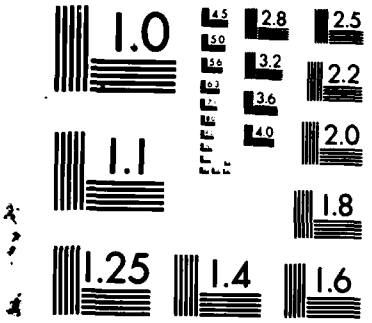
AD-A174 222 SIMULATION METHODOLOGIES FOR TRANSIENT MARKOV PROCESSES: A COMPARATIVE ST..(U) GEORGE WASHINGTON UNIV WASHINGTON D C DEPT OF OPERATIONS RESE.. 1/1

UNCLASSIFIED D GROSS ET AL. 1986 F/G 15/5 NL



Simulation Methodologies for Transient Markov Processes: A Comparative Study Based on Multi-Echelon Repairable Item Inventory Systems*

Donald Gross, Douglas R. Miller, and Christos G. Plastiras
Department of Operations Research
School of Engineering and Applied Science
The George Washington University
Washington, D.C. 20052

ABSTRACT

We consider Monte Carlo simulation of transient behavior of continuous-time Markov processes with large finite state spaces. The usual event-scheduling approach is compared to four alternative approaches which do not use event lists. Variance reduction, programming effort, and storage requirements are considered. All Monte Carlo simulation programs are written in SIMSCRIPT II.5. These approaches are then compared to a deterministic method called Randomization.

The four alternative Monte Carlo simulation methods are based on two dichotomies. The first concerns continuous versus discrete time: the process can be simulated directly in continuous time or the uniformized embedded discrete-time Markov chain can be simulated and then analytically randomized to recover the original continuous-time process. The second dichotomy concerns a table-look-up versus an algorithmic approach: the mean holding time and the transition distribution for the current state of the simulated process can be read from memory (requiring large memory storage) or computed (requiring more computation time).

The above methods are applied to the simulation of a multi-echelon repairable item provisionong system with a finite population of repairable items and limited repair capacity. Various systems are used. Probabilities of system availability are estimated. The variances of these estimates are multiplied by CPU times to get measures of computational efficiency of the different simulation methods.

INTRODUCTION

Let $\{X(t),\ t \geq 0\}$ be a continuous-time time-homogeneous Markov process on a finite state space $S = \{1,2,\ldots,m\}$. All such Markov processes can be characterized by an initial distribution $\underline{\pi}(0)$ and an infinitesimal generator

$$Q = \begin{pmatrix} -q_1 & q_{12} & q_{13} & \cdots & q_{1m} \\ q_{21} & -q_2 & q_{23} & \cdots & q_{2m} \\ q_{31} & q_{32} & -q_3 & \cdots & q_{3m} \\ \vdots & \vdots & \vdots & & \vdots \\ q_{m1} & q_{m2} & q_{m3} & \cdots & -q_m \end{pmatrix}$$

where

$$q_{ij} = \lim_{\Delta t \to 0} \frac{P(X(t+\Delta t)=j \mid X(t)=i)}{\Delta t}, \quad i \neq j$$

and

$$q_i = \sum_{j \neq i} q_{ij}.$$

The q_{ij}'s are the transition rates. The infinitesimal generator Q seems to be the most natural way to describe the stochastic nature of continuous time Markov models with denumerable state spaces. The state probability vector at time t is denoted $\underline{\pi}(t) = (\pi_1(t), \pi_2(t), \ldots, \pi_m(t))$, where $\pi_s(t) = P(X(t) = s)$, $s \in S$. These transient probabilities satisfy the Kolmogorov forward equation

$$\underline{\pi}'(t) = \underline{\pi}(t)Q, \quad t \geq 0. \tag{1}$$

This is an initial value system with $\underline{\pi}(0)$ given. (See [1,2,3] for more background on Markov processes.)

The purpose of this paper is to consider different ways of computing or estimating the above transient probabilities, $\underline{\pi}(t)$, $t \geq 0$, i.e., solution of the Kolmogorov equation (1). There are two general approaches: deterministic algorithms and Monte Carlo algorithms. We shall consider one deterministic algorithm: the Randomization Technique. We shall consider five Monte Carlo approaches for simulating the transient behavior of continuous time Markov processes.

THE RANDOMIZATION ALGORITHM

Any Markov process X on a finite state space can be represented as a discrete time Markov chain (the uniformized embedded chain) "randomized" by a Poisson process. Define

$$P = Q/\Lambda + I, \quad \text{where } \Lambda = \max_{i \in S} q_i \tag{2}$$

and I is the identity matrix; P is a stochastic matrix. Let $\{Y_n,\ n = 0,1,2,\ldots\}$ be a Markov chain on S with transition matrix P and initial distribution $\underline{\pi}(0)$. Let $\{N(t),\ t \geq 0\}$ be a Poisson process with rate Λ which is independent of $\{Y_n,\ n = 0,1,2,\ldots\}$. Then $\{Y_{N(t)},\ t \geq 0\}$ is a Markov process with generator Q and initial distribution $\underline{\pi}(0)$ and hence is probabilistically identical to $\{X(t),\ t \geq 0\}$. This construction makes it possible to compute transient probabilities of a Markov process with generator Q from transient probabilities of a Markov chain Y with transition matrix P and a Poisson process N with rate Λ. The transient probabilities of Y are denoted $\underline{\nu}(n) = (\nu_1(n), \nu_2(n), \ldots, \nu_m(n))$, where $\nu_s(n) = P(Y_n = s)$, $s \in S$. The randomization formula is

