SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14850-7501

TECHNICAL REPORT NO. 810

June 1988

NUMERICAL METHODS FOR TRANSIENT
MARKOV CHAINS

by

Bennett L. Fox*

*On leave from the University of Montreal. Address as of August 1, 1988: Department of Mathematics, Campus Box 170, University of Colorado, 1200 Larimer Street, Denver, CO 80204.

This research was supported by the Natural Sciences and Engineering Research Council of Canada.
NUMERICAL METHODS FOR TRANSIENT MARKOV CHAINS

Bennett L. Fox*

June 1988

ABSTRACT The setting is finite-horizon, continuous-time Markov chains. We find and compare the computational complexities of numerical procedures, both deterministic and numerical, for four standard classes of problems.

1 INTRODUCTION

In the setting of continuous-time, finite-horizon Markov chains, we study and compare the computational complexities of several methods to estimate

(i) expected terminal reward
(ii) expected cumulative reward
(iii) hitting-time distribution
(iv) expected reward up to absorption.

Problems (i) and (ii) are for fixed horizons, while problems (iii) and (iv) refer to random horizons. Among the numerical methods we consider, most are well known but their computational complexities have apparently not been compared previously.

Among the deterministic methods, our major innovation is to decompose problems (i) and (ii) into steady-state and transients parts. The steady-state solution can be found by well-known methods. We compute the transient solution by solving a (possibly stiff) system of differential equations related to a matrix exponential. The eigenvalues of the corresponding matrix all have

*Département d’informatique et de recherche opérationnelle, Université de Montréal, Montréal, Québec H3C 3J7. On leave at: School of Operations Research and Industrial Engineering, Upson Hall, Cornell University, Ithaca, New York 14853-7501. Address as of August 1, 1988: Dept. of Mathematics, Campus Box 170, University of Colorado, 1200 Larimer Street, Denver, CO 80204.
strictly negative real parts. Hence the norm of the solution to the differential equations converges to zero \textit{exponentially fast}. This makes the computational complexity insensitive to the horizon length $T$. Without the decomposition, the computational complexity would be sensitive to $T$; the Reibman-Trivedi (1988) approach has this defect. In addition, we shift the spectrum leftwards by a small positive constant $d$ by subtracting $dI$ from the relevant matrix; this helps to get numerical stability. Global-error damping (including \textit{A-stability}) depends on all eigenvalues of the relevant matrix having strictly negative real parts and on restricting the order of the differential-equation solver. If the original matrix had eigenvalues on or just to the left of the imaginary axis, roundoff error could (in effect) push some to its right and thereby cause a numerical disaster. This error buildup would become even more critical with the forward time-stepping going all the way to $T$ (as it would have to without our decomposition above). Though our computational-complexity estimates assume infinite precision, we do not otherwise ignore roundoff. That is why we shift. Reibman and Trivedi (1988) do not shift.

The other numerical approaches, both deterministic and simulation, are based on uniformization. We take advantage of the Fox-Glynn (1988a) way to compute Poisson probabilities and to bound the truncation error.

Deterministic numerical methods, unlike simulation, require (sometimes hard) complete state enumeration and explicit storage of the state vector. Hence, they are feasible only if the state vector can fit into computer memory, possibly including (a reasonable amount of) auxiliary storage. In complex queueing networks, for example, this condition is often far from being satisfied. When it is not, simulation may still be feasible (without state aggregation beyond that implicit in the model) because its visits only a tiny fraction of the states. Using a future-event schedule does not require state enumeration. Fox (1987a, b) suggests work-savers for an alternative to a future-event schedule: generating transitions directly from a compact representation of the currently-scanned row that does not involve complete state enumeration. With either deterministic or simulation methods generating or regenerating rows often takes much more time than scanning them. The former must generate each row of the transition matrix at least once.

To make comparison of deterministic and simulation approaches non-trivial, we therefore assume that explicit storage of the relevant matrix is feasible. In that case, our major innovation for simulation is to compute the alias table for a row of the transition matrix the first time the corresponding state is visited. The work to set up an alias table is proportional to the number of positive elements of the row. Afterwards, each variate from that row gets produced in $O(1)$ marginal time. This reduces the work to execute a simulation run by an order of magnitude relative to what can happen using a future-event schedule (stored as a heap, say). If the number of runs required to satisfy a given root-mean-square-error tolerance is large enough, then the alias-table set-up is more than amortized by subsequent savings in run times.
All the approaches tied to uniformization convert to discrete-time. For simulation, Fox and Glynn (1988b) detail the advantages of such conversion. The corresponding summations for deterministic and simulation methods are truncated at the same place, leaving the same (very small) bias.

