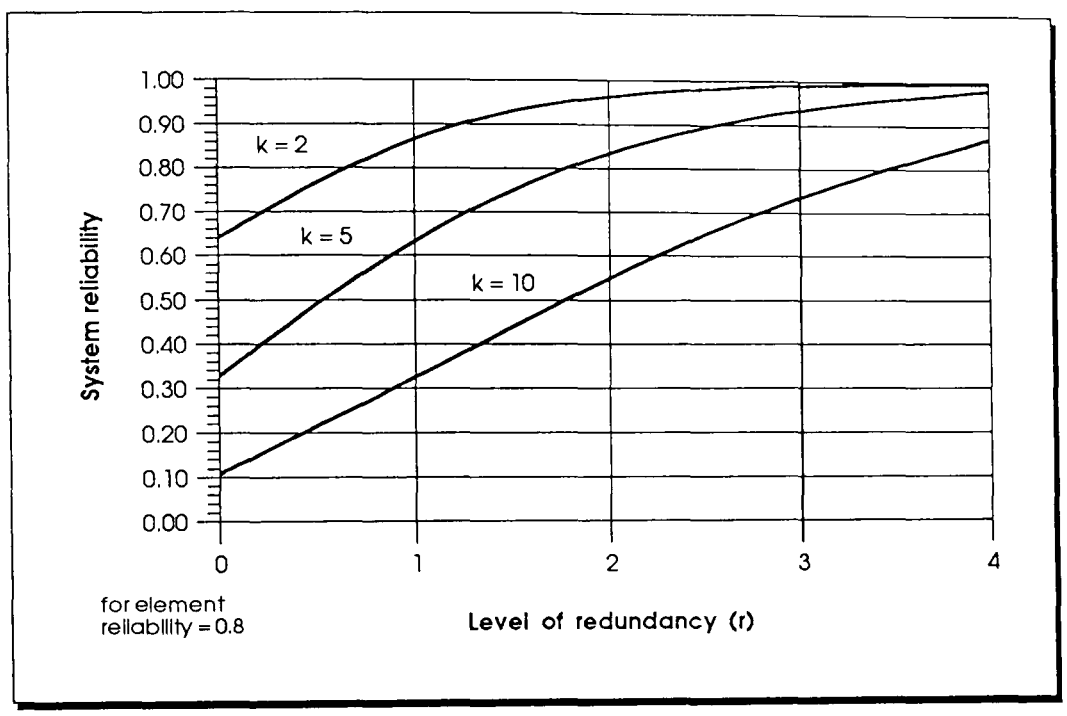


FIGURE 4.32
Example of k-out-of-n reliability, (n = k + r)

Chapter 5

Reliability Modelling of Large Multiprocessor Systems

5.1 INTRODUCTION

Multiprocessor systems have been increasing in size rapidly over the last few years. Many system control functions, routing, performance modelling, reliability modelling, etc. cannot be carried out in a large environment because of prohibitive overheads.

Reliability evaluation of a general multiprocessor network has been proved to be NP-hard to compute [BAL 86], due to the exponential growth of the system state space. An exact evaluation technique on a 'flat' network requires a very large computational effort in both, computation time and memory, which will be prohibitive if the system to evaluate is large.

The idea of decomposing the system structure in a set of smaller subsystems is a viable solution to overcome such limitations. Such

decomposition can be achieved by a m^{th}-level hierarchical clustering of the system.

Two cases have been addressed in this work:

(a) A system has been hierarchically decomposed for the purpose of simplifying control functions, routing, etc.. Reliability is evaluated for such hierarchical network, or

(b) It is desired only to simplify reliability evaluation of a large flat network; in this case by imposing a decomposable hierarchical structure we can obtain an approximation (lower bound) for each of the various reliability measures.

In both cases we can think of the entire network as a tree of hierarchies, in which each node at a higher level is made up of one or more nodes from lower levels. Once a hierarchy exists we can use a hierarchical solution to the problem of reliability modelling. The basic strategy can proceed in the same manner for both cases:

(1) To obtain a hierarchical structure by a m^{th}-level hierarchical clustering of the graph representing the system.

(2) To evaluate reliability for such structure.

In the rest of this chapter is presented a detailed description of the hierarchical clustering method and the subsequent hierarchical reliability evaluation of the system, as well as the results obtained by applying this method to some multiprocessor configurations.

5.2 HIERARCHICAL CLUSTERING

5.2.1 DEFINITIONS

A *cluster* is defined as a group of objects, entities, elements, etc. connected together according to some rules or relations. The goal of the clustering problem is to find groups containing objects most homogeneous (similar) within these groups, while at the same time the groups are heterogeneous (dissimilar) between themselves as much as possible. The homogeneity or similarity is measured by using a set of rules called the *similarity criteria*. Each criterion could be *qualitative* (e.g. small, tall, etc.) or *quantitative* (i.e. some kind of numerical measure). Clustering has been used mainly for clasiffication purposes of sets of unclassified data leading to a multitude of methods [EVE 80].

5.2.2 REVIEW OF CLUSTERING TECHNIQUES

Clustering techniques have been classified roughly into five types: hierarchical, optimisation, density, clumping and other techniques. For this work, we are concerned basically with hierarchical techniques where the data are not grouped all in only one step, rather they are grouped progressively into steps.

Essentially, hierarchical techniques may be subdivided into *agglomerative* (bottom-up) methods which proceed by a series of succesive fusions of the n objects into groups (classes, clusters, etc.), and *divisive* (top-down) methods which partition the set of n objects successively into finer partitions. Both techniques may be represented by inverted tree structures which are two dimensional diagrams illustrating the fusions or divisions that have been made at each successive step of the procedure, the only

difference between the two methods is the direction. A tree representation of agglomerative and divisive clustering is shown in Figure 5.1.

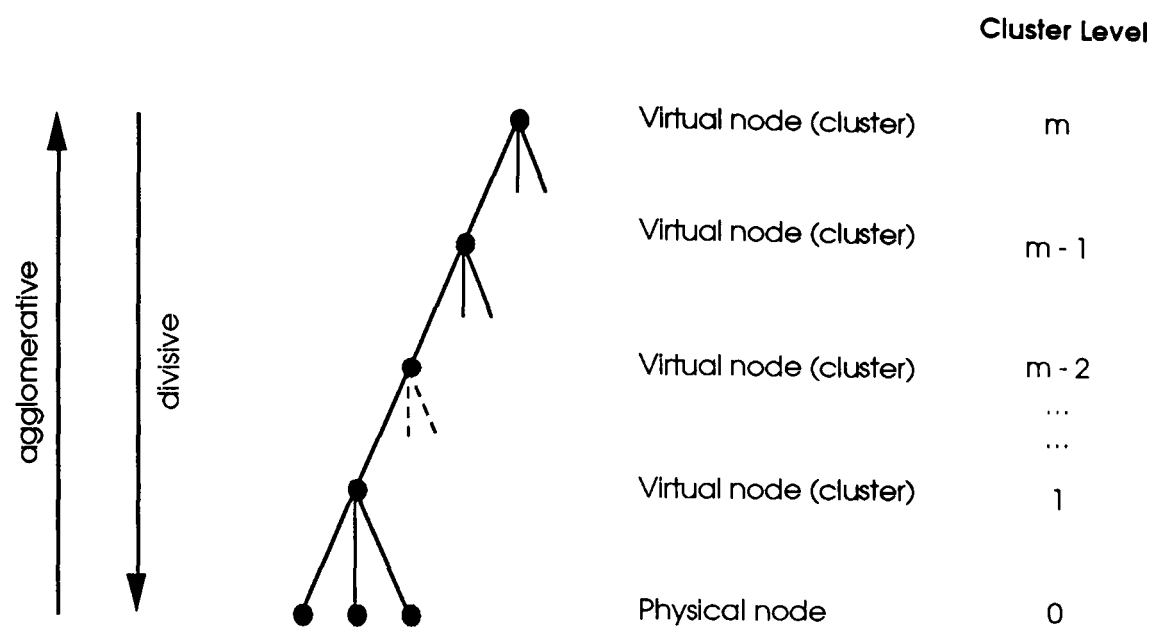


FIGURE 5.1
Tree representation of hierarchical clustering

The most commonly used methods, like single linkage, complete linkage, Ward's method, etc., are of hierarchical type and agglomerative. These methods follow the general procedure of successively pairing off the most similar objects and then replacing them by one representative, using in most cases a similarity criteria based in the smallest distance between two elements. This procedure always leads to the creation of a degree two (binary) tree, which is known to have the maximum height among all trees. The disadvantage of these methods is that the time required to execute the clustering algorithm is maximum if a binary tree is to be generated; this time succesively decreases with the degree of the tree [RAM 86].

Another disadvantage of these standard clustering algorithms is that they are suited to use with the distance matrix between all elements, requiring the recalculation of the matrix at each step of the algorithm. Therefore they can handle efficiently only a small number of objects, since

the dimension of the matrix grows proportionally to the square of the number of nodes.

From these drawbacks it is concluded that standard clustering techniques are not suitable for use in clustering of large multiprocessor networks. In [RAM 86] is presented an efficient heuristic algorithm designed for the clustering of computer networks which is suitable to adapt for our hierarchical reliability model.

5.2.3 *GENERAL MODEL*

The general model consists of objects connected by relations where clustering is done based on these relations. The basic graph model for multiprocessor systems described in section 3.4.1 can be used, where the objects are modeled as weighted nodes and the relationships between them are modelled by weighted edges (interconnection network). The weights represent the strength of the relation, in this case a reliability measure. This model is adequate enough for the clustering problem.

5.2.3.1 *Solution objectives*

Solving the clustering problem involves achieving one or more objectives. For a hierarchical network can be: to minimise communication cost, connectivity and link-failure resilience, balanced clustering structures, minimise routing tables, etc. For the approximate reliability evaluation of flat multiprocessor systems, the problem is to find an optimal clustering structure in such a way that the error in the reliability expression (or values) obtained is minimised compared to the exact expression (or values).

As it was observed with the deterministic reliability model in section 4.3, reliability factors like edge and node connectivity decrease with an increase in the diameter of the network. Therefore, intuitively, clusters should be chosen as to correspond to highly connected sets of nodes which result in a small diameter. Also, since reliability evaluation is dependent on the communication paths or trees internal to the cluster, the cluster subnetwork must contain the shortest paths between its nodes in that cluster.

The following factors have to be taken into account to find an optimal clustering structure:

- Appropriate similarity criteria
- Optimum number of clusters
- Optimum number of nodes constituting each cluster
- Optimum number of hierarchical levels

5.2.4 METHOD

As discused in section 5.2.2 there are basically two different methods of solving the hierarchical clustering problem: the divisive and the agglomerative; the former method uses graph partitioning and has been found to be NP-complete [RAM 86]. The agglomerative method starts with the original graph in which each node represents a single element (0^{th}-level cluster). These elements are grouped and merged to form 1^{st}-level clusters. Every such cluster is then collapsed and replaced by a single representative node. The process is repeated, 1^{st}-level clusters are also grouped and merged into 2^{nd}-level clusters and so on, until the graph is reduced to a single node at the top level (m^{th}-level cluster).

The agglomerative approach has been adopted for our clustering algorithm. The basic procedure and the heuristic factors considered are explained in the following subsections.

Along with the hierarchical clustering of nodes, we must select special type of nodes: the exchange nodes or *gates* for all clusters at all levels. The function of the gate in a cluster is to represent the cluster and to handle the communication between the set of nodes in that cluster and those outside in another clusters. $(k+1)^{st}$-level gates are selected among the k^{th}-level gates at any level.

5.2.4.1 *Assumptions*

The following is assumed about the communication paths between the nodes [KLE 80] :

(a) Communication between nodes in the same cluster, at any level, only take paths which are internal to that cluster (paths contained in the local subnetwork).

(b) Communication between nodes in different k^{th}-level clusters, but which belong to the same $(k+1)^{st}$-level cluster is directed via its local subnetwork to a $(k+1)^{st}$-level gate of the originating cluster; then it takes the $(k+1)^{st}$-layer subnetwork to reach a $(k+1)^{st}$-level gate of the destination cluster, then its local subnetwork is used to finally reach the destination node.

A k^{th}-layer subnetwork is defined as a network connecting k^{th}-level gates which belong to the same k^{th}-level cluster. Figure 5.2 illustrates the preceding definitions for a two-level hierarchical network. Clustering leads to the tree representation shown in Figure 5.3.

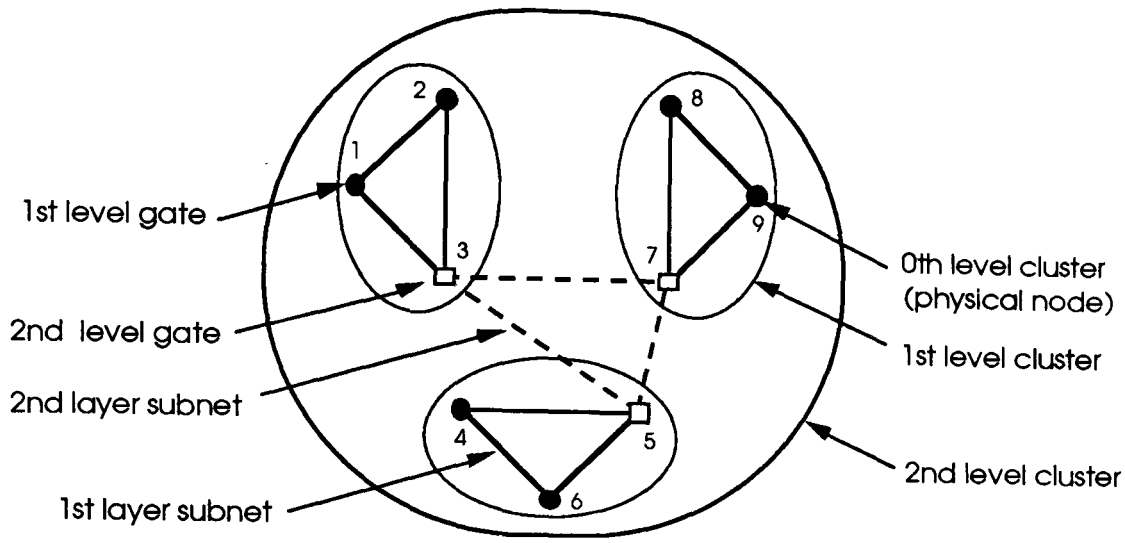


FIGURE 5.2
Two-level hierarchical network

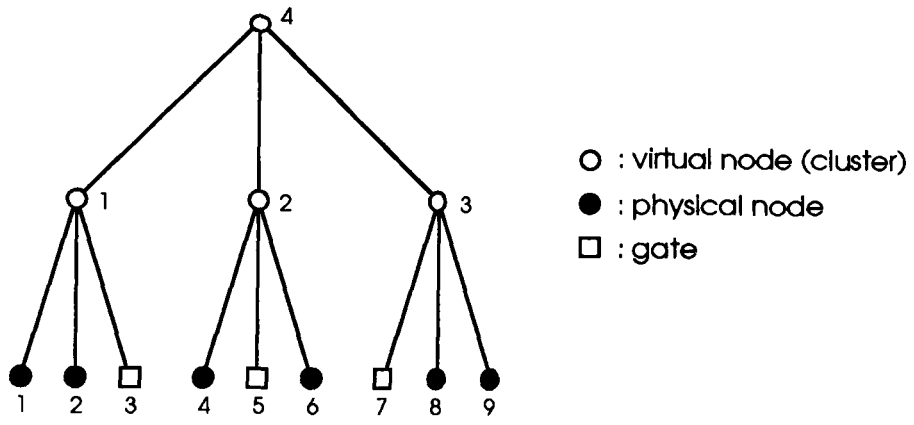


FIGURE 5.3
Tree representation of a two-level hierarchical network

5.2.4.2 Basic procedure

The procedure for our agglomerative clustering method follows four basic steps:

(1) The nodes in the current graph are sorted into a list and the first one is chosen (centre).

(2) The neighbours of the selected node (centre) are sorted as well in some manner to form a list and some of them are chosen to create a cluster together with the centre.

(3) A exchange node or gate is selected from the nodes that constitute the cluster according to some criteria.

