An Efficient Calculation Algorithm in Continuous-Time Markov Analysis Using Large-Scale Numerical Calculation

Nobuko Kosugi and Koichi Suyama

Tokyo University of Marine Science and Technology
2-1-6 Etchujima, Koto-ku, Tokyo 135-8533, Japan
{kosugi, suyama}@kaiyodai.ac.jp

Abstract

We propose a new calculation algorithm in continuous-time Markov analysis on the precondition of using large-scale numerical calculation on a computer. Because it is more efficient than the ordinary calculation in absorbing Markov analysis in memory usage and computation time, it can be applied to Markov models with enormous numbers of states especially in the fields of reliability, dependability, and safety.

Mathematics Subject Classification: 68M15, 60J27, 60J28

Keywords: continuous-time Markov analysis, absorbing state, numerical calculation

1 Introduction

The explosive development of computer technology makes large-scale numerical calculation possible, while such calculation was almost impossible a few decades ago. The objective of this paper is to improve calculation algorithm in continuous-time Markov analysis on the precondition of using large-scale numerical calculation on a computer.

Continuous-time finite Markov chains [2] have been widely used for system models in the fields of reliability, dependability, and safety, due to the power of expression. Hence, continuous-time Markov analysis has come to be regarded as one of basic and important analysis methodologies [3, 6, 8].

Recently its application to fault trees [4, 6] receives attention [1]. However, especially in the case of dynamic fault trees with dynamic gates such as
priority AND gates [5, 7, 9], the state space has enormous numbers of states [1]. It is a fairly common case where the state space includes several hundreds of thousands of states. Hence, some special device in numerical calculation algorithm is required for such Markov analysis.

In this paper, we propose a new calculation algorithm in continuous-time Markov analysis on the precondition of using large-scale numerical calculation on a computer. Because it is more efficient than the ordinary calculation in absorbing Markov analysis in memory usage and computation time, it can be applied to Markov models with enormous numbers of states. It has a significant meaning of establishing the potential of continuous-time Markov analysis especially in the fields of reliability, dependability, and safety.

The following notations are used in this paper. $\mathbb{R}$: the field of real numbers. $\mathbb{R}^{l \times m}$: the ring of $l \times m$ matrices with elements in $\mathbb{R}$. $E$: an identity matrix of appropriate dimensions. $0$: a zero matrix of appropriate dimensions. $\mathbf{t}X$: the transpose of a matrix $X$.

2 Problem statement

2.1 Calculation inputs

The following (I1) and (I2) are given as calculation inputs.

(I1) Differential equation: Consider a continuous-time Markov model with the state space consisting of $n (< \infty)$ states, States 1, \ldots, $n$. Let $p_i(t)$ denote the probability that the system described by the Markov model is in State $i$ at the time $t$, $1 \leq i \leq n$. Define the probability vector

$$ p(t) = [p_1(t) \ \cdots \ p_n(t)]. \quad (1) $$

Then, the Markov model can be described by the differential equation

$$ \frac{dp(t)}{dt} = p(t)A, \quad (2) $$

which is given to us as a calculation input. The $A (= [a_{ij}]) \in \mathbb{R}^{n \times n}$ is the transition rates matrix, where $a_{ij}$ ($i \neq j$) denotes the transition rate from State $i$ to State $j$, and

$$ a_{ii} = - \sum_{j=1, j\neq i}^{n} a_{ij}, \quad 1 \leq i \leq n \quad (3) $$

(see [6]). Hence, it follows that

$$ \sum_{j=1}^{n} a_{ij} = 0, \quad 1 \leq i \leq n. \quad (4) $$
Without loss of generality, suppose that the system is in State 1 at the initial time $t = 0$, hence the initial condition is

$$p(0) = [1 \ 0 \ \cdots \ 0].$$  \hspace{1cm} (5)

(I2) Set of state numbers of all target states: Suppose that there are $m$ ($< n$) target states in the state space, States $k_1, \ldots, k_m$. Define

$$K = \{k_1, \ldots, k_m\}.$$  \hspace{1cm} (6)

The initial state, State 1, is not a target state, hence, without loss of generality, assume $(1 <) k_1 < \cdots < k_m$ ($\leq n$).

A target state is not always an absorbing state in the transition rates matrix $A$. We assume the following.