Our complexity criterion is the order of magnitude of the work required to satisfy a given root-mean-square-error tolerance. We assume that each scalar addition, multiplication, and comparison is done exactly in $O(1)$ time. On the other hand, we do not otherwise ignore roundoff error: only numerically-stable methods are considered. No matrix structure is assumed in our complexity estimates (but see section 8.4). We assume standard computer architecture, without parallelism or pipelining.

The steady-state counterpart to problem (ii) looks at

$$\text{expected cumulative reward} = \text{(gain rate)} \cdot T + \text{“bias”} + O(1/T).$$

Lamond (1985) surveys and extends matrix-decomposition methods for steady-state dynamic programming in this (asymptotic) setting. Our results could be used to check whether for a given policy and a given $T$ (or larger), the approximation above is good.


### 2 SUMMARY OF COMPUTATIONAL COMPLEXITIES

<table>
<thead>
<tr>
<th>PROBLEM</th>
<th>METHOD</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vector-Matrix Multiplication</td>
<td>Differential Equations</td>
<td>Simulation</td>
</tr>
<tr>
<td>Fixed horizon, terminal reward</td>
<td>$n^2\Lambda$</td>
<td>$n^3 + n^2(n</td>
<td></td>
</tr>
<tr>
<td>Fixed horizon, cumulative reward</td>
<td>$n^2\Lambda$</td>
<td>$n^3 + n^2(n</td>
<td></td>
</tr>
<tr>
<td>Hitting-time distribution</td>
<td>$(n^2 + m)\Lambda$</td>
<td>$n^3 + n^2(m + (n</td>
<td></td>
</tr>
</tbody>
</table>
\[
\begin{array}{|c|c|c|}
\hline
\text{Method} & \text{Matrix Inversion} & \text{Successive Approximations} & \text{Simulation} \\
\hline
\text{Cumulative reward until absorption} & n^3 & n^2 R & n^2 + R(\sigma/\delta)^2 \\
\hline
\end{array}
\]

\[n = \text{number of states}\]
\[\theta = \text{jump - rate parameter in uniformized chain}\]
\[\Lambda = \theta T \quad (T = \text{fixed horizon})\]
\[\delta = \text{root mean square error tolerance}\]
\[f(i) = \begin{cases} 
\text{reward in state } i, \text{ for terminal - reward problems} \\
\text{reward rate in state } i, \text{ for cumulative - reward problems}
\end{cases}\]
\[k = \text{order of differential - equation solver (see section 4.3)}\]
\[m = \text{size of grid (see section 3.3)}\]
\[R = \text{truncation point (see section 3.4)}\]
\[\sigma^2 = \text{output variance per run}\]

Here we use the convention that, if \(x(i)\) is the \(i\)-th component of a vector \(x\), then \(\|x\| = \max_i |x(i)|\). For future reference, if \(A\) is a matrix, then its norm
\[
\|A\| = \max_i \sum_j |A(i,j)|.
\]

Vector and matrices are in boldface.

All our complexity estimates are scale-free. Sections 3–7 justify them. The implicit proportionality factors are universal constants, except for our differential-equation approach to problems (i) and (ii). For those two problems, the implicit proportionality factors for the terms involving \(\delta\) are \(O(T^{1+1/k})\) in principle. However, section 4.2 shows that \(T\) rarely influences these factors (when \(T\) is large), provided that the cutoff strategy given there is used. Section 8 draws conclusions.

Fox[(1987c, d), (1988)] gives counterparts of the table for gradient estimation.

3 PROBLEM SPECIFICATIONS

With deterministic approaches based uniformization, the heart of the computation is computing
\[
g(k) = \nu C^k f
\]
for $k = L, L + 1, \ldots, R$ say, where $\nu$ is a probability row vector (for the initial state) and $C$ is the transition matrix in the uniformized chain. Approaches related to matrix exponentials use the generator $G$ of the nonuniformized chain. Recall that $G = \theta C - \theta I$, so the spectrum of $C$ is that of $G$ shifted and rescaled to the unit circle. Thus, $TG = \Lambda C - \Lambda I$ and $\exp(TG) = \exp(\Lambda C) \cdot \exp(-\Lambda I)$. The connection to $g(k)$ weighted by $P(k)$, the probability that a Poisson variate with parameter $\Lambda$ equals $k$, is now apparent from the (matrix) exponential power series. Thus, apart from numerical stability, uniformization and the exponential power series are virtually equivalent for the terminal-reward problem but not for the cumulative-reward problem, as we will see. The exponential power series has mixed signs (making it vulnerable to numerical instability), but we can arrange that uniformization works solely with numbers of one sign.

