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.

Aggregation of partitions can be done in one of two ways. The first is to use exact state-by-state aggregation to aggregate each individual state within a partition. However, we discover that this approach still causes matrix density problems, albeit on a much smaller scale compared to non-partition aggregation. A second approach to the aggregation of partitions, and the one presented in this paper, is *atomic partition aggregation*. Inspired by a technique used in passage-time analysis, this collapses a whole partition into a small number of semi-Markov states and transitions.

Most partitionings produced by existing graph partitioners are not suitable for use with our atomic partition aggregation techniques, and we therefore 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 of 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 [17], aggregation/disaggregation [11], aggregation of hierarchical models [10]) but to date analogous work on semi-Markov aggregation algorithms

*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)

has been very limited. In prior work [5, 8], we presented an aggregation algorithm for semi-Markov processes which operates on each state individually. 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 gets less sparse.

In this paper, we present a number of novel approaches for overcoming the aggregation problem. Central to these is the concept of partitioning the state space, and we begin by considering different partitioning methods (initially inspired by those previously used for parallel sparse matrix–vector multiplication) and evaluating their suitability for our state-by-state aggregation algorithm. We demonstrate that by partitioning the state space in this way and then using the state-by-state aggregation algorithm on the separate partitions, as opposed to applying it directly to an unpartitioned state-space, we can reduce the computational cost and memory requirements of our exact aggregation approach.

However, even when applied to partitions of the semi-Markov process there is a central drawback of exact state-by-state aggregation. Although the result of the process is an aggregated and smaller state space, the intermediate steps can actually create more state transitions (and hence require more storage and computational effort) than were present in the original unaggregated state space. Inspired by our prior work on iterative passage-time analysis in SMPs [9], we therefore present *atomic partition aggregation* to overcome this limitation. This does not require each state in the partition to be aggregated in turn, but instead effectively calculates the passage-time distribution across an entire partition and combines this with the state holding time distributions of relevant states outside the partition. As partitioning techniques suitable for parallel sparse matrix–vector multiplication do not produce partitions suitable for the application of atomic aggregation, we also introduce a new *barrier partitioning* strategy which is better suited. 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], and also summarises our state-by-state aggregation technique [8]. Section 3 then introduces the concept of performing aggregation on partitions of the state space, and discusses the importance of the order in which partitions are chosen to be aggregated. Section 4 then presents our novel atomic aggregation approach where whole partitions are aggregated by means of a passage-style analysis. Section 5 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 6 concludes and suggests directions for future work.

2. Background

2.1. Semi-Markov Processes

Semi-Markov Processes (SMPs) are an extension of Markov processes which allow for generally distributed sojourn times [19, 20]. 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 $N(t) = \max\{n : T_n \leq t\}$, 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

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.

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 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:

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

This formulation of the random variable $P_{i\vec{j}}$ applies to an SMP with no immediate transitions. If such transitions are present, then the passage time can be stated as:

$$P_{i\vec{j}} = \inf\{u > 0 : N(u) \geq M_{i\vec{j}}\} \tag{2}$$

where $M_{i\vec{j}} = \min\{m \in \mathbb{Z}^+ : \chi_m \in \vec{j} \mid \chi_0 = i\}$ is the transition marking the terminating state of the passage.

$P_{i\vec{j}}$ 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:

$$L_{i\vec{j}}(s) = \sum_{k \notin \vec{j}} r^*_{ik}(s) L_{k\vec{j}}(s) + \sum_{k \in \vec{j}} r^*_{ik}(s) \qquad : \text{for } 1 \leq i \leq N \tag{3}$$

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

$$r^*_{ik}(s) = \int_0^\infty e^{-st} \, dR(i, k, t) \tag{4}$$

Eq. (3) 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. (3) yields:

$$\begin{pmatrix} 1 & -r^*_{12}(s) & \cdots & -r^*_{1N}(s) \\ 0 & 1-r^*_{22}(s) & \cdots & -r^*_{2N}(s) \\ 0 & -r^*_{32}(s) & \cdots & -r^*_{3N}(s) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & -r^*_{N2}(s) & \cdots & 1-r^*_{NN}(s) \end{pmatrix} \begin{pmatrix} L_{1\vec{j}}(s) \\ L_{2\vec{j}}(s) \\ L_{3\vec{j}}(s) \\ \vdots \\ L_{N\vec{j}}(s) \end{pmatrix} = \begin{pmatrix} r^*_{11}(s) \\ r^*_{21}(s) \\ r^*_{31}(s) \\ \vdots \\ r^*_{N1}(s) \end{pmatrix} \tag{5}$$

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) [9]. 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:

$$P^{(r)}_{i\vec{j}} = \inf\{u > 0 : Z(u) \in \vec{j}, 0 < N(u) \leq r, Z(0) = i\} \tag{6}$$

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.

If we have immediate transitions in our SMP model (as in Eq. (2)) then the rth transition first passage time is:

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

This is because as the firing of an immediate transitions results in zero time being spent in the state in which it was enabled, it is not meaningful to talk about the SMP being in a particular state at a particular time. Instead, we count the transitions which have happened so that we may reason about the order in which they have occurred.