*Research supported by the Office of Naval Research under Contract N00014-83-K-0217 with the George Washington University.

$$P(X(t) = s) = \sum_{n=0}^{\infty} P(X(t)=s \mid N(t)=n)P(N(t)=n)$$

$$= \sum_{n=0}^{\infty} P(Y_n = s)P(N(t) = n)$$

or equivalently,

$$\underline{\pi}(t) = \sum_{n=0}^{\infty} \underline{\phi}(n) \frac{e^{-\Lambda t}(\Lambda t)^n}{n!}. \qquad (3)$$

See Gross and Miller [4] for additional discussion and details. (Equation (3) can also be found in Çinlar [1, p. 259].)

The infinite series in Equation (3) must be truncated for computational purposes. Let

$$T(\varepsilon,t) = \min \left\{ k: \sum_{n=0}^{k} \frac{e^{-\Lambda t}(\Lambda t)^n}{n!} > 1 - \varepsilon \right\} \qquad (4)$$

where ε equals an acceptable error (specified by the user). The computational version of Equation (3) is

$$\underline{\pi}^{\varepsilon}(t) = \sum_{n=0}^{T(\varepsilon,t)} \underline{\phi}(n) \frac{e^{-\Lambda t}(\Lambda t)^n}{n!}. \qquad (5)$$

Truncation of the infinite series involves a probability loss of at most ε; thus all probabilities (of states or subsets of states) will have an error between $-\varepsilon$ and 0. Note that the randomization formula (5) reduces the calculation of transient probabilities of a Markov process to those of a Markov chain and underlying Poisson process, both of which are more amenable to exact numerical evaluation.

The ϕ's are computed recursively using the relation from standard Markov chain theory:

$$\underline{\phi}(0) = \underline{\pi}(0); \quad \underline{\phi}(n+1) = \underline{\phi}(n)P, \quad n \geq 0. \qquad (6)$$

(Note that Equation (6) involves only nonnegative numbers, a fact that contributes to numerical stability of the algorithm.) The matrix P is usually sparse and thus the above matrix multiplication should be performed by an appropriate algorithm. Such a multiplication algorithm is described by Gross and Miller [4]. The number of operations in this algorithm is proportional to the sum of the number of states and the number of transitions.

In short, the standard randomization computational algorithm computes Λ and P from the generator Q using (2). It computes the truncation point $T(\varepsilon,t)$ from (4), then the $\underline{\phi}(n)$'s using (6) recursively, accumulating in Equation (5) to give $\underline{\pi}^{\varepsilon}(t)$.

Gross and Miller [5] have computed transient probabilities for Markov process models of multi-echelon inventory systems with 20,000 states and 200,000 transitions using the randomization algorithm. Melamed and Yadin [6] have applied the method to queuing networks with a large number of states. Miller [7] has adapted the randomization algorithm to efficiently handle certain stiff systems which arise in the reliability analysis of fault-tolerant systems.

There are other deterministic approaches to the solution of the Kolmogorov equation (1) which we will not consider. Two general approaches are: (i) numerical integration techniques such as Runge-Kutta, predictor-corrector, etc.; and (ii) exponentiation [$\underline{\pi}(t) = \underline{\pi}(0)e^{Qt}$] by computing the spectrum, computing the Taylor series, or other means. The randomization technique has a distinct advantage over these approaches in that a bound on the global error can be set by the user, and it is achieved. (The only other source of error is the influence of rounding and truncations by the machine performing the calculations; by noting that the randomization algorithm mainly involves multiplication and addition of positive numbers, Grassmann [8] has bounded this error.) Furthermore, Grassmann [9] has shown empirically that randomization is more efficient for some queuing systems.

We use the randomization algorithm as a deterministic method for solving the Kolmogorov equation (1). We also use the concept of randomizing a discrete-time Markov chain (as described above) as part of a Monte Carlo simulation method.

MONTE CARLO SIMULATION METHODS

We use five Monte Carlo simulation methods. The first uses the classical world view of "future event scheduling." The others exploit the Markovian nature of the system and use no future event list.

Event-scheduling Method (ES)

The classical event-scheduling method is well known. It is used by languages such as SIMSCRIPT. An event is defined as a change of state of the system. The occurrence of one event often triggers the occurrence of another event in the future; thus, when the first event occurs the simulation algorithm schedules the future occurrence of the other event by creating an event notice and filing it in a "future event list." Advancing simulated time requires searching the event list for the next scheduled event and noting its time of occurrence. The event-scheduling approach is very general; it is applicable to virtually all discrete event systems.

Simulating Markov Processes

If the system is Markovian, there may be advantages in exploiting that property. An event-scheduling algorithm can be used, but it may consume considerable execution time for filing and retrieving notices from the future event list if the simulated system is even slightly complex. It may be more economical to simulate a Markovian system directly as a Markov process. There are various ways to do this, for example, see Hordijk, Iglehart, and Schassberger [10] or Fishman [11].

