Truncation of Passage Time Calculations in Large Semi-Markov Models

Marcel C. Guenther Nicholas J. Dingle
Jeremy T. Bradley William J. Knottenbelt *

Abstract

Calculation of passage time distributions in large semi-Markov models can be accomplished by means of a previously-presented iterative algorithm, the core of which is repeated sparse matrix–vector multiplication. The algorithm's performance is therefore highly dependent on the number of multiplications of matrix and vector elements that must be performed during each iteration. At the same time, the products of matrix and vector elements that are very small contribute little to the overall result of the multiplication. In this paper, we investigate the effect of ignoring these values on both the performance and accuracy of the iterative passage time algorithm. We show that in the models we analyse here this truncation significantly reduces the number of multiplications which must be performed, and hence significantly reduces the running time of the algorithm, with little effect on the accuracy of the final result.

1 Introduction

Semi-Markov Processes (SMPs) are an extension of Markov processes which allow for generally distributed sojourn times [12, 13]. Although the memoryless property no longer holds for state sojourn times, at transition instants SMPs still behave in the same way as Markov processes (that is to say, the choice of the next state is based only on the current state) and so share some of their analytical tractability. In prior work [4, 6] we have presented an iterative numerical algorithm for the calculation of passage time densities and distributions in structurally-unrestricted semi-Markov models. The kernel of this algorithm is repeated sparse matrix–vector multiplication, and its running time is therefore heavily influenced by the number of multiplications of individual vector and matrix elements which are performed during each iteration.

In analysing very large semi-Markov models with state spaces of the order of 10^7 states, we encountered the phenomenon of numerical underflow. In accordance

*Department of Computing, Imperial College London, 180 Queens Gate, London SW7 2BZ, UK {mcg05,njd200,jb,wjk}@doc.ic.ac.uk

with the IEE 754 floating point standard, the Intel families of processors do not by default round floating point numbers which are smaller than double precision to zero but instead continue to represent them at reduced precision [8]. Our experience in analysing very large semi-Markov models, as well as that of the authors of [11], is that this default behaviour imposes a large performance overhead on applications in the presence of underflow (in our experience, slowing computation down by a factor of up to 200). Furthermore, for our algorithm it was not necessary to represent such very small values explicitly; the convergence criterion is typically 10^{-8} or 10^{-16}, and so vector elements of of extremely small magnitude, e.g. 10^{-100} or smaller, contribute negligibly to the final result and could be set to 0 without affecting the overall accuracy.

In this paper we investigate the effect of more aggressive truncation of vector elements on the performance and accuracy of our iterative passage time algorithm, inspired by our experiences of overcoming these underflow problems. We have observed above that the number of complex multiplications performed at each iteration has a major impact on the running time of our algorithm, and this stems from the fact that floating point operations are some of the most expensive CPU instructions to perform. For example, on an Intel Core Duo CPU an integer add operation takes approximately 1 clock cycle while a floating point add takes 3 [8].

Our first optimisation is therefore to modify the implementation of our algorithm to ignore vector elements which are 0 when multiplying them with matrix elements as such values obviously contribute nothing to the final result. In our earlier implementation the vector was stored in a dense fashion with all values, including zeros, explicitly represented, and so to avoid the overhead of having to check each value before use we here employ a sparse vector storage scheme. We demonstrate that this reduces the total number of multiplications carried out during calculation.

We also investigate the effect of raising the threshold of zeroing vector elements to 10^{-32}. Again, we show that this reduces the total number of multiplications compared to both the original and modified implementations, and also show that it reduces the elapsed running time of the algorithm. Furthermore, we present results which show that the accuracy of the final passage time distributions calculated for the models considered in this paper is not affected by this truncation.

The remainder of this paper is organised as follows. Section 2 presents background theory on semi-Markov processes and summarises an iterative numerical algorithm from [4, 6] for the calculation of passage time densities and distributions in such processes. Section 3 then introduces the idea of truncating values in this algorithm during calculation and presents results to demonstrate that this improves the algorithm's performance without significantly affecting the accuracy of the final result. Finally, Section 4 concludes and suggests directions for future work.