When $A$ is large, the normal approximation to the Poisson makes it plausible that, given a positive tolerance $\epsilon$, we can find $L, R, W, w(L), \ldots, w(R)$ such that

$$|A - B| \leq \epsilon \|f\|$$

(2)

where

$$A = \sum_{k=0}^{\infty} g(k)P(k)$$

(3)

$$B = (1/W) \sum_{k=L}^{R} g(k)w(k)$$

(4)

$$W = w(L) + \cdots + w(R)$$

(5)

$$P(k) = w(k)/W$$

(6)

$$R - L = O(\Lambda^{1/2})$$

(7)

$$L = O(\Lambda).$$

(8)

Fox and Glynn (1988a) show how to find $L, R$, and the weights $w(k)$, without underflow or overflow, with

$$\epsilon = 10^{-10}, \ R - L \leq \max(600, [20\sqrt{\Lambda}]).$$

(9)

Our complexity estimates assume that (9) is consistent with the overall error tolerance $\delta$. The Fox-Glynn approach can't handle arbitrarily small $\epsilon$ (because, for large enough truncation points, the mass in the right Poisson tail is an order of magnitude greater than the mass in the corresponding normal tail). This is not a practical constraint, though it slightly abuses some big-O expressions involving $\delta$.

Via (2)–(9), uniformization is the springboard for efficient deterministic approaches (in section 4.1) and for efficient simulation approaches (in section 5).

De Souza e Silva and Gail (1986) use uniformization to compute the distribution of time during which a system is operational during $[0, T]$. They also
consider the corresponding expectation. The latter corresponds to a special case of our problem (ii) by setting \( f \) to one on good states and to zero elsewhere. Their reliability problem corresponds to our problem (iii). Their mean-time-to-failure problem corresponds to our problem (iv) by setting \( f \) to one on transient states.

Let the \( j \)-th component of

\[
y(t) = \nu[\exp(tG)]
\]

be the probability that the state at time \( t \) is \( j \). So \( y \) solves

\[
\frac{\partial y}{\partial t} = yG, \quad y(0) = \nu
\]

and that is the basis of our approach based on solving differential equations.

### 3.1 Terminal reward

The expected terminal reward is

\[
A = \sum_{k=0}^{\infty} g(k)P(k)
\]

\[
= y(T)f.
\]

### 3.2 Cumulative reward, fixed horizon

Gross and Miller (1984) and Fox and Glynn (1988) show, by different methods, that the expected cumulative reward up to \( T \) is

\[
\tilde{A} = \sum_{k=0}^{\infty} \tilde{g}(k)P(k)
\]

where

\[
\tilde{g}(k) = T\sum_{j=0}^{k} g(j)/(k + 1).
\]

Either from (15) or (more easily) directly, we get

\[
\tilde{A} = \int_0^T y(t)f \, dt.
\]

We compute \( \tilde{g}(k) \) recursively as \( \tilde{g}(k) = [k\tilde{g}(k - 1) + Tg(k)]/(k + 1) \). A counterpart of (2) holds for \( \tilde{A} \).
3.3 Hitting time

We estimate the cumulative distribution $H$ of time to hit a set $M$. Given that we hit $M$ on the $j$-th (discrete-time) transition, the (continuous-time cumulative) distribution $H(t; j, \theta)$ is Erlang with shape parameter $j$ and scale parameter $\theta$. Collapse $M$ to state 1 say. In (1), let $f(1) = 1$ and $f(i) = 0$ for $i \neq 1$. The probability of hitting $M$ on jump $k$ is the difference of two cumulative probabilities:

$$g^*(k) = g(k) - g(k - 1)$$

(17)

where $g(-1) = 0$ by definition.

Clearly

$$H(t) = \sum_{k=0}^{\infty} g^*(k)H(t; k, \theta)$$

(18)

and Neuts [(1981), p. 45] shows that

$$H(t) = 1 - y^*(t)e$$

(19)

where $e = (1, 1, \ldots, 1)'$ and

$$y^*(t) = \nu^* \left[ \exp(tG^*) \right]$$

(20)

with asterisks denoting restrictions to transient states. All states in the complement of $M$ are assumed transient.

We want to compute $H(t)$ over a grid $[0, t_1, t_2, \ldots, t_m, T]$ and then interpolate as needed. Via (20), this is straightforward if $y^*$ is found numerically by solving the corresponding differential equation over that grid.

With (18), we use counterparts of (2)–(9) with $L = 0$ and, for $k = 0, \ldots, R_T$ (specified below), compute $H(t; k, \theta)$ over the grid — say by numerical integration. Computing $[g^*(0), \ldots, g^*(R_T)]$ generally dominates the work. It is easily checked that an upper bound on truncation error is $H(T; R_T + 1, \theta) \cdot [1 - g(R_T)]$. In practice, we would increase $R_T$ until an error tolerance is satisfied. To estimate computational complexity a priori, we estimate the order of magnitude of an upper bound on $R_T$ using the normal approximation to the $k$-Erlang with scale parameter $\theta$ which has mean $k/\theta$ and variance $k/\theta^2$. Our estimate ignores the factor $1 - g(R_T)$, which is less than one and decreasing in $R_T$. Taking $R_T = \theta[T + O(1)]$ makes the truncation error negligible for large $\theta$. We are centering the normal slightly to the right of $T$ and its standard deviation goes to zero as $\theta$ decreases.

