Passage-time Computation and Aggregation Strategies for Large Semi-Markov Processes

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

Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2BZ, United Kingdom

Abstract

High-level semi-Markov modelling paradigms such as semi-Markov stochastic Petri nets and process algebras are used to capture realistic performance models of computer and communication systems but often have the drawback of generating huge underlying semi-Markov processes. Extraction of performance measures such as steady-state probabilities and passage-time distributions therefore relies on sparse matrix–vector operations involving very large transition matrices. Previous studies have shown that exact state-by-state aggregation of semi-Markov processes can be applied to reduce the number of states. This can, however, lead to a dramatic increase in matrix density caused by the creation of additional transitions between remaining states. Our paper addresses this issue by presenting the concept of state space partitioning for aggregation.

We present a new deterministic partitioning method which we term *barrier partitioning*. We show that barrier partitioning is capable of splitting very large semi-Markov models into a number of partitions such that first passage-time analysis can be performed more quickly and using up to 99% less memory than existing algorithms.

Key words: Semi-Markov processes, Aggregation, Passage-time analysis

1. Introduction

Semi-Markov processes (SMPs) are expressive tools for modelling a wide range of real-life systems. The state space explosion problem, however, hinders the analysis of large finite SMPs, as it does for many stochastic and functional modelling disciplines. One approach to addressing this problem is to use aggregation techniques to remove single states or groups of states and aggregate their temporal effect into the remaining states.

Many techniques exist in the Markovian domain for exact and approximate aggregation (e.g. lumpability [1], aggregation/disaggregation [2], aggregation of hierarchical models [3]). Sumito and Rieders [4] present a generalised lumpability result for SMPs, although, as with the application of Markovian lumpability, there is a strict structural condition on the underlying semi-Markov process. A form of aggregated semi-Markov process was first defined in Çinlar [5, 6] as a so-called *filtered* or *restricted process*. Until recently though, work on efficient algorithms for semi-Markov aggregation to match that on Markov processes has been very limited.

In prior work [7, 8], we present an aggregation algorithm for semi-Markov processes that realises the filtering process envisaged by Çinlar [5]. The technique operates on each state of the SMP indi-

*Corresponding author

Email addresses: mcg05@doc.ic.ac.uk (Marcel C. Guenther), njd200@doc.ic.ac.uk (Nicholas J. Dingle), jb@doc.ic.ac.uk (Jeremy T. Bradley), wjk@doc.ic.ac.uk (William J. Knottenbelt)

vidually, producing a smaller SMP which preserves passage-time distributions across the process. Our analysis in [8] suggests that the primary limitation of this technique is that the computational cost and memory requirements become very large as increasing numbers of states are aggregated and the transition matrices representing the SMP consequently get less sparse.

In this paper we introduce a new *barrier partitioning* strategy. We demonstrate how this enables passage-time analysis to be conducted in less time and using up to 99% less memory than before.

The remainder of this paper is organised as follows. Section 2 summarises background theory on the calculation of passage times in semi-Markov processes from [9]. Section 3 presents the barrier partitioning technique and evaluates the improvements in the memory and time required to analyse large semi-Markov models that barrier partitioning offers. Finally, Section 5 concludes and suggests directions for future work.

1.1. Motivation: Avoiding a Density Explosion

We consider a semi-Markov model of a voting system from [9], described in detail in Section 4. A population of voters interact with a voting system to cast their vote. A typical measure of interest is a first passage-time representing the length of time it takes for all the voters to vote once.

The underlying semi-Markov model is generated from a semi-Markov stochastic Petri net [10] model. In the particular version of the system that we use in this illustration, the semi-Markov process is 4 050 states in size with a transition matrix containing approximately 15 000 non-zeros (representing a transition matrix that is 0.1% dense).

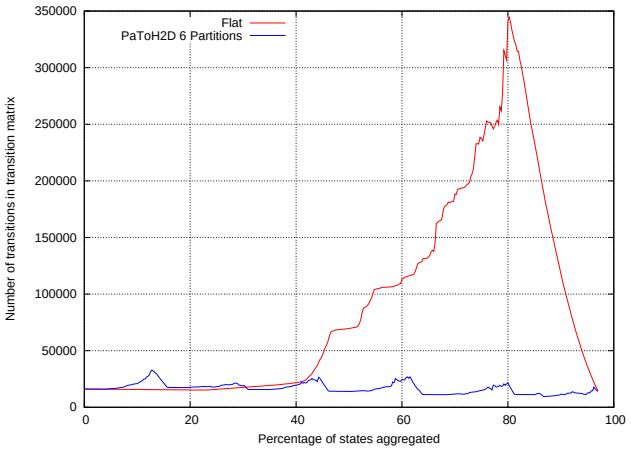

Fig. 1. The effect of partition aggregation compared to flat aggregation of the 4 050 state Voting model.

In this example, we wish to shrink the size of the semi-Markov process by state-wise aggregation, so as to highlight explicit start and end states of the passage-time measure and make the passage-time analysis faster. The complexity of sparse passage-time analysis of semi-Markov processes [9] is approximately $O(n^2 \log n)$ for an n state process, so any reduction by a factor of α in the number of states in an SMP will result in a reduction of α^2 in the time to complete a first passage-time analysis in the limit of n.

Fig. 1 shows the number of non-zeros in the transition matrix as the matrix is aggregated. The lines show the density of the transition matrix as it is being aggregated by the original state-wise algorithm outlined in [7] using two different strategies. The large upper red peak in Fig. 1 illustrates the problem encountered in applying the exact state-by-state aggregation algorithm sequentially across the *flat* state space of an SMP. The transition matrix, initially containing 15 000 non-zeros, increases to have nearly 350 000 (about 48% dense) after approximately 80% of the states have been aggregated. Therefore, the sparsity of the matrix has clearly been sacrificed even though the dimension of the matrix has been reduced dramatically. As we are potentially dealing with semi-Markov processes with many millions of states, which are by themselves memory-consuming to store and manipulate, we wish to avoid at all costs any significant increase in storage requirement that aggregation may bring.

In this paper, we investigate the idea that we can avoid this so-called density explosion by first partitioning the semi-Markov transition matrix and then aggregating the partitions one by one.

The slightly varying lower blue line in Fig. 1 shows how the non-zero density varies as the partitioned transition matrix is aggregated, one partition at a time. Each small peak in the blue line represents a small increase in transition density as each of the partitions is individually aggregated. Since the partitioning tool (in this case PaToH2D [11]) has found loosely coupled partitions, the effect of increased density on the overall system can be substantially reduced.

This is important since it is the absolute number of non-zero entries in the transition matrix that determines the storage requirement and run-time performance of our passage-time analysis algorithm.

However, the argument is not quite as straightforward as this; we cannot just arbitrarily apply a partitioning algorithm and aggregate the resulting partitions. As can be seen from Fig. 2, where we have partitioned a 106 540 state version of the Voting model using various partitioners (including METIS [12] and PaToH2D [11]), the type of partitioning and number of partitions can have a dramatic effect on the density of the transition matrix during aggregation.

In this case, a naïve row-striping partitioner produces by far the worst behaviour during aggregation, but this is not necessarily always the case. On different models, we have observed row-striping behave well while the PaToH partitioner produces the highest density.

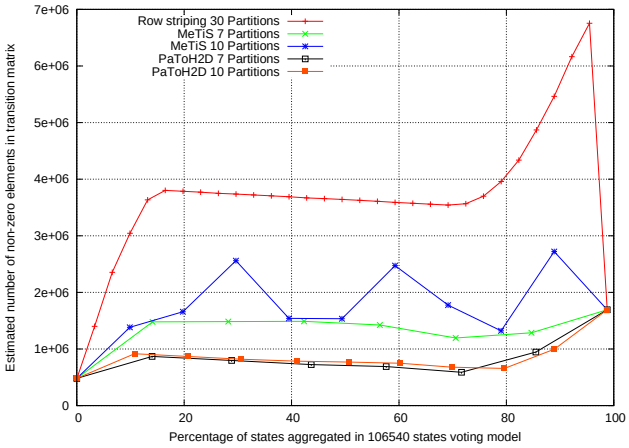

Fig. 2. The density progression of a 106 540 Voting model during aggregation, as partitioned by three types of partitioner.