2 Background

2.1 Semi-Markov Processes

Consider a Markov renewal process $\{(\chi_n, T_n) : n \geq 0\}$ where T_n is the time of the nth transition ($T_0 = 0$) and $\chi_n \in \mathcal{S}$ is the state at the nth transition. Let the kernel of this process be:

$$R(n, i, j, t) = \mathbb{P}(\chi_{n+1} = j, T_{n+1} - T_n \leq t \mid \chi_n = i)$$

for $i, j \in \mathcal{S}$. The continuous time semi-Markov process, $\{Z(t), t \geq 0\}$, defined by the kernel R, is related to the Markov renewal process by:

$$Z(t) = \chi_{N(t)}$$

where $N(t) = \max\{n : T_n \leq t\}$, i.e. the number of state transitions that have taken place by time t. Thus $Z(t)$ represents the state of the system at time t. We consider only time-homogeneous SMPs in which $R(n, i, j, t)$ is independent of n:

$$\begin{aligned} R(i, j, t) &= \mathbb{P}(\chi_{n+1} = j, T_{n+1} - T_n \leq t \mid \chi_n = i) \quad \text{for any } n \geq 0 \\ &= p_{ij} H_{ij}(t) \end{aligned}$$

where $p_{ij} = \mathbb{P}(\chi_{n+1} = j \mid \chi_n = i)$ is the state transition probability between states i and j and $H_{ij}(t) = \mathbb{P}(T_{n+1} - T_n \leq t \mid \chi_{n+1} = j, \chi_n = i)$, is the sojourn time distribution in state i when the next state is j. An SMP can therefore be characterised by two matrices $\mathbf{P}$ and $\mathbf{H}$ with elements p_{ij} and H_{ij} respectively.

2.2 Iterative Passage Time Algorithm

Consider a finite, irreducible, continuous-time semi-Markov process with N states $\{1, 2, \ldots, N\}$. Recalling that $Z(t)$ denotes the state of the SMP at time t ($t \geq 0$) and that $N(t)$ denotes the number of transitions which have occurred by time t, the first passage time from a source state i at time t into a non-empty set of target states $\vec{j}$ is defined as:

$$P_{i\vec{j}}(t) = \inf\{u > 0 : Z(t + u) \in \vec{j}, N(t + u) > N(t), Z(t) = i\}$$

For a stationary time-homogeneous SMP, $P_{i\vec{j}}(t)$ is independent of t:

$$P_{i\vec{j}} = \inf\{u > 0 : Z(u) \in \vec{j}, N(u) > 0, Z(0) = i\} \tag{1}$$

$P_{i\vec{j}}$ has an associated probability density function $f_{i\vec{j}}(t)$. The Laplace transform of $f_{i\vec{j}}(t)$, $L_{i\vec{j}}(s)$, can be computed by means of a first-step analysis. That is, we consider moving from the source state i into the set of its immediate successors $\vec{k}$ and must distinguish between those members of $\vec{k}$ which are target states and those which are not. This calculation can be achieved by solving a set of N linear equations of the form:

$$L_{i\vec{j}}(s) = \sum_{k \notin \vec{j}} r^*_{ik}(s) L_{k\vec{j}}(s) + \sum_{k \in \vec{j}} r^*_{ik}(s) \qquad : \text{for } 1 \leq i \leq N \tag{2}$$

where $r^*_{ik}(s)$ is the Laplace–Stieltjes transform (LST) of $R(i,k,t)$ from Section 2.1 and is defined by:

$$r^*_{ik}(s) = \int_0^\infty e^{-st}\, dR(i,k,t)$$

Eq. 2 has matrix–vector form $\mathbf{Ax} = \mathbf{b}$, where the elements of $\mathbf{A}$ are general functions of the complex variable s. For example, when $\vec{j} = \{1\}$, Eq. 2 yields:

$$\begin{pmatrix} 1 & -r^*_{12}(s) & \cdots & -r^*_{1N}(s) \\ 0 & 1-r^*_{22}(s) & \cdots & -r^*_{2N}(s) \\ 0 & -r^*_{32}(s) & \cdots & -r^*_{3N}(s) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & -r^*_{N2}(s) & \cdots & 1-r^*_{NN}(s) \end{pmatrix} \begin{pmatrix} L_{1\vec{j}}(s) \\ L_{2\vec{j}}(s) \\ L_{3\vec{j}}(s) \\ \vdots \\ L_{N\vec{j}}(s) \end{pmatrix} = \begin{pmatrix} r^*_{11}(s) \\ r^*_{21}(s) \\ r^*_{31}(s) \\ \vdots \\ r^*_{N1}(s) \end{pmatrix}$$