3.4 Cumulative reward, random horizon

It is well known and easily proved (e.g., see Fox and Glynn (1988b)) that the expected cumulative reward $S$ until hitting $M$ satisfies the linear systems

$$S = (1/\theta) \nu^*(I - C^*)^{-1} \mathbf{r}^*$$

(21)

$$S = \nu^*(I - D^*)^{-1} \mathbf{r}^*$$

(22)
where asterisks denote restrictions to transient states, $D$ is the transition matrix of the nonuniformized chain, and $\hat{f}(i)$ is the product of $f(i)$ and the mean holding time in state $i$. The systems (21) and (22) each can be solved in $O(n^3)$ time by direct elimination.

They can also be solved iteratively. The mappings

$$Kz = C^*z + f^*$$

and

$$Lz = D^*z + \hat{f}^*$$

are $n$-stage contractions: $\|K^n x - K^n z\| \leq c_1 \|x - z\|$ and $\|L^n x - L^n z\| \leq c_2 \|x - z\|$ with $0 \leq c_2 \leq c_1 < 1$. Therefore, iterating $L$ to approximate its unique fixed point (say $z^*$) converges faster than iterating $K$. Clearly

$$S' = \nu^*z^*.$$  

All this is well known.

In general, $c_1$ is as hard to compute as $z^*$. To get an interesting complexity bound, we now assume more structure: $M$ can be reached it one step from any state. This assumption may hold, for example, if $M$ denotes system failure and the system can fail from any state in one step. Under this assumption, $c_1 = \|D^*\|$ and is easily computed in $O(n^2)$ time. If we substitute $L^*0$ for $z^*$ in (25), the error in the corresponding estimate of $S$ is bounded by $\|D^*\|^{k+1}\|f^*\|/(1 - \|D^*\|)$. This bound takes $O(n^2)$ time to compute. Setting it equal to an error tolerance quickly determines $R$. Next $S$ is computed in $O(Rn^2)$ time by iteration or in $O(n^3)$ time directly, whichever is smaller (up to proportionality factors). To this point, the remarks in this section have been standard numerical analysis.

Now notice, however, that stopping simulation runs after $R$ transitions or hitting $M$ – whichever happens first – gives the same bias as for the deterministic iterative run, provided that the reward received up to that stopping time is counted as the output of the run and that successive approximations are initiated with the 0 vector. More importantly, this stopping rule reduces the (generally infinite) worst-case complexity to a finite worst-case complexity which we bound in section 5. We find the link, $R$, between the deterministic iterative method and simulation surprisingly close. Observing that link is another innovation of this paper.

4 DETERMINISTIC NUMERICAL APPROACHES

In section 4.1 we discuss the computation of $[g(L), \ldots, g(R)]$. Together with section 3, this shows how to use uniformization to compute the solution to problems (i)–(iii). Section 3.4 showed how to handle problem (iv); its discussion of appropriate deterministic numerical methods suffices for our purposes.
Section 4.2 gives our decomposition into steady-state and transient parts. Its implications for our differential-equation approach to problems (i)–(iii) are given in section 4.3.

4.1 The naive method and binary powering

The naive way to compute \(\{g(L), \ldots, g(R)\}\) is

\[

g(0) \leftarrow \nu f \\
k \leftarrow 1 \\
\text{WHILE } k \leq R \\
\quad \text{SET } \nu \leftarrow \nu C \\
\quad g(k) \leftarrow \nu f \\
\quad k \leftarrow k + 1
\]

which takes \(R \cdot O(n^2)\) time for unstructured matrices. The vector-matrix multiplications of the form \(\nu C\) dominate the work to execute this algorithm.

Another way, pointed out independently by Reibman and Trivedi (1988), begins with the binary representation of \(L = \beta_m \cdots \beta_k \cdots \beta_0\) say, with \(\beta_i = 0\) or 1. Now compute

\[
F \leftarrow 1 \\
k \leftarrow 0 \\
\text{WHILE } k \leq m \\
\quad \text{IF } \beta_k = 1, \text{ THEN} \\
\quad \quad \text{SET } F \leftarrow FC \\
\quad \quad \text{ENDIF} \\
\quad C \leftarrow C^2 \\
\quad k \leftarrow k + 1
\]

which takes \(O(\mu(n) \cdot \log L)\) time to produce (the final) \(F = C^L\), where \(\mu(n)\) is the time to multiply two square matrices each with \(n\) rows. Continue with the naive scheme, above to get \(\{g(L), \ldots, g(R)\}\). The overall time is \(O(\mu(n) \cdot \log L + (R - L)n^2)\). This can be competitive with our differential-equation approach only if \(\mu(n) = o(n^3)\).