**Assumption 2.1** If State $i$ is an absorbing state, i.e., $a_{ij} = 0$ ($1 \leq j \leq n$) in the transition rates matrix $A$, it is also a target state.

Then, the set of all target states includes all absorbing states.

### 2.2 Calculation results

The following (R1) and (R2) are calculation outputs to be obtained.

(R1) MTTFV (Mean Time To First Visit): We define MTTFV by the mean time from the system start at the initial time $t = 0$ to a first transition into any one target state.

In continuous-time Markov analysis in the fields of reliability, dependability, and safety, a first transition into any one target state corresponds to an occurrence of a serious event such as an overall system fault and a harmful incident. If we have the mean time from such an event occurrence to an overall system restoration, we can have the frequency of such serious events.

(R2) Possession rates\(^2\) of target states: We define the possession rate $\beta_{kj}$ of State $k_j$ ($1 \leq j \leq m$) by the probability that a first visit to a target state is a transition into State $k_j$. If the set of all target states is equal to the set of all absorbing states, the possession rate $\beta_{kj}$ is equivalent to the absorption probability $p_{kj}(\infty)$.

The possession rate $\beta_{kj}$ implies the importance of State $k_j$, i.e., the corresponding scenario leading to such a serious event. The information helps take countermeasures against such serious events.

---

\(^1\)From a practical viewpoint, we cannot assume that $m$ target states are States $n - m + 1, \ldots, n$. It takes an additional and enormous computation time to rearrange the transition rates matrix $A$ on the memory in such a manner.

\(^2\)Because a target state is not always an absorbing state, we cannot use the terms “absorption probability” and “limiting distribution”[2].
3 Ordinary calculation

Ordinarily, by regarding all target states as absorbing states, we reduce the necessary calculation to absorbing continuous-time Markov analysis.

At first obtain a new transition rates matrix $A' (=[a_{ij}']) \in \mathbb{R}^{n \times n}$ from the original one $A$ by replacing its $k_1, \ldots, k_m$-th rows by $[0 \cdots 0] \in \mathbb{R}^{1 \times n}$. That is, for State $k_j (1 \leq j \leq m)$,

$$a'_{kj,l} = 0, \quad 1 \leq l \leq n. \quad (7)$$

Then, under Assumption 1, the set of all absorbing states is equal to the set of all target states in the absorbing continuous-time Markov process described by the following differential equation with the transition rates matrix $A'$:

$$\frac{dp(t)}{dt} = p(t)A'. \quad (8)$$

There are $(n - m)$ non-target states in the state space. For simplicity, define the following set of their state numbers:

$$L = \{\ell_1 (= 1), \ell_2, \ldots, \ell_{n-m}\} = \{1, \ldots, n\} \setminus K. \quad (9)$$

Without loss of generality, assume that $(1 =) \ell_1 < \ell_2 < \cdots < \ell_{n-m} (\leq n)$.

Integrating both sides of the differential equation (8) on $t \in [0, \infty)$ gives

$$p(\infty) - p(0) = qA', \quad (10)$$

where

$$q = [q_1 \cdots q_n] = \int_0^\infty p(t)dt. \quad (11)$$

We have

$$p_i(\infty) = 0, \quad i \in L, \quad (12)$$

and

$$0 < p_i(\infty) \leq 1, \quad i \in K, \quad (13)$$

where $p_i(\infty)$ $(i \in K)$ is the absorbing probability, and

$$\sum_{i \in K} p_i(\infty) = 1. \quad (14)$$

Then, from (12) and (13) we have

$$p_i(\infty) - p_i(0) = \begin{cases} -1, & i = 1 \\ 0, & i \in L, \ i \neq 1 \\ p_i(\infty), & i \in K. \end{cases} \quad (15)$$
An efficient calculation algorithm in continuous-time Markov analysis

It follows from (7) that only \((n - m)\) elements of \(q\) in (11), \(q_{\ell_1}, \ldots, q_{\ell_{n-m}}\), make sense in (10). Combining them, we define

\[
q_{\text{sub}} = [q_{\ell_1} \cdots q_{\ell_{n-m}}].
\]

(16)

It follows from (7) that \(m\) elements of \(q\) in (11), \(q_{k_1}, \ldots, q_{k_m}\), are indeterminate in (10). It is clear from (13) that they do not make sense as \(\int_0^\infty p_{k_j}(t)dt, 1 \leq j \leq m\).