Consider a continuous time Markov process $\{X(t), t \geq 0\}$ with initial probability vector $\underline{\pi}(0)$ and infinitesimal generator Q. Let T_n equal the time of the n-th transition of X, $n = 1,2,\ldots$, $T_0 = 0$. The process $\{Z_n = X(T_n), n = 0,1,2,\ldots\}$ is an embedded discrete-time Markov chain (jump chain). The transition matrix of Z is

$$R = \begin{Vmatrix} 0 & r_{12} & r_{13} & \cdots & r_{1m} \\ r_{21} & 0 & r_{23} & \cdots & r_{2m} \\ r_{31} & r_{32} & 0 & \cdots & r_{3m} \\ \vdots & \vdots & \vdots & & \vdots \\ r_{m1} & r_{m2} & r_{m3} & \cdots & 0 \end{Vmatrix} \qquad (7)$$

where $r_{ij} = q_{ij}/q_i$, $i \neq j$. The continuous-time Markov process X is really a semi-Markov process (see [1,2, 3]) with transition matrix R and exponential holding times: $P(T_{n+1} - T_n > t \mid Z_n = i) = \exp(-q_i t)$; $i = 1, \ldots,m$; $n = 0,1,2,\ldots$. This structure will be used in two Monte Carlo simulation algorithms.

Table-look-up Continuous-time Method (TC). In this Monte Carlo algorithm we use the structure of embedded (jump) chain and exponential holding times. The transition matrix R of the jump chain and the vector of rates $(q_1,q_2,\ldots,q_m)$ are stored. To simulate one sample path $\{x(t), 0 \leq t \leq s\}$ of X we proceed as follows:

(i) let $t = 0$;
(ii) compute $x(t)$, a random variate from the probability vector $\underset{\sim}{\pi}(0)$, using the inverse cdf method, see [11];
(iii) generate j, a random variate whose probability vector is the $x(t)$-th row of R, using the inverse cdf method;
(iv) generate an exponential (rate $q_{x(t)}$) random variate and add it to t;
(v) let $x(t) = j$;
(vi) repeat steps (iii) through (v) until $t \geq s$.

This TC algorithm has the advantage of being quite simple and fast. It has the disadvantage of requiring storage of all nonzero elements of R and the vector of rates, $\underset{\sim}{q} = (q_1,q_2,\ldots,q_m)$. Also, complex systems may have generators which require sophisticated programs for their computation.

Algorithmic Continuous-time Method (AC). This algorithm is the same as TC except instead of storing the rate vector $\underset{\sim}{q}$ and all the nonzero transition probabilities in R, an algorithm computes the values as they are needed. In this approach the state x of the system (which itself may be a vector) is more descriptive than with TC, where states are simply numbered; the state descriptions x for AC are the same as or similar to ES. Suppose there are at most k events that can occur: EV_1, EV_2, ..., EV_k. To simulate a sample path using AC we proceed as follows:

(i) let $t = 0$;
(ii) generate an initial state x_0 and let $x(t) = x_0$;
(iii) compute rates of occurrence $q(x(t),i)$ of event EV_i, $i = 1,2,\ldots,k$, given the current state $x(t)$, let

$$q_{x(t)} = \sum_{i=1}^{k} q(x(t),i), \quad i \neq x(t);$$

(iv) generate the next event to occur using a Monte Carlo computation on the state descriptor $x(t)$, then call the appropriate event routine (similar to ES) to compute the change of state descriptor to get a new state x;
(v) generate an exponential (rate $q_{x(t)}$) random variate and add it to t;
(vi) let $x(t) = x$;
(vii) repeat steps (iii) through (vi) until $t \geq s$.

AC has the drawback of requiring more computation at each transition of X. However, AC has the advantage of small memory requirements, the capability of handling infinite state spaces, and simpler transition rules which are closely related to physical properties of the system.

Table-look-up Discrete-time Method (TD). This method is the same as TC except we simulate the uniformized embedded discrete-time Markov chain $\{Y_n, n = 0,1,\ldots\}$ which has initial distribution $\underset{\sim}{\pi}(0)$ and transition matrix $P = Q/\Lambda + I$; see Equation (2). We then convert this to the continuous-time process $\{X(t), t \geq 0\}$ by randomizing, i.e., using Equation (5). In particular, we simulate the sample path $\{y_n, n = 0, 1,2,\ldots,m\}$ as follows:

(i) let $n = 0$;
(ii) compute y_n, a random variate from the probability vector $\underset{\sim}{\pi}(0)$;
(iii) generate j, a random variate whose probability vector is the y_n-th row of $P = Q/\Lambda + I$;
(iv) generate a geometric random variate with parameter $p(y_n) = (1 - q_{y_n}/\Lambda)$, i.e., $P(G = m) = (p(y_n))^{m-1}(1 - p(y_n))$, and add it to n;
(v) let $y_n = j$;
(vi) repeat (iii) through (v) until $n \geq m$.