We now describe an iterative algorithm for generating passage time densities that creates successively better approximations to the SMP passage time quantity $P_{i\vec{j}}$ of Eq. 1 [4, 6]. We approximate $P_{i\vec{j}}$ as $P^{(r)}_{i\vec{j}}$, for a sufficiently large value of r, which is the time for r consecutive transitions to occur starting from state i and ending in any of the states in $\vec{j}$. We calculate $P^{(r)}_{i\vec{j}}$ by constructing and then numerically inverting [1, 2, 3] its Laplace transform $L^{(r)}_{i\vec{j}}(s)$.

Recall the semi-Markov process $Z(t)$ of Section 2.1, where $N(t)$ is the number of state transitions that have taken place by time t. We formally define the rth transition first passage time to be:

$$P^{(r)}_{i\vec{j}} = \inf\{u > 0 : Z(u) \in \vec{j}, 0 < N(u) \leq r, Z(0) = i\} \tag{3}$$

which is the time taken to enter a state in $\vec{j}$ for the first time having started in state i at time 0 and having undergone up to r state transitions.

$P^{(r)}_{i\vec{j}}$ is a random variable with associated Laplace transform $L^{(r)}_{i\vec{j}}(s)$. $L^{(r)}_{i\vec{j}}(s)$ is, in turn, the ith component of the vector:

$$\mathbf{L}^{(r)}_{\vec{j}}(s) = \left(L^{(r)}_{1\vec{j}}(s), L^{(r)}_{2\vec{j}}(s), \ldots, L^{(r)}_{N\vec{j}}(s)\right)$$

representing the passage time for terminating in $\vec{j}$ for each possible start state. This vector may be computed as:

$$\mathbf{L}^{(r)}_{\vec{j}}(s) = \mathbf{U}\left(\mathbf{I} + \mathbf{U}' + \mathbf{U}'^2 + \cdots + \mathbf{U}'^{(r-1)}\right)\mathbf{e}_{\vec{j}} \tag{4}$$

where $\mathbf{U}$ is a matrix with elements $u_{pq} = r^*_{pq}(s)$ and $\mathbf{U}'$ is a modified version of $\mathbf{U}$ with elements $u'_{pq} = \delta_{p\notin\vec{j}}\, u_{pq}$, where states in $\vec{j}$ have been made absorbing. Here, $\delta_{p\notin\vec{j}} = 1$ if $p \notin \vec{j}$ and 0 otherwise. The initial multiplication with $\mathbf{U}$ in Eq. 4 is included so as to generate cycle times for cases such as $L^{(r)}_{ii}(s)$ which would otherwise register as 0 if $\mathbf{U}'$ were used instead. The column vector $\mathbf{e}_{\vec{j}}$ has entries $e_{k\vec{j}} = \delta_{k\in\vec{j}}$, where $\delta_{k\in\vec{j}} = 1$ if k is a target state ($k \in \vec{j}$) and 0 otherwise. From Eq. 1 and Eq. 3:

$$P_{i\vec{j}} = P^{(\infty)}_{i\vec{j}} \quad \text{and thus} \quad L_{i\vec{j}}(s) = L^{(\infty)}_{i\vec{j}}(s)$$

This can be generalised to multiple source states $\vec{i}$ using, for example, a normalised steady-state vector $\boldsymbol{\alpha}$ calculated from $\boldsymbol{\pi}$, the steady-state vector of the embedded discrete-time Markov chain (DTMC) with one-step transition probability matrix $\mathbf{P} = [p_{ij}, 1 \leq i, j \leq N]$, as:

$$\alpha_k = \begin{cases} \pi_k / \sum_{j \in \vec{i}} \pi_j & \text{if } k \in \vec{i} \\ 0 & \text{otherwise} \end{cases}$$