Obtain \(A_{\text{sub}} \in \mathbb{R}^{(n-m)\times(n-m)}\) from \(A'\) by deleting its \(k_1, \ldots, k_m\)-th rows and \(k_1, \ldots, k_m\)-th columns. Note that \(A_{\text{sub}}\) is equal to the matrix obtained from \(A\) by the similar manipulations because the differences between \(A\) and \(A'\) exist only in the \(k_1, \ldots, k_m\)-th rows, which are deleted to obtain \(A_{\text{sub}}\). Then, considering (4) and (14), we can have the unique solution \(q_{\text{sub}} \in \mathbb{R}^{1\times(n-m)}\) to the linear equation

\[
q_{\text{sub}} A_{\text{sub}} = [-1 \ 0 \ \cdots \ 0].
\]

(17)

Because \(q_{\ell_j} = \int_0^\infty p_{\ell_j}(t)dt\) in (11) is equal to the mean dwell time of State \(\ell_j\), which is a non-absorbing state, we have

\[
\text{MTTFV} = \sum_{j=1}^{n-m} q_{\ell_j}.
\]

(18)

The possession rate \(\beta_{k_j}\) of State \(k_j\) \((1 \leq j \leq m)\) is equal to the absorption probability \(p_{k_j}(\infty)\), which is obtained by the \(k_j\)-th element of the \(1 \times n\) vector \(qA'\) in the right-hand side of (10). Obtain \(a_{\text{sub},k_j} \in \mathbb{R}^{(n-m)\times 1}\) from the \(k_j\)-th column of \(A'\) by deleting its elements in the \(k_1, \ldots, k_m\)-th positions. Note that \(a_{\text{sub},k_j}\) is equal to the vector obtained from the \(k_j\)-th column of \(A\) by the similar manipulations because the differences between \(A\) and \(A'\) exist only in the \(k_1, \ldots, k_m\)-th rows, which are deleted to obtain \(a_{\text{sub},k_j}\). Then, we have

\[
\beta_{k_j} = p_{k_j}(\infty) = q_{\text{sub}} a_{\text{sub},k_j} (1 \leq j \leq m).
\]

(19)

The ordinary calculation can be summarized as follows.

**Algorithm 3.1 (Ordinary calculation)**

**Step 1:** Obtain \(A_{\text{sub}}\) from the transition rates matrix \(A\) by deleting its \(k_1, \ldots, k_m\)-th rows and \(k_1, \ldots, k_m\)-th columns.

**Step 2:** Solve the linear equation (17) for \(q_{\text{sub}}\).

**Step 3:** Obtain MTTFV by (18).

**Step 4:** For \(1 \leq j \leq m\), obtain \(a_{\text{sub},k_j}\) from the \(k_j\)-th column of \(A\) by deleting its elements in the \(k_1, \ldots, k_m\)-th positions. Obtain the possession rate \(\beta_{k_j}\) of State \(k_j\) by (19).
4 Proposed calculation

4.1 Algorithm

Here is the proposed calculation algorithm. We rearrange the linear equation (10) so that \( q_{k_1} = \cdots = q_{k_m} = 0 \), where they are indeterminate in the ordinary calculation, in order to increase efficiency in memory usage and computation time.

Algorithm 4.1 (Proposed calculation)

Step 1: Obtain \( \tilde{A} \in \mathbb{R}^{n \times n} \) from \( A \) by replacing its \( k_1\)-th, \( k_2\)-th, \ldots, \( k_m\)-th columns by the \( k_j\)-th elementary unit vector

\[
e_{k_j} = t[0 \cdots 0 1 0 \cdots 0] (\in \mathbb{R}^{n \times 1}), \quad 1 \leq j \leq m, \quad (20)
\]

respectively.

Step 2: Solve the linear equation

\[
r \tilde{A} = [-1 \ 0 \ \cdots \ 0] \quad (21)
\]

for \( r = [r_1 \ \cdots \ r_n] \in \mathbb{R}^{1 \times n} \).