By taking r replicates we can estimate $P(Y_n = i)$ using the sample proportion $\hat{\pi}_i(n)$ of replicates for which $y_n = i$, $i \in S$, $n = 0,1,2,\ldots,m$. Using Equation (5), if $m \geq T(\epsilon,t)$,

$$\sum_{n=0}^{m} \hat{\pi}_i(n) \frac{e^{-\Lambda t}(\Lambda t)^n}{n!} \tag{8}$$

is an estimate of $\pi_i(t) = P(X(t) = i)$.

Algorithmic Discrete-time Method (AD). This approach is the same as TD except that the discrete-time Markov chain $\{Y_n, n = 0,1,2,\ldots\}$ is simulated using algorithms to compute and recompute the appropriate rates and probabilities when they are needed, rather than getting them from precomputed and stored values. The probabilities related to Y are then transformed into probabilities related to the continuous-time process X by using the randomization formula, Equation (8).

The two algorithms using discrete-time simulation and then randomization are attractive because they offer the potential for variance reduction of estimates based on the simulation: from the above model of the continuous-time Markov process it is clear that randomness can be attributed to two independent sources: the discrete-time chain $\{Y_n, n = 0,1,\ldots\}$ and the Poisson process $\{N(t), t \geq 0\}$. The methods TD and AD simulate Y using Monte Carlo and treat $N(\cdot)$ deterministically, thus there should be less variation in estimators based on TD and AD than in the same estimators based on TC and AC. Grassmann [12] has also pointed this out. The question is whether increased computation negates the benefit of the decreased variance of the estimator.

MULTI-ECHELON REPAIRABLE ITEM SYSTEMS

Multi-echelon repairable item provisioning systems are generalizations of the classic machine repair model. We consider a system consisting of two bases and a depot. Each base has a certain number of machines assigned to it and a certain desired number of these which should be operating. Machines fail (independently of each other) after being operated for an exponentially distributed length of time. There are repair shops at each base and the depot. When a

machine fails it has a certain probability of going to the base repair shop for repair; otherwise it goes to the depot repair shop. Each repair shop has a certain number of repair channels. Repair times are exponentially distributed. If there are more machines requiring repair than repair channels at a given repair shop, a queue forms. The depot repair shop stocks spare machines which are used on a one-to-one ordering basis to replace failed machines coming from the bases. If the depot spares pool is empty when a replacement is required, a backorder is created which will be filled when repair is completed on one of the items in depot repair. If both bases are awaiting backorders, a repaired item is sent to the base with the maximum depot backorders (ties are broken by flipping a fair coin). When neither base is awaiting a backorder, a machine completing depot repair is placed in the depot spares pool. Thus any given machine may be in any of six states: failed and in base repair shop at either base (BR1, BR2); failed and in depot repair shop (DR); operational and at either base (BU1, BU2); operational and in the depot spares pool (DU). These six states and the possible transitions a machine can make between them is illustrated in Figure 1. The different parameters of the system are described in Table 1.

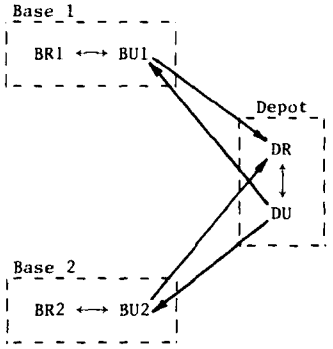


Figure 1: General schematic for a two-base multi-echelon repairable item system

Table 1:
Parameters of Multi-echelon System

Symbol	Definition
BS_i	Allocation of total stock to Base i (operating machines plus spares), i = 1,2
MS_i	Desired number of working machines at Base i
BC_i	Number of repair channels in repair shop at Base i
FB_i	Probability machine failing at Base i is base repairable
λ_i	Failure rate, Base i items
μ_i	Repair rate, Base i items
DS	Number of depot spares
DC	Number of depot repair channels
μ_D	Depot repair rate

Steady-state models and behavior of multi-echelon inventory systems are presented by Sherbrooke [13] and Muckstadt [14]. A method for computing approximate transient performance measures of the above multi-echelon system is presented in [15,16]. Gross and Miller [5] compute exact transient probabilities using the randomization algorithm. For further background on multi-echelon inventory systems, see the above references.

We used various cases of the above multi-echelon system as tests for comparing different simulation methods. Some of the cases considered are shown in Table 2. These systems are initially in perfect condition with no failed machines.

COMPUTATIONAL EXPERIENCE

We wrote SIMSCRIPT II.5 programs to simulate the above multi-echelon inventory model. Five programs were based on the five Monte Carlo methodologies. The event-scheduling (ES) program is written in the style of a typical SIMSCRIPT program: the SIMSCRIPT timing routine, random variate generators, and future event list are used. The bases and depot are modelled as permanent entities. There are three types of events: failure, completion of repair at base, and completion of repair at depot. The four methods for Monte Carlo simulation of Markov processes use SIMSCRIPT in the style of a general purpose language rather than a simulation language. The SIMSCRIPT random number generator and exponential random variate generator are used. Geometric random variates are generated by a routine that either uses the inverse cdf method or adds the integer part of an exponential random variate to 1, depending on the value of the geometric parameter (see [11]). The AD and TD programs use $\varepsilon = .0025$ to set the truncation value $T(\varepsilon,t)$ in Equation (5). The lower tail of the Poisson distribution of N(t) was also truncated, throwing away a probability of at most .0025.