The row vector with components α_k is denoted by $\boldsymbol{\alpha}$. The formulation of $L^{(r)}_{\vec{i}\vec{j}}(s)$ is therefore:

$$\begin{aligned} L^{(r)}_{\vec{i}\vec{j}}(s) &= \boldsymbol{\alpha} \mathbf{L}^{(r)}_{\vec{j}}(s) \\ &= (\boldsymbol{\alpha}\mathbf{U} + \boldsymbol{\alpha}\mathbf{U}\mathbf{U}' + \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^2 + \cdots + \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^{(r-1)})\, \mathbf{e}_{\vec{j}} \\ &= \sum_{k=0}^{r-1} \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^k \mathbf{e}_{\vec{j}} \end{aligned}$$

Note that the central operation in Eq. 5 is repeated sparse matrix–vector multiplication; we begin by calculating the vector $\boldsymbol{\nu}_{(0)} = \boldsymbol{\alpha}\mathbf{U}$ and then repeatedly multiply this with the matrix $\mathbf{U}'$ such that $\boldsymbol{\nu}_{(k)} = \boldsymbol{\nu}_{(k-1)}\mathbf{U}'$ $(k \geq 1)$ until the calculation converges.

In practice, convergence of the sum $L^{(r)}_{\vec{i}\vec{j}}(s) = \sum_{k=0}^{r-1} \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^k$ can be said to have occurred if, for a particular r and s-point:

$$|\mathrm{Re}(L^{(r+1)}_{\vec{i}\vec{j}}(s) - L^{(r)}_{\vec{i}\vec{j}}(s))| < \varepsilon \quad \text{and} \quad |\mathrm{Im}(L^{(r+1)}_{\vec{i}\vec{j}}(s) - L^{(r)}_{\vec{i}\vec{j}}(s))| < \varepsilon \qquad (5)$$

where ε is chosen to be a suitably small value, say $\varepsilon = 10^{-16}$.

2.3 Case Study: Semi-Markov Models

Throughout this paper we use two semi-Markov models as running examples. The Voting model is a model of a distributed voting system with voters, failure-prone voting booths and failure-prone central servers [4, 6]. The Web-server model represents a web content authoring system, and contains a number of clients, authors web servers and a write buffer [6]. Both models were originally represented in a high-level Semi-Markov Stochastic Petri Net (SM-SPN) [5] form, from which semi-Markov processes of varying sizes can easily be generated. Further detail can be found in [9].

3 Truncation of Passage Iteration Vector Values

We observe that the passage iteration vector, $\boldsymbol{\nu}_{(k)}$, often contains a high proportion of elements with very small complex values. These contribute little to the updated values in $\boldsymbol{\nu}_{(k+1)}$ but these multiplications still require as much time to be computed as for the larger values in $\boldsymbol{\nu}_{(k)}$; indeed, in the presence of numerical underflow they can take substantially longer. We therefore study the impact of truncating (i.e. setting to zero) these small elements on the accuracy and performance of the iterative passage time analysis algorithm.

Definition 1. We define a *negligibly small complex number* s as one for which $|Re(s)| < \varepsilon^2$ and $|Im(s)| < \varepsilon^2$, where $\varepsilon > 0$ is the precision of the iterative passage time solver in Eq. 5.

Setting an element in $\boldsymbol{\nu}_{(k)}$ to zero can create an error that is larger than the absolute value of the truncated element. This is because of the cascading effect of the sparse matrix–vector multiplication. Any non-zero element in $\boldsymbol{\nu}_{(k)}$ which corresponds to a non-target state contributes to the value of at least one other entry in $\boldsymbol{\nu}_{(k)}$ during the next iteration. As many states have more than one outgoing transition the value of one element in the $\boldsymbol{\nu}_{(k)}$ vector usually contributes to the sums of a large percentage of the elements in $\boldsymbol{\nu}_{(k+m)}$ since the number of states a single state can reach in m state transitions can be exponentially high. It is thus important to restrict truncation to elements whose absolute values are much smaller than our required precision; otherwise truncation might have a negative impact on the accuracy of the results of the passage time calculation [10].