Step 3:

\[
MTTFV = \sum_{i=1}^{n} r_i. \quad (22)
\]

Step 4: For \( 1 \leq j \leq m \), let \( a_{k_j} \in \mathbb{R}^{n \times 1} \) denote the \( k_j\)-th column of \( A \). Obtain the possession rate \( \beta_{k_j} \) of State \( k_j \) by

\[
\beta_{k_j} = ra_{k_j}. \quad (23)
\]

4.2 Proof of consistency

Consider the following column rearrangement by post-multiplying with \( V (= [v_{ij}]) \in \mathbb{R}^{n \times n} \):

\[
p(t)V = [p_{\ell_1}(t) \ \cdots \ p_{\ell_{n-m}}(t) \ p_{k_1}(t) \ \cdots \ p_{k_m}(t)], \quad (24)
\]

where

\[
v_{ij} = \begin{cases} 
1, & 1 \leq j \leq n - m, \ i = \ell_j, \text{ or} \\
1, & n - m + 1 \leq j \leq n, \ i = k_j - (n - m) \\
0, & \text{otherwise}.
\end{cases} \quad (25)
\]
The transpose of (24) gives the following row rearrangement by pre-multiplying with \( tV \):

\[
\begin{pmatrix}
  p_{\ell_1}(t) \\
  \vdots \\
  p_{\ell_{n-m}}(t) \\
  p_{k_1}(t) \\
  \vdots \\
  p_{k_m}(t)
\end{pmatrix}
\]

(26)

It holds that

\[
V^tV = E
\]

(27)
as shown next. From (24) and (26), we have

\[
p(t)V^tV^tp(t) = \begin{bmatrix}
  p_{\ell_1}(t) \\
  \vdots \\
  p_{\ell_{n-m}}(t) \\
  p_{k_1}(t) \\
  \vdots \\
  p_{k_m}(t)
\end{bmatrix}
\begin{bmatrix}
  p_{\ell_1}(t) \\
  \vdots \\
  p_{\ell_{n-m}}(t) \\
  p_{k_1}(t) \\
  \vdots \\
  p_{k_m}(t)
\end{bmatrix} + \sum_{j=1}^{n-m} p_{\ell_j}^2(t) + \sum_{j=1}^{m} p_{k_j}^2(t) = \sum_{j=1}^{n} p_j^2(t) = p(t)^t p(t),
\]

(28)
hence

\[
p(t)[V^tV - E]^tp(t) = 0.
\]

(29)
Because this holds for any \( p(t) \), we have \( V^tV - E = 0 \), which implies that (27) holds.

Post-multiplying (21) with \( V \) and using (27) lead to

\[
[r_1 \cdots r_n]V^tV\tilde{A}V = [-1 \ 0 \ \cdots \ 0]V.
\]

(30)
Here, we have

\[
[r_1 \cdots r_n]V = [r_{\ell_1} \cdots r_{\ell_{n-m}} r_{k_1} \cdots r_{k_m}]
\]

(31)

\[
V^t\tilde{A}V = \begin{bmatrix}
  A'_{sub} & 0 \\
  A'_{21} & E
\end{bmatrix}
\]

(32)

\[
[-1 \ 0 \ \cdots \ 0]V = [-1 \ 0 \ \cdots \ 0].
\]

(33)
In (32), the $(1,1)$-block of $\tilde{V} \tilde{A} \tilde{V}$ is the matrix obtained from $\tilde{A}$ by extracting its $\ell_1, \ldots, \ell_{n-m}$-th rows and $\ell_1, \ldots, \ell_{n-m}$-th columns, which is equal to the matrix obtained from $A$ by the similar manipulations. Because we replace the $k_1, \ldots, k_m$-th columns of $A$ in Step 1, the $(1,1)$-block of $\tilde{V} \tilde{A} \tilde{V}$ is equal to $A_{\text{sub}}$ obtained from $A$ by the similar manipulations in Step 1 of the ordinary calculation.

Note that $\tilde{A}_{21}$ in (32) is not the zero matrix if there exists a target and non-absorbing state in the original transition rates matrix $A$.

In (33), the element “$-1$” in the first position of the vector $[-1 \ 0 \ \cdots \ 0]$, which is the only one nonzero element, is not permuted with another element by post-multiplying with $V$ because State 1 is not a target state.