(4) The selected nodes (centre and selected neighbours) are merged into a single node, thus reducing the size of the graph. This single node is a virtual node, which is the representative at the next level of clustering of all the nodes (physical or virtual) that are its constituents.

These four steps describe a single cluster creation. They are repeated in sequence, firstly until all nodes in the current graph have been clustered, completing one level of clustering, and finally until the graph is reduced to a single node which is the top level cluster. A cluster created in such manner can be temporary if its weight has not reached the maximum weight and more nodes can be added to it. It is permanent if it has not more capacity for growing because it has reached the maximum weight or there are no more nodes which can be merged to it. Figure 5.4 shows an example of the above procedure for the network of Fig. 5.2.

This agglomerative method does not use global topology information since each node has information about its neighbours only; thus this approach is inherently heuristic.

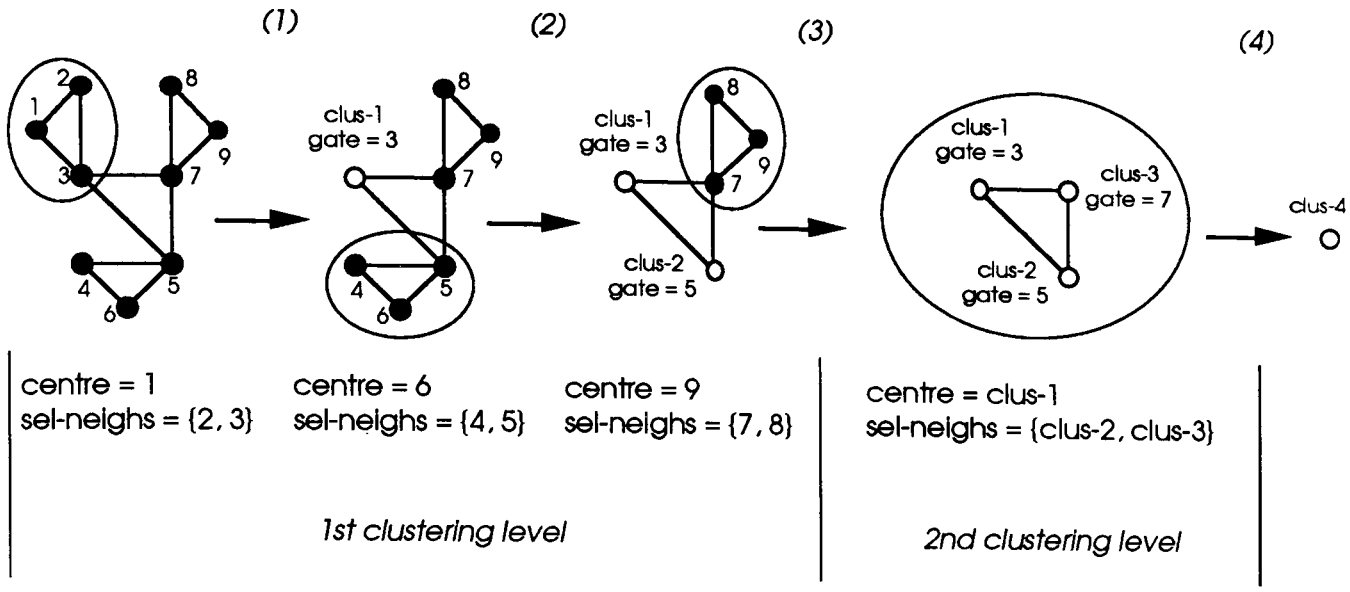


FIGURE 5.4
Example of the basic clustering procedure

5.2.4.3 Factors for clustering

The selection of centres and neighbours to be merged are very important for achieving the desired objectives. There are three main factors to be considered in agglomerative clustering [RAM 86]:

(1) Sorting of nodes.

Two parameters that can be used as keys to sort the nodes in order to select a centre are:

(a) *Degree*, or number of incident edges to a node

(b) *Weight*, or number of nodes merged to create a virtual node at the current clustering level (physical nodes have weight=1)

Both could be used simultaneously, with one being used as the primary key and the other as the secondary key. Sorting is done in a non-decreasing magnitude order; thus the node with smallest degree and weight is selected as the centre.

To sort the neighbours of a centre, another parameter is employed in addition:

(c) *Strength* between neighbour and centre, which is the number of parallel edges between each neighbour and its centre.

Sorting for parameters (a) and (b) is done in a non-decreasing magnitude order and for parameter (c) in a non-increasing order. Thus, neighbours with the smallest degree, smallest weight and largest strenght are selected first.

(2) Binary/multiple merging.

As it was discussed before in section 5.2.2, binary merging in which only one neighbour is selected to be merged with the centre every time, leads to the creation of a binary tree; in contrast, in multiple merging the aim is to select as many neighbours as possible (but not exceding the maximum size allowed to each cluster) leading to the creation of higher order trees; thus reducing the number of steps. Therefore, for our model is employed multiple merging.

(3) Centre selection.

There are devised three different approaches of centre selection: the *aggressive*, the *moderate* and the *pacific*.

In the aggressive approach, once a centre is chosen it is retained as the centre as long as its cluster can grow, but this can lead to uneven sized clusters. In the pacific approach, at every step a new centre is chosen among the candidate nodes. In the moderate approach, the centre chosen in the previous step is also put in the list of possible candidates for the next step. Its being chosen as centre again depends on whether it comes to the head of the list after subsequent sorting.

Based in the results obtained in [RAM 86] where the pacific and moderate methods gave better balanced structures, it has been chosen the moderate approach for centre selection in our algorithm.

5.2.5 *DESCRIPTION OF THE ALGORITHM*

The clustering algorithm is described in the following, as well as its most relevant local procedures. This description is detailed enough to specify all the steps involved in the clustering process, while at the same time language implementation details are not specified.

5.2.5.1 *Main procedure*

The main procedure of the clustering algorithm is described in pseudo-code in Algorithm 5.1, where *cgraph = (cnodes, cedges)* is the current graph which describes the clustering at any step of the algorithm. Initially *cgraph* represents the entire network.

Following is the terminology used:

cgraph = the current graph
cnodes = the set of nodes of the current graph
cedges = the set of interconnection edges of the current graph
cgraphrecord = **record**
    clustered : Boolean { indicates if a node is clustered or not }
    weight : integer { current weight of a node }
    degree: integer { current degree of a node }
end;

clusterlevel = the current level of clustering

candset = the set of candidates to chose the next centre from

neighset = the set of neighbours of the centre

selecset = the set of nodes selected to be merged with the centre

centre = the node chosen as the next centre

key1, key2 = each can be either the weight or the degree of a node

key3 = the strenght between centre and neighbour

htree = hierarchical tree representation of the clustering structure

clusterset = nodes constituting a cluster

```
procedure Clustering;
   cgraph = graph;                          { initially cgraph is the entire ... }
   cnodes = n;                              { ... network, graph = (n, e) }
   cedges = e;
   clusterlevel = 0;
   while cnodes > 1 do                      { start a new clustering level }
      for node = 1 to cnodes do
         with cgraphrecord[node] do
            clustered = false;
            weight = 1;
         end; {with}
      end; {for}
      clusterlevel = clusterlevel + 1;
      while Not_all_clustered (cnodes) do   { while not all nodes have been ... }
                                            { ... clustered at the current level }
         Obtain_degree (cnodes, degree);    { obtain adjacent nodes of each node }
         Obtain_candidate_set (candset);    { obtain all nodes with clustered=false }
         Sort (candset, key1, key2);        { return candset sorted }
         centre = first (candset);
         Obtain_neigh_set (centre, neighset);   { obtain adjacent nodes to centre ... }
                                                { ... not already clustered }
         Sort (neighset, key1, key2, key3);     { return neighset sorted }
         Select_neighbours (selectset);
         if |selectset| ≥1 then                 { if some neighs. selected }
            Record_cluster (htree);             { record a new cluster or update a ... }
                                                { ... cluster; select gate }
            Reduce_graph (cgraph);              { merge centre and selected neighs ... }
                                                { ... in one coalesced node (cluster) }
         end {if then}
         else                                   { if no neighs. selected }
            cgraphrecord[centre].clustered = true;
            if cgraphrecord[centre].weight = 1 then
               Record_cluster (htree);          { record a one node cluster }
         end; {else}
      end; { while Not_all_clustered }
   end; { while cnodes > 1 }
end; { Clustering }
```

ALGORITHM 5.1
Main procedure of clustering algorithm

The local procedures *Sort*, *Record_cluster* and *Reduce_graph* are described in detail in the following subsections.

5.2.5.2 *Sort*

A general sorting method has a complexity of $O(n \log_2 n)$. Since in our problem, any key in which sorting is to be done lies in the range $0 \leq \text{key} \leq n$ we can use a linear time sorting algorithm like radix distribution sort [REI 77]. This will reduce the complexity to $O(kn^2)$, where k is a constant.

Let $node_1, node_2, \ldots, node_m$ be a list of nodes in the range 1 to n. The list can be sorted for one key in the following manner.

(1) Initialise n empty queues, each queue represents a pile.

(2) Scan the list of nodes, placing the node with the key value v in the v^{th} pile.

(3) Concatenate the queues to obtain the sorted list.

Assume that a link field $link_i$ is associated with each $node_i$ and is used to link the nodes to form an input queue Q which is the list of nodes to be sorted. This field is used also to link the nodes into the queues that represent the piles Q_1 - Q_n. After the nodes have been distributed into piles, the queues representing those piles are concatenated together to reform the queue Q but now with the nodes sorted in non-decreasing order, starting with the front of queue Q. The outline of this sorting algorithm is shown in Algorithm 5.2.

```
procedure Sort;
   Input queue (Q);
   for j = 1 to num_keys do                          { for each key }
      Initialise queues Q0-Qn to be empty;
      while Q not empty do                           { distribute in piles}
         nodei = next node in Q;
         case key[j] of
            weight : v = weight (nodei );                          {non-decreasing order}
            degree : v = degree(nodei );                           {non-decreasing order}
            strenght : v = n - num_parallel_edges(nodei );         {non-increasing order}
         end; {case}
         Add nodei to Qv;
      end; {while}
      Concatenate queues Q0-Qn together to form the sorted queue Q;
   end; {for}
end; { Sort }
```

ALGORITHM 5.2
Radix distribution sort

5.2.5.3 Record_cluster

This procedure creates the hierarchical tree representation of the clustering structure and selects a gate for each cluster. Two cases can be presented:

(a) To update a cluster : add a node to a temporary cluster

(b) To create a new cluster

The data structure used for the hierarchical tree (*htree*) is an array of dynamic linked lists. Each element of the array, which corresponds to one cluster, has an associated dynamic list of the nodes forming such cluster. An example is shown in Figure 5.5.

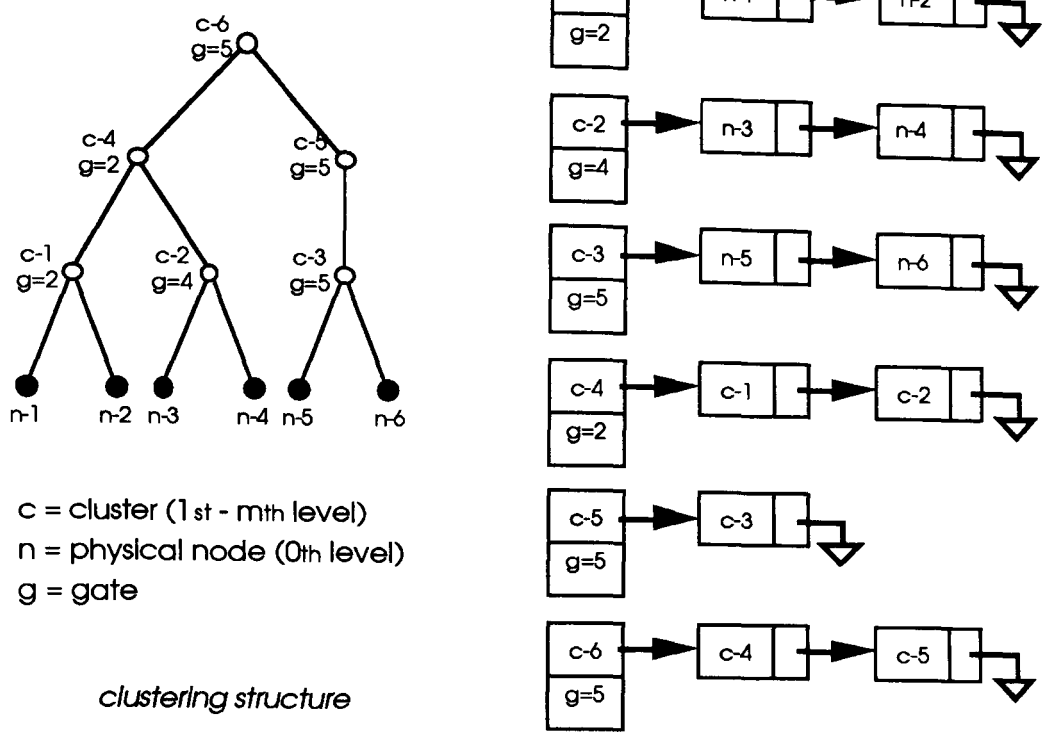


FIGURE 5.5
Data structure of htree

Gate selection

Each time a cluster is created or updated, a cluster representative or gate is chosen among the nodes constituting the cluster.

It has been used a simple criteria for gate selection: the node with more 'external' adjacent nodes (nodes in other clusters), since such node (gate) has to handle the communication between all nodes within the cluster to nodes in other clusters.

5.2.5.4 Reduce_graph

After a permanent or temporary cluster has been created or updated, the nodes constituting such cluster are merged into a single node, thus reducing the size of the graph. A description of the merging procedure is presented in

Algorithm 5.3; the basic steps involved are shown graphically in the example of Figure 5.6.

```
procedure Reduce_graph
   Select node identifier of cluster : for simplicity its chosen the centre;
   for i=1 to num_selected_nodes do                { for each node forming the cluster }
      Obtain neighbours of node_i (neigh_set);
      for j=1 to num_neigh do                      { for each neighbour of node_i }
         if neigh_j (node_i ) in selected set then { remove internal edges between ...}
            Remove edge (neigh_j, node_i);         { ... nodes in cluster }
         else if node_i <> centre then             { remove external edges ... }
            Remove edge (neigh_j, node_i);         { ... if node_i is not the centre }
            if neigh_j is external neighbour of centre then
               Update capacity_edge (neigh_j, centre)
            else
               Add edge (neigh_j, centre);
               Include neigh_j as new neighbour of centre
            end; {else}
         end; {for j}
      end; {for i}
      Update number of nodes and edges of current graph;
      Obtain new weight of centre = Σ weights of coalesced nodes
end; {Reduce_graph}
```

ALGORITHM 5.3
Reduce_graph (Merging of nodes)