Knowing in advance which partitioner to use and how many partitions to produce is not obvious and a lot of computational effort can be wasted producing such partitions fruitlessly. This paper explores a new style of *barrier* partitioning and aggregation which is tailored to passage-time analysis and produces consistently low transition matrix density behaviour during aggregation.

2. Background

2.1. Semi-Markov Processes

Semi-Markov Processes (SMPs) are an extension of Markov processes which allow for generally distributed sojourn times [13, 14]. 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.

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 $N(t) = \max\{n : T_n \leq t\}$ is 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:

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

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. Aggregate semi-Markov Processes

Numerical analysis of semi-Markov process is a computationally expensive process and any reduction in the size of the SMP can drastically lower the analysis time. Typically for first passage-time analysis, we are only explicitly interested in initial and target states of the analysis. In this section, we define an observer process to the original SMP, which focuses on an arbitrary subset of states in the original SMP, as in [5, 6]. It is a nice property of semi-Markov processes that this observer process is itself a semi-Markov process, which is something that is not true, for instance, of Markov processes unless a lumpable condition is found.

We wish to construct an aggregate process which only observes arrival instants by the original process at states in $D \subseteq \mathcal{S}$. We do so by constructing a sequence of random variables N_i that capture the ith arrival at a state in D in terms of the transition number of the original SMP.

$$\begin{aligned} N_0 &= 0 \\ N_1 &= \inf\{n > 0 \ : \ \chi_n \in D\} \\ &\vdots \\ N_i &= \inf\{n > N_{i-1} \ : \ \chi_n \in D\} \end{aligned} \tag{2}$$

We can now define the aggregate process $(\hat{\chi}_n, \hat{T}_n)$ where:

$$\hat{\chi}_n = \chi_{N_n}, \hat{T}_n = T_{N_n} \tag{3}$$

for $n > 0$. In [5], the process $(\hat{\chi}_n, \hat{T}_n)$ is called the restriction of (χ_n, T_n) to D. We assume for simplicity in this paper that our original SMP is irreducible and that $N_i < \infty$ for all i (although a straightforward extension exists if the original SMP is transient).

This is the theoretical grounding for previous aggregation algorithms on semi-Markov processes [7, 8]. In this paper, we will be looking at aggregating whole partitions of the state-space and, in these terms, constructing a restriction on the SMP that observes only entry and exit states to the partition from the remainder of the process. The actual computation of sojourn times between entry and exit states relies on an iterative passage-time algorithm, outlined below.

2.3. Iterative Passage-time Algorithm

In this section we define the first passage-time random variable used throughout the paper. We also summarise from [9] an iterative algorithm for calculating first passage-time density in semi-Markov processes.

In manipulating semi-Markov processes to extract passage-time distributions, the Laplace transforms of sojourn time distribution functions are used. This reflects the previous analysis of semi-Markov processes carried out by Pyke [13, 14], Çinlar [5, 6] and many others. One reason for this is that the Laplace transform of the convolution of two random variables is easily represented by the product of the respective Laplace transforms of the distribution functions.

However, in the case of numerical representation of general distributions and calculation of passage times in SMPs, we also work with samples of Laplace transforms. The specific sample points are dictated by the numerical Laplace transform inversion technique that is ultimately deployed (for instance the Euler technique [15]). Further details of this style of numerical representation can be found in [9].

From now on, we 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 into a non-empty set of target states $\vec{j}$ (for a stationary time-homogeneous SMP) is defined as:

$$P_{i\vec{j}} = \inf\{u > 0 : N(u) \geq M_{i\vec{j}}\} \tag{4}$$

where $M_{i\vec{j}} = \min\{m \in \mathbb{Z}^+ : \chi_m \in \vec{j} \mid \chi_0 = i\}$ is the transition count of the terminating state of the passage. $P_{i\vec{j}}$ picks the least time point at which this transition occurs, which in turn defines the passage time. This formulation of the random variable $P_{i\vec{j}}$ applies to an SMP which contains immediate transitions.[1] If such transitions are not present, then the passage time simplifies to:

$$P_{i\vec{j}} = \inf\{u > 0 : Z(u) \in \vec{j}, N(u) > 0, Z(0) = i\} \tag{5}$$

$P_{i\vec{j}}$ 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. From [9], this calculation can be achieved by solving a set of N linear equations of the form:

$$L_{i\vec{j}}(s) = \sum_{k \notin \vec{j}} r^*_{ik}(s) L_{k\vec{j}}(s) + \sum_{k \in \vec{j}} r^*_{ik}(s) \qquad : \text{for } 1 \leq i \leq N \tag{6}$$

where $r^*_{ik}(s)$ is the Laplace–Stieltjes transform (LST) of $R(i, k, t)$ of Eq. (1) and is defined by:

$$r^*_{ik}(s) = \int_0^\infty e^{-st}\, dR(i, k, t) \tag{7}$$

Eq. (6) 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. (6) yields:

$$\begin{pmatrix} 1 & -r^*_{12}(s) & \cdots & -r^*_{1N}(s) \\ 0 & 1 - r^*_{22}(s) & \cdots & -r^*_{2N}(s) \\ 0 & -r^*_{32}(s) & \cdots & -r^*_{3N}(s) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & -r^*_{N2}(s) & \cdots & 1 - r^*_{NN}(s) \end{pmatrix} \begin{pmatrix} L_{1\vec{j}}(s) \\ L_{2\vec{j}}(s) \\ L_{3\vec{j}}(s) \\ \vdots \\ L_{N\vec{j}}(s) \end{pmatrix} = \begin{pmatrix} r^*_{11}(s) \\ r^*_{21}(s) \\ r^*_{31}(s) \\ \vdots \\ r^*_{N1}(s) \end{pmatrix} \tag{8}$$

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. (5). 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 the density of $P^{(r)}_{i\vec{j}}$ by constructing and then numerically inverting its Laplace transform $L^{(r)}_{i\vec{j}}(s)$, using techniques from for example [15].

[1] Passages which terminate on a state which has no timed out-transitions are not well defined in Eq. (5) as $Z(t)$ is not well defined. $M_{i\vec{j}}$, which instead counts the number of state transitions until a target state is entered, is able to distinguish such target states. More details can be found in [9].

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:

$$P_{i\vec{j}}^{(r)} = \inf\{u > 0 : M_{i\vec{j}} \leq N(u) \leq r\} \tag{9}$$

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.

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

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

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

$$\mathbf{L}_{\vec{j}}^{(r)}(s) = \mathbf{U}(\mathbf{I} + \mathbf{U}' + \mathbf{U}'^2 + \cdots + \mathbf{U}'^{(r-1)})\,\mathbf{e}_{\vec{j}} \tag{10}$$

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. (10) is included so as to generate cycle times for cases such as $L_{ii}^{(r)}(s)$ 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. (5) and Eq. (9):

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

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:

$$\alpha_k = \begin{cases} \pi_k / \sum_{j\in\vec{i}} \pi_j & \text{if } k \in \vec{i} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

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

$$L_{\vec{i}\vec{j}}^{(r)}(s) = \boldsymbol{\alpha}\mathbf{L}_{\vec{j}}^{(r)}(s) = \sum_{k=0}^{r-1} \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^k\,\mathbf{e}_{\vec{j}} \tag{12}$$

The sum of Eq. (12) can be computed efficiently using sparse matrix–vector multiplications with a vector accumulator, $\boldsymbol{\mu}_n = \sum_{k=0}^{n} \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^k$. At each step, the accumulator, initialised as $\boldsymbol{\mu}_0 = \boldsymbol{\alpha}\mathbf{U}$, is updated as $\boldsymbol{\mu}_{n+1} = \boldsymbol{\alpha}\mathbf{U} + \boldsymbol{\mu}_n\mathbf{U}'$.

