A Monte Carlo Simulation Method for Systems With a Degraded State

Yoichi Watanabe

September 1986

UWFDM-644

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

A Monte Carlo Simulation Method for Systems With a Degraded State

Yoichi Watanabe

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

September 1986

UWFDM-644

A MONTE CARLO SIMULATION METHOD FOR SYSTEMS WITH A DEGRADED STATE

Yoichi Watanabe

Fusion Technology Institute
1500 Johnson Drive
University of Wisconsin-Madison
Madison, Wisconsin 53706

September 1986

UWFDM-644

TABLE OF CONTENTS

ABSTRACT

A computer program has been developed to simulate the availability of three-state systems with three-state components. The program utilizes a Monte Carlo method and systems trees. While the three-state model is useful to obtain more detailed performance prediction of systems, there are three major problems: lack of data on state transition probabilities, difficulty in obtaining gate logic, and high computing cost due to the use of a Monte Carlo method.

1. INTRODUCTION

An availability analysis of a system is usually performed by assuming the system and its subsystems are in either good (success) or bad (failed) states. This assumption leads to a simpler system model and consequently makes the problems analytically solvable by means of well-developed methods such as the Kinetic Tree Theory (KITT) [1].

When we look into system behavior more carefully, we discover that in many cases states of systems cannot be categorized into either good or bad. Let's consider a few examples. A fusion reactor plant is operated at full capacity (success state), and sometimes it completely fails to generate power (failed state). On many occasions, the plant may produce power partially, that is, at 40%, 60%, or 80% of the full capacity for some reason. These states (called degraded states) should be included in an availability analysis in order to estimate the plant availability with good accuracy and to predict the cost of electricity. At subsystem levels, it is reasonable to foresee that subsystems such as plasma heating systems (neutral beam injectors, wave heating systems) and magnetic coils will frequently go to degraded states. We may be able to say that the more sophisticated the system is the more likely the system will operate at least some of the time in degraded states. On the other hand, states of components which have rather simple structure can be easily divided into either good or bad states. As examples, consider electrical switches, pipes for liquid transportation, valves, and electric wires.

The necessity of availability analysis methods for systems with more than two states has been long recognized and investigated for the past ten years [2-34,65]. In Table 1, this literature is classified into four categories: general reliability analysis methods, availability analysis methods

Table 1. Classification of Literature on Reliability and Availability Analysis Methods for Multistate Systems

Category	References
Reliability analysis method	8, 9, 11, 13, 15, 25, 26, 31-34
Availability analysis method (repairable system)	2-4, 10, 14, 16, 18-21, 23, 27, 29, 30
Mathematical reliability theory	5, 8, 11, 17, 22, 24, 28, 65
Fault tree based reliability analysis method	6, 7, 12, 25, 32, 33

that are used to analyze reparable systems, mathematical theory of system reliability, and reliability analysis methods based on fault trees. There are few applications of the methods to complex systems except methods described in Refs. [19-21].

We are concerned with availability analysis of very large and complex systems such as future fusion reactor power plants. Also, detailed system operation modeling capability is desired. For these reasons, simulations by a Monte Carlo method have been carried out [35,36]. These programs employ a binary state system model; hence, in this paper we shall develop a system and operation model and a computer program utilizing a Monte Carlo method for multistate systems.

Going from binary state to multistate models in a Monte Carlo simulation is rather straightforward. However, larger computing cost and uncertainty of solutions due to data uncertainty might reduce the value of such a detailed simulation. Hence, we shall specifically address the following two topics in the present paper:

1. techniques to reduce the simulation cost,
2. advantages of multistate modeling.

In Chapter 2, a difference between continuous simulations and an availability simulation will first be clarified. Second, a governing equation for a one component system with a continuous state will be derived. The multistate equation will be considered as one of the special cases. To construct a system model containing many components and simulate it, we shall use a fault tree and the tree will be called the systems tree. The tree consists of gates and components. Several gates specific to three-state models will be proposed. Finally, the simulation methodology using Monte Carlo will be dis-

cussed. Chapter 3 will describe a computer program PROPA, utilizing the method discussed in Chapter 2. The program will be tested for a system with two gates and three components. The advantages of three-state models will be emphasized. A computing cost estimate will also be made for more complex systems. Chapter 4 will deal with programming improvements taking advantage of new computer hardware such as vector processing and multiprocessing capabilities of the Cray XMP and Cray 2 to reduce computing cost of the simulation. Further reduction of computing cost will be achieved by using variance reduction techniques. Chapter 5 will discuss the possible variance reduction techniques. Finally, Chapter 6 will conclude this work. Many future extensions will be suggested.

2. MODEL

2.1 Simulation and Availability Analysis

First we define basic terminologies used in this report.

system = everything in this world,

component = a system that composes the system being analyzed and whose internal structure is not considered,

subsystem = a system that is a subunit of the system being analyzed but is not a component.

Consider a system consisting of M components. Let $x_m(t)$ represent the state of component m. t is used to emphasize the time dependence of the state. A state of the system is associated with a vector $\underline{x}(t) = (x_1(t),\ldots, x_M(t))$. The state of the system, z(t), is given by

$$z(t) = \phi(\underline{x}(t)) \ . \tag{2.1}$$

In reliability theory, the function ϕ is called the structure function [1].

The time variation of the vector $\underline{x}$ can be obtained by solving a set of equations:

$$F_j^t(x_1(t),x_2(t),\ldots,x_M(t);c_1(t),\ldots,c_L(t)) = 0 \qquad j = 1,2,\ldots,J \tag{2.2}$$

where $c_\ell(t)$ ($\ell = 1,2,\ldots,L$) are parameters representing the time variation of the quality of a system. The superscript t of F indicates that the expression of the function F_j can vary. This represents a structural variation of a system.

In general two distinctive simulations are performed by using Eqs. (2.1) and (2.2). First, we assume that the time-variation of $c_\ell(t)$ and F_j^t is negligible for the time interval of interest. This is the simulation usually called time-dependent system simulation. For the second type of simulation, we assume the time variation of $x_m(t)$ due to the internal structure of a system is not important and consider the variation due to a quality change of the system, $c_\ell(t)$. This is the simulation usually called a reliability and availability analysis. There is a third type of simulation that simulates the time variation of a system due to the change of the system structure. This simulation is, however, not frequently performed, particularly in engineering. In the future this will become important if a machine can change its own structure to adapt to different environments or for other reasons.

Example. In order to clarify the above discussion, let us consider a simple electrical system illustrated in Fig. 2.1(a). The state of the system is the brightness of the lamp, W. Components are an electric battery V, a resistance R, an inductance L, and a resistance inside the lamp r. For this model, Eq. (2.2) becomes

$$V(t) = L(t)\,\frac{di(t)}{dt} + R_o(t)\,i(t) \tag{2.3}$$

where

$$R_o(t) = \left(\frac{1}{R(t)} + \frac{1}{r(t)}\right)^{-1}$$

and $i(t)$ is the electric current. Equation (2.1) is given by

$$W(t) = \alpha\,\frac{R_o^2(t)}{r(t)}\,i(t)^2 \tag{2.4}$$

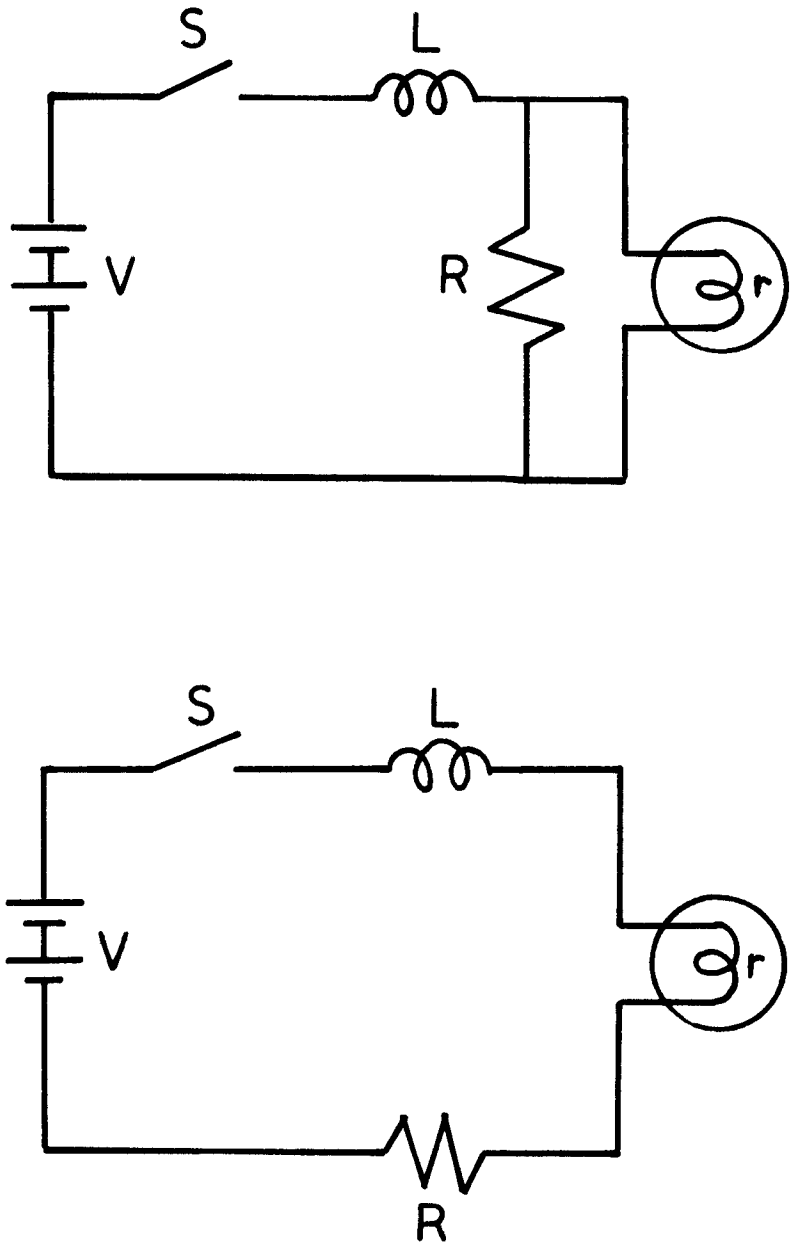

Fig. 2.1. Schematic diagram of an electric circuit.

where α is the constant converting the unit from power consumption rate to brightness.

For a standard simulation we assume that R, L, V, and r are independent of time. Then Eq. (2.3) can be solved by using an initial condition, for example, i(0) = 0. Substituting the solution of i(t) into Eq. (2.4), we have

$$W(t) = \alpha \frac{V^2}{r} \left[1 - \exp\left(-\frac{R_o}{L} t\right)\right]^2 . \tag{2.5}$$

The time variation of the brightness induced by turning the switch on and off is illustrated in Fig. 2.2. In the above analysis F_j is invariant; in other words, the circuit does not change its structure from one shown in Fig. 2.1(a) to a different structure such as one shown in Fig. 2.1(b).

Assuming the invariant F_j, let us take into account the time variations of L, V, R, and r. These variations are due to degradation of the components and repair of failed components. In this case Eqs. (2.3) and (2.4) can be solved. To simplify the analysis, however, we choose a representative state of the system; for example, W(t) at t = T given in Fig. 2.2. Here W(t) is given by

$$W(t) = \alpha \frac{V(t)^2}{r(t)} \left[1 - \exp\left(-\frac{R_o(t)}{L(t)} T\right)\right]^2 . \tag{2.6}$$

This equation can be used for the availability simulation.

2.2 General Model of Component State Transition

Let us define

w(x→y;t): transition probability of the system from state x to state y at time t.

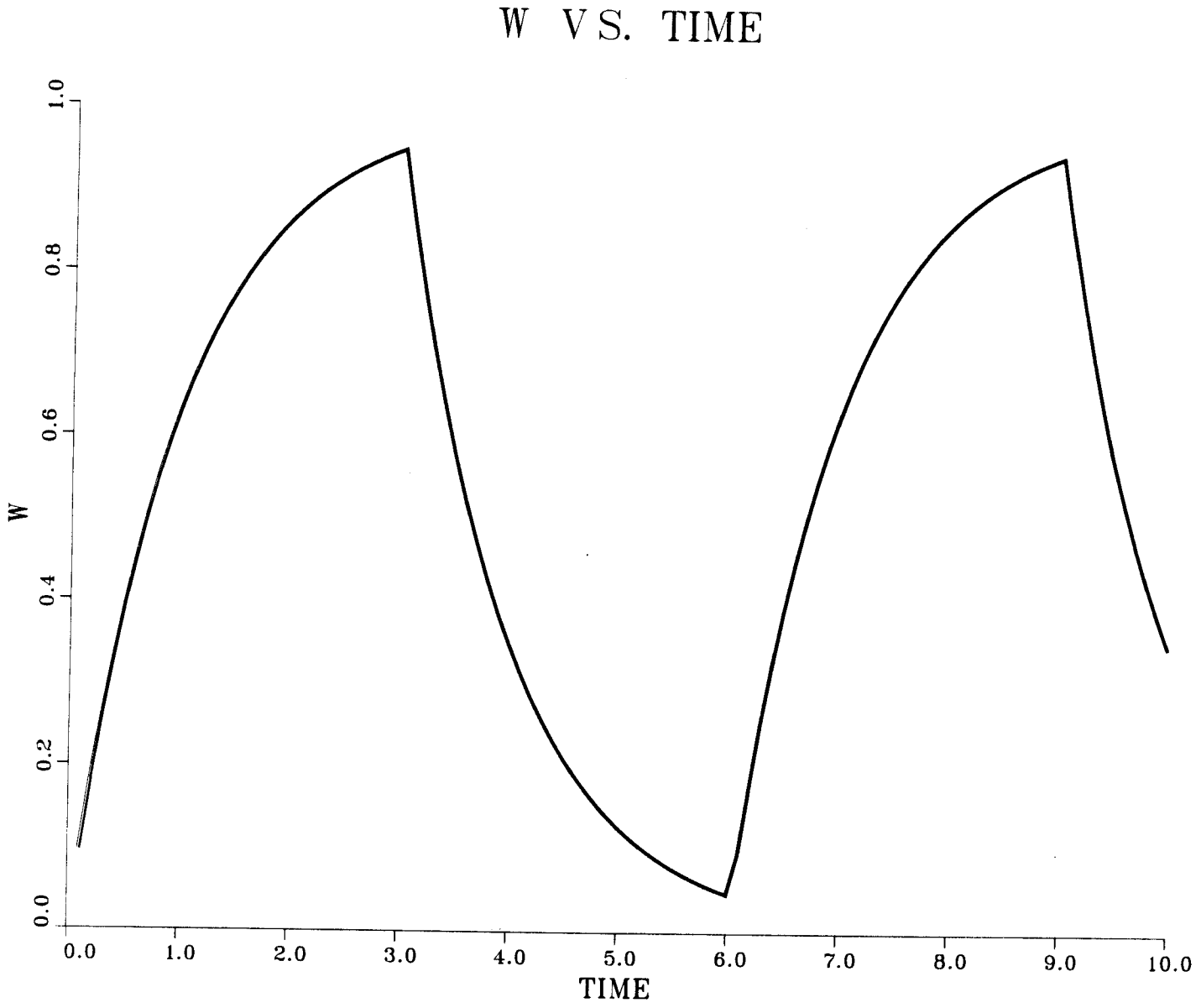

Fig. 2.2. Time variation of function W(t).

$K(y,s|x,t)$: transition probability of the system from state y at time s to state x at time t for the first time.

$f(x,t)$: state probability density that the system is in state x at time t.

Then the following equation can be obtained:

$$\frac{\partial f(x,t)}{\partial t} = \int_D w(z \to x;t)\, dz - \int_D w(x \to y;t)\, dy \tag{2.7}$$

where
$$w(x \to y;t) = \int_0^t ds \int_D dz\, K(x,s|y,t)\, w(z \to x,s) \tag{2.8}$$

and D is the range of the state variable.

For discrete states,

$$\frac{d}{dt} f_i(t) = \sum_{\substack{j \\ j \neq i}} w_{ji}(t) - \sum_{\substack{\ell \\ \ell \neq i}} w_{i\ell}(t) \tag{2.9}$$

$$w_{ji}(t) = \sum_k \int_0^t K_{ji}(s|t)\, w_{kj}(s)\, ds \tag{2.10}$$

where
$$f_i(t) = f(x_i,t)$$
$$w_{ij}(t) = w(x_i \to x_j;t)$$
and
$$K_{ij}(t|s) = K(x_i,t|x_j,s)\ .$$

Integrating Eq. (2.9) over [0,t] with respect to t, we have

$$f_i(t) = \sum_{\substack{j \\ j \neq i}} W_{ji}(t) - \sum_{\substack{\ell \\ \ell \neq i}} W_{i\ell}(t) + f_i(0) \tag{2.11}$$

where
$$W_{ij}(t) = \int_0^t w_{ij}(s)\, ds\ .$$

Equation (2.11) is identical to Eq. (5) in Ref. [3]. Equations (2.9) and (2.10) lead to a set of basic equations for binary state models (see Appendix A).