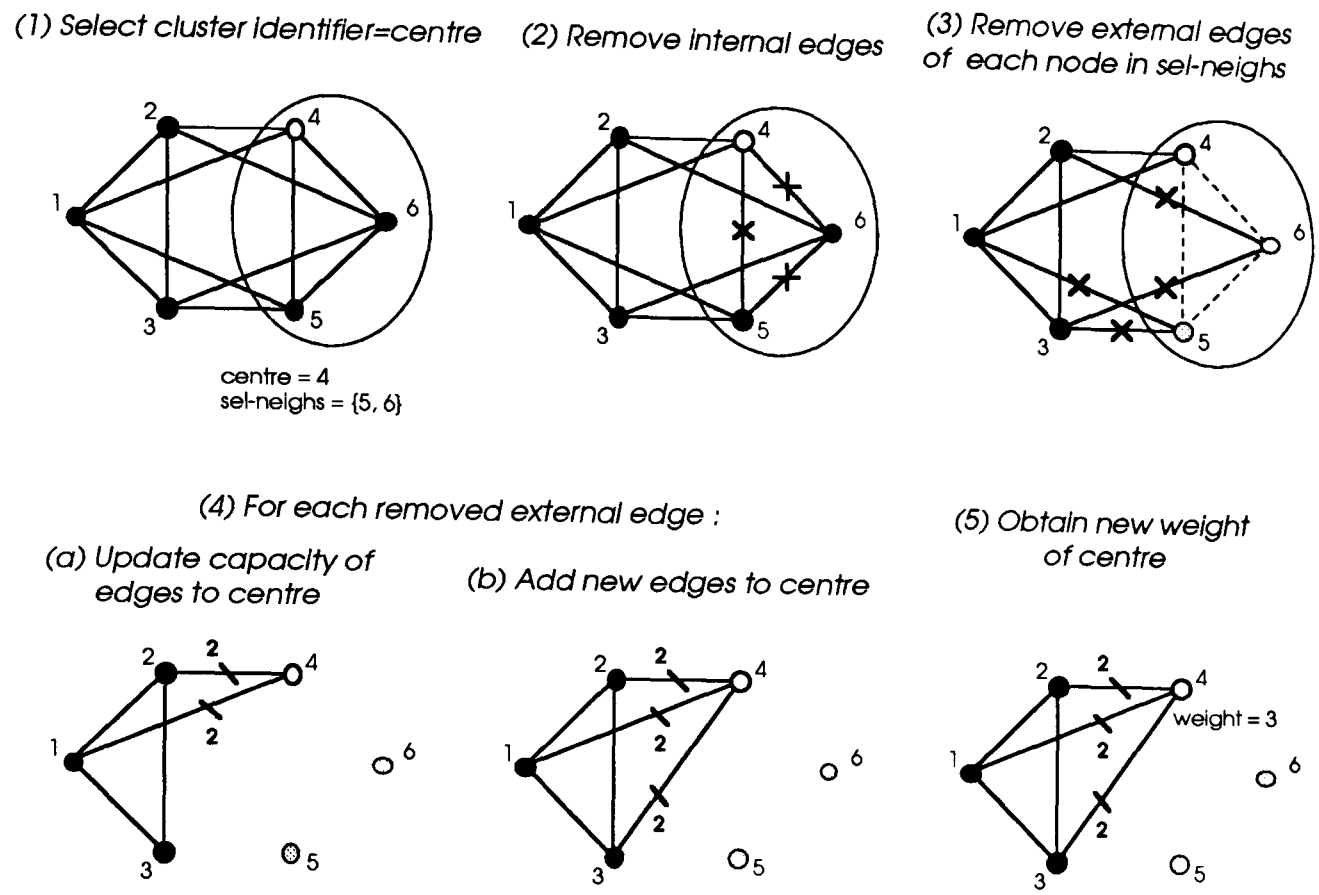


FIGURE 5.6
Example of the merging of nodes to reduce the graph

5.3 HIERARCHICAL RELIABILITY MODEL

Reliability evaluation methods using a hierarchical approach to obtain an approximation of the system reliability have been suggested by [SOI 85] and [MAN 87] but only for overall (AT) reliability evaluation of flat computer networks. In this work it is intended to solve for the various reliability problems defined in section 3.4.2 and for both cases: reliability approximation in a flat network and reliability evaluation of a hierarchical network, using a general methodology.

After decomposing the network in a hierarchical structure, reliability evaluation can proceed as follows.

(1) As 1^{st}-level clusters are composed of 0^{th}-level clusters, i.e. physical nodes, the appropriate reliability measure, relevant to the problem, for the 1^{st}-level clusters can be calculated using the general procedure (RM) described in chapter 4. These clusters are managed as independent subgraphs of the current graph.

(2) Each 1^{st}-level cluster is treated as a new virtual node with its reliability as calculated in step 1. Reliability of 2^{nd}-level clusters is obtained again by using RM on the new graphs formed by these virtual nodes.

(3) Step 2 is repeated for the subsequent levels until the reliability of the m^{th}-level cluster is obtained which is the system reliability.

This method has been called I-hierarchical reliability model (IHRM). In Figure 5.7 is illustrated an example for calculating AT reliability.

5.3.1 IHRM METHOD

The IHRM model has been implemented for reliability evaluation of the various problems. Particularly, for each of the different reliability problems is considered the following :

(a) ST, TT, SKT or KT with the nodes in the same local cluster :

- obtain the corresponding shortest paths or Steiner trees using only the local subnetwork.

(b) ST or TT with the nodes in different clusters :

(1) obtain ST or TT reliability for the 1^{st}--level source cluster from source node to the corresponding local gate and ST or TT reliability for the 1^{st}--level terminal cluster from the local gate to terminal node.

(2) repeat the above procedure for the subsequent levels until ST or TT reliability of the m^{th}--level cluster is obtained.

(c) SAT and AT :

(1) obtain SAT or AT reliability for each of the 1^{st}--level clusters using the spanning trees on the 1^{st}--layer subnetworks.

(2) repeat the above procedure for the subsequent levels until SAT or AT reliability of the m^{th}--level cluster is obtained.

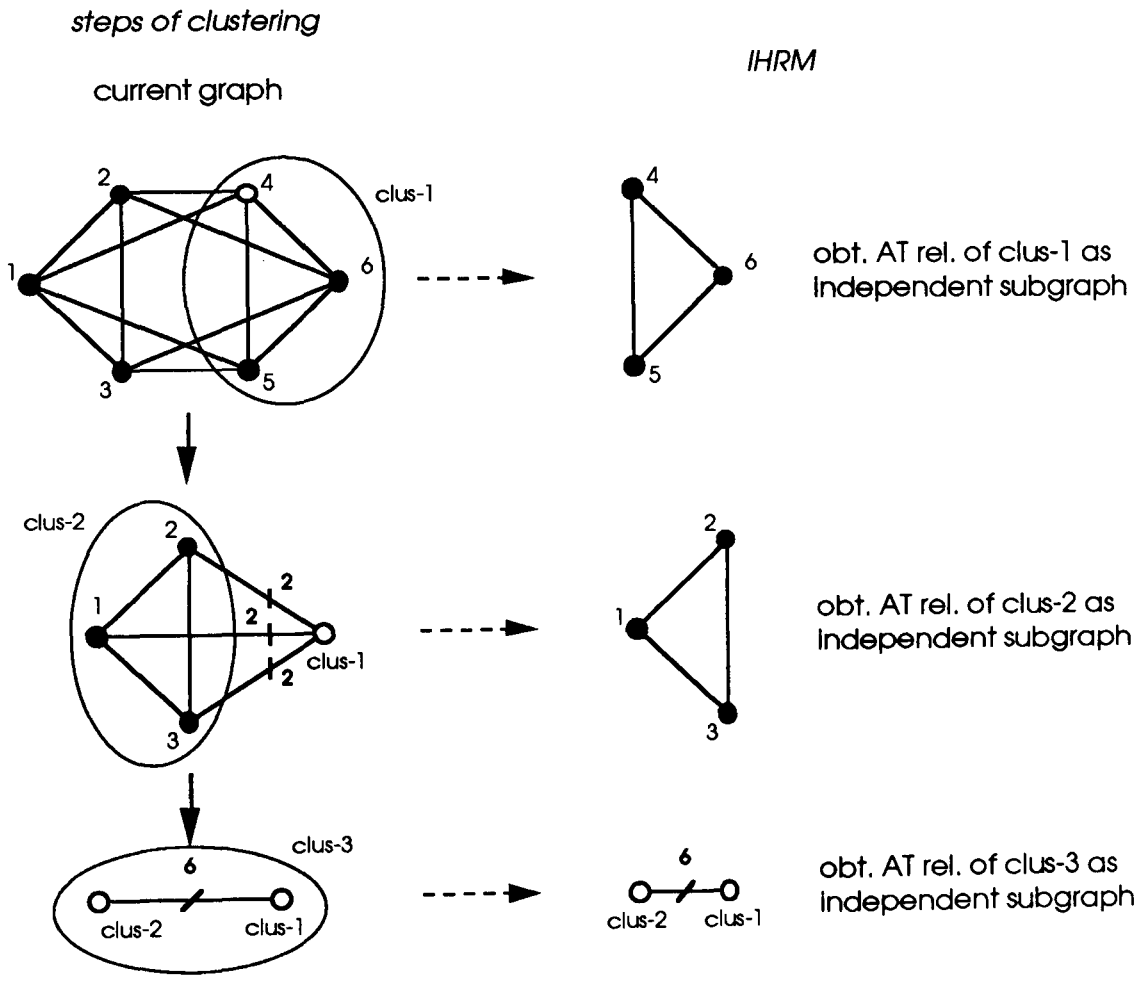


FIGURE 5.7
Example of AT reliability using IHRM

(d) SKT and KT in different clusters :

(1) obtain SKT or KT reliability for the corresponding 1^{st}--level clusters using the Steiner trees on the 1^{st}--layer subnetworks.

(2) repeat the above procedure for the subsequent levels until SKT or KT reliability of the m^{th}--level cluster is obtained.

The hierarchical paths from $(k\text{-}1)^{st}$-level source cluster to k^{th}-level local gate and from k^{th}-level local gate to $(k\text{-}1)^{st}$-level terminal cluster (using the corresponding k^{th}-layer subnetwork) are used for all these procedures.

The basic structure of the IHRM method is presented in Algorithm 5.4. It is assumed that a graph configuration representing the system has been already selected, as well as the problem, class and reliability measures to solve. In the set *problem_set* are contained the nodes concerning to the problem, i.e. all nodes for SAT or AT, k-node set for SKT or KT, and source and terminal nodes for ST or TT.

```
procedure IHRM;
   Hierarchical clustering (graph, htree);     { obtain the hierar. structure in htree }
   current_graph = graph;
   for i=1 to num_clusters do
      if ( | cluster_i | > 1) and ((cluster_i ∩ problem_set) <> ∅ ) then
         Obtain independent subgraph (current_graph, cluster_i , i_subgraph);
         RM (rel_problem, rel_class, rel_measure, i_subgraph, reliab_i);
         Reduce_graph (current_graph);
         Set reliab_i for new collapsed node;
      end; {if}
   end; {for}
   system_reliability = reliab_i
end; {IHRM}
```

ALGORITHM 5.4
I-hierarchical reliability model

5.3.2 *KHRM METHOD*

In addition to the IHRM procedure, another hierarchical method is suggested for reliability evaluation. We can solve for the AT problem by using the set of nodes of each cluster as a k-node subset of the current graph (KT problem) at each step of clustering. Since the graph is reduced in size at every step of the clustering process, the reliability calculation is simplified.

This method does not use the proper hierarchical routing through the gates and local subnetworks as described in section 5.2.4, so it is useful only to obtain a better approximation of system reliability in a flat network. A similar procedure is applied to the KT and TT problems, using only the appropriate clusters which have nodes belonging to the problem.

The procedure developed has been called the K-hierarchical reliability model (KHRM) and is described in Algorithm 5.5. The same considerations as for the IHRM model are assumed.

```
procedure KHRM;
   Hierarchical clustering (graph, htree);     { obtain the hierar. structure in htree }
   current_graph = graph;
   for i=1 to num_clusters do
      if ( |cluster_i| > 1) and ((cluster_i ∩ problem_set) <> ∅ ) then
         Obtain k-subset ⊇ cluster_i ;
         RM (KT, rel_class, rel_measure, i_subgraph, reliab_i);
         Reduce_graph (current_graph);
         Set reliab_i for new collapsed node;
      end; {if}
   end; {for}
   system_reliability = reliab_i
end; {KHRM}
```

ALGORITHM 5.5
K-hierarchical reliability model

5.4 EXAMPLES

Our hierarchical model has been tested in some graph configurations to demonstrate it and to evaluate its implementation, comparing results against the exact and the other approximation method described in section 4.4.3.

For these examples it is assumed for simplicity that nodes are perfectly reliable and all edges have the same reliability. It is calculated the numerical value of stationary system reliability with edge reliability varying in a wide range from 0.6 to 1.0, although in practical systems each edge is expected to have a value of reliability higher than 0.9.

5.4.1 MESHED RING 3X2

Our hierarchical algorithm was first tested in a 3x2 undirected meshed ring shown in figure 5.8a. In this example it is calculated the overall (AT)

reliability when (a) the exact technique is used, (b) IHRM and KHRM are used with clusters chosen to be constituted for a maximum of three nodes, thus creating a two-level hierarchical tree (figure 5.8b), and (c) with clusters chosen to be maximum of two nodes, creating a three-level hierarchical tree (figure 5.8c). The error percentage in the approximation is calculated by,

$$E_r = \frac{R_{ex} - R_{ap}}{R_{ex}} \times 100 \qquad \ldots (5.1)$$

Figures 5.9 and 5.10 show a plot of AT as a function of edge reliability for IHRM and KHRM respectively; Figures 5.11 and 5.12 show the error percentage also as a function of edge reliability. Figure 5.13 is a comparison of computer time and the number of cubes generated which is proportional to the memory required for the various methods.

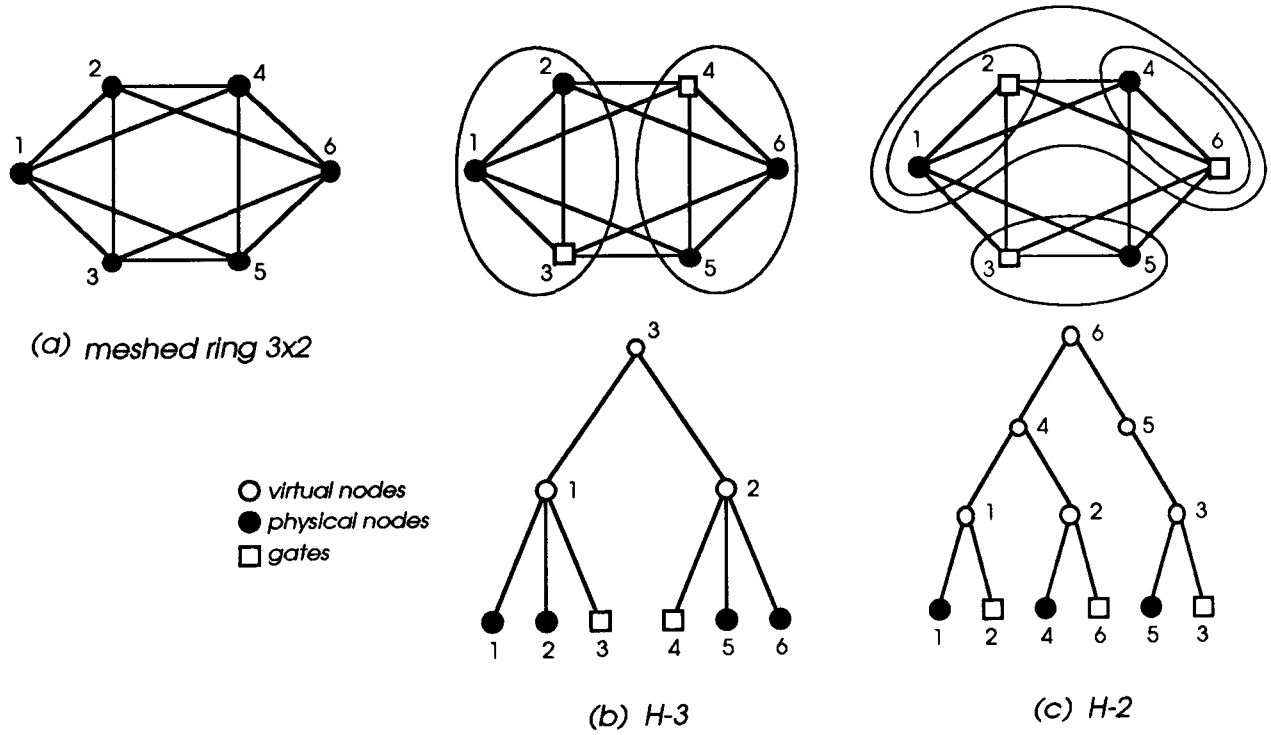


FIGURE 5.8
(a) 3 x 2 meshed ring, (b) 3-node clusters (2-level tree), (c) 2-node clusters (3-level tree)