The deterministic randomization algorithm (DET) was programmed in FORTRAN. It is described by Gross and Miller [5]. A truncation probability of $\varepsilon = .001$ is used in Equation (5) for this algorithm; this guarantees that all computed probabilities have an error between -.001 and 0. Thus we are implicitly declaring an indifference to this size of error.

Table 2:
Parameter Values of Multi-echelon Cases

#	Base 1						Base 2						Depot		
	BS	MS	BC	FB	λ	μ	BS	MS	BC	FB	λ	μ	DS	DC	μ
A	4	2	2	0.7	0.3	0.5	5	3	2	0.5	0.3	0.9	2	2	0.3
B	9	7	3	0.7	0.2	0.6	10	7	3	0.5	0.3	0.9	3	3	0.8
C	14	10	3	0.8	0.2	1.0	14	9	3	0.7	0.2	0.9	3	3	1.5
D	18	3	1	1.0	0.5	0.25	1	1	1	1.0	1.0	0.5	1	1	0.75
E	18	3	1	1.0	1.0	0.5	2	1	1	1.0	1.25	0.25	1	1	1.0
F	18	1	1	0.1	1.0	0.01	1	1	1	1.0	2.0	2.0	3	1	0.01
G	15	1	1	1.0	1.0	0.05	1	1	1	1.0	0.00	0.00	1	1	0.00
H	15	1	1	1.0	1.0	0.05	2	1	1	1.0	5.0	5.0	1	1	0.01
J	2	1	1	1.0	1.0	1.0	1	1	1	1.0	0.00	0.00	1	1	0.00
K	4	2	2	1.0	1.0	1.0	1	1	1	1.0	0.00	0.00	1	1	0.00
L	16	8	4	1.0	1.0	2.0	1	1	1	1.0	0.00	0.00	1	1	0.01

The performance measures of interest are availabilities at each base and for the whole system. For i = 1 and 2, let $M_i(t)$ equal the number of operational machines at base i; if $M_i(t) = MS_i$ we have full availability at base i. Thus we define the following availability measures:

$$Av1(t) = P(M_1(t) = MS_1)$$

$$Av2(t) = P(M_2(t) = MS_2)$$

$$Av12(t) = P(M_1(t) = MS_1 \text{ and } M_2(t) = MS_2),$$

for $t \geq 0$. We shall compare the different simulation algorithms by the efficiency with which they estimate Av1, Av2, and Av12.

All five Monte Carlo programs are structured similarly for sample path replication and data collection. Estimates of Av1, Av2, and Av12 are based on blocks of 100 replicates: for ES, AC, and TC 100 independent replicates are generated, the availability at time t of each noted, and the sample proportion computed; for AD and TD 100 independent replicates of the discrete-time chain $\{Y_n, n = 0,1,\ldots,T(\varepsilon,t)\}$ are generated, the availabilities at each discrete-time point are noted, the sample proportions for each discrete-time point computed and then converted to estimates of availabilities at the continuous-time point t using Equation (8). To get more accurate estimates, multiple blocks (of 100 replicates each) are generated and the estimates from each are averaged to obtain single estimates for Av1, Av2, and Av12.

The efficiency measure of a program is the product of the variance of the estimator and the CPU time required to execute the program:

$$\text{Efficiency} = \text{Variance} * \text{CPU time}.$$

We estimate the efficiency by observing the CPU time for each execution and estimating the variance (for AD and TD) or using the exact value of the variance (for ES, AC, and TC). The CPU times that we use for comparison do not include the time required to compute the generator in the table-look-up method (for cases A, B, and C these times were .82, 8.10, and 31.24 seconds, respectively); thus, when making final comparisons these times must be added as overhead. The DET algorithm is used to compute the exact values of Av1, Av2, and Av12. For ES, AC, and TC the variances of estimators based on 100 replicates are Av1(1-Av1)/100, Av2(1-Av2)/100, and Av12(1-Av12)/100, respectively; thus, these values are computed directly (by DET) and the only purpose of the Monte Carlo programs is to obtain estimates of CPU execution times: 10 blocks (of 100 replicates each) were computed for this purpose. For AD and TD, 60 blocks of 100 replicates each were computed, yielding 60 independent estimates of the availabilities; the variance of these 60 estimates about the true availability (computed by DET) was computed and used as an estimate of the variance of the availability estimator for each of the three availabilities.

We initially simulated three cases of the multi-echelon inventory system (Cases A, B, and C in Table 2). These have typical parameter values; state spaces of 375, 3960, and 15075 states, respectively; and time points which encompass transient and steady-state behavior. We then ran additional cases (including D through L) to gain more insight into the behavior of the "randomized discrete-time" approaches (AD and TD). All programs were run on a VAX 11/780 at the School of Engineering and Applied Science, George Washington University.