Returning to the continuous state equation (2.7), we shall show that under some assumptions this equation becomes the neutron slowing down equation well known in nuclear reactor theory. Let us assume that the transition probability is proportional to the state probability density at the state:

$$w(z \to x;t) = b(z \to x)\ f(z,t)\ . \tag{2.12}$$

Furthermore, assume that the state always decreases (or degrades): $b(z \to x) = 0$ for $z < x$. Also, let $a(x)$ be defined by

$$a(x) = \int_0^x b(x \to y)\ dy\ . \tag{2.13}$$

Then Eq. (2.7) becomes

$$\frac{\partial}{\partial t} f(x,t) + a(x)\ f(x,t) = \int_x^\infty b(y \to x)\ f(y,t)\ dy\ . \tag{2.14}$$

This is the fast neutron transport equation in an infinite medium [37].

Now let $C(t)$ be defined by

$$C(t) = \int_0^\infty f(x,t)\ dx\ . \tag{2.15}$$

$C(t)$ must be constant so that $f(x,t)$ given by Eq. (2.14) actually represents the state probability density. The constancy of the $C(t)$ is proven as fol-

lows. Multiply Eq. (2.13) by $f(x,t)$ and integrate it with respect to x over $[0,\infty)$. Then we have

$$\int_0^\infty a(x)\ f(x,t)\ dx = \int_0^\infty dx \int_x^\infty dy\ b(y\to x)\ f(y,t)\ . \tag{2.16}$$

Then by Eqs. (2.14) and (2.16),

$$\frac{d}{dt}\ C(t) = \int_0^\infty \frac{\partial}{\partial t}\ f(x,t)\ dt = 0\ . \tag{2.17}$$

Thus $C(t)$ is constant.

We can obtain analytical solutions for special cases using a standard technique. Such solutions are useful to see how a state probability density function of a nonreparable component varies. The solutions are obtained and a picture of $f(x,t)$ is given in Appendix B.

2.3 Three-State Model

As a specific case, let us consider a three discrete state model. Those states are good, degraded, and bad states and are denoted by 2, 1, and 0, respectively. The state transition diagram of a system is shown in Fig. 2.3. Transitions from 2 to 1, 2 to 0, and 1 to 0 are due to degradation, while transitions from 0 to 2, 0 to 1, and 1 to 2 are due to repair or replacement.

Although non-Markov processes will be considered in our Monte Carlo simulation model, let us assume that the transition is Markovian in the remainder of this section. For this case, λ_{ij} $(i \neq j)$ is the failure rate from state i to j and μ_{ij} $(i \neq j)$ is the repair rate from state i to j (see Fig. 2.3). Let $P_i(t)$ be the state probability that the system is in state i at time t. Then the following equations can be obtained:

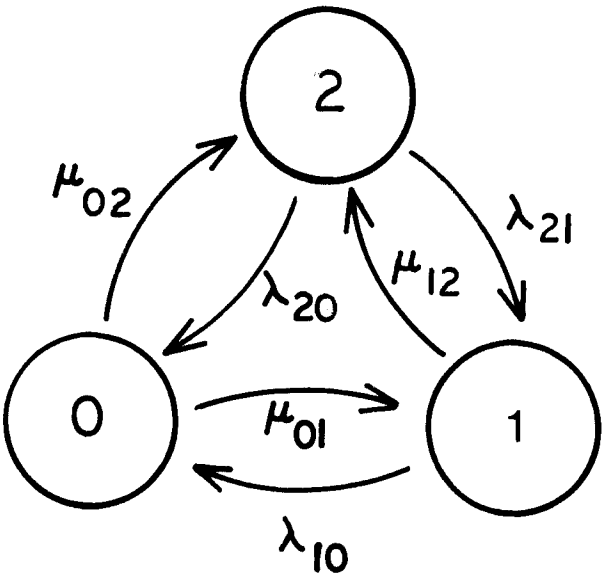

Fig. 2.3. Three-state transition diagram.

$$\frac{dp_0}{dt} = -(\mu_{01} + \mu_{02})p_0 + \lambda_{10}p_1 + \lambda_{20}p_2$$

$$\frac{dp_1}{dt} = \mu_{01}p_0 - (\lambda_{10} + \mu_{12})p_1 + \lambda_{21}p_2 \qquad (2.18)$$

$$\frac{dp_2}{dt} = \mu_{02}p_0 + \mu_{12}p_1 - (\lambda_{20} + \lambda_{21})p_2 \ .$$

Since a general solution of Eqs. (2.18) is rather complicated, in this section we shall present a solution for a special case: no repair, $\mu_{ij} = 0$ and initially $p_0(0) = p_1(0) = 0$, $p_2(0) = 1$. The solution is

$$p_0(t) = 1 + \frac{\lambda_{21}}{\lambda_{10} - \lambda_{20} - \lambda_{21}} e^{-\lambda_{10}t}$$

$$- \frac{1}{\lambda_{20} + \lambda_{21}} \left(\lambda_{20} + \frac{\lambda_{10}\lambda_{21}}{\lambda_{10} - \lambda_{20} - \lambda_{21}}\right) e^{-(\lambda_{20}+\lambda_{21})t}$$

$$p_1(t) = \frac{\lambda_{21}}{\lambda_{10} - \lambda_{20} - \lambda_{21}} \{-e^{-\lambda_{10}t} + e^{-(\lambda_{20}+\lambda_{21})t}\} \qquad (2.19)$$

$$p_2(t) = e^{-(\lambda_{20}+\lambda_{21})t} \ .$$

See Appendix C for more general solutions.

2.4 Systems Tree and Gates

In this section a system consisting of components will be discussed. As discussed in Section 2.1, the state of a system is given by using a structure function ϕ as a function of the component state vector $\underline{x}$, Eq. (2.1). Hence, in principle we can find the system state variation by knowing the components' state variation $\underline{x}(t)$ and the function ϕ.

Since the structure function ϕ becomes very complicated for large systems and on many occasions ϕ cannot be obtained, we shall take a different approach utilizing fault trees [1]. In our work we call tree systems tree. We assume that the state of a system is represented by one variable. Gates (i.e., nodes) and the lowest branches in a tree are associated with a system, either components or subsystems. The top gate (there is only one top gate) represents the state of the entire system.

To construct a systems tree, first the state of a system must be defined. Then subsystems affecting the system state are identified. Subsystems of the subsystem are identified next. This procedure is continued until the component level is reached.

An example of a systems tree is illustrated in Fig. 2.4. System A is composed of components 1 to 10. Components 1 and 2 are parts of subsystem B, components 3, 4, and 5 of subsystem C. Components 7 and 8 and components 9 and 10 make up subsystems E and F, respectively. Subsystem D is composed of component 6 and subsystems E and F. The second level of system A consists of subsystems B, C, and D. There are four levels. Symbol ○ denotes a component. Symbol ⌔ represents a subsystem. This symbol is called a gate. A gate is associated with a specific logic which relates the system state (i.e., output state) with input states. This is a generalization of the binary gates such as "and" and "or" gates.

Since a gate and its inputs make a subsystem, the output state is represented by an equation similar to Eq. (2.1). The state of the entire system is represented by a function of states of components and subsystems that are inputs to the gate. At the next level, similar equations are obtained. The equation, in general, can be represented as

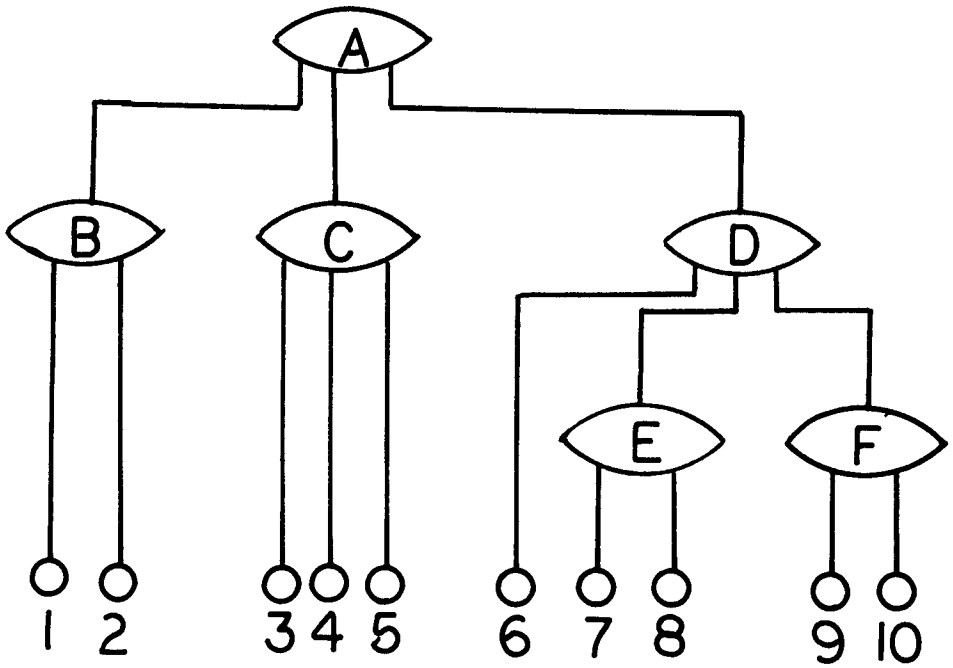

Fig. 2.4. Systems tree.

$$F_i(z_i; x_{p_{i1}}, x_{p_{i2}}, \ldots, x_{p_{iK_i}}; z_{q_{i1}}, z_{q_{i2}}, \ldots, z_{q_{iL_i}}) = 0 \qquad (2.20)$$

where $x_{p_{ij}}$ is the state of component p_{ij}, $z_{q_{ij}}$ is the state of subsystem (or gate) q_{ij}. In Eq. (2.20), $\sum_{i=1}^{M} K_i = N$, $\sum_{i=1}^{M} L_i = M - 1$ where N is the number of components and M is the number of gates. p_{ij} and q_{ij} denote the j-th component for the i-th gate and the j-th gate for the i-th gate, respectively.

For many cases Eq. (2.20) can be solved for z_i to give

$$z_i = \phi_i(\cdot) \qquad (2.21)$$

where the arguments of ϕ_i are the same as those in Eq. (2.20) except z_i. We call a gate having this property an explicit gate; otherwise it is called an implicit gate. For a continuous state model, Eq. (2.20) must be solved if it is implicit; otherwise z_i can be computed by using Eq. (2.21). For a discrete state model, a table can be given for possible combinations of $\{x_i, z_i\}$. In particular, AND, OR, and m-out-n voting gate logic tables have been constructed for a binary state model. For a multistate model, some multilogic gates have been used [7].

For convenience, we create three types of gates for multistate cases. Let a gate have I inputs and one output as illustrated in Fig. 2.5. The input states are denoted by x_i (i = 1,2,...,I), where $x_i \in \{0,1,\ldots,m\}$. The output state y is also one of the m + 1 states.

An AND gate is defined by the following logic:

$$y = \min_i \{x_i\} . \qquad (2.22)$$

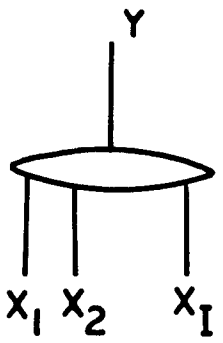

Fig. 2.5. Symbol of gate.

A SUM gate is defined as follows:

$$y = j - 1 \qquad \text{if } s_{j-1} < \sum_{i=1}^{I} x_i \leq s_j \tag{2.23}$$

where s_j (j=1,...,m+1) depends on the model. The OR gate in the binary state model is a type of SUM gate having $s_0 = s_1 = 0$ and $s_2 = I$.

A MAX gate is defined as:

$$y = \max_i \{x_i\} . \tag{2.24}$$

In addition to these standard gates, the concept of a virtual gate is useful. When there are some identical components in operation, this group of components is considered as one system and has one output state quantity; the output state is determined by the logic of the gate whose inputs are the states of the components. This gate is not included in the systems tree explicitly; hence this is called a virtual gate.

2.5 Computational Method

There are two types of computational methods for system availability analysis: deterministic methods and Monte Carlo methods [1]. The Monte Carlo methods have advantages as well as disadvantages. One of the advantages is the possibility of sophisticated modeling of large complex systems. A disadvantage is the computing cost needed to obtain sufficient accuracy for the solutions. In particular, obtaining the unreliability or unavailability of very highly reliable systems is extremely costly.

In Monte Carlo methods the reliability or availability of an entire system (i.e., the top event) can be obtained by using fault trees [35,36], system block diagrams [1,45], or the system state space [47].

There are two distinctive approaches for advancing time in a Monte Carlo simulation. In the first method, timesteps are chosen as the duration between times when a component changes its state [1,36]. In the second method, a fixed timestep is used [35,44].

In this work a Monte Carlo simulation method utilizing fixed timesteps and a systems tree will be developed for three-state systems with three-state components. The entire time duration for which a system is in operation, T, is partitioned into small time intervals with length Δt.

A simulation can be performed in the following way. Suppose we know the states of all components at a time t_ℓ. Then the states of the components at $t_\ell + \Delta t$ $(= t_{\ell+1})$ are obtained by using the following method. If a component is in state 2, a random number, r, is generated and the next state j is determined by:

$$j = \begin{cases} 0 & \text{if } r < p_{20} \\ 1 & \text{if } p_{20} \leq r < p_{20} + p_{21} \\ 2 & \text{otherwise} \end{cases} . \tag{2.25}$$

When a component is in state 1, there are two options: it is either repaired (or replaced) or not repaired. If it is not repaired, the next state j is determined by

$$j = \begin{cases} 0 & \text{if } r < p_{10} \\ 1 & \text{otherwise} \end{cases} . \tag{2.26}$$

If it is repaired with a certain repair rate, the next state j is determined by

$$j = \begin{cases} 2 & \text{if } r < p_{12} \\ 1 & \text{otherwise} \end{cases} . \tag{2.27}$$

When a component is in state 0,

$$j = \begin{cases} 2 & \text{if } p_{01} \leq r < p_{01} + p_{02} \\ 1 & \text{if } r < p_{01} \\ 0 & \text{otherwise} \end{cases} . \tag{2.28}$$

In Eqs. (2.25)-(2.28), p_{ij} is defined by

$$\begin{aligned} p_{21} &= \frac{\lambda_{21}(t_\ell)}{\lambda_{20}(t_\ell) + \lambda_{21}(t_\ell)} \{1 - \exp[-(\lambda_{20}(t_\ell) + \lambda_{21}(t_\ell))\ \Delta t]\} \\ p_{20} &= \frac{\lambda_{20}(t_\ell)}{\lambda_{20}(t_\ell) + \lambda_{21}(t_\ell)} \{1 - \exp[-(\lambda_{20}(t_\ell) + \lambda_{21}(t_\ell))\ \Delta t]\} \\ p_{10} &= 1 - \exp(-\lambda_{10}(t_\ell)\ \Delta t) \\ p_{01} &= \frac{\mu_{01}(t_\ell)}{\mu_{01}(t_\ell) + \mu_{02}(t_\ell)} \{1 - \exp[-(\mu_{01}(t_\ell) + \mu_{02}(t_\ell))\ \Delta t]\} \\ p_{02} &= \frac{\mu_{02}(t_\ell)}{\mu_{01}(t_\ell) + \mu_{02}(t_\ell)} \{1 - \exp[-(\mu_{01}(t_\ell) + \mu_{02}(t_\ell))\ \Delta t]\} \\ p_{12} &= 1 - \exp[-\mu_{12}(t_\ell)\ \Delta t] \end{aligned} \tag{2.29}$$

where $\lambda_{ij}(t_\ell)$ and $\mu_{ij}(t_\ell)$ indicate failure and repair rates at time t_ℓ. If a

component is repaired with a fixed time period, the next state j is set to a specific state when the fixed time period has passed since the last transition into the present state.

States of all components at time $t_{\ell+1}$ are obtained in this way. Next the states of the gates are consecutively obtained by using the input states and the gate logic beginning at the lowest level of the systems tree. Finally the state of the top gate that is equivalent to the system state can be obtained.

After this, the clock is advanced by Δt and the same procedure is repeated. If a scheduled maintenance period is encountered, the state of a component is forced to become a specified state. At the end of the maintenance, normal operation is resumed. One history of a simulation ends after time T has passed.

Then the i-th state availability of gates and components is computed by using

$$ {}_n\bar{A}_i = \frac{t_i}{T} \tag{2.30} $$

where t_i is the total time duration for which the system is in state i and n denotes the history number. Using ${}_n\bar{A}_i$, the average i-th state availability up to the n-th history is obtained by computing a sample mean [48]

$$ \bar{A}_i^n = \sum_{j=1}^{n} \frac{{}_j\bar{A}_i}{n} . \tag{2.31} $$

The uncertainty of the i-th state availability, C_i^n, is given by

$$ C_i^n = \sqrt{S_i^2/n} \tag{2.32} $$

where the sample variance is ${}_j\bar{A}_i$

$$S_i^2 = \sum_{j=1}^{n} \frac{({}_j\bar{A}_i - \bar{A}_i^n)^2}{(n-1)} . \tag{2.33}$$

The confidence interval of the sample mean of the state availability $\bar{A}_i^n$ can be derived according to the arguments in 9.2.2 of Ref. [48]. The confidence interval (L,U) is defined as an interval which satisfies

$$P(L < \bar{A}_i^n < U) \geq 1 - \alpha , \qquad 0 < \alpha < 1 \tag{2.34}$$

where P is the probability that $\bar{A}_i^n$ is in the interval (L,U). By means of the uncertainty C_i^n, L and U are given as follows:

$$\begin{aligned} L &= \bar{A}_i^n - t_{1-\alpha/2;n-1}\, C_i^n \\ U &= \bar{A}_i^n + t_{1-\alpha/2;n-1}\, C_i^n \end{aligned} \tag{2.35}$$

where $t_{1-\alpha/2;n-1}$ is the value at $t = 1 - \alpha/2$ of the cumulative distribution function of the students t-distribution with n degree of freedom.