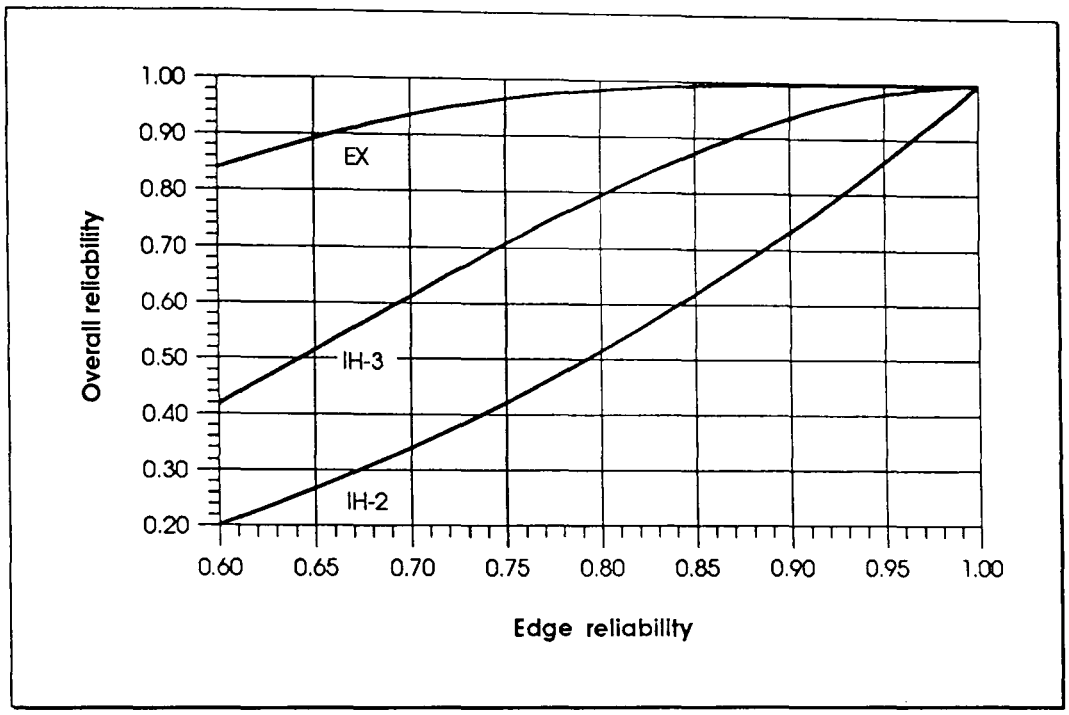


FIGURE 5.9
Overall reliability for a 3 x 2 meshed ring. EX is by using a exact method, IH-3 by using IHRM with 3 nodes per cluster and IH-2 with 2 nodes per cluster

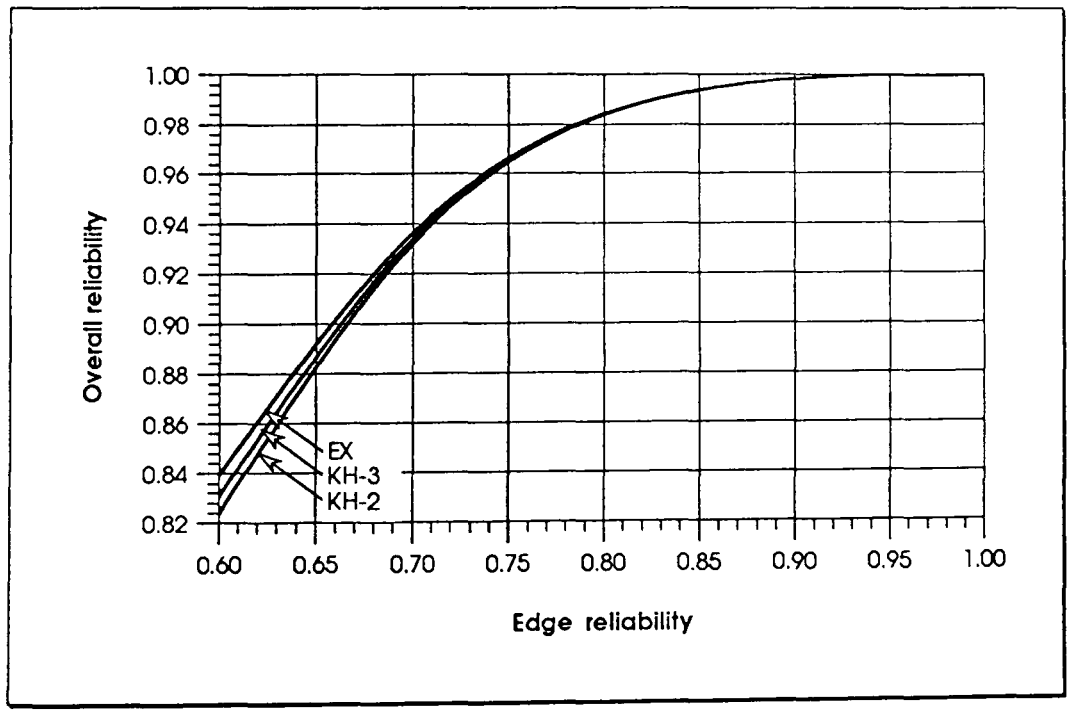


FIGURE 5.10
Overall reliability for a 3 x 2 meshed ring. EX is by using a exact method, KH-3 and KH-2 by using KHRM with 3 nodes per cluster and 2 nodes per cluster respectively

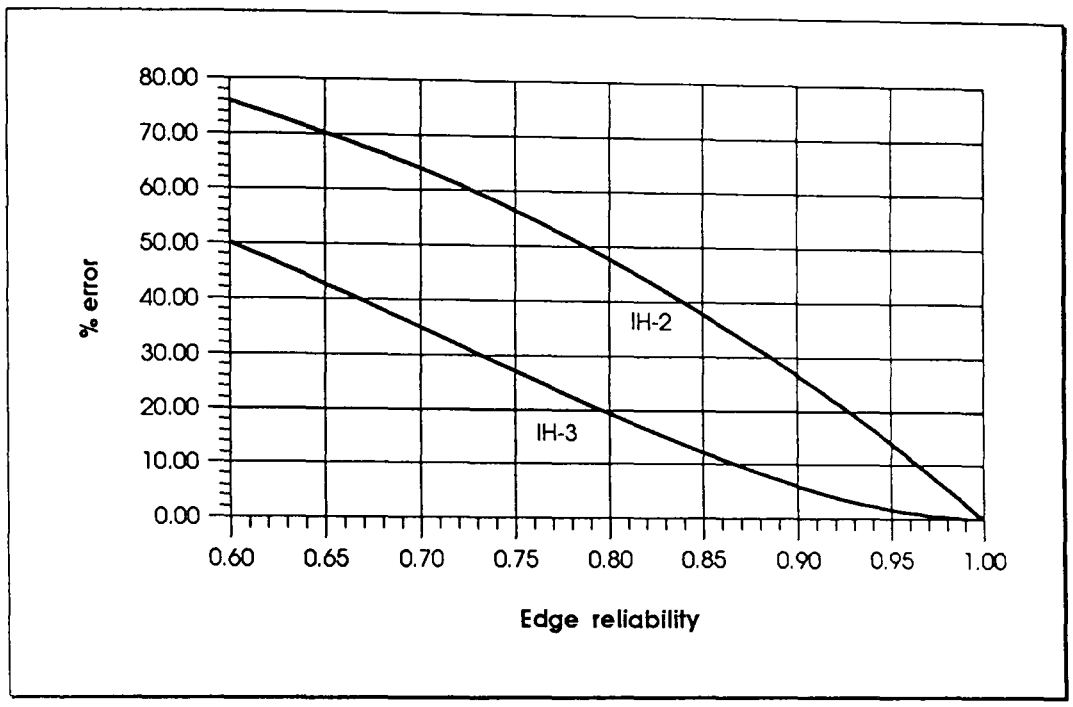


FIGURE 5.11
Error percentage of overall reliability for a 3 x 2 meshed ring by using IHRM

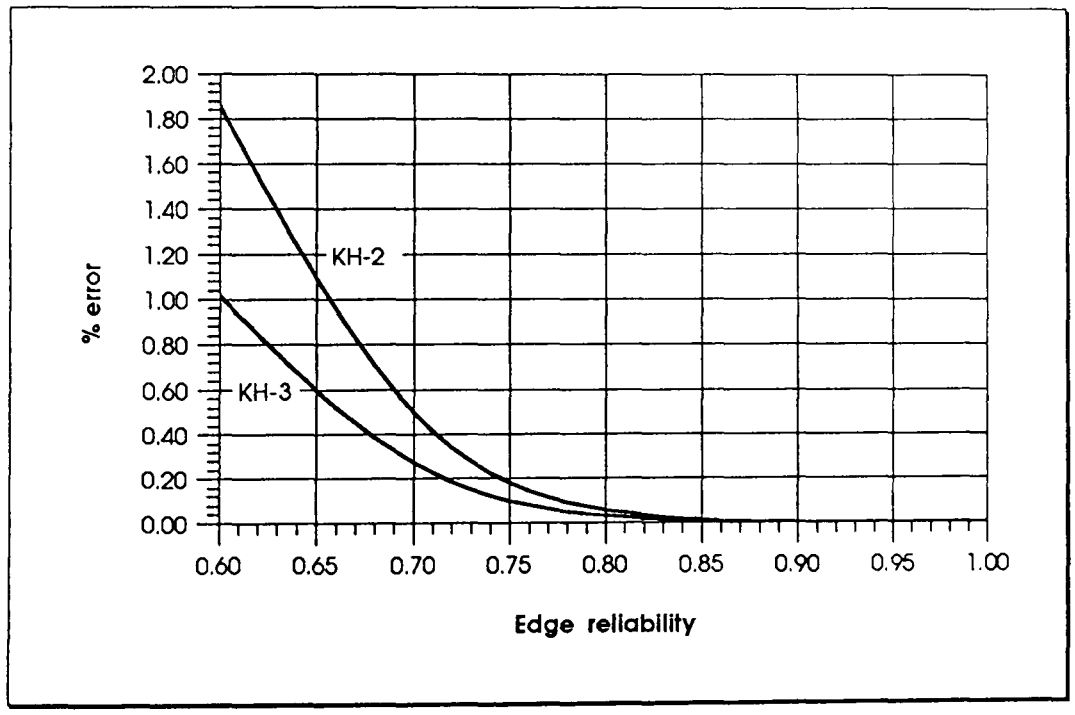


FIGURE 5.12
Error percentage of overall reliability for a 3 x 2 meshed ring by using KHRM

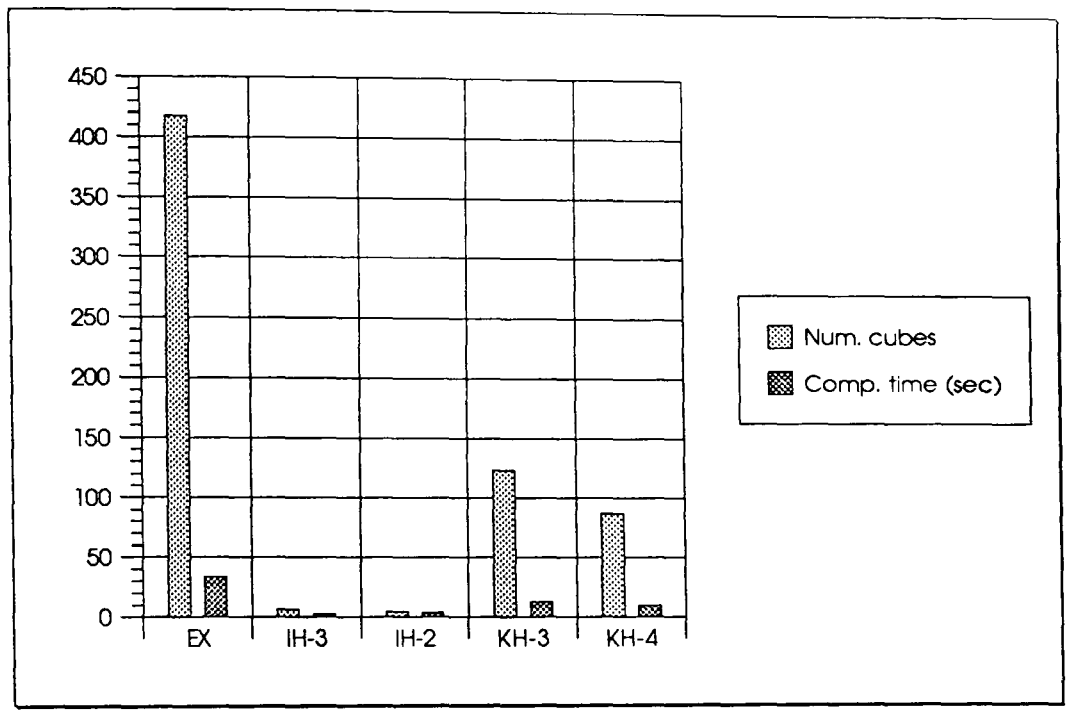


FIGURE 5.13
IHRM and KHRM computation time and memory (number of cubes) for AT reliability in a 3 x 2 meshed ring

5.4.2 MESHED RING 6X2

The second example is a 6 x 2 undirected meshed ring which has 12 nodes and 24 edges. Overall reliability and two terminal reliability from node 3 to 9 is obtained when (a) the exact technique is used, (b) IHRM and KHRM are used with clusters having 6 nodes maximum (2-level tree), (c) with clusters having 4 nodes (2-level tree) and (d) with clusters having 3 nodes (3-level tree), as illustrated in Figure 5.14.

Figures 5.15 and 5.16 show a plot of AT as a function of edge reliability for IHRM and KHRM respectively; Figures 5.17 and 5.18 show the error percentage for AT also as a function of edge reliability. Figure 5.19 is a comparison of computer time and the number of cubes generated. Figures 5.20 to 5.24 show the above measures for TT reliability.

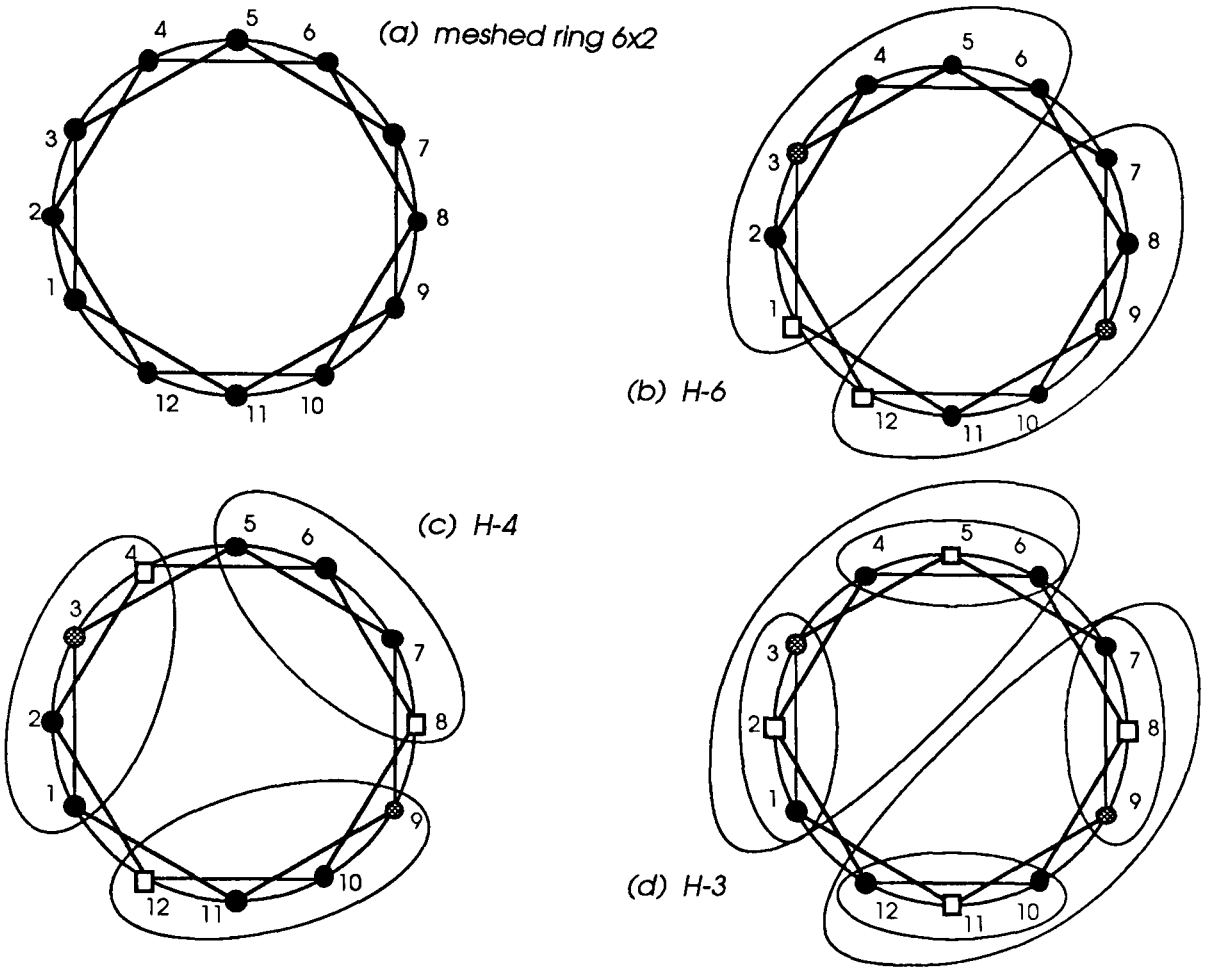


FIGURE 5.14
(a) 6 x 2 meshed ring, (b) 6-node clusters (2-level tree), (c) 4-node clusters (2-level tree), (d) 3-node clusters (3-level tree)