3.1 Methodology

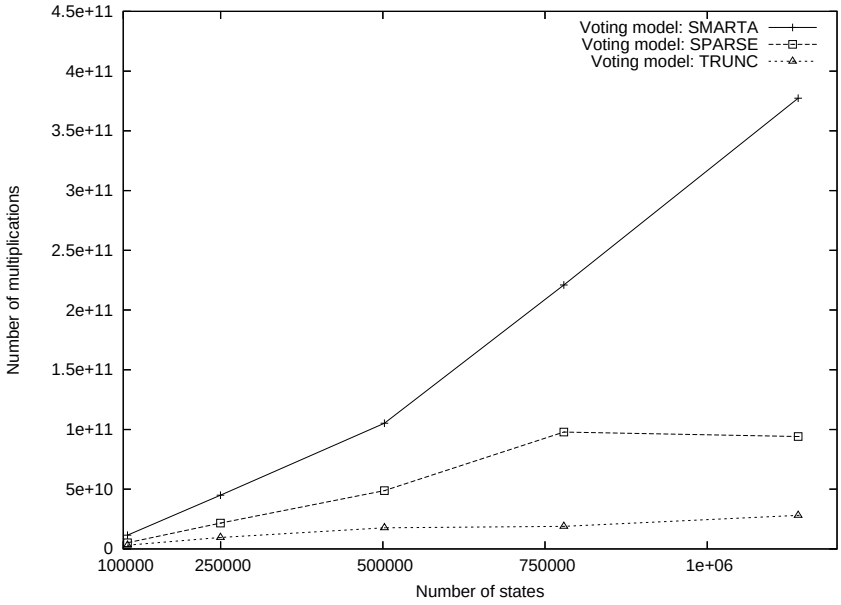
In the following analysis, we distinguish between three implementations of the iterative passage time algorithm:

- ***SMARTA***: The original implementation from [4], which was later developed as the *Semi-Markov Response Time Analyser* (*SMARTA*) tool in [9]. In *SMARTA*, every element of $\boldsymbol{\nu}_{(k)}$, including zeros, is explicitly represented in a dense vector, and multiplications with $\mathbf{U}'$ elements are carried out even when the corresponding vector element is zero. This provides our baseline for comparison.
- ***SPARSE***: A new implementation in which $\boldsymbol{\nu}_{(k)}$ is stored as a sparse vector. Under this scheme only elements which are non-zero are stored in the vector and hence multiplications of $\mathbf{U}'$ elements with vector values which are zero will not occur.
- ***TRUNC***: A second new implementation which extends the sparse vector implementation to actively truncate negligibly small complex numbers.

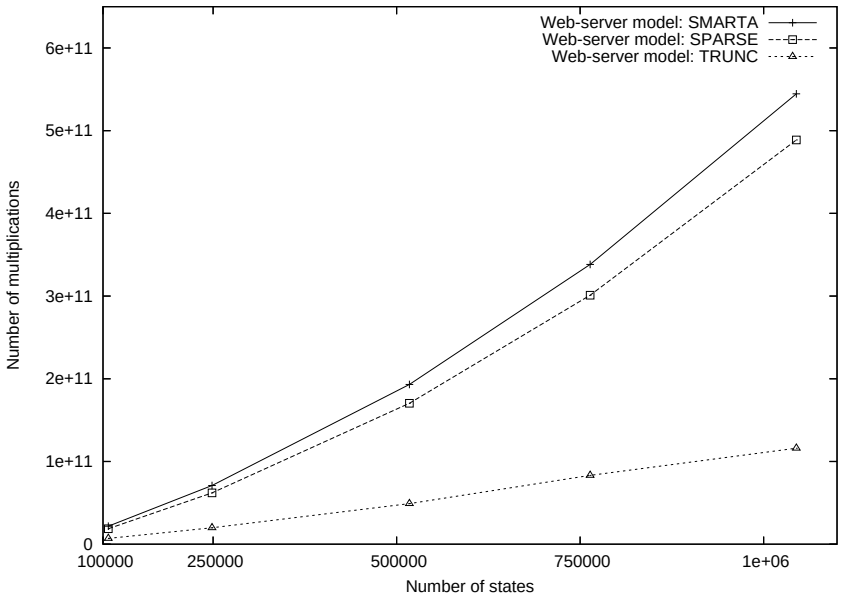
We assess the effect of truncation on the performance and accuracy of the iterative algorithm by conducting passage time analysis on the semi-Markov models described in Section 2.3 for a range of state-space sizes up to 1 100 000 states with $\varepsilon = 10^{-16}$. It would be too expensive to check all values in $\boldsymbol{\nu}_{(k)}$ on every iteration to determine if they should be rounded to zero, and so we instead choose to perform truncation every 25 iterations to minimise this overhead.