As an example, let us calculate the 95% confidence interval. Now α is 0.05. Since n is usually bigger than 100, it is reasonable to choose 1.980 as a value of $t_{0.975;n-1}$. Thus the 95% confidence interval is given by

$$(\bar{A}_i^n - 1.980\, C_i^n, \bar{A}_i^n + 1.980\, C_i^n) . \tag{2.36}$$

The effective availability $\bar{A}$ is defined by

$$\bar{A} = \alpha \bar{A}_2^n + \beta \bar{A}_1^n + \gamma \bar{A}_0^n \tag{2.37}$$

where $$0 \leq \alpha, \beta, \gamma \leq 1 .$$

The variance of $\bar{A}$ is given by

$$V(\bar{A}) = \alpha^2 V(\bar{A}_2^n) + \beta^2 V(\bar{A}_1^n) + \gamma^2 V(\bar{A}_0^n) + 2\alpha\beta \, Cov(\bar{A}_2^n, \bar{A}_1^n) + 2\beta\gamma \, Cov(\bar{A}_1^n, \bar{A}_0^n) + 2\gamma\alpha \, Cov(\bar{A}_0^n, \bar{A}_2^n) \tag{2.38}$$

where $$Cov(x,y) = E(xy) - E(x) \, E(y) . \tag{2.39}$$

3. COMPUTER PROGRAM PROPA

3.1 System Model

A computer program PROPA (Probabilistic Performance Analysis Program) has been developed utilizing the following system model.

1. Systems assume three states: good, degraded, and bad states. Good, degraded, and bad states are denoted by 2, 1, and 0, respectively.
2. A component fails from state 2 to 1, 2 to 0, and 1 to 0 with failure rates λ_{21}, λ_{20}, and λ_{10}.
3. A component can be repaired from state 0 to 1, 0 to 2, and 1 to 2 with specified fixed repair time τ_{01}, τ_{02}, and τ_{12}. Note that only one of two repairs (from state 0 to 1 and 0 to 2) is possible at a time.
4. A component may be replaced by a new unit within a certain time interval.
5. Subsystems (i.e., gates) can be replaced within a specific time interval. At present, however, replaceable gates can contain only components.
6. Spare systems are supplied within a specific time interval; in other words, some systems may be lacking spares in some period.
7. Out-of-line spare parts are not subject to normal degradation.
8. If there are n identical units, they can be treated as a subsystem composed of these n units. The logic of the virtual gate is specified by the modeler.
9. Scheduled maintenance can be carried out for components separately. During maintenance the component is in one of three states.

3.2 Program Features

A. Input Data

a. Selection of options

1. flag for systems tree plotting
2. flag for capital cost calculation
3. flag for variance reduction techniques

b. Simulation data

1. the total number of histories for a simulation
2. total time duration for one history
3. timestep length
4. an integer specifying the frequency of output (top event availability) during a simulation: NPRNT

c. System specification data

1. the number of components
2. the number of gates
3. the number of levels
4. gate ID number, name, level, number of inputs, gate type, s_1 and s_2 for SUM gate, replacement option flag, number of spares
5. gate ID number, numbers of gate inputs and component inputs, input gate ID numbers, input component ID numbers
6. component ID number, name, type of virtual gate, s_1 and s_2 for SUM gate, state after repair, the number of spares, the number of scheduled maintenance
7. time interval for renewal of spares

d. Reliability and repair data for components

   1. λ_{20}, λ_{21}, λ_{10}

   2. τ_{01}, τ_{02}, τ_{12}, τ_{00} (time required to replace a bad component), τ_{11} (time required to replace a degraded component)

   3. source of the data

e. Scheduled maintenance data for components

   1. start time of maintenance

   2. end time of maintenance

   3. component state during maintenance

f. Cost data

   1. component capital cost

   2. scheduled maintenance cost for components

B. Output

a. Input data

b. A picture of the systems tree

c. Capital cost of the system

d. Top event state availabilities printed every NPRNT histories

e. Final results

   1. state availabilities and uncertainties of components and gates

   2. effective availabilities and uncertainties for components and gates

   3. ranking of gates and components according to their effective availabilities

f. Computing time in seconds

1. central processor unit time
2. input and output
3. system

C. Special Feature -- To make a continued run possible, state availabilities, their uncertainties, and the last random number generated are written in a binary file OFILE. This file can be used for the next run by renaming if BFILE to continue the simulation when better accuracy is required.

3.3 Program Structure

A flow diagram and a block diagram of the PROPA program are shown in Figs. 3.1 and 3.2, respectively. The functions of subroutines are as follows:

PROPA: main program

TREE: reads systems tree data and find the top gate ID number

PLTTRE: draws the systems tree

PLGAT: draws a picture of a gate

GATEDAT: reads data on gates

RELDAT: reads data on failure and repair/replacement, and computes transition probabilities

MAINTDAT: reads data on scheduled maintenance

CALC: repeats the Monte Carlo simulation and computes state availabilities and uncertainties

HIST: computes states of components/units and gates at a timestep

STATE: computes state of components/units for the next timestep by playing a Monte Carlo game

NGATE33: computes gate output state from input states

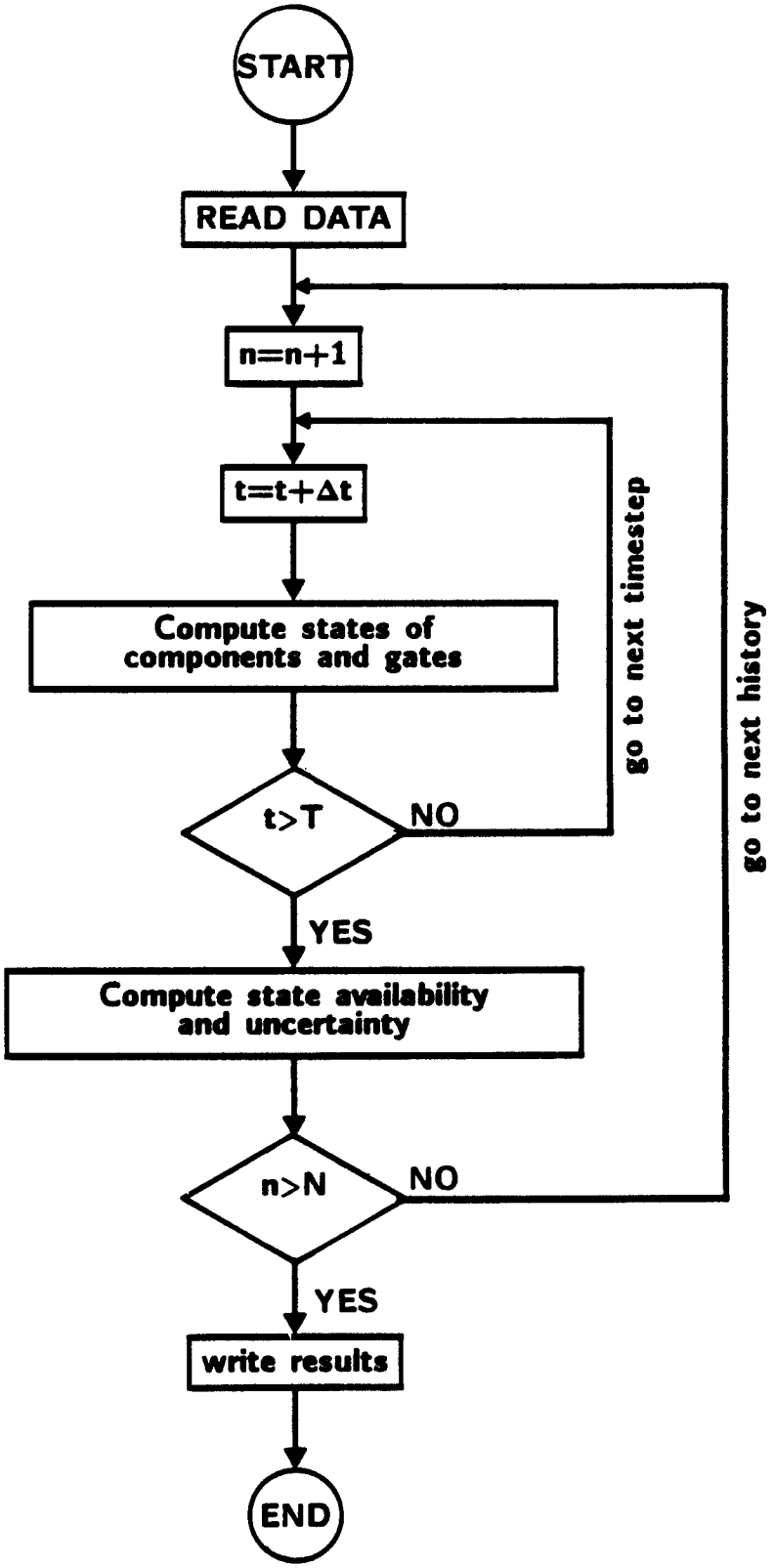

Fig. 3.1. Simplified flowchart of the PROPA program.

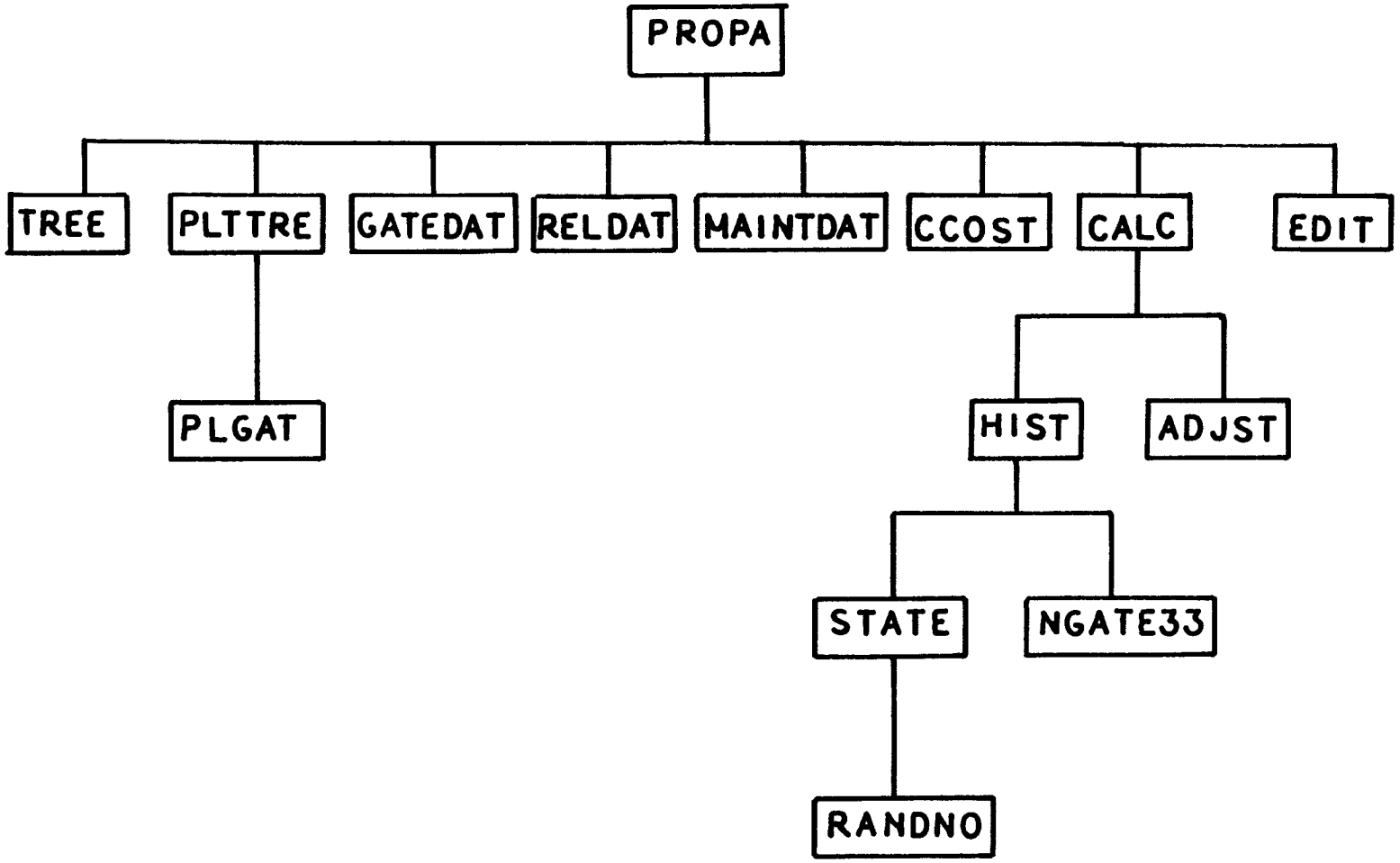

Fig. 3.2. Program structure of PROPA.

ADJST: computes adjusted time duration for components and gates using weights of a variance reduction technique

EDIT: prints the final results on availability and uncertainty

RANDNO: generates random numbers using the recursive congruential method [35]

CCOST: reads cost data and computes the system cost

3.4 Program Testing

To demonstrate the PROPA program, we simulate the performance of the illumination system illustrated in Fig. 2.1(a). Since we are interested in steady-state operation, the effect of resistance r and inductance L can be neglected. Then there are three components in this system: resistance r, battery V, and switch S. The systems tree is shown in Fig. 3.3. We introduce two gates: gate 1 represents the system output, i.e. the brightness of the light bulb, and gate 2 represents the power consumption rate, W, when the switch is working properly.

The logic of gate 2 can be determined by examining a diagram of W as a function of r and V as shown in Fig. 3.4. Now suppose an operating point is located at point A in Fig. 3.4. As we see, decrease of V affects W significantly, while a reduction in r affects W less. Clearly, zero r is destructive for the system. Hence, the logic shown in Table 3.1(a) can be derived.

To find the logic of gate 1, first it is noted that the state of the switch should be binary: good (2) or bad (0), because a degraded switch is most likely to make the system unusable. Hence, we obtain the logic shown in Table 3.2.

Table 3.1. Logic of Gate 2

a)

r \ V	0	1	2
0	0	0	0
1	0	1	2
2	0	1	2

b)

r \ V	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

Table 3.2. Logic of Gate 1

W \ S	0	1	2
0	0	-	0
1	0	-	1
2	0	-	2

Table 3.3. Reliability and Repair Time Data

Component	λ_{20}	λ_{21}	λ_{10}	τ_{02}	τ_{12}	τ_{01}
1	1.0E-5	2.0E-5	2.0E-5	24.0	N.A.	N.A.
2	4.0E-4	8.0E-4	8.0E-4	240.0	N.A.	N.A.
3	2.0E-3	N.A.	N.A.	24.0	N.A.	N.A.

Note: failure rate λ [1/hr], repair time τ [hr]

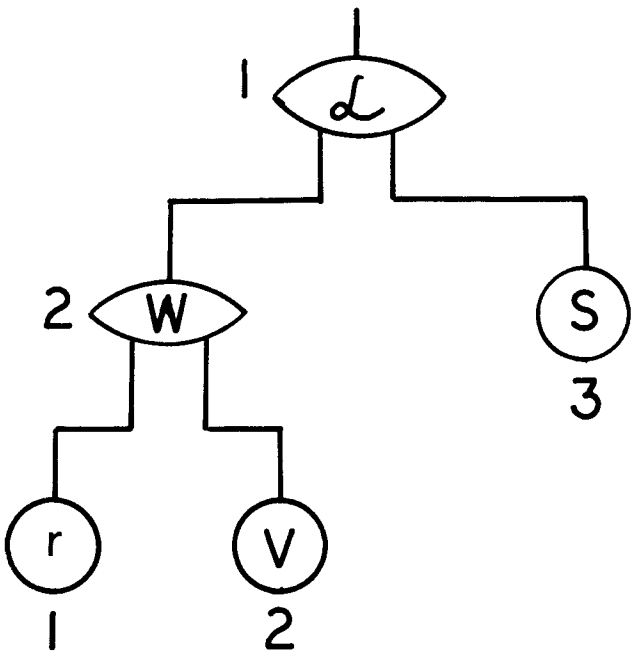

Fig. 3.3. Systems tree of sample problem.