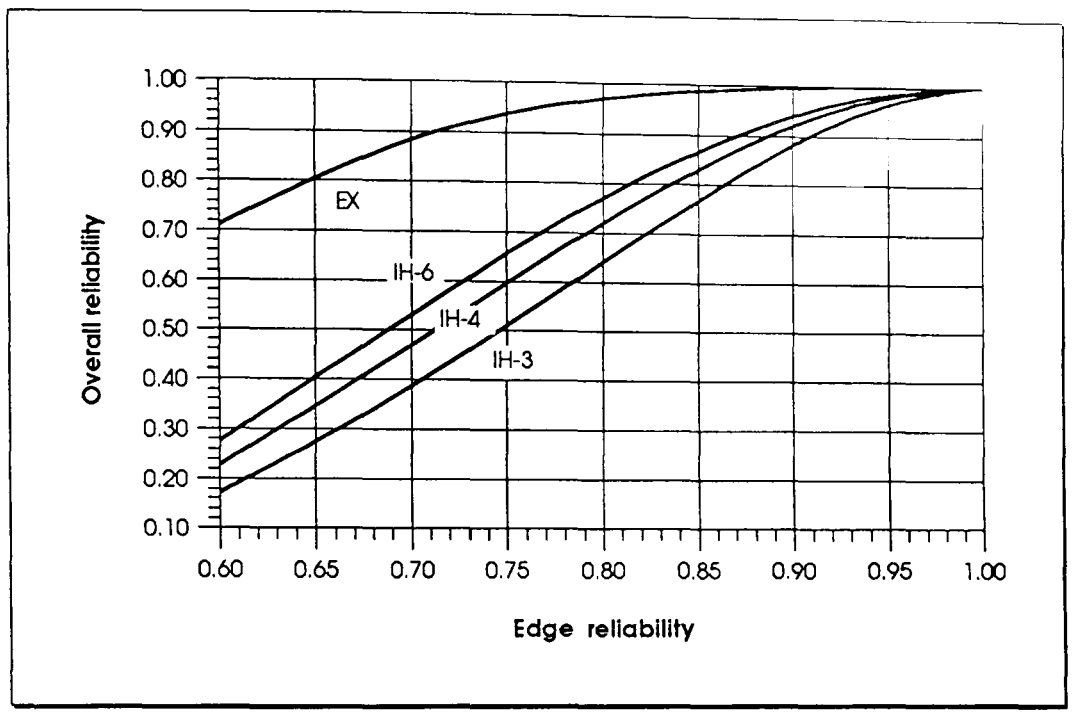


FIGURE 5.15
Overall reliability for a 6 x 2 meshed ring. EX is by using a exact method, IH-6, IH-4 and IH-3 by using IHRM with 6 nodes, 4 nodes and 3 nodes per cluster respectively.

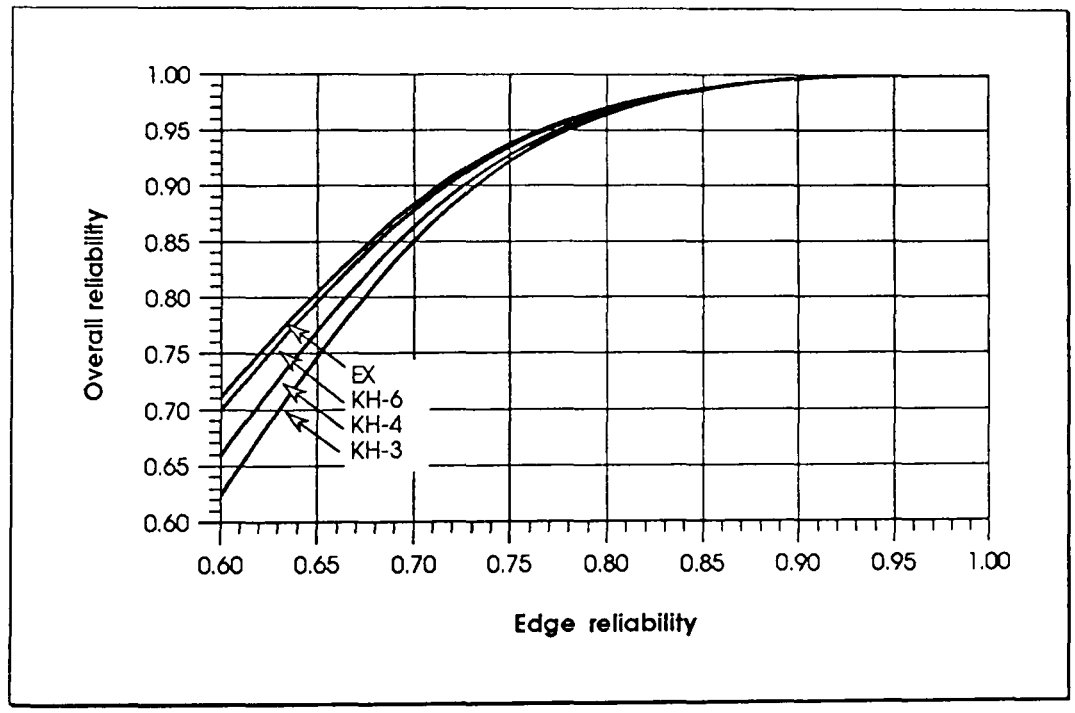


FIGURE 5.16
Overall reliability for a 6 x 2 meshed ring. EX is by using a exact method, KH-6, KH-4 and KH-3 by using KHRM with 6 nodes, 4 nodes and 3 nodes per cluster respectively.

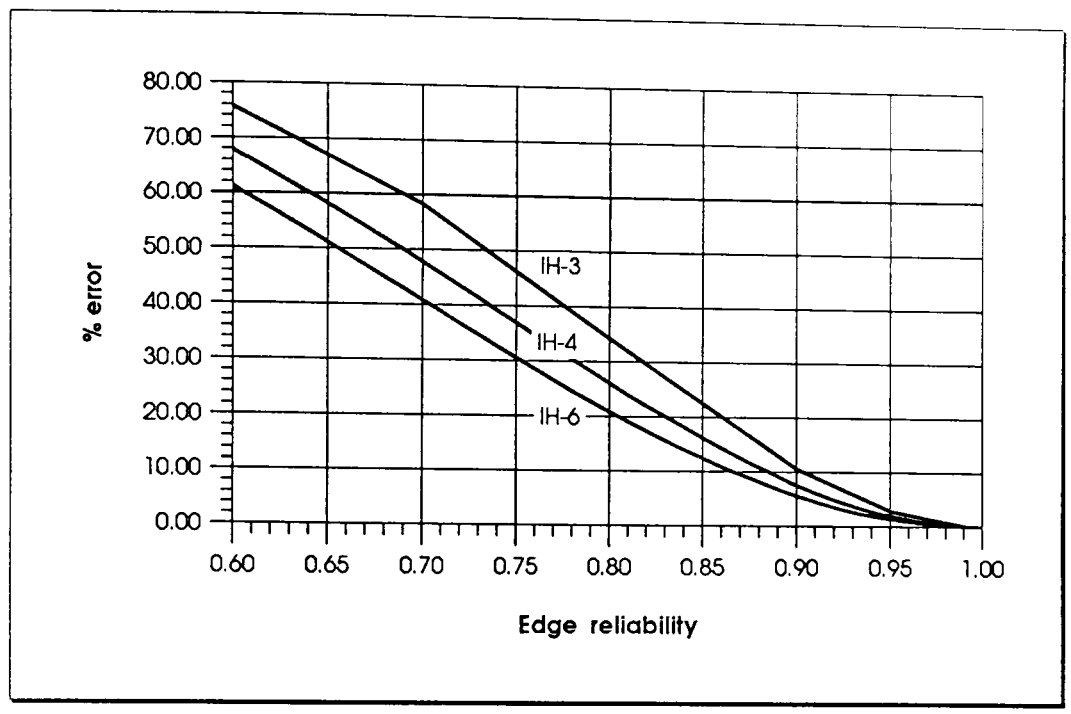


FIGURE 5.17
Error percentage of overall reliability for a 6 x 2 meshed ring by using IHRM

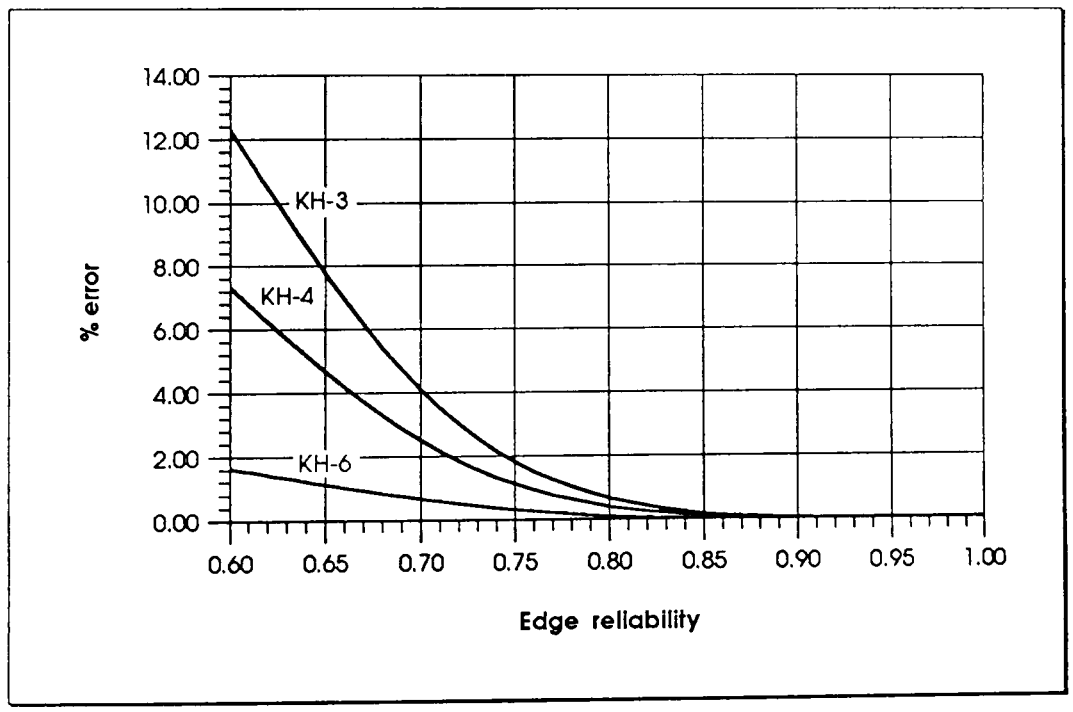


FIGURE 5.18
Error percentage of overall reliability for a 6 x 2 meshed ring by using KHRM

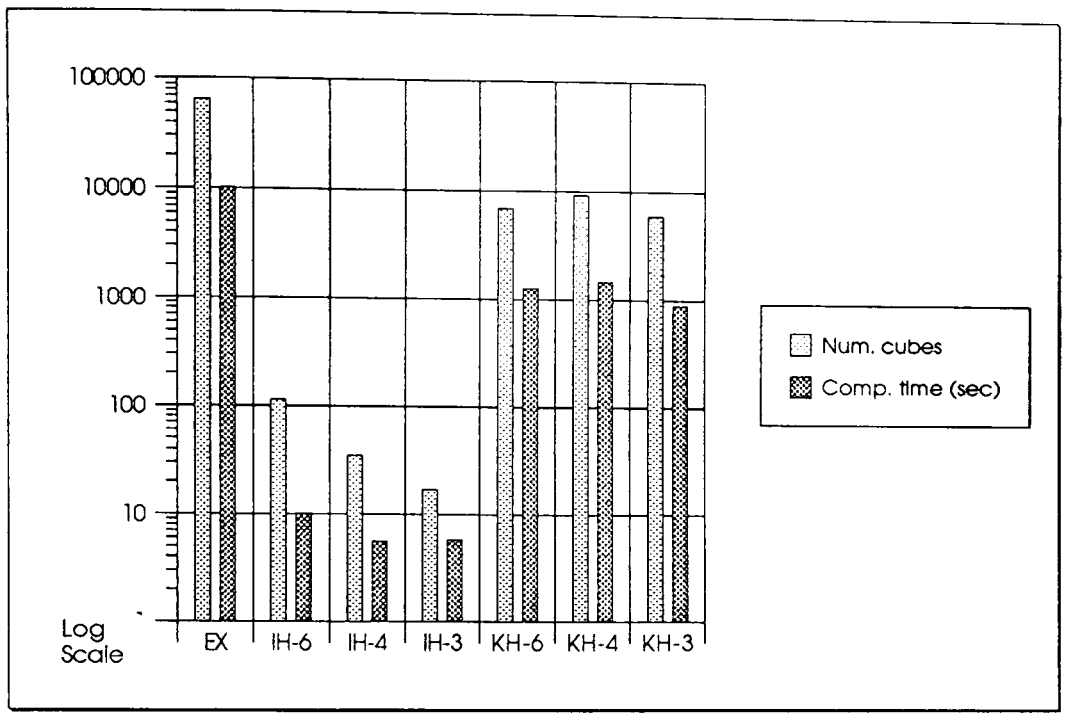


FIGURE 5.19
IHRM and KHRM computation time and memory (number of cubes) for AT reliability in a 6 x 2 meshed ring

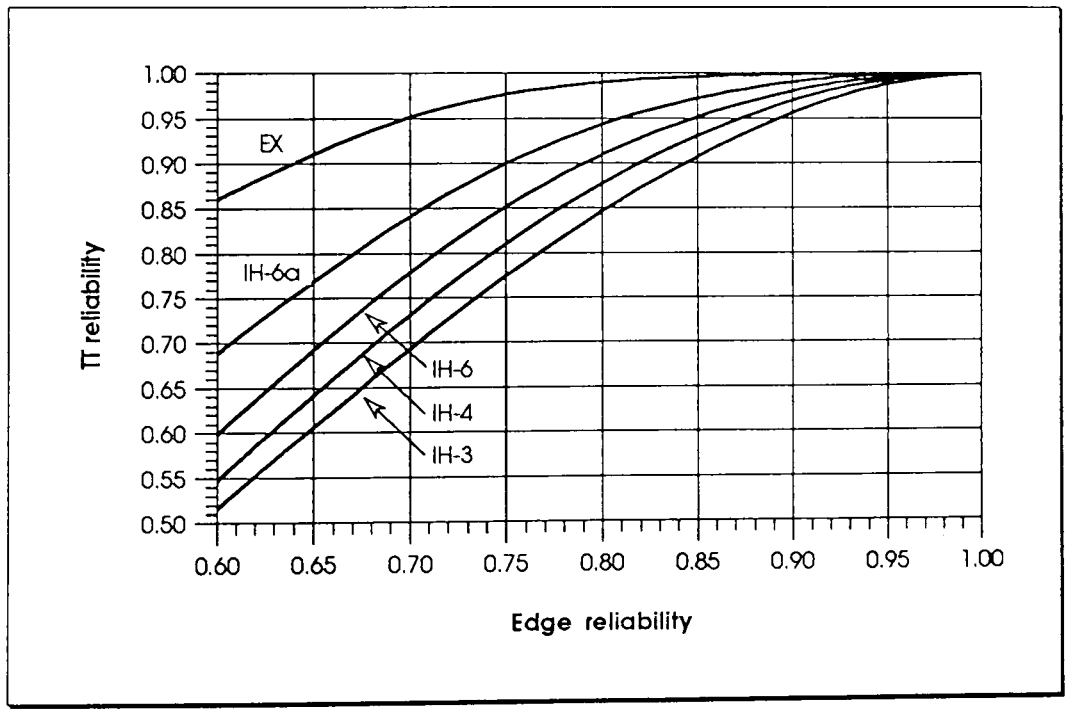


FIGURE 5.20
Two-terminal reliability for a 6 x 2 meshed ring. EX is by using a exact method, IH-6, IH-4 and IH-3 by using IHRM with 6 nodes, 4 nodes and 3 nodes per cluster respectively, the terminal nodes are at a mean distance from the gate. IH-6a is with the gates as terminal nodes.