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

$$|\mathrm{Re}(L_{\vec{i}\vec{j}}^{(r+1)}(s) - L_{\vec{i}\vec{j}}^{(r)}(s))| < \varepsilon \quad \text{and} \quad |\mathrm{Im}(L_{\vec{i}\vec{j}}^{(r+1)}(s) - L_{\vec{i}\vec{j}}^{(r)}(s))| < \varepsilon \tag{13}$$

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

Further details of this technique can be found in [9] and a formal proof of convergence can be found in [16].

3. Barrier Partitioning

3.1. Atomic partition aggregation

Compared to flat state-by-state aggregation, a partition-by-partition aggregation approach reduces the transition matrix fill-in drastically. However, there is still the problem that the maximum number of transitions generated during the aggregation of a partition is much higher than the final number of transitions in the aggregated state space. Indeed, there is also the problem that the final number of non-zeros in the aggregated state space can be higher than in the initial unaggregated one. Such density peaks are undesirable because it requires a significant amount of memory to store all temporary transitions, and the fill-in also slows down the aggregation of states as we need to perform more aggregation operations to remove states when the sub-matrix of a partition becomes dense. This observation prompted us to investigate an approach inspired by first passage-time analysis which avoids these peaks by aggregating an entire partition in one go. We term this *atomic aggregation*.

The basic premise is derived from Çinlar's restricted SMP, discussed in Section 2.2. Assume we define the set of states $\vec{i}$ to be those states that have transitions into a given partition, T, and the set of states $\vec{j}$ to be those states that have predecessor states in the transition. We can effectively eliminate the partition altogether by performing a series of passage-time computations: finding $L_{ij}(s)$ for all $i \in \vec{i}, j \in \vec{j}$ and incorporating these passage-times as transition sojourn times in a new process that does not include the partition. This calculation can be accelerated by including all the start states in $\vec{i}$ in the linear system of Eq. (8), but even this improvement results in a calculation of $|\vec{j}|$ sets of $|\vec{i}| + |T|$ linear equations to obtain the necessary passage times.

3.2. Barrier Partition Aggregation

In this section we introduce a type of atomic partitioning called *barrier partitioning*, which does not require the high number of individual passage-time computations of a naïve atomic aggregation. This technique takes advantage of common features of the passage-time calculation to improve the partition quality and still permit exact passage-time analysis.

To perform first passage-time analysis on an SMP with n states we need to solve n linear equations to obtain $L_{\vec{ij}}(s)$ (see Section 2.3). We observe that first passage-time analysis can be done forward, that is, from each source state to the set of target states, as usual, as well as in the reversed process (from the set of target states to the individual source states) to get an identical result. The transition matrix of the reversed process can be obtained by transposing the forward SMP transition matrix and swapping source and target states in the passage time specification. The barrier partitioning method exploits this duality between the forward and reverse calculation of the first passage-time distribution and allows us to split the first passage-time calculation into two separate calculations. Although the overall number of passage-time iterations is the same, the combined computational cost of doing the two separate calculations is much less than the cost of the original first passage-time calculation. This is because the two separate calculations require only half the amount of memory as the original and can be performed independently and thus also in parallel.

Assume we have an SMP with a set of start states $\vec{i}$ and a set of target states $\vec{j}$. If any state is both a source and a target state, it can be split into separate source and target states, with a connecting immediate transition between them. Any in-transitions to the state become in-transitions to the target state and any out-transitions become out-transitions from the start state. This does not change any passage-based measure defined on the SMP, but we make this change prior to any partitioning. We are now in a position to define a barrier partitioning, a schematic representation of which is shown in Fig. 3.

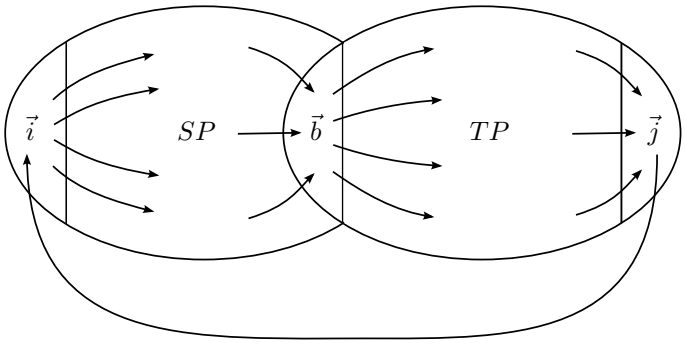

Fig. 3. A barrier partitioning, showing a set of start states $\vec{i}$ and target states $\vec{j}$ for a passage-time calculation. The remainder of the state space is split into a source partition SP, a target partition TP which contains the barrier, $\vec{b}$. Passages from SP to TP have to pass through $\vec{b}$ and cannot return, except via the target set, $\vec{j}$.

Definition 1 (Barrier partitioning)**.** A (2-way) barrier partitioning consists of state-space partitions SP and TP. SP contains all source states and a proportion of the intermediate states such that any outgoing transition from SP to TP goes through a set of barrier states $\vec{b}$ in TP. Furthermore the only outgoing transitions from states in TP to states in SP are from target states $\vec{j}$ to source states $\vec{i}$. Note that $\vec{b}$ and $\vec{j}$ may intersect.

We proceed to show that a system-wide passage-time calculation can be split into two separate and smaller passage-time calculations across the partitions created by the barrier partitioning. Assume that we can divide the state space $\mathcal{S}$ of a connected SMP graph into two partitions such that the resulting partitioning is a barrier partitioning. Clearly we have $\vec{i} \cap \vec{j} = \emptyset$, $SP \cup TP = \mathcal{S}$. We denote the set of source states as $\vec{i}$, the set of barrier states as $\vec{b}$ and the set of target states as $\vec{j}$.

In the proposition below, $L^R_{ib}(s)$ denotes a restricted first passage-time distribution from state i to state $b \in \vec{b}$, the barrier states, where all states in $\vec{b}$ are made absorbing for the calculation of $L^R_{ib}(s)$. This ensures that we only consider paths of the form $i \to k_1 \to \cdots \to k_m \to b$, with $k_r \in SP$. In other words we do not consider paths through TP for the calculation of $L^R_{ib}(s)$.

Proposition 1. The result of a first passage-time calculation from a source state i to the set of target states $\vec{j}$ is the same as the result obtained by summing the first passage-times from i to $b \in \vec{b}$ convolved with the first passage-time from the barrier state b to the set of target states $\vec{j}$. In the Laplace domain this translates to:

$$L_{i\vec{j}}(s) = \sum_{b \in \vec{b}} L^R_{ib}(s) L_{b\vec{j}}(s) \tag{14}$$

Proof. Restricting our set of equations to consider passage times from states $i \in SP$ to the target set $\vec{j}$, by Eq. (6) we have:

$$L_{i\vec{j}}(s) = \sum_{k \in (SP \cup TP) \setminus \vec{j}} r^*_{ik}(s) L_{k\vec{j}}(s) + \sum_{k \in \vec{j}} r^*_{ik}(s)$$

hence:

$$L_{i\vec{j}}(s) = \sum_{k \in (SP \cup TP)} r^*_{ik}(s) L_{k\vec{j}}(s) \tag{15}$$

where $L_{k\vec{j}}(s)$ is equal to 1 if $k \in \vec{j}$. We can rewrite $k \in SP \cup TP$ since $k \in SP \cup \vec{b}$ as there is no

transition from any state in SP to any state in $TP \backslash \vec{b}$ by construction of the barrier.