Applying (31), (32), and (33) to (30), we have

$$\begin{bmatrix} r_{\ell_1} & \cdots & r_{\ell_{n-m}} & r_{k_1} & \cdots & r_{k_m} \end{bmatrix} \begin{bmatrix} A_{\text{sub}} & 0 \\ \tilde{A}_{21} & E \end{bmatrix} = [-1 \ 0 \ \cdots \ 0].$$

Using the inverse matrix of a lower-triangular block matrix

$$\begin{bmatrix} X_{11} & 0 \\ X_{21} & X_{22} \end{bmatrix}^{-1} = \begin{bmatrix} X_{11}^{-1} & 0 \\ -X_{22}^{-1}X_{21}X_{11}^{-1} & X_{22}^{-1} \end{bmatrix},$$

we have

$$\begin{bmatrix} r_{\ell_1} & \cdots & r_{\ell_{n-m}} \\ r_{k_1} & \cdots & r_{k_m} \end{bmatrix} = [-1 \ 0 \ \cdots \ 0] A_{\text{sub}}^{-1} \quad \text{(36)}$$

$$\begin{bmatrix} r_{\ell_1} & \cdots & r_{\ell_{n-m}} \end{bmatrix} = [0 \ \cdots \ 0]. \quad \text{(37)}$$

Hence, from (17) and (36) we have

$$q_{\text{sub}} = [q_1 \ \cdots \ q_{n-m}] = [r_{\ell_1} \ \cdots \ r_{\ell_{n-m}}]. \quad \text{(38)}$$

It follows from (37) and (38) that the calculation result of MTTFV by (18) is equal to the one by (22).

Using (27), (24), (26), (37), and (38) in this order, we have

$$r a_{k_j} = r V^t V a_{k_j} = \begin{bmatrix} a_{k_j \ell_1} & \cdots \\ a_{k_j \ell_{n-m}} & a_{k_j k_1} & \cdots \\ a_{k_j k_m} \end{bmatrix} \begin{bmatrix} a_{k_j \ell_1} \\ \vdots \\ a_{k_j \ell_{n-m}} \end{bmatrix} = q_{\text{sub}} a_{\text{sub},k_j}. \quad \text{(39)}$$

Hence, the possession rate $\beta_{k_j}$ calculated by (23) is equal to the one calculated by (19).
4.3 Comparison with ordinary calculation

4.3.1 Advantages

(i) Economy of memory usage: In Step 1, we can obtain \( \tilde{A} \) by overwriting \( A \). Hence, the proposed calculation needs additional memory usage only to restore \( \mathbb{R}^{n \times m} \) data of \( a_{kj} \in \mathbb{R}^{1 \times n} \), the \( k_j \)-th column of \( A \), before the overwriting. On the other hand, in addition to \( \mathbb{R}^{(n-m) \times m} \) data of \( a_{\text{sub},kj} \in \mathbb{R}^{1 \times (n-m)} \), the ordinary calculation needs additional memory usage for enormous data amount of \( A_{\text{sub}} \in \mathbb{R}^{(n-m) \times (n-m)} \), which is obtained from \( A \) by deleting its \( k_1, \ldots, k_m \)-th rows and \( k_1, \ldots, k_m \)-th columns.\(^3\)

Roughly speaking, especially in the case \( n \gg m \), the proposed calculation saves the memory usage for \( A_{\text{sub}} \), which is necessary in the ordinary calculation, and thus reduces memory usage by about one-half. See Section 5 (vi).

(ii) Reduction of computation time: The proposed calculation can reduce computation time for the following reasons.

- In Steps 1 and 4, only column manipulations are required, while both column and row manipulations are required in the ordinary calculation.

- In Step 1, it suffices to overwrite directly the \( k_j \)-th column with the \( k_j \)-th elementary unit vector \( e_{kj} \) in (20), where \( k_j \) belongs to the given set of state numbers of all target states \( K \). On the other hand, in the ordinary calculation, it is necessary to check whether \( i \) does not belong to \( K \) before extracting the \( i \)-th column/row of \( A \) to obtain \( A_{\text{sub}} \).

Thus, especially in Step 1, the proposed calculation reduces computation time to less than \( m/(2n) \).

However, almost all matrices/vectors handled in the two calculations are sparse in general. A special and efficient data storage for such sparse matrices/vectors is absolutely indispensable to large-scale numerical calculation. Thus, it is difficult to compare them in (ordinary) computational quantity.

4.3.2 Disadvantage

The only disadvantage to the proposed calculation is that the dimension of the linear equation to be numerically solved in Step 2 is equal to \( n \), while it is equal to \( (n-m) \) in the ordinary calculation. However, in spite of this disadvantage, the proposed calculation is considerably more efficient than the ordinary calculation in the total computation time for Steps 1–4 as shown in Section 5.