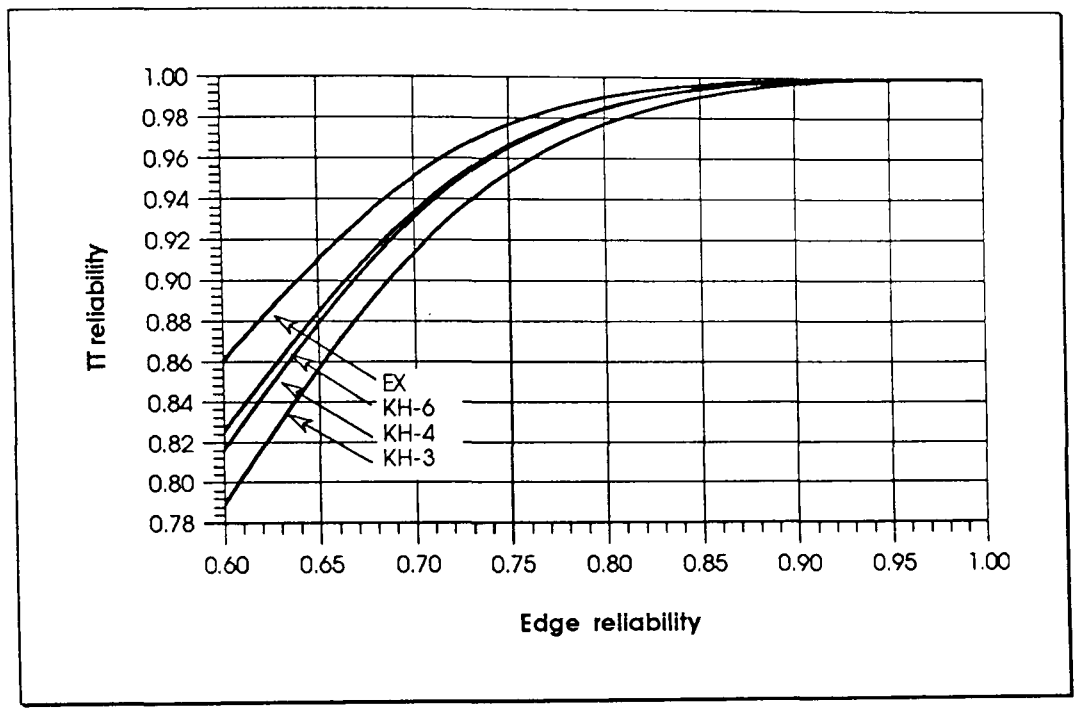


FIGURE 5.21
Two-terminal reliability for a 6 x 2 meshed ring. EX is by using a exact method, KH-6, KH-4 and KH-3 by using KHRM with 6 nodes, 4 nodes and 3 nodes per cluster respectively.

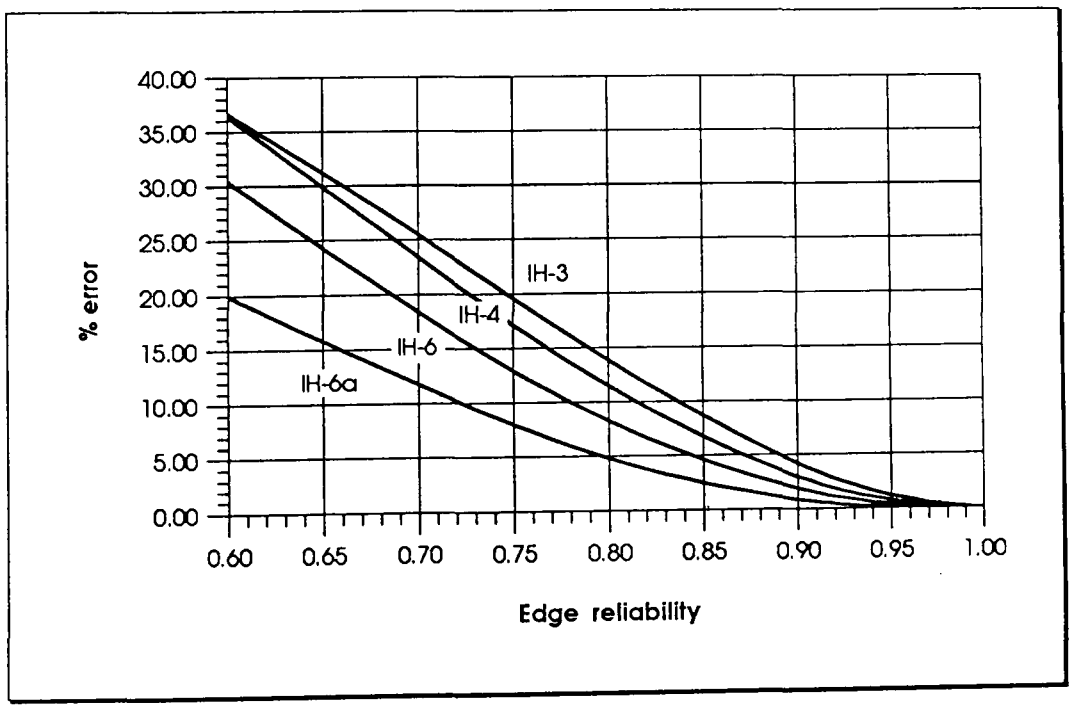


FIGURE 5.22
Error percentage of two-terminal reliability for a 6 x 2 meshed ring by using IHRM

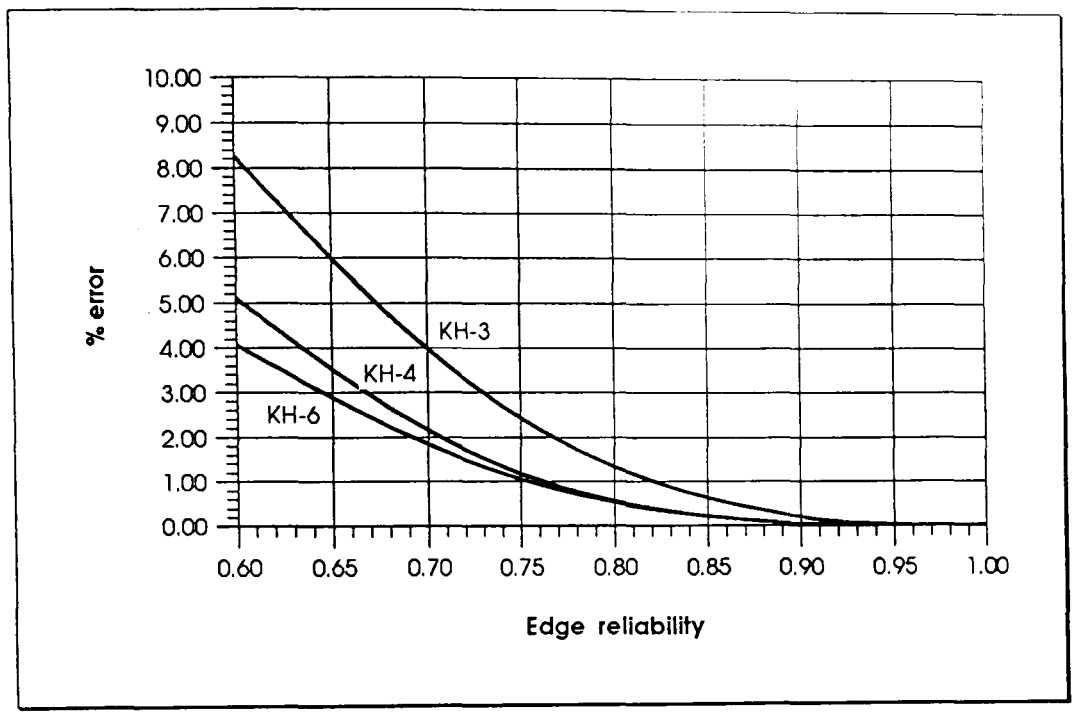


FIGURE 5.23
Error percentage of two-terminal reliability for a 6 x 2 meshed ring by using KHRM

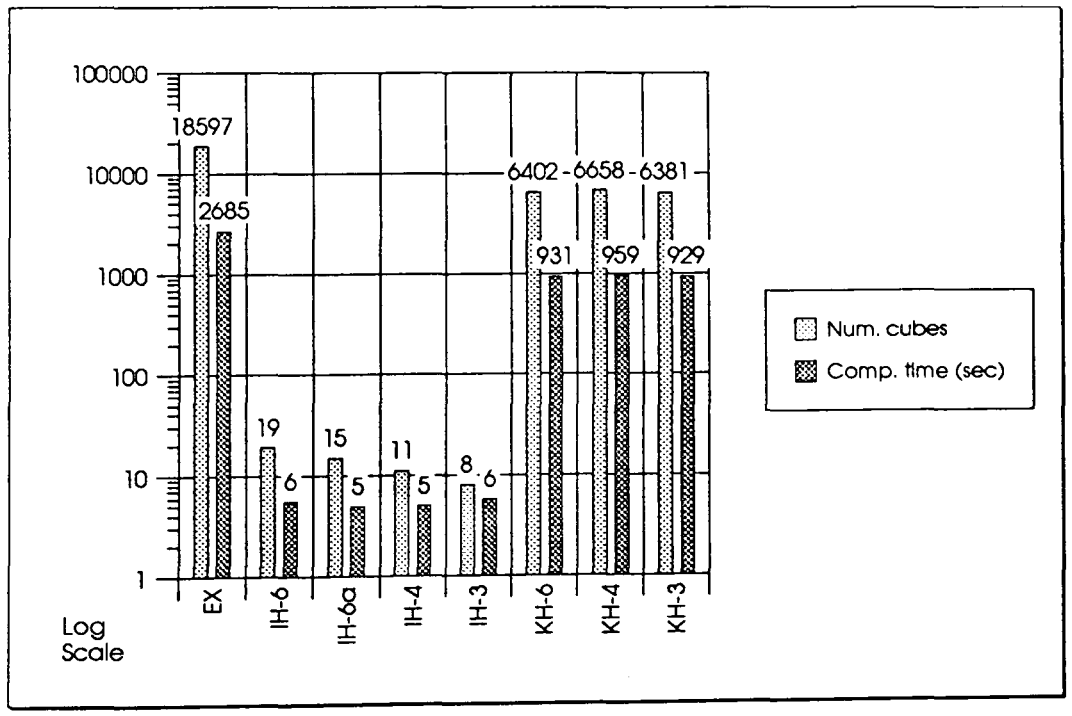


FIGURE 5.24
IHRM and KHRM computation time and memory (number of cubes) for TT reliability in a 6 x 2 meshed ring

5.4.3 *RING 12*

A ring network is an example of a sparse graph. Figure 5.25 shows such graph with 12 nodes and 12 edges and the hierarchical structure tested : clusters formed by 4 nodes maximum. Figure 5.26 show a plot of overall system reliability against edge reliability for the 3 methods: exact, IHRM and KHRM; and Figure 5.27 shows the computer time and number of cubes.

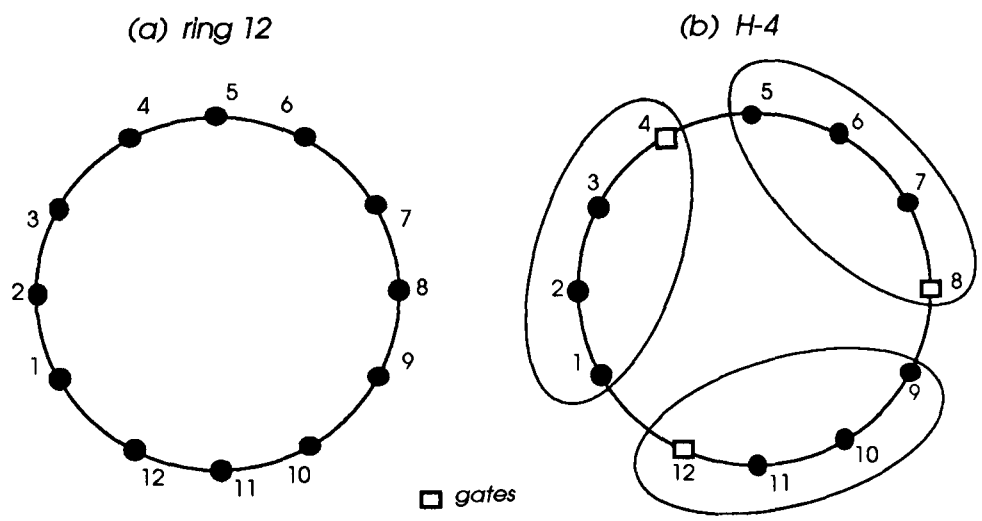


FIGURE 5.25
(a) 12 node ring, (b) clustering with 4 nodes per cluster (2-level tree)

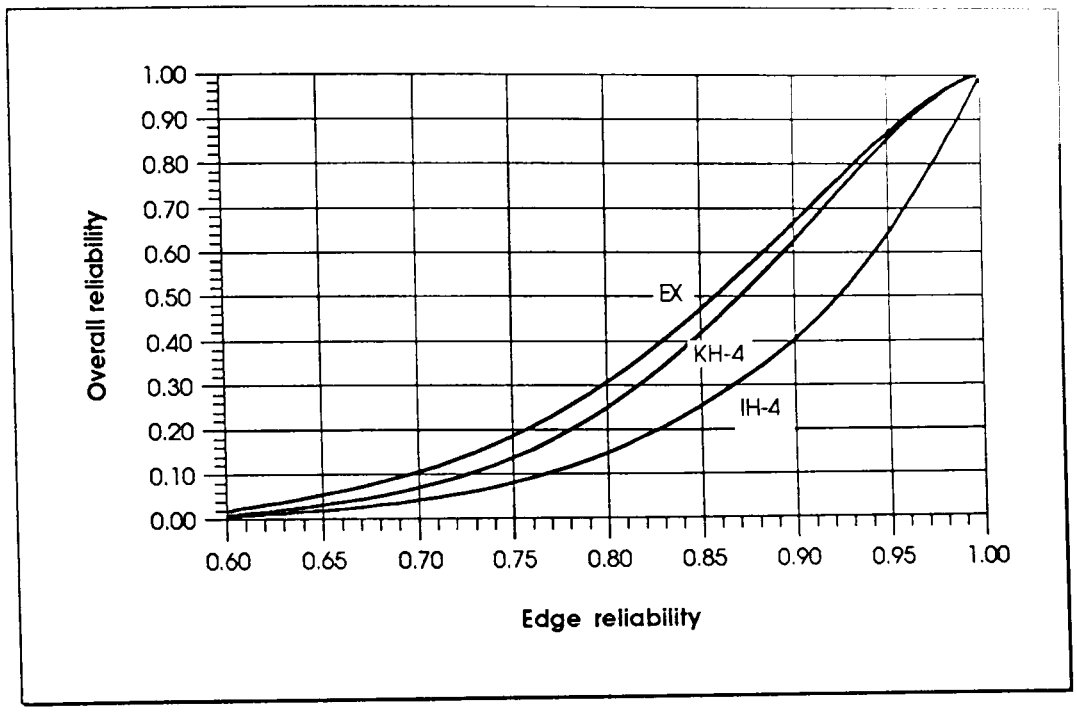


FIGURE 5.26
Overall reliability for a 12 node ring. EX is by using a exact method, IH-4 and KH-4 by using IHRM and KHRM with 4 nodes per cluster.