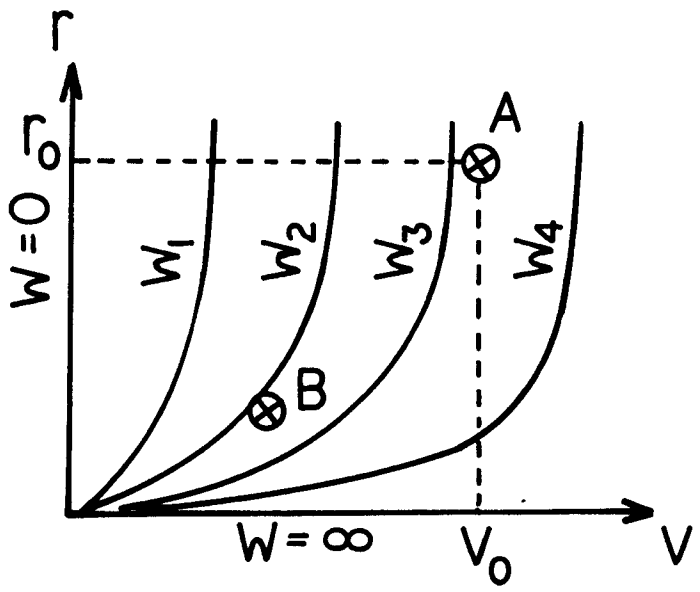

Fig. 3.4. W versus r and V, $W_1 < W_2 < W_3 < W_4$.

We use the reliability and repair time data shown in Table 3.3, unless otherwise stated. A series of simulations are carried out by varying some variables.

First, a computation for a binary state model is performed by setting λ_{21} to 1.0×10^{-99} for the resistance and the battery. This is case 1. Cases 2 to 12 will now be discussed.

Case 2 (base case): All components in state 0 are repaired to state 2. No scheduled maintenance is performed.

Case 3: Same as case 2 except a battery in state 0 is replaced by a new unit within 24 hours.

Case 4: Same as case 3 except a battery in state 1 is also replaced by a new unit within 24 hours.

Case 5: Same as case 4 except a battery in state 1 is replaced by a new unit instantaneously.

Case 6: Same as case 2 except now there are two identical batteries connected in parallel. The gate used to obtain the state of these batteries as one system is a MAX gate.

Case 7: Same as case 6 except that there are three identical batteries.

Case 8: Same as case 6 except that there are four identical batteries.

Case 9: Same as case 2 but scheduled maintenance is performed. Maintenance of length 100 hours is carried out once a year, six months from start for all components. The state during maintenance is set to 0. In addition, the battery is maintenanced every month with one day duration.

Case 10: Same as case 9 except the monthly maintenance of the battery is carried out by setting the state to 1.

Case 11: Same as case 2, but the operating point of the resistance and the battery is chosen at point B in Fig. 3.4. Consequently the logic shown in Table 3.1(b) is used for gate 2.

For cases 1 to 11, 1000 trial runs are carried out for 8760 hours (1 year) of operation. For case 12, 200 trial runs are carried out for 87600 hours (10 years) using the same data as the base case. For all the cases the timestep for the Monte Carlo simulations is 24 hours. For convenience, data on the battery is shown in Table 3.4. Effective availabilities of components and gates (notice gate 1 is the entire system) are shown in Table 3.5. Since state availabilities have uncertainties less than 5%, the accuracy of effective availabilities is good enough to make comparisons among different cases.

Observations of the results lead to the following conclusions:

1. The binary state model results in much higher availability (compare case 1 with case 2).
2. Replacement of a failed battery instead of repair slightly increases the system availability. A more effective way, however, is to replace a degraded battery. Instantaneous replacement does not significantly improve the system availability compared with one day replacement (compare cases 2 through 5).
3. Adding redundant batteries in parallel is a very effective way to improve availability (compare cases 2, 6, 7, and 8).
4. Frequent short scheduled maintenance of the battery, in fact, increases the system availability. The state of the battery during the maintenance is insignificant because the total length of the maintenance is much shorter than the total operation time (compare cases 2, 9, and 10).

Table 3.4. Data on Battery

Case	Repair Time, τ_{02} hrs	Replacement Time, τ_{00} hrs	Replacement Time, τ_{11} hrs	Number of Units	State at Scheduled Maintenance
1	240.	N.A.	N.A.	1	N.A.
2	240.	N.A.	N.A.	1	N.A.
3	N.A.	24.	N.A.	1	N.A.
4	N.A.	24.	24.	1	N.A.
5	N.A.	24.	0.	1	N.A.
6	240.	N.A.	N.A.	2	N.A.
7	240.	N.A.	N.A.	3	N.A.
8	240.	N.A.	N.A.	4	N.A.
9	240.	N.A.	N.A.	1	0
10	240.	N.A.	N.A.	1	1
11	240.	N.A.	N.A.	1	N.A.
12	240.	N.A.	N.A.	1	N A.

Table 3.5. Effective Availability

Case	Resistance	Battery	Switch	Resistance + battery	System
1	99.981	91.4	95.6	91.3	87.3
2	95.8	67.8	95.6	67.8	64.8
3	96.4	75.5	95.6	75.5	72.2
4	96.6	98.2	95.6	98.1	93.8
5	96.2	99.1	95.6	99.0	94.7
6	95.6	82.3	95.6	85.2	81.5
7	95.4	92.3	95.6	92.3	88.2
8	96.6	95.9	95.5	95.9	91.6
9	96.3	78.4	95.3	78.4	76.2
10	96.3	80.6	95.3	80.6	76.3
11	95.9	67.6	95.6	65.8	63.0
12	80.5	65.9	95.6	65.8	63.0
AVSYS*	99.98	91.5	95.5	91.5	87.4

*The solution of the AVSYS program (a binary state model) [35].

5. Increasing the total length of simulation from one year to 10 years results in a very different availability of the resistance. Since the mean-time-to-failure of this component is about 10 years, one year is too short to observe a failure of the resistance. Fortunately, the system availability does not change much for the longer length of simulation; hence, the discussions so far are still correct.
6. There is a possibility of finding the best operating point of a system from the availability point of view by using this program as indicated in case 11; operating point A is better than operating point B although the difference is small.

Central processor times are shown in Table 3.6 for the 12 cases. These are measured by calling a utility routine TIMEUSED in the FORTLIB library on Cray 1 computers at the National Magnetic Fusion Energy Computer Center.

The input and output files of the PROPA program for case 9 are reproduced in Figs. 3.5 and 3.6.

3.5 Computing Cost

Central Processor Unit (CPU) time for a simulation, T_c, can be estimated by the following formula:

$$T_c = \alpha N_s N_h N_g N_c + \beta \qquad \text{[minutes]} \tag{3.1}$$

where α and β are constants, N_s is the total number of timesteps, N_h is the total number of histories, N_g is the total number of gates, N_c is the total number of components.

To compute α and β, consider two very different systems: the first system is case 2 discussed in Section 3.4, and the second case is a fusion

Table 3.6. CPU Seconds

Case	CPU
1	33.2
2	33.3
3	33.4
4	33.4
5	33.3
6	39.2
7	42.7
8	46.2
9	32.6
10	32.1
11	---
12	66.2

Table 3.7

Case	N_s	N_h	N_g	N_c	T_c (minutes)
1	365	1000	2	3	0.55
2	365	100	12	109	1.15

```
test problem 1  (case 5) - 05/27/85
iccost=1
ncomp=3
ngate=2
level=2
nhist=1000
ibias=0
nprnt=100
ipltree=0
mantmx=12
trenew=0.
tstep=24.0
total=8760.0
$
  1 system          1   2 -1   0   0   0   0
  2 power           2   2 -1   0   0   0   0
 0 0 0 0 0 0 0 1 2
 0 0 0 0 1 1 0 2 2
  1   1   1
  2   3
  2   0   2
  1   2
  1 resistance      1   0   0   0   2   0   1   0
  2 battery         1   0   0   0   2   0  12   0
  3 switch          1   0   0   0   2   0   1   0
          0.0         2.0e-5          1.0e-5 arbitary
          0.0            0.0          2.0e-5 arbitary
         24.0            0.0             0.0 arbitary
          0.0         8.0e-4          4.0e-4 arbitary
          0.0            0.0          8.0e-4 arbitary
        240.0            0.0             0.0 arbitary
          0.0        1.0e-99          2.0e-3 arbitary
          0.0            0.0         1.0e-99 arbitary
         24.0            0.0             0.0 arbitary
0
 4380.0   4548.0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  720.0    744.0   1440.0   1464.0   2160.0   2184.0   2880.0   2904.0   3600.0   3624.0
 4380.0   4548.0   5268.0   5292.0   5988.0   6012.0   6708.0   6732.0   7428.0   7452.0
 8148.0   8172.0   8736.0   8760.0
0
 4380.0   4548.0
   1          9.5      180.0
   2         55.2      125.0
   3         12.0      125.0
```

Fig. 3.5. Sample input for PROPA.

```
**********************************************************************************************************************
***  program propa1.3  (06/26/85) *


-  test problem 1  (case 5) - 06/24/85
 (ncomp/ngate/level/nhist)=          3/         2/         2/      1000/

- time step for simulation =     24.000 hours
- total time interval for simulation =   8760.000 hours


- no variance reduction is utilized


- gate information -

  id  name           level   input no.   type   nsumg   nsumg   replace opt.   number       time for
                                                for 0   for 1   for 0/1        of spares    replace. hrs
   1  system           1         2         -1      0       0         0            0              0.00
   2  power            2         2         -1      0       0         0            0              0.00

- gate logic input -

 gate id      gate output
     1      0 0 0 0 0 0 0 1 2
     2      0 0 0 0 1 1 0 2 2

- systems tree data -

 gate id      input gate id/ input comp. id
       1          2
                  3
       2
                  1  2

- component data -

 id      name         number     logic      nsum       nsum       replacement type     number of      number       importance
                      of units   type       for 0      for 1      for state 0 and 1    sch. maint.    of spares    factor
  1     resistance        1          0          0          0          2   0                  1             0          0.0000
  2     battery           1          0          0          0          2   0                 12             0          0.0000
  3     switch            1          0          0          0          2   0                  1             0          0.0000

- 3 states transition matrices (input) -

id     (0,0)        (1,0)        (2,0)        (0,1)        (1,1)        (2,1)        (0,2)        (1,2)        (2,2)        data base
1       0.0000   2.0000e-05   1.0000e-05      0.0000       0.0000   2.0000e-05     24.0000       0.0000   0.             arbitary
2       0.0000   8.0000e-04   4.0000e-04      0.0000       0.0000   8.0000e-04    240.0000       0.0000   0.             arbitary
3       0.0000   1.0000e-99   2.0000e-03      0.0000       0.0000   1.0000e-99     24.0000       0.0000   0.             arbitary
```

Fig. 3.6. Sample output from PROPA.

- scheduled maintenace data -

(state/start time/end time)

n\id	1			2			3		
1	0	4380.00	4548.00	0	720.00	744.00	0	4380.00	4548.00
2	0	0.00	0.00	0	1440.00	1464.00	0	0.00	0.00
3	0	0.00	0.00	0	2160.00	2184.00	0	0.00	0.00
4	0	0.00	0.00	0	2880.00	2904.00	0	0.00	0.00
5	0	0.00	0.00	0	3600.00	3624.00	0	0.00	0.00
6	0	0.00	0.00	0	4380.00	4548.00	0	0.00	0.00
7	0	0.00	0.00	0	5268.00	5292.00	0	0.00	0.00
8	0	0.00	0.00	0	5988.00	6012.00	0	0.00	0.00
9	0	0.00	0.00	0	6708.00	6732.00	0	0.00	0.00
10	0	0.00	0.00	0	7428.00	7452.00	0	0.00	0.00
11	0	0.00	0.00	0	8148.00	8172.00	0	0.00	0.00
12	0	0.00	0.00	0	8736.00	8760.00	0	0.00	0.00

- time interval for spare supply = 0.000 hrs

*** simulation starts ***

history no.	availability and uncertainty of top event					
	state 0		state 1		state 2	
100	1.6337e-01	(3.6841e-03)	1.5351e-01	(7.1245e-03)	6.8312e-01	(7.2377e-03)
200	1.5855e-01	(2.6537e-03)	1.6037e-01	(5.4640e-03)	6.8108e-01	(5.6311e-03)
300	1.5583e-01	(2.0709e-03)	1.6311e-01	(4.5429e-03)	6.8106e-01	(4.5896e-03)
400	1.5804e-01	(1.8040e-03)	1.6112e-01	(3.8036e-03)	6.8084e-01	(3.8650e-03)
500	1.5719e-01	(1.6295e-03)	1.6245e-01	(3.4597e-03)	6.8036e-01	(3.5404e-03)
600	1.5598e-01	(1.4567e-03)	1.6332e-01	(3.1859e-03)	6.8070e-01	(3.2273e-03)
700	1.5734e-01	(1.3573e-03)	1.6207e-01	(2.8941e-03)	6.8059e-01	(2.9448e-03)
800	1.5675e-01	(1.2724e-03)	1.6304e-01	(2.7332e-03)	6.8021e-01	(2.7949e-03)
900	1.5615e-01	(1.1885e-03)	1.6334e-01	(2.5926e-03)	6.8051e-01	(2.6324e-03)
1000	1.5705e-01	(1.1334e-03)	1.6247e-01	(2.4280e-03)	6.8048e-01	(2.4715e-03)

*** final results print **

(state 0 = failure/state 1 = degraded/state 2 = normal)

- gate id

		availability					
		state 0		state 1		state 2	
1	system	1.5705e-01	(1.1334e-03)	1.6247e-01	(2.4280e-03)	6.8048e-01	(2.4715e-03)
2	power	1.3296e-01	(1.1286e-03)	1.6688e-01	(2.4822e-03)	7.0016e-01	(2.5292e-03)

- component id

		availability					
		state 0		state 1		state 2	
1	resistance	1.9395e-02	(2.3381e-05)	3.6030e-02	(3.2147e-03)	9.4458e-01	(3.2124e-03)
2	battery	1.3279e-01	(1.1304e-03)	1.6694e-01	(2.4825e-03)	7.0027e-01	(2.5296e-03)
3	switch	4.6515e-02	(2.5205e-04)	0.	(0.)	9.5348e-01	(2.5205e-04)

- effective availability of whole system = 76.1718 %

- ranking of effective availability -

rank	gate id	name	eff-avail %
1	2	power	78.3600
2	1	system	76.1718

rank	comp. id	name	eff-avail %
1	1	resistance	96.2590
2	3	switch	95.3485
3	2	battery	78.3738

* computing time in seconds *
(cpu/i&o/sys)= 32.58165/ 4.46803/ 0.00420/

reactor model. Four parameters for these systems and CPU times on a Cray 1 are shown in Table 3.7.

Using values in Table 3.7, we obtain the following values of α and β:

$$\alpha = 1.32 \times 10^{-8} \tag{3.2a}$$

$$\beta = 0.52 \ . \tag{3.2b}$$

As an example of the largest systems, let us consider a system with $N_g = 100$, $N_c = 1000$. Then Eq. (3.1) becomes

$$T_c = 1.32 \times 10^{-3} \ N_s N_h + 0.52 \ . \tag{3.3}$$

There are two controllable parameters N_s and N_h. N_s is actually determined by

$$N_s = \frac{T}{\Delta t} \tag{3.4}$$

where T is the total time length of a history, and Δt is the timestep length. The timestep should be chosen so that time variation of the system is clarified. The minimum characteristic time scale such as the minimum repair time and mean-time-to-failure is the best choice. Δt = 24 hours may be sufficiently small for most system simulations.

The total time length of a simulation should be longer than the time at which the system reaches steady state. For a system which has a very long characteristic time such as the mean-time-to-failure of a very reliable component, T may be very long; 10 years or more. Sometimes it must be simulated over the lifetime of the system. If T is 10 years (= 3650 days), then

N_s = 3650 and Eq. (3.3) becomes

$$T_c = 4.82\ N_h \tag{3.5}$$

where the second term is neglected because the first term is much larger than that for $N_h >> 1$. For N_h = 100, T_c = 482 minutes (~ 8 hours). For N_h = 1000, T_c = 80 hours. N_h is determined by requirement for the accuracy of results. For systems with moderately high availability, 100 histories might be enough to get an error of 10% or less. If one needs unavailability of very highly available system, however, N_h must become very large.

In any case, the computing efficiency of the present simulation program is too low to perform a detailed simulation of large systems even on a supercomputer such as a Cray 1. A 1000 times faster computer is required to reduce CPU usage down to the order of 10 minutes. To overcome the difficulties associated with computing cost, several approaches can be proposed. Reducing complexity of the model is not the desirable path because one of the advantages of a Monte Carlo simulation resides in the complex modeling capability. Hence, we fix the timestep to 1 day. There are three approaches:

1. reduce the computing time for a timestep in a history,
2. reduce T,
3. reduce N_s.

Approach (1) is closely related to computer programming, computing algorithm, and computer hardware. Among these taking advantage of computer power of new types of computers such as vectorization and parallel processing is the most important task. Approaches (2) and (3) can be achieved by employing variance reduction techniques.

4. VECTOR AND PARALLEL PROCESSING

4.1 Introduction

Modern supercomputers such as the Cray 1 and Cyber 205 utilize vector processing capability [49]. Computer programs taking advantage of this feature may run many times faster than non-vectorized programs. Since at present, a compiler takes care of the vectorization [50], what a programmer should do mainly is to get rid of portions in a program for which vectorization is prohibited.