$$\begin{aligned} L_{i\vec{j}}(s) &= \sum_{k \in (SP \cup \vec{b})} r^*_{ik}(s) L_{k\vec{j}}(s) \\ &= \sum_{b \in \vec{b}} r^*_{ib}(s) L_{b\vec{j}}(s) + \sum_{k \in SP} r^*_{ik}(s) L_{k\vec{j}}(s) \end{aligned} \tag{16}$$

also by construction of the barrier partitioning and the fact that target states are absorbing states we know that once we have entered TP (i.e. reached a state in $\vec{b}$) we cannot find a path back to a state in SP. Hence:

$$\begin{aligned} L_{i\vec{j}}(s) &= \sum_{b \in \vec{b}} r^*_{ib}(s) L_{b\vec{j}}(s) + \sum_{k \in SP} r^*_{ik}(s) \sum_{b \in \vec{b}} L^R_{kb}(s) L_{b\vec{j}}(s) \\ &= \sum_{b \in \vec{b}} \left[\left(\sum_{k \in SP} r^*_{ik}(s) L^R_{kb}(s) + r^*_{ib}(s) \right) L_{b\vec{j}}(s) \right] \end{aligned} \tag{17}$$

by definition $\sum_{k \in SP} r^*_{ik}(s) L^R_{kb}(s) + r^*_{ib}(s)$ is the restricted first-passage time from state i to barrier state b. Therefore:

$$L_{i\vec{j}}(s) = \sum_{b \in \vec{b}} L^R_{ib}(s) L_{b\vec{j}}(s) \tag{18}$$

■

The following result demonstrates the separability of the passage-time calculation, an aspect that facilitates *divide-and-conquer* parallel computation. We will also need the following result to ease the extension to the k-way partition. We define $L^R_{i\vec{j}}(s)$ to be the passage time from i to $\vec{j}$ restricted by making all the states in $\vec{j}$ absorbing, a natural extension of $L^R_{ij}(s)$, defined earlier.

Corollary 1.1.

$$L^R_{i\vec{j}}(s) = \sum_{b \in \vec{b}} L^R_{ib}(s) L^R_{b\vec{j}}(s) \tag{19}$$

Proof. We have, for all states b in the barrier set, $\vec{b}$:

$$L^R_{b\vec{j}}(s) = L_{b\vec{j}}(s) \tag{20}$$

since target states are absorbing states by assumption and because none of the outgoing transitions of non-target barrier states go into SP. Furthermore:

$$L^R_{i\vec{j}}(s) = L_{i\vec{j}}(s) \tag{21}$$

as the restricted first passage-time distribution on the entire state space is by definition also the standard passage-time distribution. The result follows from Eq. (18). ■

We can similarly extend this separable result to cover passage-times from multiple sources states, $\vec{i}$, to multiple target states, $\vec{j}$. Let $L^R_{\vec{i}\vec{b}}(s) = \{L^R_{\vec{i}b_1}(s), \ldots, L^R_{\vec{i}b_l}(s)\}$, where $L^R_{\vec{i}b_m}(s) = \{\alpha_1 L^R_{i_1 b_m}(s) + \cdots + \alpha_l L^R_{i_l b_m}(s)\}$ and $L_{\vec{b}\vec{j}}(s) = \{L_{b_1\vec{j}}(s), \ldots, L_{b_l\vec{j}}(s)\}$.

Corollary 1.2. In steady-state, we have:

$$L_{\vec{i}\vec{j}}(s) = \sum_{b \in \vec{b}} L^R_{\vec{i}b}(s) L_{b\vec{j}}(s) = L^R_{\vec{i}\vec{b}}(s) \cdot L_{\vec{b}\vec{j}}(s) \tag{22}$$

Proof. Let $\alpha_1, \alpha_2, \ldots, \alpha_l$ be the normalised steady-state probabilities of the source states $\vec{i} = (i_1, i_2, \ldots, i_l)$ as defined in Eq. (11). By Eq. (12) we have:

$$\begin{aligned} L_{\vec{i}\vec{j}}(s) &= \alpha_1 L_{i_1\vec{j}}(s) + \alpha_2 L_{i_2\vec{j}}(s) + \cdots + \alpha_l L_{i_l\vec{j}}(s) \\ &= \sum_{b\in\vec{b}} \left(\alpha_1 \left(L^R_{i_1 b}(s) L_{b\vec{j}}(s) \right) + \cdots + \alpha_l \left(L^R_{i_l b}(s) L_{b\vec{j}}(s) \right) \right) \\ &= \sum_{b\in\vec{b}} \left(\alpha_1 L^R_{i_1 b}(s) + \cdots + \alpha_l L^R_{i_l b}(s) \right) L_{b\vec{j}}(s) \\ &= L^R_{\vec{i}\vec{b}}(s) \cdot L_{\vec{b}\vec{j}}(s) \end{aligned}$$

■

3.3. Barrier Partitioning in Practice

To compute the first passage-time distribution of a model whose state space has been split into partitions SP and TP, we start by calculating $L_{\vec{i}\vec{b}}(s)$ using iterative first passage-time calculation. For this the source states remain unmodified, but the barrier states become absorbing target states. Also as this calculation is part of the final first passage-time calculation we need to weight the source states by their normalised steady state probabilities. Having calculated $L_{\vec{i}\vec{b}}(s)$ we use it as our $\boldsymbol{\mu}_0$ (see Section 2.3) in the subsequent first passage-time calculation from the set of barrier states $\vec{b}$ to the set of target states $\vec{j}$.

This technique reduces the amount of memory that we need for a first passage-time calculation as we only have to keep either the sub-matrix of the source partition or the target partition in memory at any point in time. Another advantage of barrier partitioning is that we can easily find barrier partitions in large models at low cost.

Algorithm 1 describes a general method for finding balanced barrier partitionings given a transition matrix of a semi-Markov or Markov model. Firstly, since we are doing first passage-time analysis we can discard the outgoing transitions from all target states. Secondly, we explore the entire state space using breadth-first search, with all source states being at the root level of the search. We store the resulting order in an array. To find a barrier partitioning we first add all non-target states among the first m states in the array to our source partition. Note that m has to be larger than the number of source states in the SMP. We then create a list of all predecessor states of the resulting partition. In the next step we add all predecessor states in the list to the source partition and recompute the list of predecessor states. We repeat this until we have found a source partition with no predecessor states. Since we discarded all outgoing edges of the target states, this method must give us a barrier partitioning. In the worst case this partitioning has all source and intermediate states in SP, while TP only contains the set of target states.

3.4. k-way Barrier Partitioning

The idea of barrier partitioning described in the previous section is a huge improvement to the straightforward passage-time calculation, as it reduces the amount of memory needed for the passage-time computation while introducing very little overhead. In this section we investigate the idea of *k-way barrier partitioning*. In practice a k-way barrier partitioning is desirable since it allows us to reduce the amount of memory needed to perform passage-time analysis on Markov and semi-Markov models by even more than 50%.