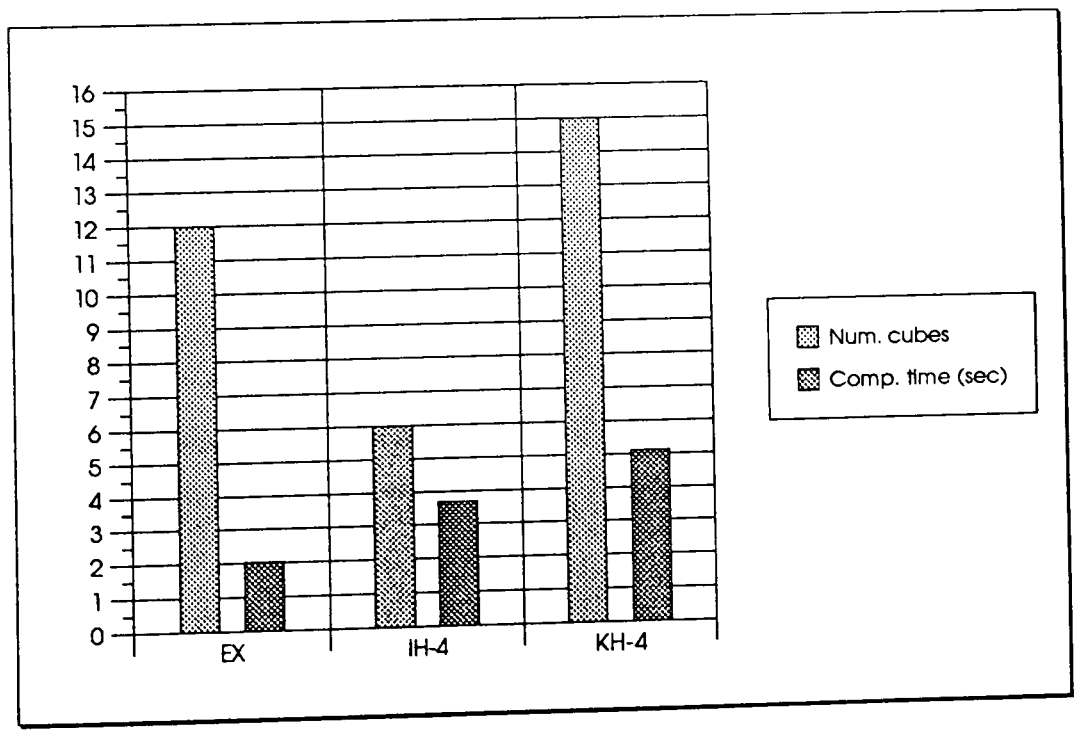


FIGURE 5.27
IHRM and KHRM computation time and memory (number of cubes) for AT reliability in a 12 node ring

5.5 DISCUSSION OF RESULTS

The following observations can be made about the results obtained in the above examples.

(a) The precision of reliability evaluation when using the IHRM or KHRM methods for approximate evaluation of flat systems depends on the choice of the clustering structure, i.e. number of levels in the hierarchical tree, number of nodes per cluster and connectivity of each cluster. Therefore, in general, to obtain better results, closer to those obtained by exact methods, the number of levels should be low, the number of nodes should be high, and the subgraph constituting the cluster should be highly connected.

(b) The error percentage of system reliability is always decreasing as the value of element reliability increases. Therefore for practical values of edge reliability, i.e. from 0.9 to 1.0, the hierarchical method gives approximate results quite close to that obtained by the exact method.

(c) For IHRM the computer time and memory is greatly reduced, but the results are not very exact when compared to the results obtained by the exact method for lower values of element reliability.

(d) KHRM gives always more precise results than those obtained by IHRM, but the savings in computation time and memory are less. For some problems and some classes of graphs it is worthwhile to use KHRM, but for some others like ring networks it is not, since the computation time and number of cubes is higher than for the exact method as is explained below in (f).

(e) For TT and KT reliability approximation using IHRM and KHRM, the accuracy also vary according to the nodes being chosen as gates in each cluster, since TT and KT reliability of each local subnetwork is

dependent on the distance and network structure between the corresponding nodes and the local gate. Generally with nodes closer to the gate a higher reliability is obtained. In our example it was used a mean distance between gate and nodes.

(f) The system configuration also affects the approximation. Some structures are not suitable to decompose hierarchically to simplify its reliability evaluation when no highly connected subnetworks can be formed, such as ring and sparse structures. For this class of graphs the number of cubes generated for the exact method, which is dependent of the number of communication paths, is always low since there are few paths. Thus, in this case, by employing hierarchical decomposition, we can create an overhead, more evident with KHRM where the number of cubes and computation time are higher than for the exact method as it could be seen in Figure 5.27.

Chapter 6

Summary and Conclusions

6.1 ANALYSIS OF WORK

The main goals of this work are the study and implementation of models for reliability and fault tolerance analysis of multiprocessor systems; basically of their intercommunication structure, i.e. the interconnection network. Two classes of models were defined: deterministic and probabilistic. Both are based on graph theory concepts and the criteria of reliability and fault tolerance as measures of connectivity, i.e. the successful communication between the nodes of the system. Different connectivity problems were identified and classified into: unrooted problems, like two-terminal (TT), overall (AT) and k-node (KT) connectivity; and rooted problems, such as source to terminal (ST), source to all terminal (SAT), source to k-terminal (SKT) and k-source to k-terminal (KSKT) connectivity. Another problem of interest is k-out-of-n reliability as the general model of redundancy.

In the deterministic model, reliability is dependent upon denseness, distance and degree but above all on the number of edge and node disjoint paths (edge and node connectivity) required for the intercommunication among some nodes, according to the connectivity problem. Efficient algorithms were implemented to compute the different deterministic parameters.

For the probabilistic model it was assumed that the system components (edges and nodes) fail with some known probability distribution in an environment of statistically independent failures. A stationary measure of reliability is the probability of success; dynamic (time dependent) measures of interest are: reliability and MTTF for closed systems (non-repairable), and availability, MTBF and steady-state availability for repairable systems.

An efficient general combinatorial method for probabilistic reliability modelling (RM) was developed to deal with all reliability problems, this method consists basically of three steps:

(1) Obtain the paths corresponding to the connectivity problem, take them as the events in the probability space, represent these paths as cubes in Boolean algebra.

(2) Perform the "sharp" Boolean operation on the cubes to arrive at a Boolean algebraic expression.

(3) Interpret the Boolean expression as a disjoint sum of terms, i.e. a symbolic probability expression. From this expression any stationary or dynamic reliability measure can be easily calculated for any given probability distributions by direct substitution of their values into the expression.

In RM, steps (1) and (2) are executed recursively in order to gradually obtain the Boolean expression; the advantage of this method is that it reduces considerably the computer requirements: storage and computer time.

For large multiprocessor systems probabilistic reliability calculations require enormous computational resources, therefore approximation techniques have to be employed. The first approach was as described above, but taking only those paths that contribute more significantly to the symbolic expression. This is easily done in the recursive method RM by limiting the depth of the computation to a certain predefined limit.

The second approach was to employ hierarchical decomposition of the system. First, by the use of hierarchical clustering, the system is partitioned into smaller subsystems or clusters; second, the general reliability model RM is hierarchically applied in a bottom-up fashion to each cluster in order to obtain an approximation of reliability. For each connectivity problem different hierarchical connectivity strategies were identified. This hierarchical approach led to the development of two methods: IHRM and KHRM. The latter method gives results closer to the exact method, but in some cases the savings in computation time and memory are insignificant; on the other hand, with IHRM the computer requirements are greatly reduced, but the results are not very exact for lower values of element reliability.

6.2 MODEL PERFORMANCE

It is difficult to directly compare the performance of our model implementation to other published models, since performance is determined by several factors: (a) the algorithms, (b) the implementation, (c) the compiler and (d) the host computer. Also, quite often, the computation time and memory requirements reported do not include the whole computation, including the generation of paths, numerical reliability calculations, etc.

computation, including the generation of paths, numerical reliability calculations, etc.

It is believed that our implementation, although quite general for several reliability problems and measures, is quite efficient. For several examples tested in medium size configurations, satisfactory solution times can now be obtained on a Macintosh personal computer whereas previously, mainframe computers might have been required. It is also a recursive method that requires less memory.

6.3 APPLICATIONS

The models explained above can be applied to evaluate reliability of systems of different granularity as long as they can be represented as simple graphs, from VLSI embedded multiprocessors to geographically distributed computer networks. Some application examples are:

- *Computer networks*, such as national networks, telephone networks, LANs, etc., where all connectivity problems, rooted and unrooted are of interest.

- *Distributed systems*, as the computer resources (processes, databases, etc.) are distributed among the nodes (computers, memories, etc.) of the system. It is desirable to obtain reliability for connectivity problems such as: KT, TT and AT, or k-out-of-n redundancy like in the example of section 4.4.8.

- *Multistage interconnection networks*, where the switches, inputs and outputs can be represented as nodes of a directed graph. In this case,

rooted problems such as SKT and KSKT can be used to obtain the reliability of communication from the inputs to the outputs.

- For *medium-power multiprocessors and VLSI multiprocessor arrays* several parallel architectures have been proposed. Basic configurations are ring, rectangular mesh, binary tree, binary cube, etc.. These architectures can be augmented or combined by adding links in order to improve their reliability and fault tolerance, like meshed rings, meshed trees, etc. All reliability problems are of interest, particularly overall reliability and the degree of fault tolerance, since it is desired to compare the different architectures and their fault tolerant variations.

6.4 RECOMMENDATIONS FOR FUTURE WORK

It has been stated previously the difficulties associated to the evaluation of reliability in multiprocessor systems; this has led to the development of simplified models such as the one presented in this report.

Some parameters have not been considered here, but are important areas for future research needed for the reliability modelling of multiprocessor systems. Among them are:

(a) Development of *parallel algorithms* to improve the computation efficiency.

(b) Exploiting *fault tolerant routing and control algorithms* to help develop more realistic reliability models to establish simple and practical paths of communication between the remaning nodes in case of node or link failures. Reliability calculations can be simplified if

only the real paths (those generated for the routing algorithm) are considered.

(c) *Optimisation and reinforcement of reliability.* Investigation of applicable methods for optimisation in redundancy allocation, subjected to some reliability constraints; and the reinforcement techniques, i.e. if the topology does not meet a specified level of reliability then an identification and reinforcement of the weak points of the system is required.

(d) Development of better models for *software/hardware reliability and availability* in distributed systems.

(e) Development of *unified reliability and performance models.*

(f) Inclusion of *fault coverage* analysis.

(g) *Statistical dependency* among failures of different components. A hierachical model can be used for failure dependency problems in which several modules are dependent upon each other, as when they are placed in a single unit.

References

[AGG 81] K.K. Aggarwal and S. Rai, "Reliability evaluation in computer - communication networks", *IEEE Trans. Reliab.*, Vol. R-30, No. 1, April 1981, pp. 32-35.

[AVI 78] A. Avizienis, "Fault-tolerance : the survival attribute of digital systems", *Proc. IEEE,* Vol. 66, No. 10, Oct. 1978, pp. 1109-1125.

[AVI 86] A. Avizienis, "Dependable computing: from concepts to design diversity", *Proc. IEEE,* Vol. 74, No. 5, May 1986, pp. 629-638.

[BAL 86] M.O. Ball, "Computational complexity of network reliability analysis: an overview", *IEEE Trans. Reliab.*, Vol. R-35, No. 3, 1986, pp. 230-239.

[BAR 84] R.E. Barlow and K.D. Heidtmann, "Computing *k*-out-of-*n* system reliability", *IEEE Trans. Reliab.*, Vol. R-33, No. 4, Oct. 1984, pp. 322-323.

[BEA 78] M.D. Beaudry, "Performance-related reliability measures for computing systems", *IEEE Trans. Computers*, Vol. C-27, No. 6, Jun. 1978, pp. 540-547.

[BIL 83] R. Billington and R.N. Allan, *Reliability evaluation of engineering systems : concepts and techniques*, Pitman, London, 1983.

[CHE 85] Y. Chen and T. Chen, "DFT : Distributed fault tolerance - analysis and design", *Dig. 15th Int'l Symp. Fault-Tolerant Computing (FTCS-15)*, 1985, pp. 280-285.

[CHU 81] R.F. Churchhouse, Ed., *Handbook of applicable mathematics, Vol. 3 : Numerical methods*, Wiley, Chichester, 1981.

[COL 87] C.J. Colbourn, *The combinatorics of network reliability*, Oxford University Press, Oxford, 1987.

[DEP 77] P.G. Depledge, "Reliability considerations for airborne microcomputers", PhD Thesis, UMIST, 1977.

[EVE 80] B. Everitt, *Cluster Analysis* (2nd Ed.), Heinemann Educational Books, Halsted Press, London, 1980.

[GEI 83] R.M. Geist and K.S. Trivedi, "Utrahigh reliability prediction for fault-tolerant computer systems", *IEEE Trans. Computers*, Vol. C-32, No. 12, Dec. 1983, pp. 1118-1127.

[GIB 85] A. Gibbons, *Algorithmic graph theory*, Cambridge University Press, Cambridge, 1985.

[GRN 80] A. Grnarov, L. Kleinrock and M. Gerla, "A new algorithm for symbolic reliability analysis of computer communication networks", *Pacific Telecomm. Conf.*, Jan. 1980, pp. 1A.11-1A.19.

[HAR 86] S. Hariri and C.S. Raghavendra, "SYREL : A symbolic reliability algorithm based on path and cut set methods", *IEEE Infocom 86*, Miami, Fla., Apr. 1986, pp. 293-302.

[HAY 76] J.P. Hayes, "A graph model for fault-tolerant computing systems", *IEEE Trans. Computers,* Vol. C-25, No. 9, Sep. 1976, pp. 875-884.

[HWA 81] C.L. Hwang, F.A. Tillman and M.H. Lee, "System-reliability evaluation techniques for complex/large systems - A review", *IEEE Trans. Reliab.*, Vol. R-30, No. 5, Dec. 1981, pp. 416-423.

[JAI 85] S.P. Jain and K. Gopal, "Recursive algorithm for reliability evaluation of *k*-out-of-*n* :G system", *IEEE Trans. Reliab.*, Vol. R-34, No. 2, Jun. 1985, pp. 144-147.

[JOH 84] B.W. Johnson, "Fault-tolerant microprocessor-based systems", *IEEE Micro*, Vol. 4, No. 6, Dec. 1984, pp. 6-21.

[KLE 80] L. Kleinrock and F. Kamoun, "Optimal clustering structures for hierarchical topological design of large computer networks", *Networks*, Vol. 10, 1980, pp. 221-248.

[KUH 86] J.G. Kuhl and S.M. Reddy, "Fault-tolerance considerations in large, multiple processor systems", *IEEE Computer*, Vol. 19, No. 3, Mar. 1986, pp. 56-67.

[LAW 76] E. Lawler, *Combinatorial optimization : Networks and matroids,* Holt, Rinehart and Winston, New York, 1976.

[LOC 84] M.O. Locks, "Comments on: Improved method of inclusion exclusion applied to *k*-out-of-*n* systems", *IEEE Trans. Reliab.*, Vol. R-33, No. 4, Oct. 1984, pp. 321-322.

[MAE 86] E. Maehle, et.al., "A graph model for diagnosis and reconfiguration and its application to a fault-tolerant multiprocessor system", *Dig. 16th Int'l Symp. Fault-Tolerant Computing (FTCS-16),* 1986, pp. 292-297.

[MAK 83] S.V. Makam and C.S. Raghavendra, "Dynamic reliability modeling and analysis of computer networks", *Proc. 1983 Int'l Conf. Parallel Processing,* pp. 496-502.

[MAL 81] Y.K. Malaiya and S.Y.H. Su, "Reliability measure of hardware redundancy fault-tolerant digital systems with intermittent faults", *IEEE Trans. Computers,* Vol. C-30, No. 8, Aug. 1981, pp. 600-604.

[MAN 87] D. Mandaltsis and J.M. Kontoleon, "A decomposition technique for the overall reliability evaluation of large computer communication networks", *Microelectron. Reliab.,* Vol. 27, No. 2, 1987, pp. 299-312.