Another feature, which is beginning to appear in the new generation of computers, is multiprocessing (parallel processing) capability [49]. There are many different architectural approaches to accomplish parallel processing. Since we have easy access to a Cray XMP (with 2 processors) and a Cray 2 (with 4 processors), it is worth investigating a way to take advantage of the parallel processing capability of the Crays. The Crays are equipped with very few, but very fast processors compared with other parallel processing machines such as HEP and others shown in Table 4.1.

4.2 Limit of Vectorization

In this section we shall discuss vectorization of the PROPA program. However, the following discussion is appropriate only for the program on the Cray 1 computer with the CFT compiler. For programs on different vector computers such as the Cyber 205, the effect of vectorization may be different.

There are three levels of approach to vectorization as discussed in Ref. [51]. The first step is to look at a program written for scalar computers and to modify portions which cannot be vectorized. Vectorization can be done for the innermost DO loops if the loops satisfy the specific conditions given in Ref. [50]. The second level is to look for algorithms which are effectively

Table 4.1. Parallel Computers

Machine	Developer	Number of Processors	Cycle Time nanoseconds	Maximum Speed	Memory	Year Operational
Cray XMP	CRI	2	9.5	300 MFLOPS[1]	shared	1983
Cray 2	CRI	4	4	800 MFLOPS	shared	1985
HEP	Denelcor	16				
MPP	Goodyear Aerospace	16,384	100	6.5 BIPS[2]	dist.	1983
Connection	Thinking Machines	64,000	1000	10 BIPS	dist.	1985
NonVon	N.Y. Univ.	8,000	1500	16 BIPS	dist.	?
IPSC	Intel	32-128	100	2-8 MFLOPS	dist.	1985
Butterfly	Bolt Beranek and Newman	to 128	?	to 200 MIPS[3]	shared	1984
Sigma 1	Japan's National Electrotechnical Laboratory	256	100	100 MFLOPS	dist.	1986
Cedar	Univ. of Illinois	32	100	10 MFLOPS	shared	1985

[1]Million floating-point operations per second

[2]Billion instructions per second

[3]Million instructions per second

References [59,60]

vectorized. The third level is to rearrange the entire structure of a program into the form suitable for vectorization.

A few attempts have been made to vectorize Monte Carlo simulation programs in the field of neutron transport [52,53] and availability analysis [54]. The speedup factor by vectorization ranges from 1.2 to 50. Naturally the most successful result is obtained by working up to the third level of vectorization [53].

So far we have done only the first level of vectorization. Figure 4.1 shows DO loops included in program routines computing system availability in the PROPA program. Loops 2, 3, 5, 6, 9, and 10 can be vectorized. However, loops 7 and 8, which are the most important in terms of computing time, are not vectorizable. Loop 7, which finds component states at the timestep by using the states from the previous timestep, includes GO TO, IF-THEN-ELSE-IF, and CALL statements. Loop 8, which computes the states of the gates by tracing the systems tree, includes IF-THEN-ELSE-IF statements and vector subscripts which themselves are vectors.

As an example, the CPU times for the problem discussed in Section 3.4 are shown in Table 4.2. The original program requires 33 seconds. By modifying DO loops so that these are vectorized, a 7% speedup is accomplished (row 2). Row 4 shows the CPU by a non-vectorized program made by the CFT compiler with off = v option. This is much faster than the vectorized program. A reason is that the setup time of vectorization surpasses the gain by vectorization of DO loops. The present problem has very few gates and components. Since almost all the vectorized DO loops are for repetitive computation with respect to gates and components, significant speedup will be observed for problems with very large numbers of gates and components.

Table 4.2. CPU Gain*

	Comments	CPU (seconds)	Gain Factor
1)	Base program	33.24	1.0
2)	Level-2 vectorization	30.94	1.07
3)	Use RANF*	28.67	1.16
4)	Non-vectorization	25.90	1.28

*3 gates, 6 components, 365 timesteps, 1000 histories
The CFT 111g compiler is used.

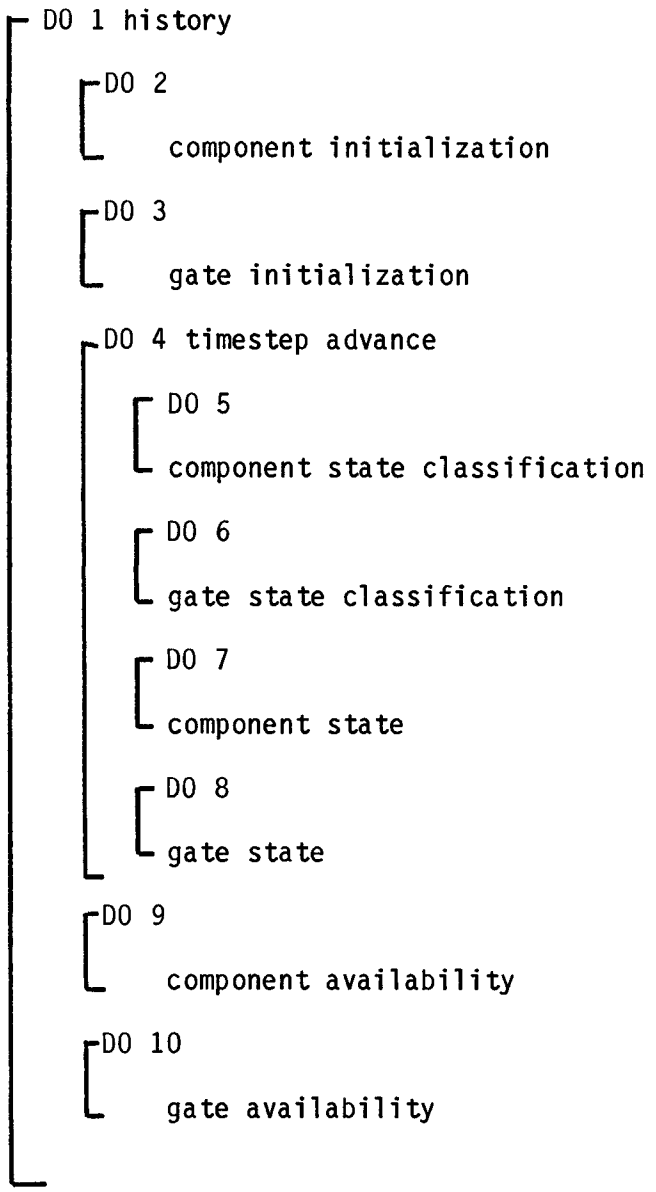

Fig. 4.1. Simplified structure of a major PROPA algorithm.

The algorithms in DO loops 7 and 8 must be modified to accomplish further speedup. Also, a more efficient random number generator should be used. For example, using the utility random number generator, RANF, instead of our generator, leads to a 10% reduction of computing time as shown at low 3 in Table 4.2. Without the third level of vectorization, however, a factor of 10 speedup may be the largest gain. Hence, we need to depend on speedup by other means.

4.3 Parallel Processing

As we discussed in Chapter 3, Monte Carlo simulations must be performed 1000 times faster than the present program so that large and complex systems can be modeled and sufficiently accurate solutions can be obtained. To accomplish this goal by using only new computer hardware, we need a 100 GFLOPS (100 billion floating-point operations per second) machine since the Cray 1 achieves 100 MFLOPS.

Suppose ten times faster processors, i.e. processors with one nanosecond clock period, are developed. If the speed of computation is proportional to the number of processors, then 100 processors are required to achieve 100 GFLOPS. In fact the cost of 100 processors with such a clock period will be too high; hence, it is better to assume that 1000 processors with 10 nanosecond clock periods will be used. A simultaneous use of 1000 processors can be accomplished by making 1000 identical tasks in an execution. In a Monte Carlo simulation, perform 1000 histories concurrently. Since each task requires its own data arrays whose data keeps varying during a simulation, much memory is needed for such multiprocessing. For example, 20 M words of memory is required to store the variable arrays for the present problem (see Fig. 4.2). If the speed of the processors is slower, more memory is required.

AVAILG(3,ngate), STDVG(3,ngate), GSTATE(3,ngate)

AVAILC(3,ncomp), STDVC(3,ncomp), CSTATE(3,ncomp)

STOCC(ncomp) TGATE(3,ngate)

STOCG(ngate) TCOMP(3,ncomp)

KKK(ncomp)

ELPSCT(ncomp)

ELPSGT(ngate)

ELPSGT(nnunt)

The total memory required to store these arrays

= 3 x ncomp x 4 + ncomp x 3 + 3 x ngate x 4 + ngate x 2 + nnunt x 2

= 15 x ncomp + 14 x ngate + 2 x nnunt

= 15 x 1000 + 14 x 100 + 2 x 1000

= 18,400

for ncomp = nnunt = 1000 and ngate = 100

Fig. 4.2. History dependent variable arrays in the PROPA program.

It should be noted, however, that the order of 10 M words memory will be available only on computers with very high speed processors. Unfortunately, such machines won't utilize as many as 100 processors.

Now we can imagine two types of future computers which will meet our needs. One is a computer with a 1 nanosecond clock period, 100 processors and 100 M words of memory. The other is a computer with a 100 nanosecond clock period, 10,000 processors, and 10 G words of memory. The former will be very expensive, while the latter requires an unrealistic amount of memory. In the remainder of this section, we shall discuss utilization of the multiprocessing capability of a Cray XMP and Cray 2.

Since the innermost DO loops are efficiently vectorizable, the best approach to multiprocessing is to process the outermost DO loops concurrently (e.g., DO loop 1 in Fig. 4.1). If there are N processors, N histories, which we call N tasks [55], can be simulated concurrently. Thus a factor N speedup can be accomplished.

At this point several cautions should be noted:

Since there are variable arrays dependent on a particular history, N copies of the variables must be made. For the problem we are talking about, 20 × N K words are required just to save these variables (see Fig. 4.2).

We use the recursive congruential random number generation [35]. This generator requires a seed number to create a new random number. Since N tasks use the same random number generator without any regular order, a seed for a task may not be one that the task generated at the previous call to the generator. In other words, there is no deterministic recursiveness for a particular task. Hence, the uniformness of the random num-

bers is not guaranteed. To overcome this difficulty, random number generators for parallel processors have been proposed [56,57].

In the timesharing environment, a factor of N gain would be rarely accomplished [58]. One reason is that a specific program cannot occupy all the processors for the entire execution. During some periods only one processor may be available.

Since all the tasks do not finish simultaneously, creating multitasks in a task results in a more efficient use of multiprocessors.

In the PROPA program, DO loops 2 and 3, 5 and 6, and 9 and 10 can be concurrently processed by two processors because these are independent. Furthermore, the algorithm in DO loop 8 can be modified so that multitasking is effectively utilized. For example, paths A-C, A-B-E, and A-B-D-F in the systems tree shown in Fig. 4.3 can be concurrently processed.

The techniques proposed above will be implemented in the PROPA program and tested on the Cray XMP and the Cray 2 in the future.

Since N = 2 and 4 for the Cray XMP and the Cray 2, respectively, a factor of 4 speedup will be the best attainable. Hence, only a 40 times faster program will be made by utilizing both vector and multiprocessing capabilities. Hence, a 25 times reduction must be attained by other means. In the next chapter variance reduction techniques will be discussed to gain such a speedup.

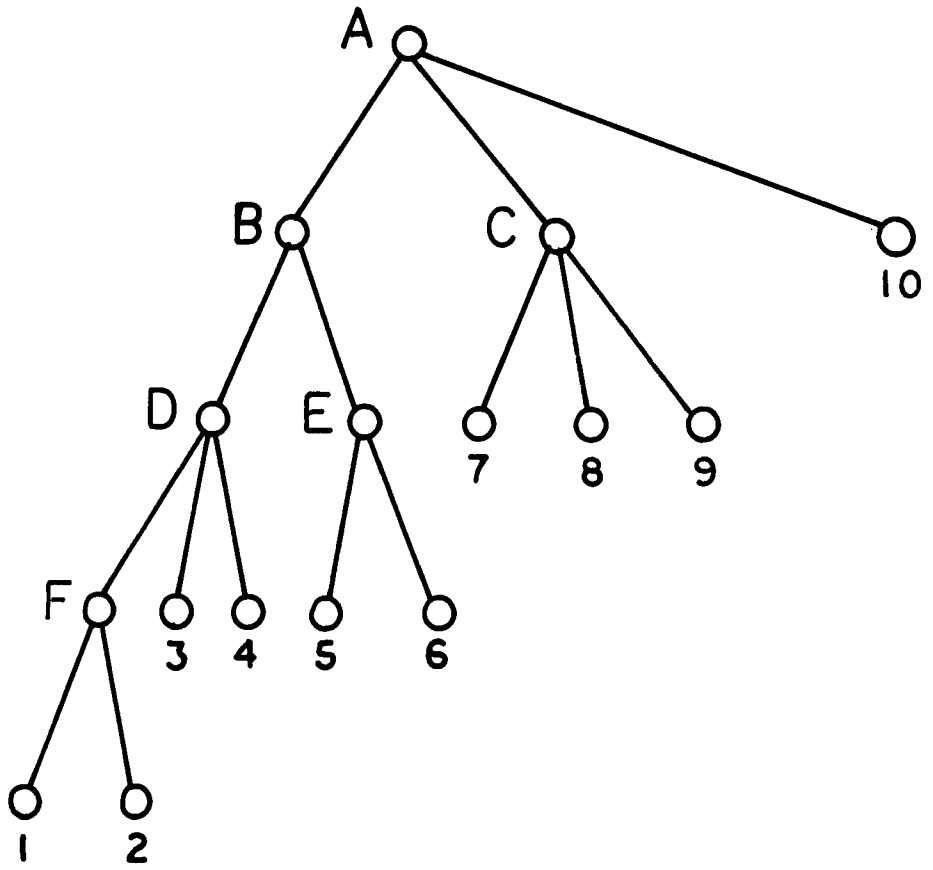

Fig. 4.3. A systems tree.

5. VARIANCE REDUCTION TECHNIQUES

5.1 Introduction

The uncertainty of results by a Monte Carlo simulation can be reduced by either increasing the number of histories or reducing the variance as we see by Eq. (2.32). If the same uncertainty is required, reducing the variance leads to a smaller number of histories, and consequently shorter computing time.

Several variance reduction techniques have been proposed for system reliability and availability simulations [1,45,47,61,62]. In this chapter we shall discuss fundamental mathematics behind the variance reduction techniques. Also, some methods will be proposed in the case of binary state models. Then suggestions will be made about an extension of the techniques to multistate models.

5.2 Unavailability Estimate of Reparable Components

We consider a component with binary states and assume the component is repaired within a fixed time τ. The purpose for the simulation is to obtain the unavailability of the component during a time period [0,T].

Obviously the total number of failures during [0,T] is a random variable. Let this random variable, K, have a probability distribution function (p.d.f.), p(K). The function p(K) satisfies

$$\sum_{k=0}^{\infty} p(K=k) = 1 \tag{5.1a}$$

$$p(K=k) > 0 \qquad 0 < k < \infty . \tag{5.1b}$$

The expected value of K is given by

$$E(K) = \int K\, p(K)\, dK = \sum_{k=0}^{\infty} k\, p(K=k) \ . \tag{5.2}$$

Also the variance of K is

$$V(K) = E(K^2) - E(K)^2 \ . \tag{5.3}$$

In order to reduce the variance, consider a different p.d.f., $\hat{p}(K)$.

We introduce the weight w(k) so that the expected value by the new p.d.f. is the same as E(K):

$$E(\hat{K}) = \sum_{k=0}^{\infty} w(k)\, k\hat{p}(K=k) = \sum_{k=0}^{\infty} kp(K=k) = E(K)$$

hence

$$w(k) = \frac{p(K=k)}{\hat{p}(K=k)} \ . \tag{5.4}$$

For this p.d.f., the variance is

$$V[\hat{K}] = \sum \left(w(k)k\right)^2 \hat{p}(K=k) - E(K)^2 = \sum \frac{p(K=k)}{\hat{p}(K=k)}\, k^2 p(K=k) - E(K)^2 \ .$$

We see that $V[\hat{K}] = 0$ if $\hat{p}(K) = (k/E(K))\, p(K)$. However, this technique is of no use because E(K), as well as p(K), is unknown.

The unavailability, ξ, is also a random variable and represented by means of K as

$$\xi = \frac{\tau}{T} K \ . \tag{5.5}$$

An unbiased estimation of ξ is

$$E[\xi] = \frac{1}{N} \sum_{\ell=1}^{N} \xi_\ell \tag{5.6}$$

where ξ_ℓ is the ξ by the ℓ-th trial.

5.3 Time-Dependent Unavailability Estimate

The time-dependent unavailability, Q(t), is given by

$$Q(t) = \int_0^t [w(s) - v(s)]\, ds \tag{5.7}$$

where the failure density w(t) is a solution of

$$w(t) = \int_{-\tau}^{t-\tau} K(t,s)\, w(s)\, ds + f(t) \tag{5.8}$$

and the repair density is given by

$$v(t) = w(t - 2) \ . \tag{5.9}$$

Here we see that once w(t) is estimated, Q(t) can be computed by Eqs. (5.7) and (5.9). Hence, we must solve Eq. (5.8). We solve Eq. (5.8) by a Monte Carlo simulation. As discussed in Ref. [63], we can use the following theorem to obtain an estimate of w(t).