Definition 2 (k-way Barrier Partitioning)**.** In a k-way barrier partitioning, partition P_0 contains the source states, partition T the target states. There are $k-2$ intermediate partitions and $k-1$ barriers in total. In general partition P_m is sandwiched between its predecessor partition P_{m-1}

```
1  Define Make target states absorbing(matrix, target states);
2  Define Find breath-first ordering(matrix, source states);
3  Define Get number of rows(matrix);
4  Define Get first m non-target states(array, stopIndex);
5  Define Get predecessor states(matrix, partition, target states);
6  Define Merge arrays(array, array);
7  Define Count number of transitions(matrix, optional array);

   input  : Sparse SMP transition row matrix matrix, source states s⃗, target states t⃗
   output: Barrier source partition

8  Make target states absorbing(matrix, t⃗);
9  bforder = Find breath-first ordering(matrix, s⃗);
10 numSourceStates = |s⃗|;
11 numStates = Get number of rows(matrix);
12 m = numStates / 2;
13 mStep = numStates / 4;
14 partition = ∅;
15 foundBalancedBarrierPartitioning = false;
16 while foundBalancedBarrierPartitioning == false && mStep > 1 do
17 |  partition = Get first m non-target states(bforder, m);
18 |  predecessors = Get predecessor states(matrix, partition, t⃗);
19 |  while predecessors is not empty do
20 |  |  partition = Merge arrays(partition,predecessors);
21 |  |  predecessors = Get predecessor states(matrix, partition, t⃗);
22 |  end
23 |  SPTPBalance = Count number of transitions(matrix,partition) / Count number of
   |  transitions(matrix);
24 |  if SPTPBalance < 0.45 then
25 |  |  m += mStep;
26 |  end
27 |  else if SPTPBalance > 0.55 then
28 |  |  m -= mStep;
29 |  end
30 |  else
31 |  |  foundBalancedBarrierPartitioning = true;
32 |  |  break;
33 |  end
34 |  mStep = mStep / 2;
35 |  partition = ∅;
36 end
37 return partition;
```

Algorithm 1: Automatic balanced barrier partitioning

and its successor partitions P_{m+1} and T. There are no transitions from partition P_n to P_m if $n > m$, hence the barrier property is satisfied in the sense that once we have reached P_m the only way to get back to any state in P_{m-1} is to go through T. T is the only predecessor partition of P_0. The barrier states of partition P_m are the union of T and the states of P_{m+1} that have incoming transitions from states in P_m.

Definition 2, shown in Fig. 4, generalises Definition 1. The latter definition corresponds to a 2-way barrier partitioning. In Definition 1 we did not define the set of barrier states to be the union of states that separate SP from TP and the set of states in T. However, this generalisation has no impact on Proposition 1 as we assumed that B and T may intersect.

The difference between the standard 2-way barrier partitioning and the general k-way barrier partitioning with $k > 2$ is the way we compute the passage time on the transition matrix of a model that has been partitioned into k barrier partitions. The following proposition verifies the correctness of the passage-time analysis on a k-way barrier partitioning. In the proposition below, m_i is the size of the ith barrier set and we drop the s-parameter from the Laplace transforms for brevity.

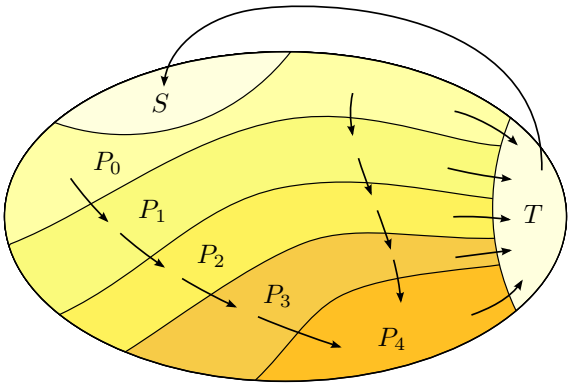

Fig. 4. A schematic representation of a k-way barrier partition

As before, we now look to construct the system-wide passage time in terms of the smaller inter-barrier passage time calculations. In Proposition 2, $L^R_{i\vec{b}_1}$ is the $1 \times m_1$ row vector containing the Laplace transforms of the restricted passage-time analysis from start state i to the states in the first barrier $\vec{b}_1$. $L^R_{\vec{b}_{k-1}\vec{j}}$ is a $m_{k-1} \times 1$ column vector of the Laplace transforms of the passage time from the states in the $(k-1)$th barrier to the set of target states $\vec{j}$ and:

$$M^R_{\vec{b}_{n-1}\vec{b}_n} = \begin{pmatrix} L^R_{\vec{b}_{n-1,1}\vec{b}_n} \\ L^R_{\vec{b}_{n-1,2}\vec{b}_n} \\ \vdots \\ L^R_{\vec{b}_{n-1,m_{n-1}}\vec{b}_n} \end{pmatrix} = \begin{pmatrix} L^R_{\vec{b}_{n-1,1}\vec{b}_{n,1}} & \cdots & L^R_{\vec{b}_{n-1,1}\vec{b}_{n,m_n}} \\ \vdots & & \vdots \\ L^R_{\vec{b}_{n-1,m_{n-1}}\vec{b}_{n,1}} & \cdots & L^R_{\vec{b}_{n-1,m_{n-1}}\vec{b}_{n,m_n}} \end{pmatrix}$$

$m_{n-1} \times m_n$ matrix containing the Laplace transform samples from the restricted passage-time analysis from barrier $n-1$ to barrier n for each pair of barrier states, i.e. pairs (a, b) where a lies in barrier $n-1$ and b in barrier n. Note that if state k is a target state then $L^R_{\vec{b}_{n-1,k}\vec{b}_{n,k}} = 1$ and $L^R_{\vec{b}_{n-1,k}\vec{b}_{n,l}} = 0$ for all $l \neq k$ as k must be an absorbing state.

Proposition 2. We can compute the aggregate passage-time distribution as the product of the inter-barrier passage times as follows:

$$L_{i\vec{j}} = L^R_{i\vec{b}_1} \; M^R_{\vec{b}_1\vec{b}_2} \; \cdots \; M^R_{\vec{b}_{k-2}\vec{b}_{k-1}} \; L^R_{\vec{b}_{k-1}\vec{j}} \tag{23}$$

Proof. First we show that:

$$L^R_{i\vec{b}_2} = L^R_{i\vec{b}_1} \; M^R_{\vec{b}_1\vec{b}_2}$$

by Corollary 1.1 we have

$$L^R_{ib_{2,n}} = \sum_{l=1}^{m_1} L^R_{ib_{1,l}} L^R_{b_{1,l}b_{2,n}}$$

then

$$\begin{aligned} L^R_{i,\vec{b}_2} &= \left(\sum_{l=1}^{m_1} L^R_{ib_{1,l}} L^R_{b_{1,l}b_{2,1}}, \; \cdots, \sum_{l=1}^{m_1} L^R_{ib_{1,l}} L^R_{b_{1,l}b_{2,m_2}} \right) \\ &= L^R_{i\vec{b}_1} \; M^R_{\vec{b}_1\vec{b}_2} \end{aligned} \tag{24}$$

using this argument repeatedly reduces Eq. (23) to:

$$\begin{aligned} L_{i\vec{j}} &= L^R_{i\vec{b}_{k-1}} L^R_{\vec{b}_{k-1}\vec{j}} \\ &= \sum_{l=1}^{m_{k-1}} \left(L^R_{ib_{k-1,l}} L^R_{b_{k-1,l}\vec{j}} \right) \end{aligned} \tag{25}$$

which holds by Proposition 1 since

$$L^R_{b_{k-1,l}\vec{j}} = L_{b_{k-1,l}\vec{j}}$$

as target states are absorbing states during first passage-time analysis. ■

Corollary 2.1.

$$L_{\vec{i}\vec{j}} = L^R_{\vec{i}\vec{b}_1} \; M^R_{\vec{b}_1\vec{b}_2} \; \cdots \; M^R_{\vec{b}_{k-2}\vec{b}_{k-1}} \; L^R_{\vec{b}_{k-1}\vec{j}}$$

Proof. Similar argument as in Corollary 1.2. ■

We now describe how sequential passage-time analysis can be performed on a k-way barrier partitioning. The basic idea is to initialise $\boldsymbol{\mu}_0^{(0)}$ (see Section 2.3) with the α-weighted source states, compute $L^R_{\vec{i}\vec{b}_1} = \boldsymbol{\mu}_0^{(1)}$ using $\boldsymbol{\mu}_0^{(0)}$ and subsequently use $\boldsymbol{\mu}_0^{(1)}$ as the new start vector for the calculation of $L^R_{\vec{i}\vec{b}_2} = \boldsymbol{\mu}_0^{(2)}$ until we obtain $\mathbf{L}_{\vec{j}} = \boldsymbol{\mu}_0^{(k)}$ (see Section 2.3). $L_{\vec{i}\vec{j}}(s)$ is computed by summing the Laplace transforms which make up this vector as in Eq. (10).