[MEY 85] F.J. Meyer, "Dynamic testing strategy for distributed systems", *Dig. 15th Int'l Symp. Fault-Tolerant Computing (FTCS-15),* 1985, pp. 84-90.

[MIL 65] R.E. Miller, *Switching theory, vol. I : Combinatorial circuits,* Wiley, New York, 1965.

[PAG 88] L.B. Page and J.E. Perry, "A practical implementation of the factoring theorem for network reliability", *IEEE Trans. Reliab.,* Vol. R-37, No. 3, Aug. 1988, pp. 259-267.

[PRA 86] D.K. Pradhan, "Fault-tolerant multiprocessor and VLSI-based system communication architectures", Chapter 7 in *Fault-Tolerant Computing, Theory and Techniques,* Vol. II, D.K. Pradhan, (Ed.), Prentice-Hall, Englewood Cliffs, NJ, 1986.

[PRE 67] F.P. Preparata, G. Metze and R.T. Chien, "On the connection assignement problem of diagnosable systems", *IEEE Trans. Electr. Computers,* Vol. EC-16, No. 6, Dec. 1967, pp. 848-854.

[RAM 86] C.V. Ramamoorthy, J. Srivastava and W-T. Tsai, "Clustering techniques for large distributed systems", *Proc. IEEE Infocom 86,* Miami, Fla., Apr. 1986, pp. 395-404.

[REI 77] E.M. Reingold, J. Nievergelt and N. Deo, *Combinatorial algorithms : Theory and practice,* Prentice-Hall, Englewood Cliffs, NJ, 1977.

[REN 80] D.A. Rennels, "Distributed fault-tolerant computer systems", *IEEE Computer*, Vol. 13, No. 3, Mar. 1980, pp. 55-65.

[RIS 87] T. Risse, "On the evaluation of the reliability of k-out-of-n systems", *IEEE Trans. Reliab.*, Vol. R-36, No. 4, Oct. 1987, pp. 433-435.

[SAT 82] A. Satyanarayana, "A unified formula for analysis of some network reliability problems", *IEEE Trans. Reliab.*, Vol. R-31, No.1, Apr. 1982, pp. 23-32.

[SOI 85] I.M. Soi and K.K. Aggarwal, "Overall reliability evaluation for large computer communication networks: An MHC approach", *Microelectron. Reliab.*, Vol. 25, No. 2, 1985, pp. 215-222.

[STI 86] J.J. Stiffler, "Computer-aided reliability estimation", Chapter 9 in *Fault-Tolerant Computing, Theory and Techniques,* Vol. II, D.K. Pradhan, (Ed.), Prentice-Hall, Englewood Cliffs, NJ, 1986.

[TOR 83] J. Torrey, "A pruned tree approach to reliability computation", *IEEE Trans. Reliab.,* Vol. R-32, No. 2, Jun. 1983, pp. 170-174.

[XU 86] W. Xu and X. Lin, "A new algorithm for the reliability evaluation of computer communication networks", *Microelectron. Reliab.*, Vol. 26, No. 6, 1986, pp. 1013-1017.

[YAN 86] R.M. Yanney and J.P. Hayes, "Distributed recovery in fault-tolerant multiprocessor networks", *IEEE Trans. Computers,* Vol. C-35, No. 10, Oct. 1986, pp. 871-879.

Appendix A

Basic Concepts of Graph Theory

A *graph* $G = (N, E)$ consists of a set of objects $N = \{x_1, x_2, \ldots, x_n\}$ called *nodes* or *vertices*, which are interconnected by another set $E = \{e_1, e_2, \ldots, e_m\}$ whose elements are called *edges*. Each edge e_k is identified with a pair (x_i, x_j) of nodes which are called the *end-nodes* of e_k . The number of nodes in a graph is denoted by $n = |N|$ and the number of edges by $e = |E|$. An example of a graph is shown in Figure A.1.

An edge having the same node as both its end-nodes is called a *self-loop* (edge e_1 in Fig. A.1). If more than one edge is associated with a given pair of nodes, these edges are referred as *parallel edges*, such as edges e_4 and e_5 in Fig. A.1. A graph that has neither self-loops nor parallel edges is called a *simple graph*.

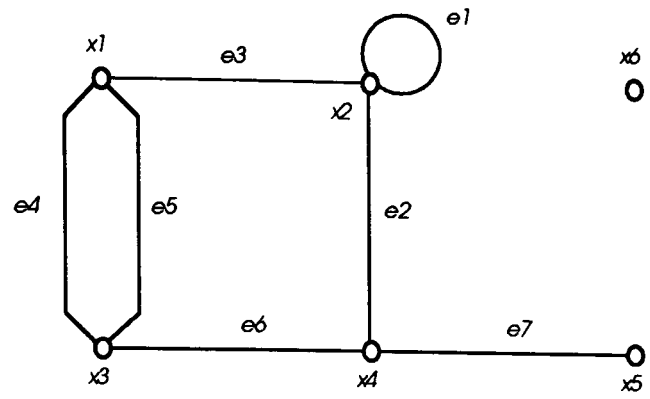


FIGURE A.1
Graph with 6 nodes and 7 edges

If an edge e_k has x_i as an end-node, then e_k is *incident* with x_i; if $(x_i, x_j) \in$ E then node x_j is *adjacent* or *neighbour* to x_i. For example in Fig. A.1 edges e_2, e_6 and e_7 are incident with x_4 which is adjacent to x_2, x_3 and x_5. Also, two non-parallel edges are adjacent if they have a common end-node, such as e_2 and e_6 in Fig. A.1.

The *degree* of a node x_i, denoted as $d(x_i)$, is the number of edges incident with x_i. A node x_i for which $d(x_i) = 0$ is called an *isolated* node, if $d(x_i) = 1$ is called a *pendant* node (x_6 and x_5 respectively in Fig. A.1). A graph is *regular* if every node has the same degree.

Two graphs G_1 and G_2 are said to be *isomorphic* if there is a one-to-one correspondence between their nodes such that the number of edges joining any two nodes in G_1 is equal to the number of edges joining the corresponding two nodes in G_2. A (proper) *subgraph* of G is a graph obtainable by the removal of a number of edges and/or nodes of G. The removal of a node necessarily implies the removal of every edge incident to it.

A *path* from x_1 to x_i is a sequence $P = x_1, e_1, x_2, e_2, \ldots, e_{i-1}, x_i$ of alternating nodes and edges such that for $1 \leq j < i$, e_j is incident with x_j and x_{j+1}. If $x_1 = x_i$ then P is a *circuit*. If in a path each node only appears once, then is called

simple path. Two paths are *edge-disjoint* if they do not have any edges in common.

The *length* of a path or circuit is the number of edges it contains, and *distance* between two nodes is the length of the shortest path.

A graph G is said to be *connected* if there is at least one path between every pair of nodes in G. Otherwise G is *disconnected*.

A *tree* T is a connected graph without any circuit, so a *simple path* can be seen also as a tree. A tree is said to be a *spanning tree* of a connected graph G if T is a subgraph of G and contains all nodes of G. A tree T is a *Steiner tree* if T spans over a subset of nodes of G.

A *directed graph* or *digraph* is a graph in which edges have assigned a direction. If $e_k = (x_i, x_j)$ is an edge of a digraph, then e_k is understood to be directed from the first node x_i to the second node x_j (e_k is *incident from* x_i and *incident to* x_j). x_j is called a *succesor* of x_i , and x_i is the *predecessor* of x_j .

The number of edges incident from a node x_i is called the *out-degree* of x_i and is written as $d+ (x_i)$; the number of edges incident to x_i is called the *in-degree* and is written as $d- (x_i)$. An *out-tree* is a connected digraph that has no circuits and there is precisely one node R of zero in-degree. So, in an out-tree there is a directed path from the root R to every other node. Similarly, an *in-tree* is obtained reversing the direction of every edge.

Finally, in a graph G, when a number or weight is assigned to each edge and/or node, G is called a *weighted graph*.

Appendix B

Computer Implementation Details

The computer program for both reliability models, deterministic and probabilistic, have been implemented on an Apple Macintosh™ personal computer. The computer program has been written in the language Pascal, using Think Lightspeed Pascal™ version 2.0 as the integrated environment for development (compiler, linker, editor and debugger).

The entire program consists of about 3,000 lines of source code, including documentation and blank lines.

The program is divided into the following modules (Figure B.1):

(a) *Generation of the graph representation*

(b) *Deterministic evaluation*

(c) *Probabilistic evaluation*, which is subdivided into:

  (1) *Generation of the symbolic Boolean expression*

  (2) *Evaluation of the probabilistic measures.*

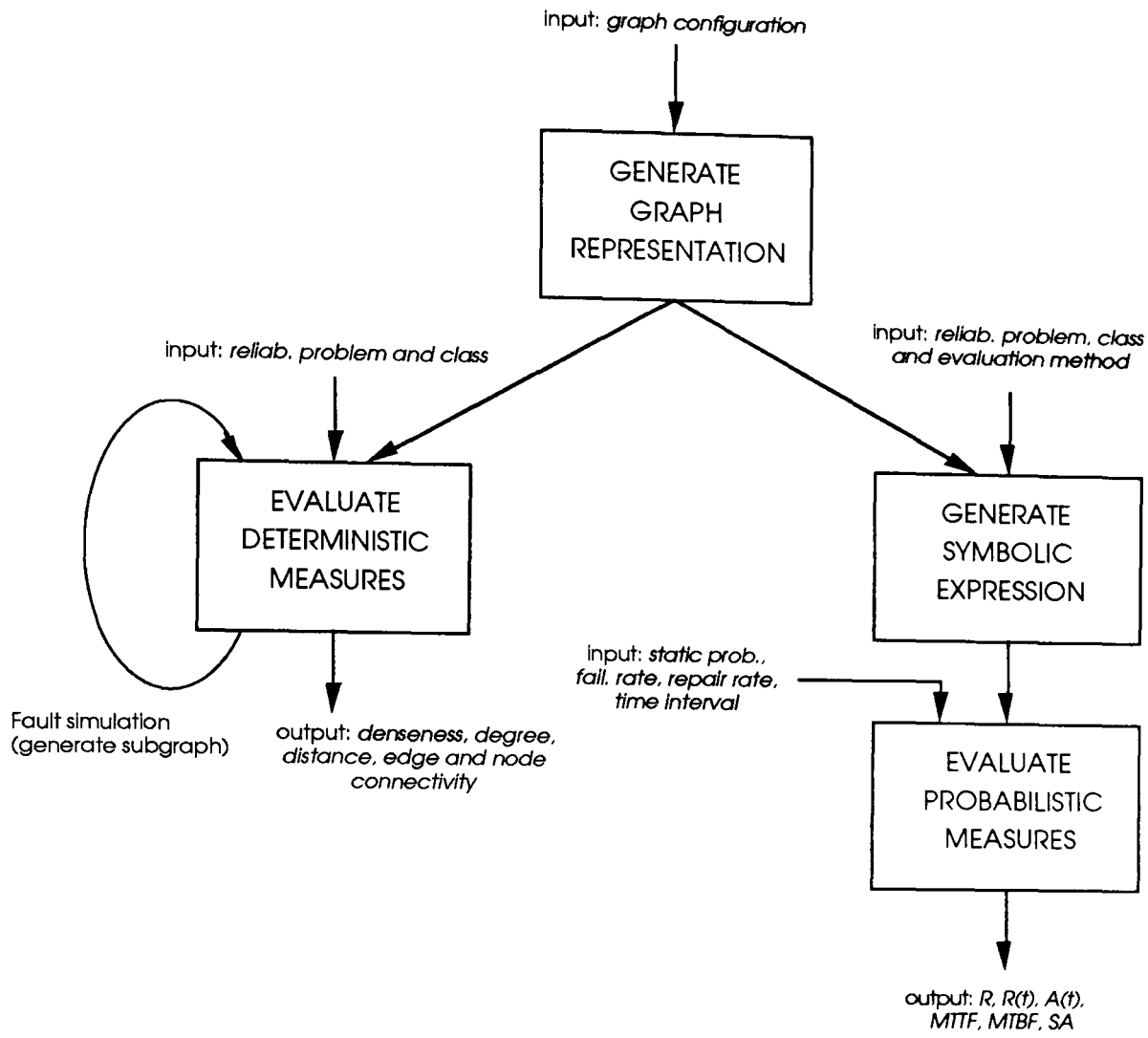


FIGURE B.1
Main program modules

Each of these modules is subdivided into units; each unit contains the global and/or local constants, variables, data structures and procedures corresponding to each module.

Figure B.2 shows the major units in each module. There is also another set of global units, which contains the common structures to all the modules; these structures are: global constants, variables and data structures and global procedures for error handling, initialisation and several graph manipulation routines: input a graph, transform its data representation, add and remove nodes and/or edges, generate a subgraph, obtain paths and trees, etc. It also contains other global procedures.

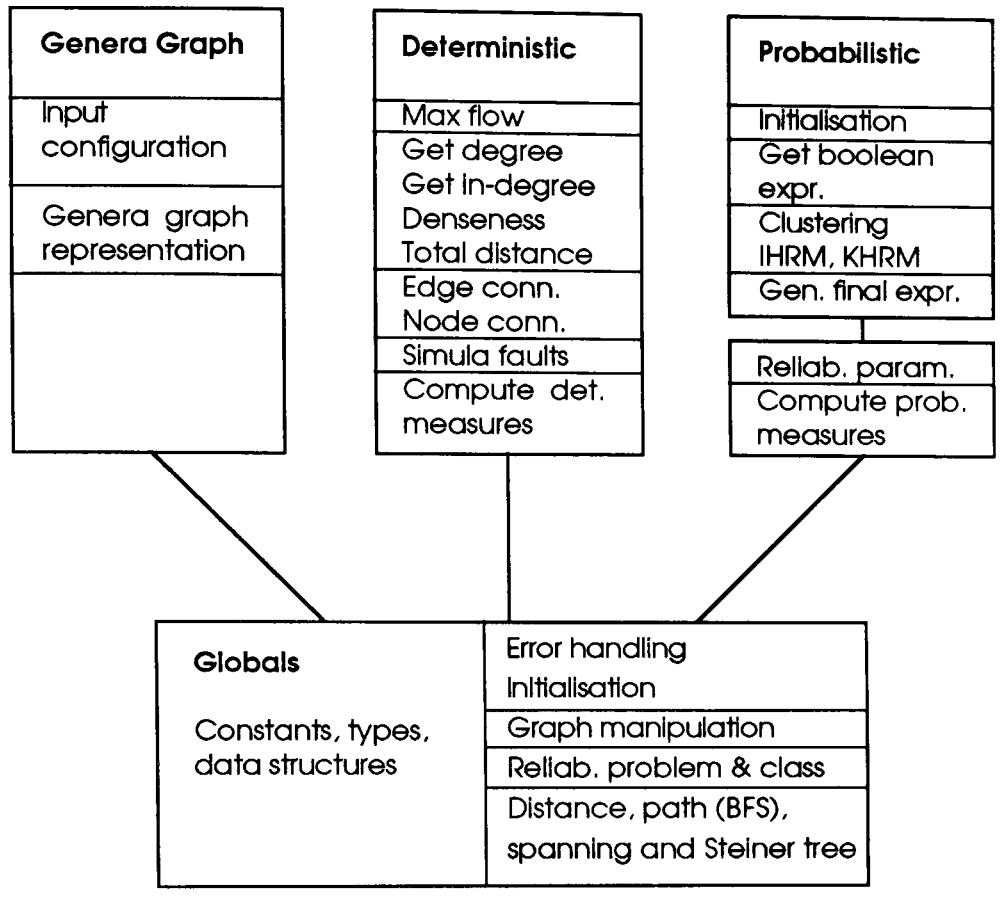


FIGURE B.2
Module units

Due to the memory and speed limitations of the Macintosh computer (Mac Plus with 68000 processor and 1M memory running at 6.7 MHz), the maximum number of nodes and edges combined cannot exceed 64 in our implementation.

The program has been coded in standard Pascal and the user interface for interactive input and output is text based, simple and self explained; therefore the program can be easily transferred to any other computer system. A more sophisticated graphics interface is out of the scope of this work.