\(^3\)It is possible to obtain \( A_{\text{sub}} \) by overwriting the memory space for \( A \) to economize on memory usage. However, such calculation requires complicated manipulations and enormous amount of computation time.
5 Example

(i) Dynamic fault tree: Consider a dynamic fault tree shown in Figure 1. Each basic event, the top event, and each gate assume two states, a non-occurrence state and an occurrence state. An occurrence from a non-occurrence state to an occurrence state [a restoration from an occurrence state to a non-occurrence state] probabilistically happens in each basic event $B_k$ in accordance with the exponential distribution with the occurrence rate $\lambda_k$ and the restoration rate $\mu_k$, $1 \leq k \leq 11$, given as follows:

$$
\begin{align*}
\lambda_1 &= 0.2 \text{ [1/hr]}, \quad \mu_1 = 0.05 \text{ [1/hr]}, \quad \lambda_2 = 0.1 \text{ [1/hr]}, \quad \mu_2 = 0.07 \text{ [1/hr]} \\
\lambda_3 &= 0.2 \text{ [1/hr]}, \quad \mu_3 = 0.05 \text{ [1/hr]}, \quad \lambda_4 = 0.1 \text{ [1/hr]}, \quad \mu_4 = 0.07 \text{ [1/hr]} \\
\lambda_5 &= 0.5 \text{ [1/hr]}, \quad \mu_5 = 0.10 \text{ [1/hr]}, \quad \lambda_6 = 0.1 \text{ [1/hr]}, \quad \mu_6 = 0.06 \text{ [1/hr]} \\
\lambda_7 &= 0.2 \text{ [1/hr]}, \quad \mu_7 = 0.05 \text{ [1/hr]}, \quad \lambda_8 = 0.2 \text{ [1/hr]}, \quad \mu_8 = 0.1 \text{ [1/hr]} \\
\lambda_9 &= 0.1 \text{ [1/hr]}, \quad \mu_9 = 0.2 \text{ [1/hr]}, \quad \lambda_{10} = 0.2 \text{ [1/hr]}, \quad \mu_{10} = 0.3 \text{ [1/hr]} \\
\lambda_{11} &= 0.1 \text{ [1/hr]}, \quad \mu_{11} = 0.3 \text{ [1/hr]}. 
\end{align*}
$$

(ii) State space: The $G_2$, $G_5$, and $G_9$ are priority AND (PAND) gates. A PAND gate is in an occurrence state if and only if all its inputs have occurred in sequence from left to right. Hence, for the shadow basic events, $B_1, \ldots, B_7$, which are located in the subtree with a PAND gate as its topmost element, their orders in an occurrence sequence have an influence on the top event. For the rest, $B_8, \ldots, B_{11}$, only their states have an influence on the top event. Hence, we describe the state of the Markov model of the dynamic fault tree.
by \((i_1, \ldots, i_7; i_8, \ldots, i_{11})\), where \(i_k\) \((1 \leq k \leq 11)\) represents the state of \(B_k\) as follows:

\[
i_k = \begin{cases} 
0, & B_k \text{ is in a non-occurrence state} \\
j (1 \leq j \leq 7), & B_k \text{ has been in an occurrence state from the } j\text{-th earliest time out of } B_1, \ldots, B_7 \text{ in occurrence states,} \\
1 \leq k \leq 7,
\end{cases}
\]

and

\[
i_k = \begin{cases} 
0, & B_k \text{ is in a non-occurrence state} \\
1, & B_k \text{ is in an occurrence state,} \\
8 \leq k \leq 11.
\end{cases}
\]

Then, there are \((n =) 219200\) states in the state space. The initial state is State 1: \((0, \ldots, 0; 0, \ldots, 0)\).

(iii) Target states: In fault tree analysis, top-event occurrence states are target states. Then, MTTFV corresponds to MTTFO (Mean Time To First Occurrence) of the top event. In this case, there are \((m =) 174\) top-event occurrence states in the state space such as the three ones in Table 1.

(iv) Transition rates matrix: The \(G_4\) is a d-OR gate, which is located in the subtree with the PAND gate \(G_2\) as its topmost element. The d-OR gate problem is that owing to the existence of a d-OR gate, it is impossible to properly understand the occurrences (and restorations) of the top event [5]. Thus, in order to solve such a problem, we should obtain a transition rates matrix \(A\) by using the extended transition rule [5]. Hence, there exist no absorbing states in the obtained \(A\).