Theorem. For any given vector h

$$E[n_k(h)] = \langle h, w^{(k)}(t) \rangle \tag{5.10}$$

where the r.v. $n_k(h)$ is defined on the Markov chain $t_1 \to t_2 \to \ldots \to t_k$, such that

$$\eta_k(h) = \frac{h(t_1)}{p(t_1)} \sum_{m=1}^{k} W_m f(t_m) \tag{5.11}$$

$$W_m = \frac{K(t_1,t_2)\ K(t_2,t_3)\ \ldots\ K(t_{m-1},t_m)}{P(t_1,t_2)\ P(t_2,t_3)\ \ldots\ P(t_{m-1},t_m)} \tag{5.12}$$

and $W_1 = 1$.

The main point for use of this theorem is to choose a proper Markov chain.

Suppose a component is in state 1 at $t = 0$. It fails at $t = t_1$. Then it is repaired for period τ, but it fails again at $t = t_2$, and so on. Hence, a time sequence

$$\{t_1,t_2,\ldots,t_k\} \tag{5.13}$$

represents a Markov random walk process. Here t_i is the time when the i-th failure occurs. Let us assume a constant failure rate, λ. Then the probability for the first failure

$$p(t_1) = \lambda e^{-\lambda t_1} \tag{5.14}$$

and the transition probability between t_i and t_{i+1} can be given by

$$P(t_i,t_{i+1}) = \lambda e^{-\lambda(t_{i+1}-t_i-\tau)} \ . \tag{5.15}$$

Since $\int_0^\infty p(t)\ dt = 1$ and $\int_0^\infty P(t_i,t)\ dt = 1$, the random walk (5.13) satisfies necessary conditions. Having known $K(t_i,t_{i+1}) = P(t_i,t_{i+1})$, we have

$$\eta_k(h) = \frac{h(t_1)}{p(t_1)} \sum_{m=1}^{k} f(t_m) \ . \tag{5.16}$$

Now the inner product in Eq. (5.10) is estimated by a Monte Carlo simulation using the following estimator

$$\theta_k = \frac{1}{N} \sum_{\ell=1}^{N} \eta_k^{(\ell)}(h) \tag{5.17}$$

where $\eta_k^{(\ell)}(h)$ is for the ℓ-th trial.

In order to reduce the variance, we use $\hat{\lambda}$ for the simulation. Now a new r.v., $\hat{\eta}_k(h)$, is given by

$$\hat{\eta}_k(h) = \frac{h(t_1)}{\hat{p}(t_1)} \sum_{m=1}^{k} \hat{W}_m \, F(t_m) \tag{5.18}$$

where

$$\hat{W}_m = \frac{K(t_1,t_2) \ \ldots \ K(t_{m-1},t_m)}{\hat{P}(t_1,t_2) \ \ldots \ \hat{P}(t_{m-1},t_m)} \tag{5.19}$$

$$\hat{p}(t_1) = \hat{\lambda} e^{-\hat{\lambda} t_1} \tag{5.20}$$

$$\hat{P}(t_1,t_{i+1}) = \hat{\lambda} e^{-\hat{\lambda}(t_{i+1}-t_i-\tau)} \ . \tag{5.21}$$

Since the new random walk process is also a Markov chain, the expected value of the random variable $\hat{\eta}_k$ is identical to that of the r.v. η_k by the above theorem:

$$E[\hat{\eta}_k] = E[\eta_k] \ . \tag{5.22}$$

We need to prove the variance of $\hat{\eta}_k$ is smaller than that of η_k:

$$V[\hat{n}_k] < V[n_k] \ . \tag{5.23}$$

However, what we can show is that no Markov chain can be found so that inequality (5.23) always holds. To prove this, we shall use the following equation:

$$\left(\sum_{i=1}^{n} x_i\right)^2 = \sum_{i=1}^{n} x_i^2 + 2 \sum_{\substack{i=1 \\ i\neq j}}^{n} \sum_{j=1}^{n} x_i x_j \ . \tag{5.24}$$

Now we start the proof. By Eq. (5.22), we need to show

$$E[\hat{n}_k^2] < E[n_k^2] \tag{5.25}$$

to prove the inequality (5.23).

$$E[n_k^2] = \int_1 \int_2 \ldots \int_k \left(\frac{h(t_1)}{p(t_1)}\right)^2 \left(\sum_{m=1}^{k} f(t_m)\right)^2 p(t_1)\, P(t_1,t_2)\ldots P(t_{k-1},t_k)\, dt_1 dt_2 \ldots dt_k$$

$$= \int_1 \int_2 \ldots \int_k \left(\frac{h(t_1)}{p(t_1)}\right)^2 \left\{\sum_{m=1}^{k} f^2(t_m)\right.$$

$$\left. + 2 \sum_{i}^{k} \sum_{j}^{k} f(t_i)\, f(t_j)\right\} p(t_1)\, P(t_1,t_2)\ldots P(t_{k-1},t_k)\, dt_1 dt_2 \ldots dt_k$$

$$= \sum_{m=1}^{k} \int_1 \int_2 \ldots \int_m \left(\frac{h(t_1)}{p(t_1)}\right)^2 f^2(t_m)\, p(t_1)\, P(t_1,t_2)\ldots P(t_{m-1},t_m)\, dt_1 dt_2 \ldots dt_m$$

$$+ 2 \sum_{\substack{i=1 \\ i>j}}^{k} \sum_{j=1}^{k} \int_1 \int_2 \ldots \int_i \left(\frac{h(t_1)}{p(t_1)}\right)^2 f(t_i) f(t_j) p(t_1) P(t_1,t_2)\ldots P(t_{i-1},t_i) dt_1 dt_2 \ldots dt_i$$

$$+ 2 \sum_{\substack{i=1 \\ i<j}}^{k} \sum_{j=1}^{k} \int_1 \int_2 \ldots \int_j \left(\frac{h(t_1)}{p(t_1)}\right)^2 f(t_i) f(t_j) p(t_1) P(t_1,t_2)\ldots P(t_{j-1},t_j) dt_1 dt_2 \ldots dt_j \tag{5.26}$$

where we use $\int_0^\infty P(t,s)\,ds = 1$. Meanwhile,

$$E[\hat{n}_k^2] = \int_1\int_2\cdots\int_k \left(\frac{h(t_1)}{\hat{p}(t_1)}\right)^2\left\{\sum_{m=1}^{k}\hat{W}_m f(t_m)\right\}^2 \hat{p}(t_1)\,\hat{P}(t_1,t_2)\cdots\hat{P}(t_{k-1},t_k)\,dt_1\cdots dt_k$$

$$= \int_1\int_2\cdots\int_k \left(\frac{h(t_1)}{\hat{p}(t_1)}\right)^2\left\{\sum_{m=1}^{k}\hat{W}_m^2 f^2(t_m) + 2\sum\sum W_i W_j f(t_i)f(t_j)\right\}$$

$$* \hat{p}(t_1)\,\hat{P}(t_1,t_2)\cdots\hat{P}(t_{k-1},t_k)\,dt_1\cdots dt_k$$

$$= \sum_{m=1}^{k}\int_1\int_2\cdots\int_m \frac{p(t_1)}{\hat{p}(t_1)}\,\frac{h^2}{p(t_1)^2}\,f^2(t_m)\,\frac{P(t_1,t_2)\cdots P(t_{m-1},t_m)}{\hat{P}(t_1,t_2)\cdots\hat{P}(t_{m-1},t_m)}$$

$$* p(t_1)\,P(t_1,t_2)\cdots P(t_{m-1},t_m)\,dt_1\cdots dt_m + 2\sum_{i<j}\sum_{j}\int_1\int_2\cdots\int_j \frac{p(t_1)}{\hat{p}(t_1)}\left(\frac{h(t_1)}{p(t_1)}\right)^2$$

$$* f(t_i)f(t_j)\,\frac{P(t_1,t_2)\cdots P(t_{i-1},t_i)}{\hat{P}(t_1,t_2)\cdots\hat{P}(t_{i-1},t_i)}\,p(t_1)\,P(t_1,t_2)\cdots P(t_{j-1},t_j)\,dt_1\cdots dt_j$$

$$+ 2\sum_{i>j}\sum_{j}\int\int\cdots\int_i \frac{p(t_1)}{\hat{p}(t_1)}\left(\frac{h(t_1)}{p(t_1)}\right)^2 f(t_i)f(t_j)\,\frac{P(t_1,t_2)\cdots P(t_{j-1}\ t_j)}{\hat{P}(t_1,t_2)\cdots\hat{P}(t_{j-1},t_j)}$$

$$* p(t_1)\,P(t_1,t_2)\cdots P(t_{i-1},t_i)\,dt_1\cdots dt_i\ . \qquad (5.27)$$

Since
$$\frac{p(t_1)}{\hat{p}(t_1)} = \frac{\lambda}{\hat{\lambda}}\,e^{-\Delta\lambda t_1} \qquad (5.28)$$

$$\frac{P(t_{i-1},t_i)}{\hat{P}(t_{i-1},t_i)} = \frac{\lambda}{\hat{\lambda}}\,e^{-\Delta\lambda(t_i - t_{i-1} - \tau)}\ , \qquad \Delta\lambda = \lambda - \hat{\lambda} < 0\ , \qquad (5.29)$$

comparing Eq. (5.27) with Eq. (5.26) suggests that if $\hat{\lambda}$ is chosen so that $p(t_1)/\hat{p}(t_1) < 1$ and $\left(P(t_{i-1},t_i)/\hat{P}(t_{i-1},t_i)\right) < 1$ for any i $(= 2,3,4,\ldots,k)$, then $E[\hat{n}_k^2] < E[n_k^2]$; hence $V[\hat{n}_k^2] < V[n_k^2]$. That is, by Eqs. (5.28) and (5.29) the following inequalities are sufficient.

$$\frac{\lambda}{\hat{\lambda}} e^{-(\lambda-\hat{\lambda})t_1} < 1 \tag{5.30}$$

$$\frac{\lambda}{\hat{\lambda}} e^{-(\lambda-\hat{\lambda})(t_i - t_{i-1} - \tau)} < 1 \ . \tag{5.31}$$

Now let us define a function g(x) by

$$g(x) = \frac{1}{x} e^{+a(x-1)} \tag{5.32}$$

where $$x = \frac{\hat{\lambda}}{\lambda}$$

$$a = \begin{cases} \lambda t_1 & \text{for Eq. (5.30)} \\ \lambda(t_i - t_{i-1} - \tau) & \text{for Eq. (5.31)} \end{cases} .$$

The function g(x) assumes the minimum at x = 1/a and g(1) = 1, $\lim_{x\to 0} g(x) = \lim_{x\to\infty} g(x) = \infty$. The graphs of g(x) are drawn in Figs. 5.1(a) and (b) for the case of 1/a > 1 and 1/a ≤ 1, respectively. For a given λ, $\hat{\lambda}$ is chosen; i.e. x is chosen. Since t_i's are random variables, a is a r.v. Hence the corresponding g(x) may be given by either Fig. 5.1(a) or (b). If the x happens to be in $[1,x_0]$ or $[x_1,1]$, the inequality (5.30) or (5.31) holds. There is, however, no prediction about when all the inequalities hold. To satisfy the inequality (5.23), Eqs. (5.29) and (5.30) do not need to hold at the same time. But any $\hat{\lambda}$ cannot be chosen in advance so that Eq. (5.23) holds. The only way to find an optimal λ is to perform numerical experiments.

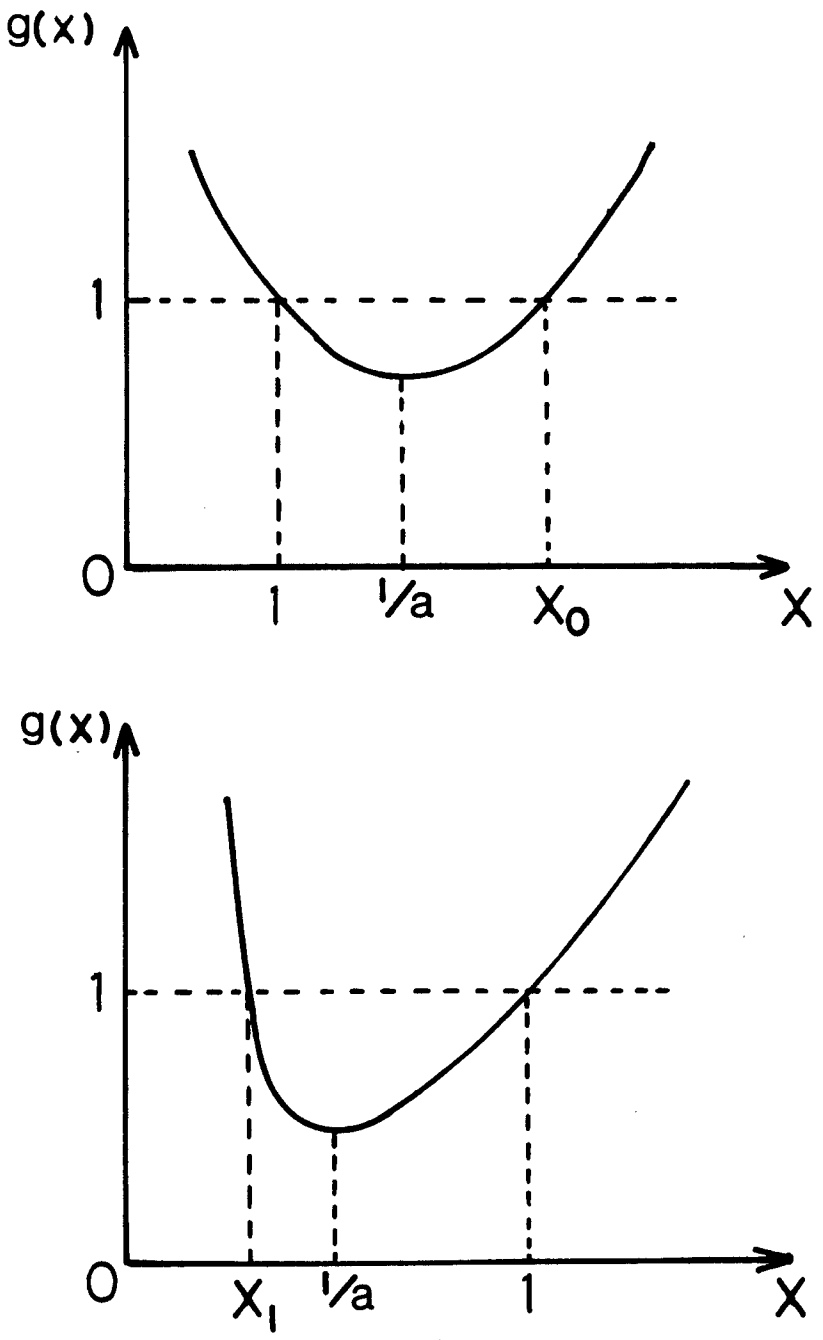

Fig. 5.1. Sketches of function g(x).

For a multistate model, the failure density w(t) is replaced by a state density function $f_i(t)$ and a similar variance reduction technique can be developed.

5.4 System Availability Estimate

We consider a multistate model of a system with M multistate components. Let a sequence of discrete times with timestep Δt be denoted by $\{t_0,t_1,\ldots,t_k\}$. We define notation as follows:

X_{mk}: the random variable (r.v.) of the state of component m at time t_k

Z_k: the r.v. of system state at time t_k

N: the total number of histories (or trials)

M: the number of components

K: the total number of timesteps

T: the total time period being simulated

Δt: timestep

$C = \{Z_1,Z_2,\ldots,Z_k\}$: a random walk process for the system

$C_m = \{X_{m1},X_{m2},\ldots,X_{mk}\}$: a random walk process for component m

f(C): the probability density function (p.d.f.) for the random walk C

$f_m(C_m)$: the p.d.f. for the random walk C_m

x_{mk}: a value of the r.v. X_{mk}, $x_{mk} \in \{0,1,\ldots,I\}$

z_k: a value of the r.v. Z_k, $z_k \in \{0,1,\ldots,I\}$

$\phi(x_{1k},x_{2k},\ldots,x_{mk})$: the structure function of the system and

$$z_k = \phi(x_{1k},x_{2k},\ldots,x_{mk}) \ . \tag{5.33}$$

Now we introduce the following r.v.:

$$n_i(C) = \frac{\Delta t}{T} \sum_{k=1}^{K} \delta(Z_k - i) \ . \tag{5.34}$$

The expected value of $n_i(C)$ is given by

$$A_i = E[n_i(C)] = \int n_i(C)\ f(C)\ dC \tag{5.35}$$

where A_i can be considered as the i-th state availability of the system. An estimator for A_i can be given by

$$A_i = \frac{1}{N} \sum_{n=1}^{N} n_i^{(n)} \tag{5.36}$$

where n denotes the n-th trial in a Monte Carlo simulation. For components equations similar to Eqs. (5.34) (5.35) and (5.36) can be obtained.