Intuitively this approach makes sense because $\boldsymbol{\mu}_0^{(n)}$ always contains the Laplace transform distribution from the initial set of source states to the states of the nth barrier, and when this is used as the start vector for the next iterative restricted passage-time analysis we obtain the Laplace transform of the distribution from the set of source states to all states that lie in the nth partition and the states of the $(n+1)$th barrier.

3.5. Constructing a k-way Barrier Partitioning

There are various ways of creating k-way barrier partitionings for SMPs. One way is recursive bi-partitioning to split sub-partitions into two balanced barrier partitions at each step. Alternatively we can modify our barrier partition algorithm to obtain the maximum number of barriers for a given transition matrix. The modified partitioner works as follows. First we make all target states absorbing. We then add the source states and all their predecessor states to the first partition. Subsequently we add the predecessor states of the predecessor states of the source states to the partition and so on. Once we have no more predecessor states we have found the first partition. The non-target successor states, i.e. non-target barrier states, of that partition are then used to construct the second partition in the same manner. However, we now only consider those predecessor states of the non-target barrier states that have not been explored yet, i.e. those that have not been assigned to any partition. We continue partitioning the state space until all states have been assigned to a partition.

We claim that this partitioning approach yields the maximum number of barrier partitions for a given transition graph as we only include the minimum number of states in every barrier partition. We call this a k_{max}-way barrier partitioning, but we will also refer to it as a *max-way barrier partitioning*. Suppose k_{max}-way partitioning does not yield the maximum number of partitions. Then it must be possible to join N adjacent barrier partitions in the k_{max}-way partitioning and split them into $N+1$ barrier partitions where $N \geq 2$ is minimal. Let the predecessor partition of the joint partition of these N partitions be partition P and the successor partition be partition S. Now if we use the successor states of partition P as the seed states to create the first of the $N+1$ partitions out of the joint partition then this partition is exactly the same as it was before

the merger and hence it must be possible to split the joint partition made out of $N-1$ partitions into N partitions. But this is not possible as N was chosen to be minimal. Hence the seed states of the successor partition R of P have to be changed so that R has a different set of seed states than it had in the original k_{max}-way partitioning. For this to be true, states are added or taken away from the seed set of R.

If we add states to the set of seed states of R then partition R must contain at least the same amount of states which it had in the original k_{max}-way partitioning and R will also have at least the same amount of successor states as it did in the k_{max}-way partitioning. The successor partition of R thus covers at least the same set of states that it covered in the original k_{max}-way partitioning. Similarly for all other successor partitions and hence we cannot generate more than N partitions from the joint partition.

So in order to create $N+1$ partitions we need to take away states in the set of seed states of R. However the seed states of R in the k_{max}-way partitioning only contains states that are non-target successor states of P and thus we cannot take away states from the seed set without violating the barrier property of the partition. This argument holds all the way down to the source partition which also contains the minimum number of seed states, namely the source states. Hence it is not possible to split N adjacent barrier partitions into $N+1$.

Note that from the max-way partitioning we can generate any k-way partitioning with $k < k_{max}$ since joining two neighbouring barrier partitions creates a new larger barrier partition. The k_{max}-way barrier partitioning also minimises the maximum partition size among the barrier partitionings.

4. Evaluation

We demonstrate the applicability of our barrier partitioning method to large semi-Markov models. We discuss the complexity of computing the partitioning itself and compare the run-times of passage-time analysis on aggregated and unaggregated models.

4.1. Semi-Markov Case Studies

Semi-Markov processes are particularly useful for capturing realistic generally-distributed sojourn times in software and hardware models. However, where general distributions are concurrently enabled, a probabilistic selection is used to choose which is executed. This is useful, for example, where a concurrent program is running on a single-threaded architecture, e.g. a Java Runtime Environment on a single core processor. The task switching between the threads can be captured by probabilistic selection of generally-distributed activities.

A second use for semi-Markov models is in the modelling of mutual exclusion, e.g. failure–recovery modes, where normal concurrent operation is suspended and a single recovery process is repairing the system. This recognises the fact that the modeller will still want to have proper Markovian concurrency at their disposal. Markov processes are a strict subset of SMPs and thus it is possible to let the modeller have true Markovian or phase-type concurrency when there are no general distributions enabled.

The evaluation of our barrier partitioning approach is discussed in the context of two semi-Markov models of concurrent systems. Both models are represented in a high-level Semi-Markov Stochastic Petri Net (SM-SPN) [10] form, from which semi-Markov processes of varying sizes can easily be generated. Further details of this formalism can be found in [10, 17].

The Voting model is a model of a distributed voting system with CC voters, MM failure-prone voting booths and NN failure-prone central servers [18, 9]. Voting agents vote asynchronously,

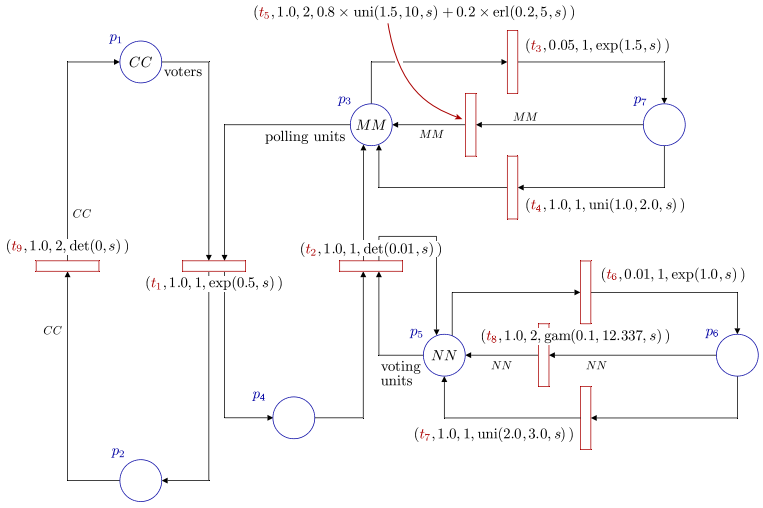

Fig. 5. The Voting Model SM-SPN [18].

moving from place p_1 to p_2 as they do so. A restricted number of polling units which receive their votes transit t_1 from place p_3 to place p_4. At t_2, the vote is registered with as many central voting units as are currently operational in p_5. Both polling units and central voting units can suffer breakdowns, represented by transitions t_3 and t_6, from which there are soft recovery mechanisms, t_4 and t_7. The system is considered to be in a failure mode if either all the polling units have failed and are in p_7 or all the central voting units have failed and are in p_6. If either of these complete failures occur, then a high priority repair is performed, which resets the failed units to a fully operational state. If some (but not all) the polling or voting units fail, they attempt self-recovery. The system will continue to function as long as at least one polling unit and one voting unit remain operational.

The Web-server model in Fig. 6 represents an SM-SPN model of a web server with RR clients (readers), WW web content authors (writers), SS parallel web servers and a write-buffer of BB in size [9]. Clients can make read requests to one of the web servers for content (represented by the movement of tokens from p_8 to p_7). Web content authors submit page updates into the write buffer (represented by the movement of tokens from p_1 onto p_2 and p_4), and whenever there are no outstanding read requests all outstanding write requests in the buffer (represented by tokens on p_4) are applied to all functioning web servers (represented by tokens on p_6). Web servers can fail (represented by the movement of tokens from p_6 to p_5) and institute self-recovery unless all servers fail, in which case a high-priority recovery mode is initiated to restore all servers to a fully functional state. Complete reads and updates are represented by tokens on p_9 and p_2 respectively.