The standard matrix-multiplication method takes order \(n^3\) work, so binary powering is not competitive when combined with it. Several matrix-multiplication methods have been proposed recently that have \(\mu(n) = o(n^3)\); see, for example, Coppersmith and Winograd (1987). To date, all such methods have huge implicit constants and their numerical stability is questionable. Therefore, our complexity table in section 2 does not reflect binary powering; future developments in matrix-multiplication methods could require updating it.

Binary powering does beat the naive scheme if \(n = o(R/\log R)\) even when \(\mu(n) = O(n^3)\). By (7) and (8), we get \(R = O(\Lambda)\). Knuth [(1981), pages 441–
surveys alternatives to binary powering; while their complexities are still \( O(\mu(n) \cdot \log L) \), sometimes the implicit proportionality factor is less.

4.2 Decomposition into steady-state and transient parts

To assure that a steady state exists (independent of the initial state), we assume that the chain has just one recurrence class. Let \( \pi \) be the steady-state (row) vector corresponding to \( G \) and set

\[
\Pi = \begin{pmatrix}
\pi \\
\pi \\
\vdots \\
\pi
\end{pmatrix}
\]

(26)

\[ W = G - \Pi. \]

(27)

**Theorem.** The eigenvalues of \( W \) all have strictly negative real parts and

\[
y(t) = \pi f - \pi f e^{-t} - \nu [\exp(Wt)] f.
\]

(28)

This theorem, proved in section 7 and of interest in its own right, decomposes the computation into steady-state (\( \pi f \)) and transient parts. To use it, we must solve \( \pi G = 0 \) for \( \pi \). For unstructured matrices, this takes \( O(n^3) \) time using direct elimination. The method of Grassmann, Taksar, and Heyman (1985) may be appropriate; Heyman (1987) studies this method further. If iterative methods are used to compute \( \pi \), then the computational complexity depends on the error tolerance \( \delta \) and on the convergence rate; under certain conditions, Mitra and Tsoucas (1987) give the latter. See also Lubachevsky and Mitra (1986); Jackson networks (e.g., see Heyman and Sobel ([1982], chapter 12)) are prominent sources of models where \( \pi \) is often easily computed.

From the theorem, the transient part of \( y(t) \), namely

\[
x(t) = \pi f e^{-t} + \nu [\exp(W(t)) f],
\]

(29)

converges in norm to zero exponentially fast—though possibly not monotonely. To help get numerical stability when solving the differential-equation system

\[
\frac{\partial z(t)}{\partial t} = zW
\]

(30)

\[ z(0) = \nu \]

(31)

for

\[ z(t) = \nu \cdot \exp(Wt), \]

(32)

we choose a small positive constant \( d \) and set

\[ \tilde{W} = W - dI \]

(33)
\[ \tilde{z} = \nu \cdot \exp(\tilde{W}t). \]

The eigenvalues of \( \tilde{W} \) are all at least \( d \) leftwards of the imaginary axis. We solve the system analogous to (30) and (31); then
\[ z(t) = \tilde{z}(t)e^{dt}. \]

We stop the forward time-stepping when we get to \( T \) or when \( z(t) \leq 10^{-10}\|f\| \), say, whichever comes first. When \( T \) is large, almost always the cutoff occurs far to its left. Since \( \|z(t)\| \) converges exponentially fast to zero, we believe that this is an excellent heuristic; it is not foolproof, because of possible non-monotone convergence of \( \|z(t)\| \). The spectrum shift in (33) makes the (recursive) computation of \( \tilde{z} \) numerically stable. Multiplying by \( e^{dt} \) in (35) to get \( z \) does not cause numerical instability, because \( z \) is not computed recursively (taking \( \tilde{z} \) as given).

In view of (28), we can rewrite the expected cumulative reward as
\[ \tilde{A} = \pi f[T - 1 + e^{-T}] - \int_0^T z(t)fdt, \]
where we took the expectation inside the integral. We compute the integral in (36) numerically over the same grid used by the differential-equation solver. To simplify the complexity estimates, we assume that the order of the quadrature is at least as high as that of the quadrature routine. If the set \( S = \{t \leq T: \|z(t)\| \leq 10^{-10}\|f\| \cdot (T - t)\} \) is empty, we take the cutoff point \( \tau = T \); otherwise (the usual case), we set \( \tau = \min\{t \in S\} \). For the same reasons as before, we think that this is an excellent heuristic.