Comparison of Simulation Methods

We expected certain patterns in the performance comparisons of the different estimators. Using "table-look-up" should be more efficient than "algorithmic" if the time to compute the table is ignored. AC should be slightly better than ES because they use similar logic but AC does not spend time managing an event list. We expect the "randomized discrete-time" approach to do better for large t than small t because of $T(\varepsilon,t)$ in Equation (5): the rate Λ of the underlying Poisson process is computed as

$$\Lambda = MS_1*\lambda_1 + BC_1*\mu_1 + MS_2*\lambda_2 + BC_2*\mu_2 + DC*\mu_D.$$

The expected number of events of N (and discrete-time values for Y) in $[0,t]$ is Λt, but the "randomized discrete-time" approach must simulate Y for $0 \leq n \leq T(\varepsilon,t)$. The relative extra time $(T(\varepsilon,t) - \Lambda t)/\Lambda t$ can be large, especially for small t. Thus, while expecting a smaller variance using AD or TD, we also expect larger CPU times, especially for small Λt. To get a better idea of how the methods perform we must look at some comparison data.

Markov Simulations vs. ES. The efficiencies of the four Markov-oriented Monte Carlo simulations are compared with ES in Table 3. All four are apparently superior to ES. Note, however, that AC is about the same as ES for large times. The others (AD, TC, and TD) require significantly more programming effort than ES. (The ratios in Table 3 are estimates of the ratios of efficiencies. They are approximate because CPU times are approximate and because the variance is estimated for AD and TD. Under the hypothesis of unit ratios and deterministic CPU times, these two sample ratios have a $\chi^2/df(60)$ distribution. Since $\chi^2/df(60;.05) = .720$ all the ratios for AD/ES and TD/ES are statistically significant.)

Algorithmic vs. Table-look-up. The ratios of efficiencies AC/TC and AD/TD are shown in Table 3. There is a clear pattern of superiority of "table-look-up" over "algorithmic." Note, however, that the "table-look-up" is generally more difficult to program. (Under the hypothesis of equal efficiencies, the ratio AD/TD has an F(60,60) distribution. Since $F(60,60;.95) = 1.53$, most of these values are significant.)

Table 3:
Comparisons of Markov vs. ES and A vs. T

Case	t	Efficiency Ratios of Estimators of Av12					
		AC/ES	AD/ES	TC/ES	TD/ES	AC/TC	AD/TD
A	3	.58	.72	.33	.39	1.74	1.83
A	14	.87	.56	.50	.45	1.74	1.25
A	95	.95	.52	.55	.26	1.74	2.05
B	1	.32	.39	.18	.30	1.81	1.30
B	6	.70	.55	.52	.41	1.34	1.34
B	40	.90	.38	.58	.22	1.55	1.72
C	5	.63	.63	.40	.46	1.57	1.37
C	30	.90	.55	.54	.21	1.67	2.68

Continuous-time vs. Randomized Discrete-time. Table 4 presents comparisons of variances, CPU times, and efficiencies for AD vs. AC and for TD vs. TC. We see that the "discrete-time" method always produces a significant decrease in the variance of the estimators of availability, but also a large increase in CPU time. Resulting efficiencies are sometimes improved (made smaller) and sometimes not. For Cases A, B, and C we see a pattern whereby the "randomized discrete-time" method works well for large t but not for small t. Initially we believed this pattern was caused only by the value of $\Lambda = (T(\varepsilon,t)-\Lambda t)/\Lambda t$ discussed earlier. But further investigation (Cases D through L) showed that while a large value of Λ usually doomed the randomization approach, a small value of Λ was no guarantee of success. For Cases A, B, and C the randomization approach works well for cases where the process has reached steady-state equilibrium behavior but not during initial transient periods. It is interesting to note that Nordijk, Iglehart, and Schassberger [10] used regenerative steady-state estimates based on the uniformized embedded chain Y and achieved approximately the same amount of variance reduction.

Consider the behavior at base 2 in Cases D, E, F, and H: In D and F the number of good machines is a two-state Birth and Death process, in E and H a three-state Birth and Death process. Initial transients are very strong in Cases D and E, while in Cases F and H the systems have reached equilibrium steady-state behavior. In Cases F and H the randomization improves efficiency but in D and E it makes it worse, in spite of the fact that variances were decreased.

Table 4:
Comparisons of Continuous-time vs. Discrete Time

Case	t	Perf.	Ratios for Estimators of Performance: Variance AD/AC	Variance TD/TC	CPU Time AD/AC	CPU Time TD/TC	Efficiency AD/AC	Efficiency TD/TC
A	3	Av12	.48	.40	2.59	2.96	1.249	1.187
A	14	Av12	.38	.46	1.70	1.93	.641	.894
A	95	Av12	.45	.35	1.23	1.33	.551	.468
B	1	Av12	.44	.46	2.77	3.71	1.220	1.697
B	6	Av12	.44	.47	1.80	1.68	.785	.786
B	40	Av12	.33	.31	1.28	1.24	.423	.381
C	5	Av12	.60	.66	1.67	1.74	1.004	1.150
C	30	Av12	.51	.30	1.20	1.26	.614	.384
D	.5	Av2	.61	.42	2.83	2.88	1.716	1.213
E	1.5	Av2	.60	.56	2.24	2.69	1.356	1.514
F	21	Av1	.50	.66	1.35	1.56	.680	1.031
F	21	Av2	.042	.049	1.35	1.56	.057	.076
G	15	Av1	.076	.092	1.49	1.86	.113	.171
H	15	Av1	.59	.74	1.33	1.35	.787	.998
H	15	Av2	.063	.051	1.33	1.35	.084	.069
J	12	Av1	.13	.15	1.88	2.78	.249	.415
K	12	Av1	.17	.21	1.64	1.81	.283	.372
L	16	Av1	.40	.35	1.21	1.21	.483	.428

Consider the behavior of base 1 in Cases J, K, and L: We have 3, 5, and 17 state Birth and Death processes, respectively. They are all in steady state and there is no activity at the depot or base 2. We get approximately the same improvement using randomization, suggesting that the size of the state space may not have much effect.