3.2 Performance

We evaluate the effect of truncation on the performance of the iterative passage time algorithm in two ways. Firstly, we consider the total number of complex



(a) Voting model



(b) Web-server model

Figure 1: Absolute number of complex multiplications carried out by the three implementations for the Voting (top) and Web-server (bottom) models across a range of state space sizes.

O(no. of states)	Voting model		Web-server model	
	SPARSE	*TRUNC*	*SPARSE*	*TRUNC*
100 000	45%	27%	86%	32%
250 000	48%	21%	87%	28%
500 000	46%	17%	88%	25%
750 000	44%	9%	89%	25%
1 000 000	25%	7%	90%	21%

Table 1: Percentage of complex multiplications needed by the two new implementations of the iterative passage time algorithm relative to *SMARTA*.

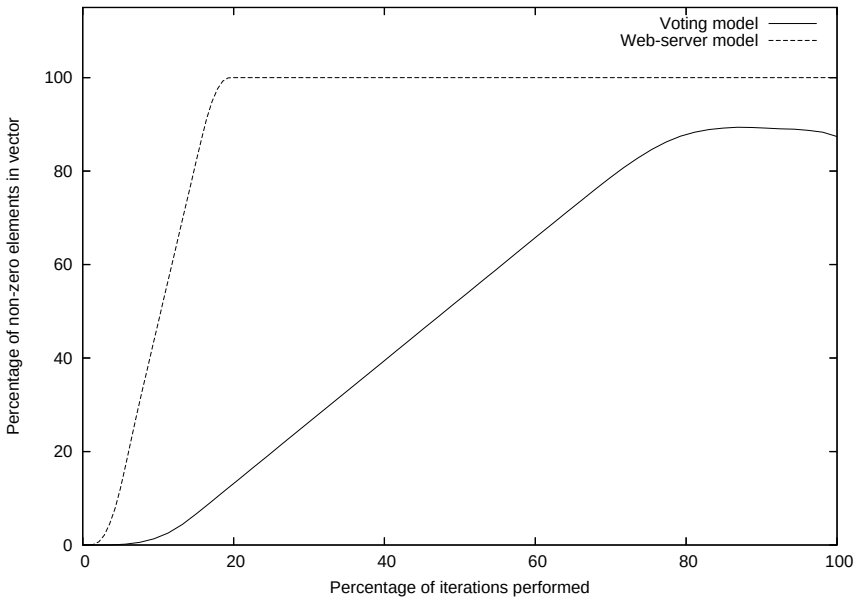


Figure 2: Percentage of non-zero elements in $\boldsymbol{\nu}_{(k)}$ during iterative solution. Results were calculated for both models with *c.* 250, 000 states without truncation of negligible values.

multiplications carried out during analysis. Figure 1 shows the total number of complex multiplications required by each of the three implementations to solve the Voting and Web-server models for a range of state space sizes, and Table 1 summarises these figures as percentages. These results demonstrate that truncation significantly reduces the amount of complex multiplication needed for the iterative passage time algorithm. Furthermore, we can observe that in both models the saving becomes larger as we increase the size of the model, which implies that truncation is likely to be scalable for SMPs in general.

Of particular interest is the difference in performance of the *SPARSE* implementation between the Voting and Web-server models. In the Voting model it performs approximately half the number of multiplications of *SMARTA*, but for the Web-server model the saving is only of the order of 10%. We believe this is due to the particular passage time selected for analysis; in the example

Method	Voting model (1 100 000 states)		Web-server model (1 000 000) states	
	Run-time (seconds)	% of *SMARTA* run-time	Run-time (seconds)	% of *SMARTA* run-time
SMARTA	5 475	100%	10 024	100%
SPARSE	4 756	87%	18 120	180%
TRUNC	2 167	40%	5 370	54%

Table 2: Running times for the three implementations on two specific models.

chosen for the Voting model there is only a single start state, and hence the vector $\boldsymbol{\alpha}$ contains only one non-zero entry, while for the Web-server example there are multiple start states. The passage time in the Voting model also has more target states, and because these are made absorbing to ensure only first passage times are calculated this renders a larger proportion of the state space unreachable. As shown in Figure 2, therefore, the vector $\boldsymbol{\nu}_{(k)}$ fills in much faster in the Web-server example, and hence the savings from only storing non-zeros elements of $\boldsymbol{\nu}_{(k)}$ are smaller. The solution for the Web-server model also takes more iterations than for Voting models with the same number of states, and this further increases the number of multiplications performed. Further experimentation with a range of passage time measures will be required to confirm this.