In a possible variance reduction technique, the p.d.f. for components, $f_m(C_m)$, will be altered so that state transitions which otherwise rarely occur take place more frequently. Since f(C) can be constructed by using $f_m(C_m)$, the p.d.f. f(C) itself is altered. An explicit form of f(C) must be known to modify $n_i(C)$ so that A_i is invariant under the new $f_m(C_m)$. Unfortunately, obtaining f(C) from $f_m(C_m)$ cannot be accomplished, in particular, for large complex systems. Therefore, proving the effectiveness of variance reduction techniques mathematically cannot be carried out even though some variance reduction techniques will turn out to be effective. However, it might be worthwhile to mathematically demonstrate the actual effectiveness of a certain variance reduction technique for simple systems. This will be done in the future.

6. CONCLUSIONS AND FUTURE WORK

We have developed a Monte Carlo simulation method for three-state systems with three-state components. A computer program PROPA has been written. The program is quite useful for obtaining detailed performance prediction for systems and it may be regarded as the first step towards more detailed system simulations.

There are many areas for further investigation. As for the computing cost of simulations:

1. Find an efficient algorithm to obtain the next states of the components.
2. Find an efficient algorithm to obtain the states of the gates in a systems tree.
3. Use a more efficient random number generator.
4. Devise variance reduction techniques; when the techniques actually reduce the variance should be theoretically and experimentally clarified.
5. Restructure the program so that the vector and multiprocessing capabilities of the Cray 2 computer are fully utilized.

As for the system model, the following models should be added:

a. time-dependent transition probability;

b. sophisticated maintenance scheme including deferred maintenance;

c. dependent components;

d. more than one state variable for systems.

A general m-state model can be easily constructed; however, at first the power of the three-state model must be further investigated through simulations of more complex and large systems. In the future, a fusion reactor power plant will be simulated by the PROPA program.

ACKNOWLEDGEMENTS

The author thanks Prof. C.W. Maynard for his continuous encouragement and valuable advice. He also thanks Dr. Z. Musicki with whom he had many useful discussions. The work was supported by the Office of Magnetic Fusion in the Department of Energy.

APPENDIX A. DERIVATION OF BASIC EQUATIONS FOR BINARY STATE MODELS

Suppose 0 and 1 denote the failed and success states of a system, respectively. For the binary state model, Eq. (2.10) becomes

$$w_{10}(t) = \int_0^t K_{10}(s|t)\, w_{01}(s)\, ds + \int_0^t K_{10}(s|t)\, w_{11}(s)\, ds \qquad (A.1)$$

$$w_{01}(t) = \int_0^t K_{01}(s|t)\, w_{10}(s)\, ds + \int_0^t K_{01}(s|t)\, w_{00}(s)\, ds\ . \qquad (A.2)$$

Assume that the system is in state 1 at time t = 0, then Eqs. (A.1) and (A.2) become

$$w_{10}(t) = \int_0^t K_{10}(s|t)\, w_{01}(s)\, ds + f(t) \qquad (A.3)$$

$$w_{01}(t) = \int_0^t K_{01}(s|t)\, w_{10}(s)\, ds \qquad (A.4)$$

where f(t) is the first failure density. Equations (A.3) and (A.4) are identical to Eqs. (4-64) in Ref. [1].

As a special case, suppose that $K_{10}(s|t)$ and $K_{01}(s,t)$ depend only on t - s. Also assume a constant failure rate λ and a fixed repair time τ. For such a system, K_{10} and K_{01} can be given by the following:

$$K_{10}(s|t) = \lambda e^{-\lambda(t-s)} \qquad (A.5)$$

$$K_{01}(s|t) = H(t - s - \tau) \qquad (A.6)$$

where

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}\ .$$

APPENDIX B. SOLUTIONS OF EQUATION (2.14)

$$\frac{\partial}{\partial t} f(x,t) + a(x)\, f(x,t) = \int_x^\infty b(y\to x)\, f(y,t)\, dy \ . \tag{2.14}$$

If we Laplace transform in time for Eq. (2.14), we have

$$[s + a(x_1)]\, \hat{f}(x,s) = \int_x^\infty b(y\to x)\, \hat{f}(y,s)\, dy + f(x,0) \ . \tag{B.1}$$

Solving for $\hat{f}$,

$$\hat{f}(x,s) = \frac{\int_x^\infty b(y\to x)\, \hat{f}(y,s)\, dy}{s + a(x)} + \frac{f(x,0)}{s + a(x)} \ . \tag{B.2}$$

Let us assume

$$b(y\to x) = b(y) \tag{B.3}$$

and define $\hat{h}(x,s)$ by

$$\hat{h}(x,s) = \int_x^\infty b(x)\, \hat{f}(x,s)\, dx \ . \tag{B.4}$$

Now multiply Eq. (B.2) by $b(x\to z)$ and integrate with respect to x over $[x,\infty)$. Using the assumption (B.3) and $\hat{h}(x,s)$ for this equation leads to

$$\hat{h}(x,s) = \int_x^\infty \frac{b(x)\, \hat{h}(x,s)}{s + a(x)}\, dx + \int_x^\infty \frac{b(x)\, f(x,0)}{s + a(x)} \ . \tag{B.5}$$

Differentiating Eq. (B.5) with respect to x, we have

$$\frac{d\hat{h}(x,s)}{dx} = -\xi(x,s)\, \hat{h}(x,s) + \xi(x,s)\, f(x,0) \tag{B.6}$$

where $$\xi(x,s) = \frac{b(x)}{s + a(x)} .$$

Equation (B.6) is easily integrated. The result is

$$\hat{h}(x,s) = \exp\left[-\int^{x} \xi(y)\, dy\right] \times \left\{C + \int^{x} \xi(y)\, f(y,0) \exp\left[\int^{y} \xi(z)\, dz\right] dy\right\} \qquad (B.7)$$

where C is a constant. Equations (B.1) and (B.4) give

$$\hat{f}(x,s) = \frac{1}{s + a(x)} \hat{h}(x,s) + \frac{f(x,0)}{s + a(x)} . \qquad (B.8)$$

Thus, the function f(x,t) is obtained by performing inverse Laplace transformation for $\hat{f}$ by using Eqs. (B.7) and (B.8).

Example. Let us apply the method to the following problem:

$$f(x,0) = \delta(x - z) , \qquad \text{the initial state is } z \qquad (B.9)$$

$$a(x) = ax \qquad (B.10)$$

$$b(x) = a . \qquad (B.11)$$

Then Eq. (B.7) becomes $$\hat{h}(x,s) = \frac{C + a}{s + ax} . \qquad (B.12)$$

Equation (B.8) is $$\hat{f}(x,s) = \frac{C + a}{s + ax} + \frac{\delta(x - z)}{s + ax} . \qquad (B.13)$$

The inverse Laplace transform of $\hat{f}$ is easily obtained:

$$f(x,t) = [(C + a)t + \delta(x - z)]\, e^{-axt} . \qquad (B.14)$$

To determine the constant C, we use a condition for f(x,t):

$$\int_0^\infty f(x,t)\, dx = 1 \ . \tag{B.15}$$

By using Eqs. (B.14) and (B.15), we have C = 0. Thus the solution is

$$f(x,t) = [at + \delta(x - z)]\, e^{-axt} \ . \tag{B.16}$$

Note here that $x \leq z$ since the state can only decrease by the nature of Eq. (2.14). Observing Eq. (B.16), we see f approaches a δ-function as time goes on; that is, a system in perfect state, z, at time 0 dies out an infinite time later. The behavior of the state between these extreme cases can be seen in Fig. B.1.

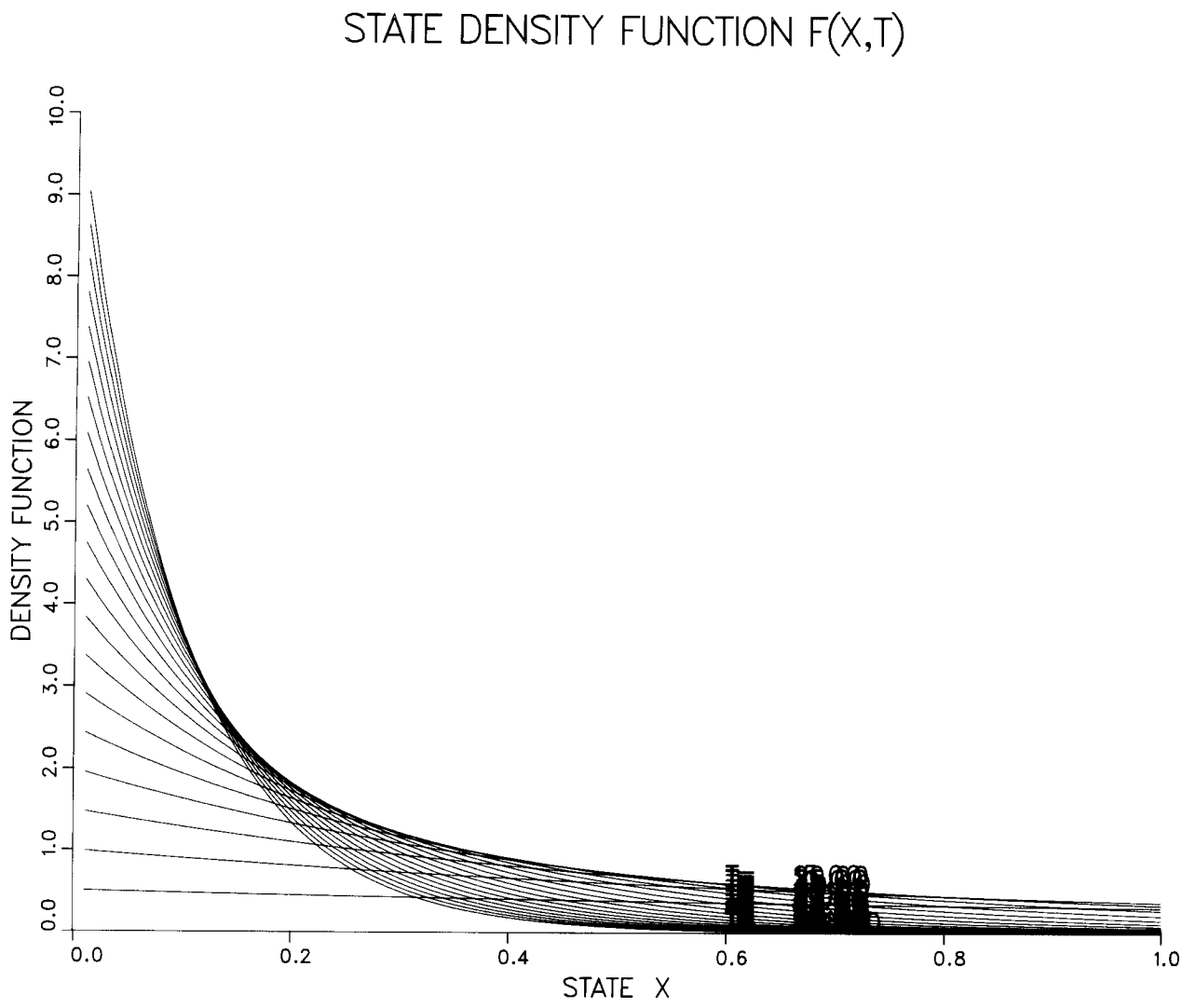

Fig. B.1. Sketch of function f(x,t) given by Eq. (B.16).

APPENDIX C. SOLUTIONS OF EQUATION (2.18)

We solve the following differential equation:

$$\frac{d}{dt}\underline{P} = \underline{\underline{A}}\,\underline{P} \tag{C.1}$$

$$\underline{P}(0) = (0,0,1)^T \tag{C.2}$$

where $\underline{P}(t) = (p_0(t), p_1(t), p_2(t))^T$,

$$\underline{\underline{A}} = \{a_{ij}\}$$

and $a_{11} = -(\mu_{01} + \mu_{02})$, $a_{12} = \lambda_{10}$, $a_{13} = \lambda_{20}$,

$a_{21} = \mu_{01}$, $a_{22} = -(\lambda_{10} + \mu_{12})$, $a_{23} = \lambda_{21}$,

$a_{31} = \mu_{02}$, $a_{32} = \mu_{12}$, $a_{33} = -(\lambda_{20} + \lambda_{21})$.

First let us consider an equation

$$|\underline{\underline{A}} - \beta\underline{\underline{I}}| = 0 . \tag{C.3}$$

This is expanded and rearranged to give a cubic equation

$$\beta(\beta^2 + C_2\beta + C_1) = 0 \tag{C.4}$$

where $c_1 = a_{11}a_{22} + a_{22}a_{33} + a_{11}a_{33} - a_{31}a_{13} - a_{12}a_{21} - a_{32}a_{23}$

$$c_2 = -(a_{11} + a_{22} + a_{33}) .$$

Let D be defined by $$D = c_2^2 - 4c_1 . \tag{C.5}$$

If $D > 0$, Eq. (C.4) has three real roots. If $D < 0$, Eq. (C.4) has one real root ($\beta = 0$) and a pair of complex roots (complex conjugates of each other). According to the sign of D, we use two different methods for the solution:

1. Eigenvector expansion method if $D > 0$.
2. Laplace transformation method if $D < 0$.

Readers should consult any standard applied mathematics book about these methods; for example Ref. [C.1].

Eigenvector expansion method. Suppose that real eigenvalues β_i (i=1,2,3) and three independent eigenvectors $\underline{x}_i$ (i=1,2,3) have been found. Make a matrix $\underline{B}$ by setting vectors $\underline{x}_i$ to rows in the matrix. Multiplying Eq. (C.1) by $\underline{B}^{-1}$ from the left, we have

$$\frac{d}{dt} \underline{\underline{B}}^{-1} \underline{P} = \underline{\underline{B}}^{-1} \underline{\underline{A}} \underline{\underline{B}} \underline{\underline{B}}^{-1} \underline{P} .$$

Since $\underline{\underline{B}}^{-1} \underline{\underline{A}} \underline{\underline{B}}$ is a diagonal matrix with β_i as the elements, this equation is easily integrated

$$\underline{\underline{B}}^{-1} \underline{P} = (d_1 e^{\beta_1 t}, d_2 e^{\beta_2 t}, d_3 e^{\beta_3 t}) .$$

Multiplying this by $\underline{\underline{B}}$ from the left and using the initial condition, the solution vector $\underline{P}(t)$ is obtained.

Laplace transformation method. Let the Laplace transform of a function f be denoted by $\hat{f}$. Then the Laplace transform of Eq. (C.1) can be represented by

$$\hat{\underline{P}}(s) = \underline{\underline{A}}\,\hat{\underline{P}}(s) + \underline{P}(0) \tag{C.6}$$

where $\hat{P}(s) = (\hat{p}_0, \hat{p}_1, \hat{p}_2)^T$,

$$\hat{p}_0 = (a_{13}s + a_{12}a_{23} - a_{13}a_{22})/\Delta$$

$$\hat{p}_1 = (a_{23}s + a_{13}a_{21} - a_{11}a_{23})/\Delta$$

$$\hat{p}_2 = (s^2 - (a_{11} + a_{22})s + a_{11}a_{22} - a_{12}a_{21})/\Delta$$

and $\Delta = s(s^2 + C_2 s + C_1)$ (Eq. (C.4)) .

The inverse Laplace transforms of these functions are easily obtained if $D < 0$. See for example Appendix A in Ref. [C.1]. A computer program has been developed by utilizing the methods discussed above. The time-dependent state probabilities $p_0(t)$ (state 0), $p_1(t)$ (state 1), and $p_2(t)$ (state 2) are plotted in Figs C.1 and C.2 for a nonreparable case and a reparable case.

Reference

C.1. L.A. Pipes and L.R. Harvill, Applied Mathematics for Engineers and Physicists, 3rd Edition, McGraw-Hill, Kogakusha, Tokyo (1970).