4.2. Complexity of Barrier partitioning

In both the Voting and the Web-server model it is possible to compute a 2-way barrier partitioning such that each partition contains roughly 50% of the total number of transitions. Even more surprisingly, we easily found balanced partitions (those where SP and TP contain a similar number of transitions) for large versions of the Voting and Web-server models with several million transitions. In addition our barrier partitioning algorithm is very fast. The computation of a balanced barrier partitioning for the 1 100 000 state Voting model takes less than 10 seconds on an Intel Core2 Duo machine with two 1.8GHz processors and 1Gbyte of RAM. By comparison, the computation of a 2-way partitioning with the PaToH2D hypergraph partitioner takes about 60 seconds on the same machine, but the resulting partitioning is not suitable for atomic aggregation as both partitions have large numbers of predecessor and successor states.

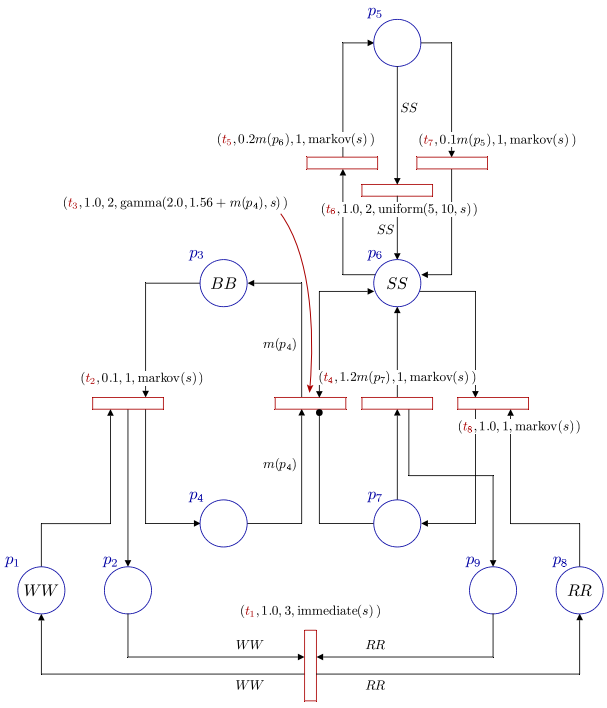

Fig. 6. The Web-server Model SM-SPN [9].

We also tested the k_{max}-way partitioning method on the 1 100 000 state Voting model and the 1 000 000 state Web-server model. In the Voting model we found a 349-way barrier partitioning, whose largest partition contains only 0.6% of the total number of transitions. In the Web-server model a 332-way barrier partitioning exists in which the largest partition contains about 0.5% of the total number of transitions. For both models it is thus possible to compute the exact first-passage time while saving 99% of the memory needed by the standard iterative passage-time analysis that works on the unpartitioned transition matrix. This is because of the fact that our k-way barrier partitioning algorithm only ever has to hold the matrix elements of one single partition in memory.

The general k_{max}-way barrier partitioning method is very fast. For the 1 100 000 state Voting model the max-way barrier partitioner needs 72 seconds on an Intel P4 3GHz with 4Gbyte of RAM to find the barrier partitioning with the maximum number of partitions. In the 1 000 000 state Web-server model the partitioner takes 35 seconds to find the max-way barrier partitioning.

The complexity of barrier partitioning is linear in the number of state transitions, and our results bear this out as the Voting model has about twice as many transitions as the Web-server model. Hence barrier partitioning does not only allow us to save an enormous amount of memory during passage-time analysis but also the partitioning method itself has a much lower complexity than, for instance, graph and hypergraph partitioners. The computation of a 2-way partitioning with the PaToH2D hypergraph partitioner takes about 60 seconds on the same machine for the Web-server model, but the resulting partitioning is not even suitable for atomic aggregation.

Another important thing to note is that the partitioner is very memory efficient as we never have to hold the entire matrix in memory during the partitioning process. A disk-based partitioning approach is also feasible as we only have to scan every transition twice: once when we look for the predecessor states of a state and a second time when we look for its successor states. This is a huge advantage compared to our 2-way barrier partitioning algorithm, for which a disk-based

solution is less feasible, since we need to scan large parts of the matrix multiple times in order to create two balanced partitionings.

4.3. Run-times and Accuracy

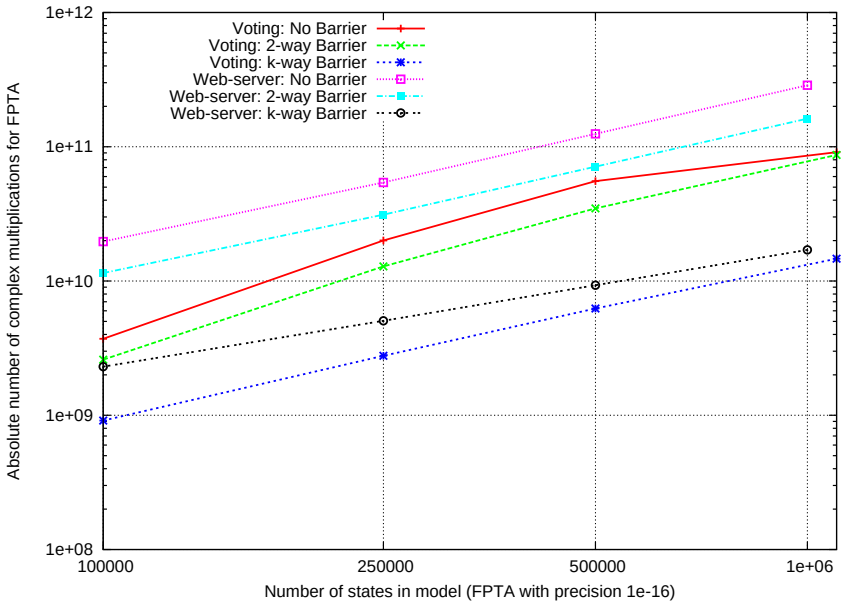

Fig. 7. Log–log comparison of the absolute number of multiplications required under different barrier aggregation strategies in the Voting and Web-server models.

The log–log plot in Fig. 7 compares the number of complex multiplications needed for our different aggregation methods to calculate the 165 Laplace transform samples required to compute 5 t-points that are representative of the distribution. It is interesting to observe that the Barrier methods generally seem to require fewer complex multiplications than the NoBarrier method in both models.

Secondly, we compare the running times of first passage-time calculations under different barrier partitionings. Tab. 1 shows the times taken to barrier partition and analyse two specific models on an Intel Core2 Duo 2.66GHz. In the Voting model the k_{max}-way barrier partitioning was a 349-way partitioning, while in the Web-server model it was a 332-way partitioning. In both cases 165 Laplace transform samples were calculated with a convergence precision $\varepsilon = 10^{-16}$. The results show that the k_{max}-way barrier approach is faster than both the unpartitioned and 2-way barrier approaches in both models investigated. In the Web-server model, the k_{max}-way barrier passage-time analyser is nearly ten times faster than the unpartitioned solver, while in the Voting model it is approximately two-and-a-half times faster. 40-way partitioning is slightly faster than k_{max}-way partitioning in these models because the smaller number of barriers results in a lower overhead in the construction of lookup tables for each barrier.

An important consideration is the effect that barrier partitioning has on the accuracy of the final passage-time result. The final column in Tab. 1 compares the first 32 decimal places of the samples of the first passage-time distributions produced under the various aggregations using the Kolmogorov–Smirnov (K–S) statistic (maximum absolute difference) against the corresponding

Method	Voting model (1 100 000 states)		
	Complex mults.	Run time (s)	K–S error
No Barrier	90 953 967 754	6 400	0
2-way Barrier	87 544 776 992	6 706	2.32602e-13
40-way Barrier	23 085 035 695	2 062	1.77547e-12
k_{max}-way Barrier	14 675 308 020	2 447	1.00372e-11
Method	Web-server model (1 000 000 states)		
	Complex mults.	Run time (s)	K-S error
No Barrier	287 181 545 505	26 921	0
2-way Barrier	160 559 878 808	16 230	2.63041e-13
40-way Barrier	29 768 374 425	2 635	1.25518e-12
k_{max}-way Barrier	17 070 767 235	2 722	1.48844e-12

Tab. 1. Computational cost, run-times and accuracy for partitioning and subsequent first passage-time analysis for two different models with varying number of barriers.

results from the unaggregated model (the No Barrier case). We conclude that, for these examples, there is negligible loss of accuracy, even with the largest number of partitions.