Consider the behavior of base 1 in Cases F, G, and H: In each case, as t increases the availability at base 1 remains very close to 1.00 and then decreases rapidly to close to 0.0 (around t = 21 for F and t = 15 for G and H). Thus we are estimating Av1 at nonequilibrium. Note that significant improvement in efficiency occurs in Case G but not in Cases F and H, and further note that there is no activity in the rest of system G and much activity in systems F and H. System G at t = 15 is the only example we have found where the randomized discrete-time approach improves efficiency in a transient situation; this example is characterized by an almost deterministic embedded Markov chain and a moderate value of $(T(\varepsilon,t)-\Lambda t)/\Lambda t$.

ES vs. Deterministic Randomization Algorithm. In order to compare an exact method (DET) with a Monte Carlo method (ES) we modify the comparison procedure. Consider confidence interval estimation of p, the parameter of a Bernoulli population: In general, a 95% confidence interval with half-width of .03 requires a sample of size 1068. In general, a 99% confidence interval with half-width of .01 requires 16590. In order to compare ES to DET, a desired confidence interval is specified such as one of the above two and the comparison of ES to DET equals the fraction of CPU.DET which ES needs to generate the necessary sample size. Table 5 gives such a comparison. It can be seen that the comparison depends on the desired level of accuracy; for example, to estimate Av12 for system C within ±.03 with 95% confidence would require at most only 19% of the CPU time required by DET; but to have ±.01 accuracy with 99% confidence may require triple the CPU time. The deterministic algorithm is preferred for smaller state spaces or when a high degree of accuracy is needed. Monte Carlo simulation is generally infeasible for estimating probabilities with ±.001 error, see [17].

Table 5:
Comparison of ES vs. Deterministic Algorithm

Case	No. of States	t	CPU Seconds ES*	CPU Seconds DET	$r = \frac{CPU.DET}{CPU.ES}$ †	Efficiency Ratio‡ (ES/DET) 1068/r	Efficiency Ratio‡ (ES/DET) 16590/r
A	375	14	.0460	5.51	119.70	8.92	138.60
A	375	95	.2705	32.81	121.28	8.81	136.79
B	3960	6	.0562	55.49	986.84	1.08	16.81
B	3960	40	.3136	305.79	975.21	1.10	17.01
C	15075	5	.0596	332.80	5586.70	.19	2.97
C	15075	30	.2651	1604.20	6049.70	.18	2.74

*Average time per replicate.

†r equals the number of ES replicates executed in time required for DET execution.

‡Ratios of CPU times required to achieve at least 95% confidence interval width ±.03 and 99% confidence interval width ±.01, respectively.

The stiffness of the system plays a major role in the comparison of ES vs. DET. If the diagonal of the P, the transition matrix of Y, has at least one zero element but many or most of the other elements are close to 1, Y will have many null transitions. Randomization (DET) loses much of its efficiency for such systems; see Miller [7] for a discussion and a modified algorithm.

CONCLUSIONS

It is difficult to accurately compare different algorithms. CPU times are machine, language and compiler dependent. In a time-share/multiprogramming environment these timings tend to exhibit randomness. Behavior of the algorithms is highly dependent on the system being analyzed. So comparisons such as these made in this study can only serve as rough guidelines. In view of this we can still draw some conclusions:

Using a table-look-up method significantly improves efficiency, but it involves considerable additional programming and cannot be used with systems which have huge state spaces. The randomized discrete-time approach does not seem to improve efficiency except when the system has reached steady state, and it requires considerable extra programming. The deterministic randomization algorithm works well for systems that have a small to moderate number of states and are not too stiff and when a high precision is desired.

REFERENCES

[1] Çinlar, E., *Introduction to Stochastic Processes*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975, 402 pages.

[2] Heyman, D. P. and M. J. Sobel, *Stochastic Models in Operations Research, Volume 1*, McGraw-Hill, New York, 1982, 548 pages.

[3] Ross, S. M., *Introduction to Probability Models*, 2 Ed., Academic Press, New York, 1980, 376 pages.

[4] Gross, D. and D. R. Miller, "The randomization technique as a modeling tool and solution procedure for transient Markov processes," *Operations Research*, to appear, 1984.

[5] Gross, D. and D. R. Miller, "Multi-echelon repairable item provisioning in a time-varying environment using the randomization technique," *Naval Research Logistics Quarterly*, to appear, 1984.

[6] Melamed, B. and M. Yadin, "Randomization procedures in the computation of cumulative time distributions over discrete state Markov processes," Bell Laboratories, Holmdel, New Jersey, 1980.

[7] Miller, D. R., "Reliability calculation using randomization for Markovian fault-tolerant computing systems," *13th Annual International Symposium on Fault-tolerant Computing, Digest of Papers*, IEEE Computer Society, New York, 1983, 284-289.