Secondly, we compare the running times of the three different implementations. Table 2 shows the time taken to conduct iterative passage time analysis for two specific models on an Intel Core2 Duo 2.66GHz. In both models 165 Laplace transform samples were calculated with a convergence precision $\varepsilon = 10^{-16}$. Note that as described in [9] *SMARTA* is a parallel program featuring a master-slave architecture; to allow comparison with the serial *SPARSE* and *TRUNC* implementations it was run with a single slave process that performed all complex multiplications itself.

Once again the performance of *SPARSE* stands out. For the Voting model it displays a lower running time than *SMARTA*, but for the Web-server model it actually takes longer than the original implementation. We believe this is due to the added overhead of accessing and maintaining the sparse representation of $\boldsymbol{\nu}_{(k)}$ over the explicit dense representation over the vector in *SMARTA*. As observed previously, $\boldsymbol{\nu}_{(k)}$ appears to fill in faster and take more iterations to converge for the Web-server model compared to the Voting model, and this added overhead of extra non-zeros outweighs the savings of not performing multiplications with vector elements which are zero. In contrast, for the Voting model the saving in multiplications is greater than the overhead of the sparse representation. These results are particularly encouraging as previous attempts to use sparse vector representations in the numerical solution of stochastic models have experienced slowdowns of up to 4 orders of magnitude [7].

We also note that *TRUNC* displays better performance than *SPARSE* and records a lower running time than the original *SMARTA* implementation for both models. It is noticeable, however, that the percentage of time saved is lower than the percentage reduction in number of complex multiplications for the same model.

Method	Voting model (1 100 000 states)		Web-server model (1 000 000) states	
	Run-time (seconds)	% of *SMARTA* run-time	Run-time (seconds)	% of *SMARTA* run-time
SMARTA	5 475	100%	10 024	100%
SPARSE-M	5 285	97%	10 282	103%
TRUNC-M	5 190	95%	9 419	94%

Table 3: Running times for the implementations which mimic sparse storage of $\boldsymbol{\nu}_{(k)}$ on two specific models.

We also compare the running times of the sparse vector storage implementations (with and without truncation) against modified versions of *SMARTA* to determine whether or not the extra overhead of the sparse representation of $\boldsymbol{\nu}_{(k)}$ outweighs the savings in the number of multiplications which must be performed. To accomplish this we implemented two variants of *SMARTA* which retained the dense vector storage of the original implementation but attempted to mimic a sparse vector storage scheme by checking each vector value prior to multiplication and only proceeding if it was non-zero. This yielded approximately the same number of complex multiplications when calculating the passage times in the two models as the sparse matrix implementations, but did not have the same overheads in managing and accessing the vector.

Table 3 contains the running times for these two additional implementations, which are denoted *SPARSE-M* and *TRUNC-M* for those which mimic *SPARSE* and *TRUNC* respectively. We no longer observe the dramatic slow-down observed in the Web-server model for *SPARSE*, but the need to check every element of the dense vector prior to multiplication clearly imposes an overhead of its own (approximately 1 clock cycle per comparison, compared with the 5 clock cycles required to perform a floating point multiplication [8]). In the case of *SPARSE-M* and the Voting model this overhead appears larger than the overhead of maintaining the sparse representation of $\boldsymbol{\nu}_{(k)}$, which can probably be attributed to relatively small number of non-zero elements in $\boldsymbol{\nu}_{(k)}$ for that model. For the Web-server model, the *SMARTA-M* implementation outperforms the *SPARSE* implementation but is still slower than *SMARTA*. When truncation is employed (in *TRUNC-M*) we see that the there is a small time saving compared to *SMARTA*, but that this saving is much smaller than in the sparse vector implementation of *TRUNC*. Clearly, therefore, the savings from storing the vector in sparse form do outweigh the extra storage and accessing overheads.