(v) Verification of analysis results: Table 1 shows the results obtained by Markov analysis using the proposed calculation (Algorithm 4.1) and by Monte Carlo simulation with taking the d-OR problem into consideration as in [5] (carried out until the top event fell into an occurrence state one million times). There is good agreement between these results within 0.25% errors. This implies that the proposed calculation gives the correct analysis results in spite of the existence of the d-OR gate \(G_4\).

<table>
<thead>
<tr>
<th></th>
<th>Markov analysis (Algorithm 4.1)</th>
<th>Monte Carlo simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTTFV</td>
<td>(5.9694 \times 10^3) [hr]</td>
<td>(5.9618 \times 10^3) [hr]</td>
</tr>
<tr>
<td>Possession rates</td>
<td>((1, 2, 3, 4, 6, 0, 5; 1, 1, 0, 1))</td>
<td>(2.9262 \times 10^{-2}) (2.9206 \times 10^{-2})</td>
</tr>
<tr>
<td>Top-three (target states)</td>
<td>((1, 2, 3, 4, 5, 6, 0; 1, 1, 1, 0))</td>
<td>(2.3295 \times 10^{-2}) (2.3345 \times 10^{-2})</td>
</tr>
<tr>
<td>(1, 2, 3, 4, 5, 0, 6; 1, 1, 0, 1)</td>
<td>(2.2704 \times 10^{-2}) (2.2647 \times 10^{-2})</td>
<td></td>
</tr>
</tbody>
</table>

(vi) Efficiency in memory usage: Table 2 shows the dimensions and the numbers of nonzero elements in the handled matrices/vectors. Even if we
use an efficient storage for the sparse matrices/vectors, $\mathbf{A}$, $\mathbf{A}_{\text{sub}}$, $\mathbf{a}_{\text{sub},k_j}$, and $
abla \mathbf{k_j}$, the ordinary calculation and the proposed calculation need at least about 88MB and 48MB of memory for data storage, respectively, because $\tilde{\mathbf{A}}$ is obtained from $\mathbf{A}$ by overwriting it. The proposed calculation is considerably more efficient than the ordinary calculation in memory usage.

Table 2: Dimensions and numbers of nonzero elements of matrices/vectors.

<table>
<thead>
<tr>
<th>Ordinary calculation (Algorithm 3.1)</th>
<th>Proposed calculation (Algorithm 4.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbf{A} \in \mathbb{R}^{219200 \times 219200}$: 2630400</td>
<td></td>
</tr>
<tr>
<td>$\mathbf{A}_{\text{sub}} \in \mathbb{R}^{219026 \times 219026}$: 2627718</td>
<td>(\tilde{\mathbf{A}} \text{ (overwriting $\mathbf{A}$)}: 2629260)</td>
</tr>
<tr>
<td>$\mathbf{a}_{\text{sub},k_j} \in \mathbb{R}^{219026 \times 1}$: 594 (1 ≤ $j$ ≤ 174)</td>
<td>$\mathbf{a}_{\text{k_j}} \in \mathbb{R}^{219200 \times 1}$: 1314 (1 ≤ $j$ ≤ 174)</td>
</tr>
<tr>
<td>$\mathbf{q}_{\text{sub}} \in \mathbb{R}^{219026 \times 1}$: 219026</td>
<td>$\mathbf{r} \in \mathbb{R}^{219200 \times 1}$: 219026</td>
</tr>
</tbody>
</table>

(vii) Efficiency in computation time: Table 3 shows the computation time for the ordinary calculation and for the proposed calculation. This implies that the proposed calculation is considerably more efficient than the ordinary calculation in the computation time.

Table 3: Computation time.

<table>
<thead>
<tr>
<th></th>
<th>Ordinary calculation (Algorithm 3.1)</th>
<th>Proposed calculation (Algorithm 4.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Solo 1.20GHz CPU, 2GB Memory Windows XP (x32), Matlab 2007a (x32)</td>
<td>484 [sec]</td>
<td>113 [sec]</td>
</tr>
<tr>
<td>Xeon 2.66GHz CPU, 32GB Memory Windows Vista (x64), Matlab 2007a (x64)</td>
<td>267 [sec]</td>
<td>68 [sec]</td>
</tr>
</tbody>
</table>

References


Received: August, 2011