For any given problem instance and \( T \) large enough, \( T \) does not affect \( \tau \). However, it appears that \( \tau \) is not universally negligible relative to \( T \), over all conceivable problem instances. This affects only the second term in the complexity estimates for our differential equation approach. In our table in section 2, there is no \( T \) in that term – with the understanding that in pathological instances, there is an implicit proportionality factor of order \( T^{1+1/k} \). We believe that this is the least misleading way to present the table.

### 4.3 Solving stiff differential equations

Integrating the differential equations defining \( y \) is one of the few methods that Moler and Van Loan (1978) do not condemn. They are skeptical about most matrix-theoretic methods. Solving the differential equations is especially convenient for the cumulative-reward problem, where we need \( y \) over a grid and not just at \( T \). Enright [(1978a, b), (1979)] tailors general algorithms for solving stiff systems of differential equations to those which are linear with constant coefficients; this is the matrix-exponential case and covers the system (11). A one-time matrix decomposition takes \( O(n^3) \) time. It allows solving the relevant systems of linear equations with \( O(n^3) \) work per time step. This leads to

11
a straightforward complexity analysis, as we show below. For sparse systems, iterative methods are probably better to solve (approximately) the related systems at successive time steps, starting at step \( k \) roughly where they quit at step \( k - 1 \); complexity analyses of such methods are beyond the scope of this paper.

Once one uses an (implicit) "A-stable" tailored for stiff systems, numerical instability can arise only if there are eigenvalues on or near the imaginary axis (regardless of how far leftwards of that axis some eigenvalues may be). Our spectrum shift of section 4.2 eliminates this problem. The function \( \tilde{z}(t) \) can be represented as a linear combination of (time-independent) vectors with weights of the form \( \exp(\rho_i t) \). Each \( \rho_i \) is an eigenvalue of \( \tilde{W} \) and hence its real part is strictly less than \(-d\). Therefore, computing \( z \) in (35) is numerically stable. The solution \( z \) can be significantly influenced, even for large time, by more than the eigenvalue(s) closest to the imaginary axis; see Aldous (1988) for an example where a (simplistic) use of "relaxation" time, depending only on the eigenvalue closest to the imaginary axis, misleads.

Many practical algorithm are variable order and variable step, the two being chosen adaptively by an error-control monitor. In complexity estimates, we (pessimistically) assume that the lowest order (say \( k \)) usable by the algorithm is always used, with a corresponding fixed step size — say \( h \). When accuracy to a few digits suffices, this is not critical.

Strang [(1986), pages 574-576] shows that the global error at time \( t \) is \( O(th^k) \). We want the absolute error in the inner product \( y(t)f \) less than \( \delta \). This holds if \( n||f||h^{k} \tau \leq \delta \), with cutoff \( \tau \). So we set \( h = (\delta/n||f||\tau)^{1/k} \). The number of time steps is \( \tau/h \). Since the work per step is \( O(n^2) \), we get the estimate in the table — with the understanding that \( \tau^{1+1/k} \) is regarded as a proportionality factor rather than appearing explicitly.

For general background on stiff ODE solvers, see for example Aitken (1985), Byrne and Hindmarsh (1987), Enright, Hall, and Lindberg (1975), Johnson and Riess (1982), and Shampine and Hair (1979). Charles Van Loan (personal communication) pointed out that, with ODE approaches, we can easily let \( f \) be time dependent. However, letting \( G \) be time dependent would significantly increase complexity.

5 SIMULATION

In equation (1) we see that \( g(k) \) is the expected reward or reward rate in the \( k \)-th state visited. To get a simulation counterpart, redefine \( g(k) \) as the actual (random) reward or reward rate in the \( k \)-th state visited. Formulas (12), (14), and (18) respectively now produce unbiased estimates of the expected terminal reward, expected cumulative reward, and cdf at \( t \) of the hitting-time distribution. The respective truncated versions of these formulas produce estimators with the same (small) bias as the deterministic version. If we expand \((I-C^*)^{-1}\) in (21) or \((I-D^*)^{-1}\) in (22) in a power series and each summand substitute

\[ 12 \]
the actual reward received during the k-th visit for its expectation, we get an unbiased estimator. This is a "backwards" approach to discrete-time conversion in simulation settings. All these estimators can be found directly by computing conditional expectations given the sequence of states visited.

Fox and Glynn (1988b) use discrete-time conversion as a springboard for further efficiency-increase techniques for problems (i), (ii), and (iv), that reduce the (asymptotic) variance constant (= expected work per run • variance per run), sometimes by an order of magnitude. Ross and Schechner (1985) propose efficiency – increase techniques for problem (iii). Our complexity estimates handicap simulation by not considering these techniques.