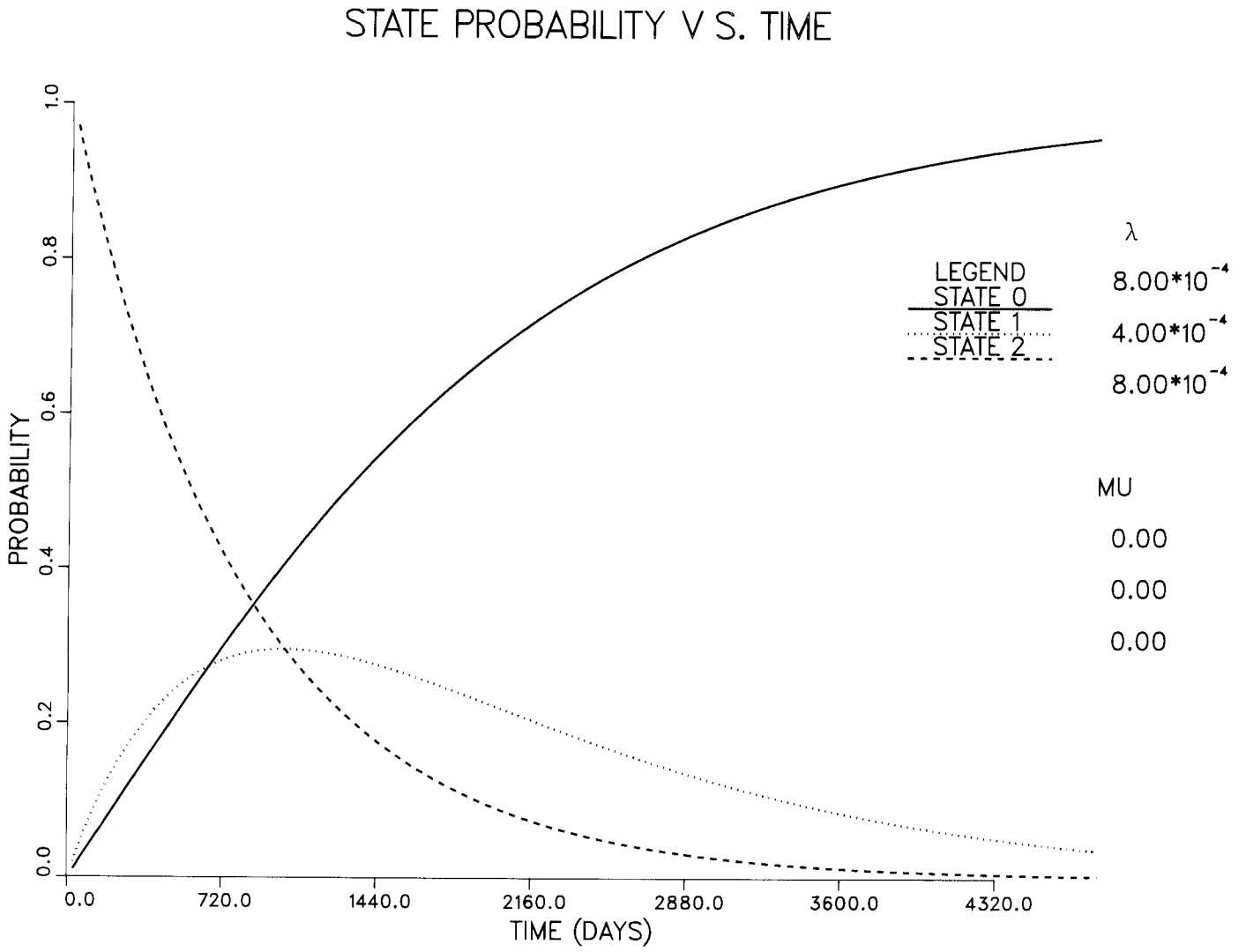

Fig. C.1. Sketch of state probabilities for a nonreparable case, $\lambda_{10} = \lambda_{21} = 8.0 \times 10^{-4}$, and $\lambda_{20} = 4.0 \times 10^{-4}$.

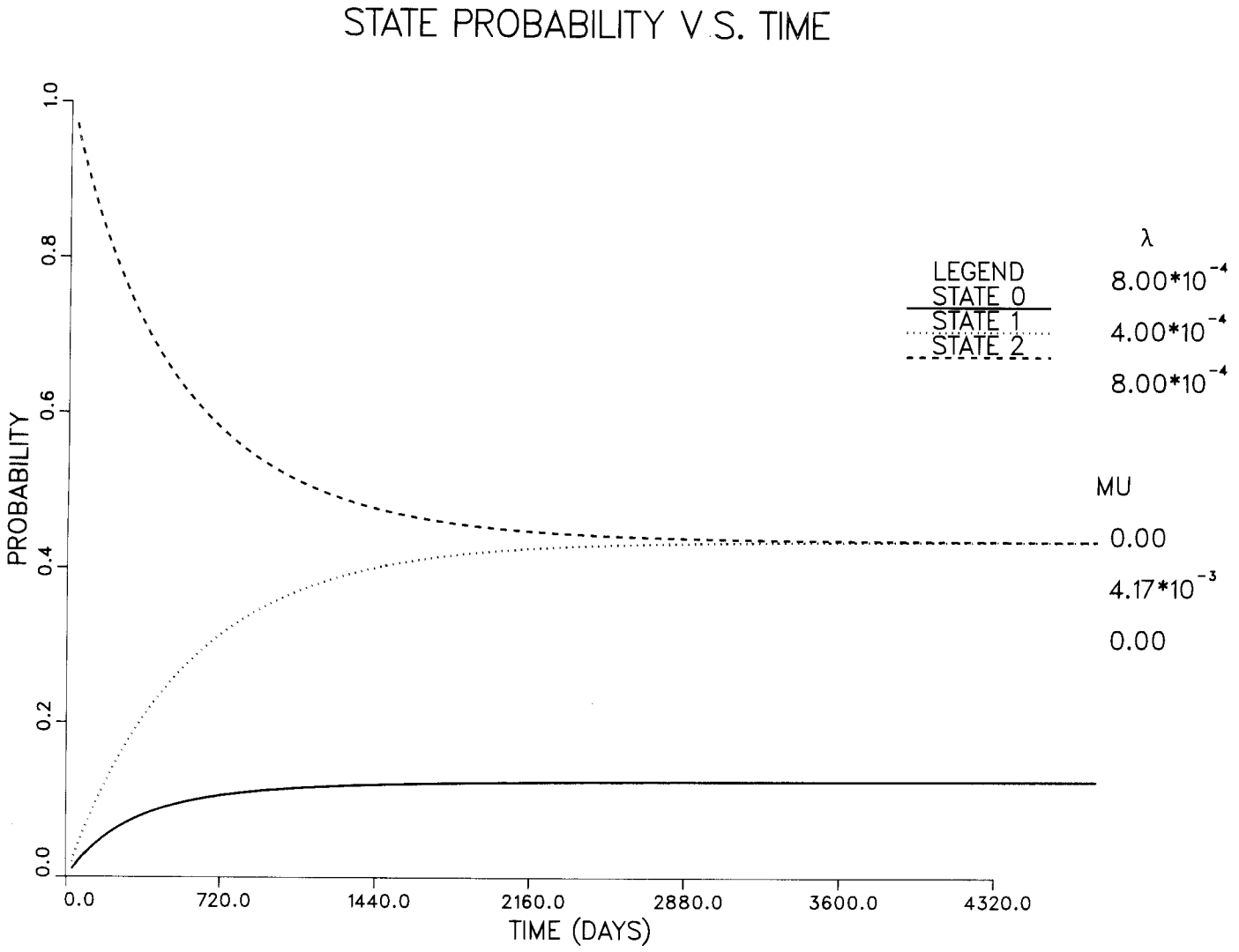

Fig. C.2. Sketch of state probabilities for a reparable case, $\lambda_{10} = \lambda_{21} = 8.0 \times 10^{-4}$, $\lambda_{20} = 4.0 \times 10^{-4}$, and $\mu_{02} = 4.17 \times 10^{-3}$.

REFERENCES

1. E.J. Henley and H. Kumamoto, Reliability Engineering and Risk Assessment, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1981).

2. R.R. Britney, "The Reliability of Complex Systems With Dependent Subsystem Failures: An Absorbing Markov Chain Model," Technometrics 16, 245-250 (1974).

3. J.D. Murchland, "Fundamental Concepts and Relations for Reliability Analysis of Multistate Systems," Reliability and Fault Tree Analysis, pp. 581-618, SIAM, Philadelphia, (1975).

4. C.L. Proctor and B. Singh, "The Analysis of a Four-State System," Microelectronics and Reliability 15, 53-55 (1976).

5. E. El-Neweihi, F. Proschan and J. Sethuraman, "Multistate Coherent Systems," J. Applied Probability 15, 675-688 (1978).

6. L. Caldarola, "Fault Tree Analysis of Multistate Systems With Multistate Components," Proc. ANS Topical Meeting on Probabilistic Analysis of Nuclear Reactor Safety, Los Angeles, CA, Paper VIII.1 (May 1978).

7. S. Contini, S. Fumagalli, P. Mussio, F. Naldi, S. Garriba, and G. Volta, "Multiple-Valued Logic in Modeling Nuclear Safety Systems and Automated Search for Fault Tree," Proc. ANS Topical Meeting on Probabilistic Analysis of Nuclear Reactor Safety, Los Angeles, CA, Paper XIII.2 (May 1978).

8. R.E. Barlow and A.S. Wu, "Coherent Systems with Multistate Components," Mathematics of Operations Research 3, 275-281 (1978).

9. K. Gopal, K.K. Agarwal and J.S. Gupta, "Reliability Analysis of Multistate Device Networks," IEEE Trans. Reliability R-27, 233-236 (1978).

10. E.A. Elsayed and A. Zebid, "A Reparable Multistate Device," IEEE Trans. Reliability R-28, 81-82 (1979).

11. Y. Hatoyama, "Reliability Analysis of 3-State Systems," IEEE Trans. Reliability R-28, 386-393 (1979).

12. L. Caldarola, "Coherent Systems With Multistate Components," Nucl. Engr. and Design 58, 127-139 (1980).

13. J. Ansell, A. Bendell, S. Humble and C.S. Mudhar, "3-State and 5-State Reliability Models," IEEE Trans. Reliability R-29, 176-177 (1980).

14. M. Yamashiro, "A Reparable Multistate Device With Repair Time," IEEE Trans. Reliability R-29, 276 (1980).

15. B.S. Dhillon, "A System With Two Kinds of 3-State Elements," IEEE Trans. Reliability R-29, 345 (1980).

16. M. Yamashiro, "A Multistate System With General Repair Time Distribution," IEEE Trans. Reliability R-29, 433 (1980).

17. M.N. Fardis and C.A. Cornell, "Analysis of Coherent Multistate Systems," IEEE Trans. Reliability R-30, 117-122 (1981).

18. W.S. Griffith, "A Multistate Availability Model: System Performance and Component Importance," IEEE Trans. Reliability R-31, 95-96 (1982).

19. M. Neely, "Guide for the Assessment of the Availability of Gasification-Combined-Cycle Power Plants," EPRI-AP-2202, Electric Power Research Institute, Palo Alto, CA (Jan. 1982).

20. M. Neely and N. Gardner, "Reliability and Availability Analysis of Coal-Fired Units: Validation of a Predictive Methodology," EPRI-AP-2938, Electric Power Research Institute, Palo Alto, CA (March 1983).

21. "User's Guide for the UNIRAM Availability Assessment Methodology," EPRI-AP-3305-CCM, Electric Power Research Institute, Palo Alto, CA (Dec. 1983).

22. H.W. Block and T.H. Savits, "A Decomposition for Multistate Monotone Systems," J. Appl. Prob. 19, 391-402 (1982).

23. A. Lesanovsky, "Availability of a 2-Unit Cold-Standby System With Degraded State," IEEE Trans. Reliability R-31, 123 (1982).

24. D.A. Butler, "Bounding the Reliability of Multistate Systems," Operations Research 30, 530-544 (1982).

25. H. Xizhi, "Fault Tree Analysis Method of a System Having Components of Multistate Failure Modes," Microelectronics and Reliability 23, 325-328 (1983).

26. J.C. Hudson and K.C. Kapur, "Modules in Coherent Multistate Systems," IEEE Trans. Reliability R-32, 183 (1983).

27. L.R. Goel, N.K. Jaiswal and R. Gupta, "A Multistate System with Two Repair Distribution," Microelectronics and Reliability 23, 337-340 (1983).

28. B. Natvig and A. Streller, "The Steady-State Behavior of Multistate Monotone," J. Appl Prob. 21, 826-835 (1984).

29. P.P. Gupta and S.C. Agarwal, "Cost Function Analysis of a 3-State Reparable System," Microelectronics and Reliability 24, 51-53 (1984).

30. P.P. Gupta and S.C. Agarwal, "Cost Analysis of a 3-State 2-Unit Reparable System," Microelectronics and Reliability 24, 55-59 (1984).

31. K.S. Park and S.S. Kim, "Graphic Composition of Three-State Device Redundancies," Microelectronics and Reliability 24, 461-464 (1984).

32. H. Xizhi, "The Generic Method of the Multistate Fault Tree Analysis," Microelectronics and Reliability 24, 617-622 (1984).

33. A. Bossche, "The Top-Event's Failure Frequency for Non-Coherent Multi-state Fault Trees," Microelectronics and Reliability 24, 707-715 (1984).

34. J.L. Roca, "Reliability Analysis of Two Special Three-State Devices," Microelectronics and Reliability 24, 899-900 (1984).

35. Z. Musicki and C.W. Maynard, "AVSYS, A Computer Program for Fusion Availability Calculations," UWFDM-531 (Jan. 1984).

36. L.F. Parsly, "Computer Simulation of Fusion Power Plant Availability," Oak Ridge National Laboratory (1984).

37. J.J. Duderstadt and W.R. Martin, Transport Theory, Section 5.3, John Wiley & Sons, NY, (1979).

38. W.H. Widawsky, "Reliability and Maintainability Parameters Evaluated With Simulation," IEEE Trans. Reliability R-20, 158 (1971).

39. P.W. Becker, "Finding the Better of Two Similar Designs by Monte Carlo Techniques," IEEE Trans. Reliability R-23, 242-246 (1974).

40. G.J. Cadwallader et al., "Nuclear Reliability/Availability Monte Carlo Analysis," Proceeding 1975 Annual Reliability and Maintainability Symposium, (1975).

41. M. Mazumada, "Importance Sampling in Reliability Estimation, Reliability and Fault Tree Analysis," SIAM, 153-163, Philadelphia (1975).

42. S.J. Kamat and N.W. Riley, "Determination of Reliability Using Even-Based Monte Carlo Simulation: Part I," IEEE Trans. Reliability R-24, 73-75 (1975).

43. S.J. Kamat and W.E. Franzmeier, "Determination of Reliability Using Event-Based Monte Carlo Simulation: Part II," IEEE Trans. Reliability R-25, 254-255 (1976).

44. P.L. Noteri et al., "Monte Carlo Methods for Power System Evaluations in Transmission and Generator Planning," Proceeding 1975 Annual Reliability and Maintainability Symposium (1975).

45. H.K. Kumamoto, K. Tanaka and K. Inoue, "Efficient Evaluation of System Reliability of Monte Carlo Method," IEEE Trans. Reliability R-26, 311-315 (1977).

46. M.C. Easton and C.K. Wong, "Sequential Destruction Model for Monte Carlo Evaluation of System Reliability," IEEE Trans. Reliability R-29, 27-32 (1980).

47. E.E. Lewis and F. Böhm, "Monte Carlo Simulation of Markov Uncertainty Models," Nucl. Engr. and Design 77, 49-62 (1984).

48. I.F. Blake, An Introduction to Applied Probability, Section 9.1.3, John Wiley & Sons, NY, (1979).

49. R.W. Hockney and C.R. Jesshope, Parallel Computing, Adam Hilger Ltd., Bristol, (1981).

50. "CFT Compiler Manual," National Magnetic Fusion Energy Computing Center, Livermore, CA, (March 7, 1984).

51. A. Chandak and J.C. Brown, "Vectorization of Discrete Event Simulation," Proceeding of the 1983 International Conference on Parallel Processing, Ohio, Aug. 23-26, 1983.

52. F.B. Brown, "Vectorized Monte Carlo Methods for Reactor Lattice Analysis," Proc. of a Topical Meeting on Advances in Reactor Computations, Salt Lake City, March 28-31, 1983.

53. K. Asai, K. Higuchi, J. Katakura and Y. Kurita, "Vectorization of KENO IV Code," Proc. of the Intern. Meeting on Advances in Nucl. Engineering Comp. Methods, Knoxville, TN, April 9-11, 1985.

54. Z. Musicki, private communication.

55. "MPDOC," National Magnetic Fusion Energy Computer Center, (Sept. 27, 1984).

56. S.W. Brock, L. Dekker, H. De Swaan Arons, "Parallel Random Number Generators for the Use in Parallel Processors," in Parallel Computers-Parallel Mathematics, edited by M. Feilmeir, International Association for Mathematics and Computers in Simulation, pp. 113-117 (1977).

57. P. Fredricson et al., "Pseudo Random Freeze in Monte Carlo," Parallel Computing 1, 175-180 (1984).

58. P. Pearson, "Multitasking - A Worked Example," in NMFECC Buffer 9, No. 5, 1-6 (May 1985).

59. E.J. Lerner, "Parallel Processing: Gets Down to Business," High Technology 5, No. 7, 20-28 (July 1985).

60. H. Bruijnes, "Anticipated Performance of the CRAY-2," in NMFECC Buffer 9, No. 6, 1-3 (June 1985).

61. M. Mazumdar, "Importance Sampling in Reliability Estimation, Reliability and Fault Tree Analysis," in Reliability and Fault Tree analysis, R.E. Barlow et al. (Eds.), SIAM, Phialedphia, pp. 153-163 (1975).

62. Z. Musicki, Ph.D. Thesis, University of Wisconsin-Madison (1984).

63. R Y. Rubinstein, Simulation and the Monte Carlo, John Wiley & Sons, NY, (1981).

64. J. Spanier and E.M. Gelbard, Monte Carlo Principles and Neutron Transport Problems, Addison-Wesley, Reading, MA, (1969).

65. H.W. Block and T.H. Savits, "Continuous Multistate Structure Functions," Operations Research 32, 703-714 (1984).