3.3 Accuracy

Comparing the first 32 decimal places of the samples of the first-passage time distributions produced by the three implementations using the Kolmogorov–Smirnov statistic, we found that in all cases the maximum absolute difference between the distributions was 0. Hence we conclude that for the examples considered our truncation technique does not appear to have a negative impact on the accuracy of the first-passage time distribution.

4 Conclusion

We have presented numerical optimisations to our previously-published iterative passage time algorithm for the calculation of passage time densities and distributions in semi-Markov processes. Through case studies of two semi-Markov models of varying sizes, we have demonstrated that it is possible to discard negligibly small values during computation, which reduces the number of multiplications that must be performed and hence the running time of the algorithm. Furthermore, our results suggest that this can be done without affecting the accuracy of the final result to at least 32 decimal places, at least for the examples considered in this paper.

In the future we will investigate further improvements to our truncation technique. We will seek to reduce its overhead and thus reduce the overall time required, with the aim that the time saving be brought more into line with the saving in the number of complex multiplications. This might be accomplished by relaxing Definition 1 or by increasing the frequency with which we remove negligibly small values from $\boldsymbol{\nu}_{(k)}$. We will also revisit our investigation in [6] of the convergence behaviour of the iterative algorithm to determine what (if any) difference truncation of negligibly small values makes. Finally, it would be interesting to attempt to improve performance further by exploiting the multiple floating point units and/or vector processing units found on modern multi-core processors.

References

[1] J. Abate, G.L. Choudhury, and W. Whitt. On the Laguerre method for numerically inverting Laplace transforms. *INFORMS Journal on Computing*, 8(4):413–427, 1996.

[2] J. Abate and W. Whitt. The Fourier-series method for inverting transforms of probability distributions. *Queueing Systems*, 10(1):5–88, 1992.

[3] J. Abate and W. Whitt. Numerical inversion of Laplace transforms of probability distributions. *ORSA Journal on Computing*, 7(1):36–43, 1995.

[4] J.T. Bradley, N.J. Dingle, P.G. Harrison, and W.J. Knottenbelt. Distributed computation of passage time quantiles and transient state distributions in large Semi-Markov models. In *Proceedings of the International Workshop on Performance Modeling, Evaluation and Optimization of Parallel and Distributed Systems (PMEO-PDS'03)*, Nice, April 26th 2003.

[5] J.T. Bradley, N.J. Dingle, P.G. Harrison, and W.J. Knottenbelt. Performance queries on semi-Markov stochastic Petri nets with an extended Continuous Stochastic Logic. In *Proceedings of 10th International Workshop on Petri Nets and Performance Models (PNPM'03)*, pages 62–71, Urbana-Champaign IL, USA, September 2nd–5th 2003.

[6] J.T. Bradley, N.J. Dingle, W.J. Knottenbelt, and H.J. Wilson. Hypergraph-based parallel computation of passage time densities in large semi-Markov models. *Linear Algebra and its Applications*, 386:311–334, 2004.

[7] P. Buchholz and P. Kemper. Compact representations of probability distributions in the analysis of superposed GSPNs. In *Proceedings of the 9th international Workshop on Petri Nets and Performance Models (PNPM'01)*, pages 81–90, Washington DC, USA, September 2001.

[8] Intel Corporation. Intel 64 and IA-32 architectures optimization reference manual. Technical Report 248966-018, March 2009.

[9] N.J. Dingle. *Parallel Computation of Response Time Densities and Quantiles in Large Markov and Semi-Markov Models*. PhD thesis, Imperial College London, United Kingdom, 2004.

[10] N.J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, Second edition, 2002.

[11] O. Lawlor, H. Govind, I. Dooley, M. Breitenfeld, and L. Kale. Performance degradation in the presence of subnormal floating-point values. In *Proceedings of the International Workshop on Operating System Interference in High Performance Applications*, September 2005.

[12] R. Pyke. Markov renewal processes: Definitions and preliminary properties. *Annals of Mathematical Statistics*, 32(4):1231–1242, December 1961.

[13] R. Pyke. Markov renewal processes with finitely many states. *Annals of Mathematical Statistics*, 32(4):1243–1259, December 1961.