Since we neglect bias here, we neglect the difference between asymptotic variance and mean square error. Even though the run lengths and sample variance of the outputs are generally dependent, asymptotic variance

\[ \omega^{-1} \cdot (\text{expected work per run}) \cdot (\text{output variance per run}) \]  

(37)

where \( \omega \) is the work to execute the completed runs; see Glynn and Whitt (1986) or Fox and Glynn (1988a). We want the root-mean-square error less than a tolerance \( \delta \). To get via (37) an upper bound, up to proportionality factors, on the work required is easy:

\[
\text{overall work} \approx \frac{[\text{expected work per run} \cdot \text{variance per run}]}{\delta^2} + \text{setup}
\]

\[
\leq \frac{[\text{maximum work per run} \cdot \text{variance per run}]}{\delta^2} + \text{setup}.
\]  

(38)

The setup is of order \( n^2 \) (to compute \( n \) alias tables) plus the work to compute the weights or, for problem (iv), the iteration bound \( R \). For problems (i) and (ii), the work to compute the weights is \( O(\Lambda^{1/2}) \) from (7). For problem (iii), the discussion near the end of section 3.3, shows that \( R_T = O(\Lambda) \). Since \( H(t; k, \theta) \) has to be computed for \( O(mn) \) pairs \( (t_i, k) \) and the incremental work to compute \( H(t_{i+1}; k, \theta) \) given \( H(t_i; k, \theta) \) is \( O(1) \), we account for the \( m\Lambda \) term in the table. For problem (iv), computing \( R \) takes \( O(n^2) \) work.

For problems (i)–(iii), there are \( O(\Lambda) \) transitions per run. Each takes \( O(1) \) marginal work to generate. Notice that the \( O(n^2) \) work to set up alias tables does not hurt simulation relative to deterministic methods; the latter require work of at least order \( n^2 \). On the other hand, if transitions were generated via a future-event schedule say, the term involving \( (\sigma/\delta)^2 \) would be blown up by a potentially large factor.

6 EMPIRICAL COMPARISONS

Gross, Miller, and Plastiras (1984), while aware of matrix-exponential approaches, consider in detail only vector-matrix multiplication and several implementations of simulation. They give no estimates of computational complexity but conclude from empirical work that
(i) vector-matrix multiplication "works well for systems that have a small to moderate number of states and are not too stiff and when a high precision is required", where systems with large jump-rate spreads are usually stiff.

(ii) implementing simulation by generating transitions via a table lookup beats using a future event schedule when table setup costs are not prohibitive.

(iii) implementing simulation via uniformization and discrete-time conversion "does not seem to improve efficiency except when the system has reached steady state" despite substantial variance reduction.

Their conclusions (i) and (ii) are consistent with our complexity estimates. Given their conclusion (ii), their conclusion (iii) can be due only to the setup work to compute Poisson probabilities not being amortized by only a small number of runs. After setup, the remaining work required is $O(\Delta/\delta^2)$. For small to moderate $\delta$ this dominates the $O(\Lambda^{1/2})$ setup to compute Poisson probabilities. Given this setup, null-jump sequences (from a state to itself) coming from uniformization do not decrease efficiency provided that they are each generated in constant time via geometric variates and not explicitly.

7 PROOF OF THE THEOREM

The theorem follows easily from lemmas 1–5 below. All the lemmas were pointed out to the author by Peter Glynn.

Lemma 1.

$$\Pi^2 = \Pi$$

(39)

$$\Pi G = G\Pi$$

(40)

$$= 0.$$

Proof. The components of $\pi$ sum to one, and the rows of $G$ sum to zero. □

Lemma 2.

$$e^{\Pi t} = \Pi e^t + I - \Pi.$$  (41)

Proof.

$$e^{\Pi t} = I + \Pi t + \Pi^2 t^2/2 + \cdots$$

$$= I + \Pi t + \Pi^2 t^2/2 + \cdots$$

$$= I + \Pi e^t - \Pi. □$$

14
Lemma 3.
\[ e^{(G - \Pi)t} = e^{Gt} + \Pi e^{-t} - \Pi. \quad (42) \]

Proof. Use \( e^{(G - \Pi)t} = e^{Gt} e^{-\Pi t} \) and lemma 2, where (40) justifies factoring the matrix exponential. \( \square \)

Lemma 4. As \( t \to \infty \),
\[ e^{(G - \Pi)t} \to 0. \quad (43) \]

Proof. Use \( e^{Gt} \to \Pi \) and lemma 3. \( \square \)

Lemma 5. All eigenvalues of \( G - \Pi \) have negative real parts.