[8] Grassmann, W., Personal communication, 1984.

[9] Grassmann, W., "Transient solutions in Markovian queueing systems," *Computers & Operations Research*, Vol. 4, 47-56, 1977.

[10] Nordijk, A., D. L. Iglehart, and R. Schassberger, "Discrete event methods for simulating continuous time Markov chains," *Advances in Applied Probability*, Vol. 8, 772-778, 1976.

[11] Fishman, G. S., *Principles of Discrete Event Simulation*, Wiley, New York, 1978, 514 pages.

[12] Grassmann, W. K., "Simulating transient solutions of Markovian systems," University of Saskatchewan, Saskatoon.

[13] Sherbrooke, C. C., "METRIC: A multi-echelon technique for recoverable item control," *Operations Research*, Vol. 16, 122-141, 1968.

[14] Muckstadt, J. A., "A model for a multi-item, multi-echelon, multi-indenture inventory system," *Management Science*, Vol. 20, 472-481, 1973.

[15] Hillestad, R. J. and M. J. Carrillo, "Models and techniques for recoverable item stockage when demand and the repair process are nonstationary--Part I: Performance measurement," N-1482-AF, Rand, Santa Monica, California, 1980.

[16] Hillestad, R. J., "Dyna-METRIC: Dynamic multi-echelon technique for recoverable item control," WD-911-AF, Rand, Santa Monica, California, 1981.

[17] Horn, S. A., "An empirical comparison of an analytical technique and a computer simulation," class project, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1983.

Decision Tree Models Using Monte-Carlo Analysis

RAYMOND P. O'CONNOR
BOEING COMPUTER SERVICES
(A DIVISION OF THE BOEING COMPANY)

Decision Tree Models Using Monte-Carlo Analysis

Abstract

A decision tree is a graphic representation of the process of evaluating the value of alternatives leading to an optimum solution. The process of decision analysis involves (1) graphic description, (2) assigning value and probability, (3) backward induction and (4) decision making or selection of the path through the tree. A computer is effectively used to solve the algorithms of Step 3, the backward induction. The problem of reliability, however, is not diminished by the high precision of computer calculations. The results can be no better than the best guess values assigned to the various probabilities and outcomes. Monte Carlo analysis provides a tool which can utilize the range between the best case and the worst case with best guess and variability to quantify the probability of achieving anticipated outcomes. The added dimension of probability, significantly improves the reliability of the decision tree analysis. A methodology is proposed for a computer model to solve decision tree models using Bonner & Moore's Monte Carlo analysis program, PAUS. A standard variable naming convention is offered which facilitates programming decision tree models and a sample PAUS program is presented as a guide for solving decision tree models using Monte Carlo analysis.

Overview

Probabilistic dynamic programming models sound significantly more important than decision tree analysis but substantially mean the same thing. People/managers make decisions all the time. In response to the morning alarm clock: (a) get up and go to work, (b) call in sick and go back to sleep, or (c) call in a vacation day and go to the beach. The logical manager makes business decisions based upon the financial value of the alternatives. A decision tree analysis is a technique of quantifying the values of various possible outcomes and the probability of achieving those outcomes.

When represented in mathematical notation, a decision tree model is awe inspiring. However; if represented in plain business language, a decision tree is a simple graphic representation of the decision process. The computer is a tool for solving mathematical relationships that have been defined by people. A manager reaches a decision in the alarm clock problem based upon many factors: work load at the office, other managers attendance, personal attitude, the availability of sick leave or vacation, and many other factors. To attempt to define all the factors which contribute to the value of each alternative would create an extremely complex model. People can make decisions involving many conflicting factors with relative ease while a computer program to solve the same problem would be excessively complex. Proper model design implies, let the superior capability of the human mind do the complex logical work and let the computer do the dull, boring repetitions work (after all its only a machine). The decision tree example which follows is large but each component is easy to understand.

There are a few simple rules which guide the logic of the decision tree solution. (1) A deterministic decision is represented by a square in the graphic exhibits. The logical manager will always choose the deterministic path which yields the greatest return. (2) A probabilistic decision is represented by a circle in the graphic exhibits. The value of a probabilistic decision point is the sum of each path's probability multiplied by its value. (3) The sum of the probabilities of each path from a probabilistic decision point must be 1.0.

The application of these three rules makes the solution of the decision tree a simple (but tedious) task.

Reliability

Since the arithmetic calculations involved in the backward induction are relatively simple, a computer can easily be programmed to compute the results. Unfortunately, the accuracy of computer arithmetic infers a degree of reliability which is not justified. The value of the answers can be no better than the reliability of the estimates of the costs, revenues and probabilities. Since the computer calculates the backward induction values, Monte Carlo analysis can be applied which results in an unlimited number of solutions. Using Monte Carlo techniques; cost, revenue and probabilities can be defined as statistical distributions as well as discrete points. The results increase reliability by adding the dimension of the probability of achieving the various decision point values.

Monte Carlo Analysis

Bonner and Moore have developed a Monte Carlo modeling tool called PAUS. The PAUS model allows the various values of the decision paths to be represented as any one of 22 different distribution types. The most common financial distribution, the BETA format requires the user to specify the minimum, maximum and most likely

END

12-86

DTIC