4.4. Very Large SMPs

We now compare the run time of the barrier-partitioned iterative passage-time analysis with that of the parallel implementation of the iterative algorithm previously presented in [9, 17] for very large SMPs.

The parallel scheme was implemented in the Semi-Markov Response Time Analyser (SMARTA) [17]. The SMARTA results presented here were produced on a Beowulf Linux cluster with 64 dual-processor nodes. Each node has two Intel Xeon 2.0GHz processors and 2GB of RAM. The nodes are connected by a Myrinet network with a peak throughput of 2Gbps. The barrier partitioning and analysis was executed on one core of a machine with a four-core AMD Opteron 1.9GHz processor and 32GB of RAM.

For the 10 991 440 state Voting model, the passage-time distribution was calculated at 31 values of t and this required $L_{\vec{ij}}(s)$ to be evaluated at 1 023 s-points. Using SMARTA this took 15 hours and 7 minutes on 64 processors, for a total cost of just over 455 processor-hours. This excludes the time taken to partition the state space using the ParMETIS parallel graph partitioning library [19] prior to computation. In contrast, it took 3 hours and 12 minutes to calculate a 599-way barrier partition of the same model and a further 4 days and 32 minutes to solve for the required distribution t-points on a single processor, for a total cost of just over 99 processor-hours. With barrier partitioning, therefore, the solution time was approximately 6.5 times longer than that of SMARTA but required only one sixty-fourth of the number of processors and cost approximately 4.5 times less in processor-hours. The maximum absolute difference between calculated passage-time distribution results was 3×10^{-6}.

5. Conclusion

In this paper we have presented barrier partitioning, which deterministically partitions the SMP's state space into a number partitions and allows first passage-time analysis to be conducted saving up to 99% of the memory required for the unaggregated SMP. Our results show that it also saves a considerable amount of time compared with the calculation of results on the unpartitioned state space. We have demonstrated that this can be achieved on SMPs with up to 10.9 million states. Barrier partitioning is suitable for population-based passage-time definitions in SMPs of models

with large populations of similarly operating cooperating components. In the case of the models presented here, the passage times were defined in terms of a percentage of the population of components having evolved to a completed state. The progression of component populations was an explicit feature of the Web-server and Voting models and is often a common feature of large SMP models, derived from higher-level formalisms. However, where this is not an explicit feature, modellers can still encode global passage-time questions by embedding counting sub-models in Petri nets and process algebras (for example, by using stochastic probes [20]). These augmented models would in turn be amenable to passage-time analysis via barrier partitioning.

For the future, we would like to explore to extent to which k-way passage-time computation can be conducted in parallel. Recall from Section 3 that passage-time calculations can be conducted in both the forward and reverse directions. For the 2-way barrier case, this suggests a simple parallelisation scheme where one machine calculates $L^R_{\vec{i}\vec{b}}(s)$ and the other $L_{\vec{j}\vec{b}}(s)$, with the final result calculated according to Corollary 1.2. We cannot simply extend this to the use of k machines in the k-way case, however, as calculation of the Laplace transforms of passage-time distributions across the $(n+1)$th partition (except for the source and target partitions) requires the Laplace transform of the passage-time across the previous nth partition as its starting point. Instead we envisage the use of two groups of machines, each performing passage-time analysis in parallel. One group does the forward passage-time calculation starting from the start states, the other one does the reverse passage-time calculation starting from the target states. Just as in the 2-way barrier case, the two groups of processors will stop when they have reached the middle barrier. This would have the advantage of being able to deal with very large partitions whose state spaces could not be held within the memory of a single machine; such partitions could arise from the analysis of extremely large SMPs with global state spaces of perhaps billions or even trillions of states.

Acknowledgements

We would like to acknowledge the very diligent efforts of our three anonymous referees, who helped us substantially improve the paper.

Bradley and Knottenbelt are supported in part by the EPSRC on the Analysis of Massively Parallel Stochastic Systems (AMPS) project, EP/G011737/1. `http://aesop.doc.ic.ac.uk/projects/amps/`. Dingle is supported by the EPSRC on the Intelligent Performance Optimisation of Virtualised Data Storage Systems (iPODS) project, EP/F010192/1. `http://aesop.doc.ic.ac.uk/projects/ipods/`.

References

[1] J. G. Kemeny and J. L. Snell, *Finite Markov Chains*. Van Nostrand, 1960.

[2] W.-L. Cao and W. J. Stewart, "Iterative aggregation/disaggregation techniques for nearly uncoupled Markov chains," *Journal of the ACM*, vol. 32, pp. 702–719, July 1985.

[3] P. Buchholz, "Hierarchical Markovian models: Symmetries and aggregation," *Performance Evaluation*, vol. 22, pp. 93–110, 1995.

[4] U. Sumita and M. Rieders, "First passage times and lumpability of semi-Markov processes," *Journal of Applied Probability*, vol. 25, pp. 675–687, Dec. 1988.

[5] E. Çinlar, "Markov renewal theory," *Advances in Applied Probability*, vol. 1, no. 2, pp. 123–187, 1969.

[6] E. Çinlar, "Markov renewal theory: A survey," *Management Science*, vol. 21, pp. 727–752, Mar. 1975.

[7] J. T. Bradley, "A passage-time preserving equivalence for semi-Markov processes," in *Lecture Notes in Computer Science 2324: Proceedings of the 12th International Conference on Modelling, Techniques and Tools (TOOLS'02)*, (London), pp. 178–187, Springer-Verlag, April 14th–17th 2002.

[8] J. T. Bradley, N. J. Dingle, and W. J. Knottenbelt, "Strategies for exact iterative aggregation of semi-Markov performance models," in *Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'03)*, (Montreal, Canada), pp. 755–762, July 20th–24th 2003.

[9] 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*, vol. 386, pp. 311–334, 2004.

[10] 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)*, (Urbana-Champaign IL, USA), pp. 62–71, September 2nd–5th 2003.

[11] U. V. Catalyürek and C. Aykanat, "PaToH: A multilevel hypergraph partitioning tool," Tech. Rep. BU-CE-9915, Version 3.0, Department of Computer Engineering, Bikent University, Ankara, 06800, Turkey, 1999.

[12] G. Karypis and V. Kumar, *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0*. University of Minnesota, September 1998.

[13] R. Pyke, "Markov renewal processes: Definitions and preliminary properties," *Annals of Mathematical Statistics*, vol. 32, pp. 1231–1242, December 1961.

[14] R. Pyke, "Markov renewal processes with finitely many states," *Annals of Mathematical Statistics*, vol. 32, pp. 1243–1259, December 1961.

[15] J. Abate and W. Whitt, "The Fourier-series method for inverting transforms of probability distributions," *Queueing Systems*, vol. 10, no. 1, pp. 5–88, 1992.

[16] J. T. Bradley and H. J. Wilson, "Convergence and correctness of an iterative scheme for calculating passage times in semi-Markov processes," *Performance Evaluation*, vol. 60, pp. 237–254, May 2005.

[17] 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.

[18] 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), p. 281, April 2003.

[19] G. Karypis, K. Schloegel, and V. Kumar, *ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 2.0*. University of Minnesota, September 1998.

[20] A. Argent-Katwala, J. T. Bradley, and N. J. Dingle, "Expressing performance requirements using regular expressions to specify stochastic probes over process algebra models," in *Proceedings of the 4th International Workshop on Software and Performance (WOSP'04)*, (Redwood Shores CA, USA), pp. 49–58, January 14th–16th 2004.