Proof. Let \( \alpha \) be an eigenvalue of \( G - \Pi \) with corresponding eigenvector \( x \). Then \( (G - \Pi)^k t^k = \alpha^k t^k x \) for \( k = 0, 1, 2, \ldots \) and all \( t \). Hence \( \exp(\alpha t) \) is an eigenvalue of \( \exp((G - \Pi)t) \). So
\[
|\exp(\alpha t)| \cdot \|x\| = \|\exp(\alpha t)x\| \\
= \|(\exp((G - \Pi)t)x)\| \\
\leq \|\exp((G - \Pi)t)\| \cdot \|x\|. 
\]

But, by lemma 4, there is a \( \tau \) such that \( \|\exp((G - \Pi)t)\| < 1 \). Hence the real part of \( \alpha \) is negative. \( \square \)

8 DISCUSSION

For problems (i)–(iii), the table in section 2 shows that simulation wins – provided that \( \sigma/\delta \ll n \). We discuss this proviso shortly. For problem (iv), we compute the parameter \( R \) in \( O(n^2) \) time – when the target set can be reached in one step from any state; section 3.4 shows how. This preliminary computation does not affect the complexity of any of the three methods for problem (iv) considered in the table of section 2. Given \( R \), we choose the method from that table with the smallest complexity.

8.1 The error tolerance \( \delta \)

Given approximations in the model and inaccuracies in its input-parameter estimates, a tiny error tolerance \( \delta \) wastes work, especially for simulation, where the number of runs required is proportional to \( \delta^{-2} \). It misleads about the resulting precision in model solution relative to its accuracy for the real system. It misleads about the competitiveness of simulation relative to deterministic numerical approaches.
8.2 The output variance per run $\sigma^2$

If $\sigma$ is huge relative to the relevant mean, the problem is not well posed: knowing only the mean would have little practical value. Pilot runs may give ballpark estimates of $\sigma$, enough to check whether the problem is well posed and, if so, whether simulation is competitive. By themselves $n$ and $\theta$ reveal nothing about $\sigma$ – contrary to some folklore.

8.3 The ratio $\sigma/\delta$

When $\delta$ is defined relative to the relevant mean (as we assume), then $\sigma/\delta$ is scale-invariant. Suppose we take $\delta = 10^a \cdot \hskip 1.5pt \hbox{relevant mean}$ and $\sigma = 10^b \cdot \hskip 1.5pt \hbox{relevant mean}$. Then $(\sigma/\delta)^2 = 10^{2(b-a)}$, without having to estimate the relevant mean a priori. In view of section 8.1, it is reasonable to take $a \geq -3$ say. In view of section 8.2, it is reasonable to suppose that, if we don't abort the study because ballpark estimates of $\sigma$ indicate that the problem is ill posed, then $\beta \leq 1$ say. Even without variance reduction techniques, this gives $(\sigma/\delta)^2 \leq 10^8$.

We saw that simulation wins for problem (i)–(iii) when $\sigma/\delta \ll n$. If one accepts our guidelines above, this says that simulation wins when $n \gg 10^4$. From the table in section 2, it generally loses when $n \ll \sigma/\delta$, i.e., when $n \ll 10^4$. If a variance reduction technique reduces $\sigma$ by a factor of $\rho$, without significantly increasing expected work per run, then the critical threshold for $n$ becomes $10^4\rho^{-1}$. Efficiency-increase techniques in Fox and Glynn (1988b), for example, easily have the potential to get $\rho$ in the tens or hundreds.

8.4 Sparseness

No matrix structure is assumed in our complexity estimates. Other than for vector-matrix multiplication, it seems hard to sharpen our analysis to account for such structure, except perhaps for certain particular cases. Suppose that the number of positive elements of the transition matrix is $O(n)$. A crude rule of thumb subtracts one from each exponent of $n$ greater than one to adjust our complexity estimates for sparseness. Doing that increases the threshold for $n$ in section 8.3 from $\sigma/\delta$ to $(\sigma/\delta)^2$, i.e., to $10^8\rho^{-1}$. Though $W$ in (27) is dense, it suffices to store $G$ and the vector $\pi$.

8.5 The number $n$ of states

Simulation loses for all toy models, i.e., those with $n$ in the hundreds or less. At the other extreme, it is the only feasible method for $n$ in the millions or more (as can easily occur with large queueing networks). This holds no matter how sparse the transition matrix, because only simulation does not require space at least proportional to $n$ (when rows are generated only as needed). In between, our table in section 2 offers guidance. Since it ignores proportionality factors, it
discriminates in practice only when the ratio of two entries in a row is outside the interval (1/10, 10) say. If their ratio falls inside that interval, the choice between them is often immaterial. Doubling the computer time used to satisfy a given error tolerance may well add negligibly to the overall cost of a study that includes model building, data collection, and computer programming. A hundredfold increase in computer time, however, can be intolerable. Comparison of computational complexities guards against disastrously — wrong choices. Extrapolating from numerical experiments with other problems is speculative; disastrously-bad inferences may be drawn.

Acknowledgment Discussions with Peter Glynn were invaluable. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

References