$P^{(r)}_{i\vec{j}}$ 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:

$$\mathbf{L}^{(r)}_{\vec{j}}(s) = \mathbf{U}\left(\mathbf{I} + \mathbf{U}' + \mathbf{U}'^2 + \cdots + \mathbf{U}'^{(r-1)}\right)\mathbf{e}_{\vec{j}} \tag{7}$$

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. (7) 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. (6):

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

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

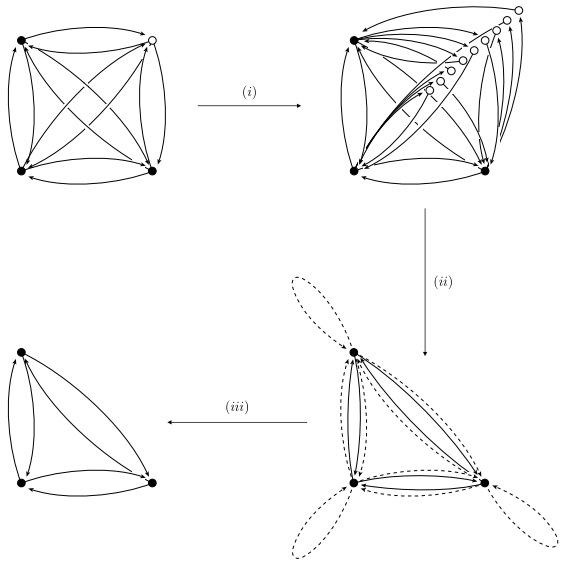


Fig. 1. Reducing a complete 4 state graph to a complete 3 state graph.

(DTMC) with one-step transition probability matrix $\mathbf{P} = [p_{ij}, 1 \leq i, j \leq N]$, as:

$$\alpha_k = \begin{cases} \pi_k / \sum_{j \in \vec{i}} \pi_j & \text{if } k \in \vec{i} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

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

$$\begin{aligned} L^{(r)}_{\vec{i}\vec{j}}(s) &= \boldsymbol{\alpha} \mathbf{L}^{(r)}_{\vec{j}}(s) \\ &= \left(\boldsymbol{\alpha}\mathbf{U} + \boldsymbol{\alpha}\mathbf{U}\mathbf{U}' + \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^2 + \cdots + \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^{(r-1)}\right) \mathbf{e}_{\vec{j}} \\ &= \sum_{k=0}^{r-1} \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^k \, \mathbf{e}_{\vec{j}} \end{aligned} \tag{9}$$

The sum of Eq. (9) can be computed efficiently using sparse matrix–vector multiplications with a vector accumulator, $\boldsymbol{\mu}_r = \sum_{k=0}^{r} \boldsymbol{\alpha}\mathbf{U}'^k$. At each step, the accumulator (initialised as $\boldsymbol{\mu}_0 = \boldsymbol{\alpha}\mathbf{U}$) is updated as $\boldsymbol{\mu}_{r+1} = \boldsymbol{\alpha}\mathbf{U} + \boldsymbol{\mu}_r \mathbf{U}'$.

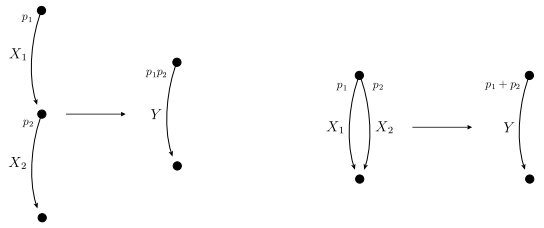
In practice, convergence of the sum $L^{(r)}_{\vec{i}\vec{j}}(s) = \sum_{k=0}^{r-1} \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^k$ can be said to have occurred if, for a particular r and s-point:

$$|\mathrm{Re}(L^{(r+1)}_{\vec{i}\vec{j}}(s) - L^{(r)}_{\vec{i}\vec{j}}(s))| < \varepsilon \quad \text{and} \quad |\mathrm{Im}(L^{(r+1)}_{\vec{i}\vec{j}}(s) - L^{(r)}_{\vec{i}\vec{j}}(s))| < \varepsilon \tag{10}$$

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

2.3. Exact State Aggregation

In order to control the state space explosion which occurs when generating the state transition matrix for a semi-Markov process, we have previously developed an exact aggregation algorithm that acts on the semi-Markov state space directly [5, 8]. The aim is to apply the aggregation before performing any passage-time or transient analysis and thus reduce the calculation time required to solve the system of linear equations shown in Eq. (5).



(a) Sequential transitions. (b) Branching transitions.

Fig. 2. Aggregating transitions in an SMP.

The method, illustrated in graphical terms in Fig. 1, works as follows: first, a state is chosen to be aggregated. Then, from the transition graph, all paths of length two centred on that state are identified (step (i)) and aggregated into stochastically equivalent, single transitions (step (ii)). The newly-created transitions (shown dashed in Fig. 1), which duplicate the route of existing transitions, are combined with the existing transitions. Finally, cyclic transitions are eliminated (step (iii)).

The result is to remove the chosen state and thus reduce the order of the transition matrix by one. Repeated application of this algorithm on different states will reduce the SMP to an arbitrary size (≥ 2 states), while still preserving the exact passage-time distributions between all pairs of the remaining states. This style of aggregation is not possible in a Markovian context as aggregation operations of this type do not have a closed form in the Markov domain (i.e. the convolution of two Markovian delays is not itself Markovian).

There are three basic reduction steps for aggregating a single state of an SMP. These deal with convolutions, branching and cycles as follows:

Sequential Reduction

In Fig. 2(a), $Y = X_1 + X_2$ is a convolution and therefore in Laplace form $L_Y(s) = L_{X_1}(s)L_{X_2}(s)$. In order to extract the path from an SMP we have to take into account the probabilities p_1 and p_2 of the first transition and second transitions of the path being selected. This gives us the overall path probability of p_1p_2.

Branch reduction

In Fig. 2(b), we can sum the respective probabilities to get the overall selection probability for the aggregate path. Thus the aggregate probability for the branch is $p_1 + p_2$. Our aggregate distribution, Y, is given by:

$$L_Y(s) = \frac{p_1}{p_1 + p_2} L_{X_1}(s) + \frac{p_2}{p_1 + p_2} L_{X_2}(s)$$

so that for both aggregate and unaggregated forms the total sojourn-time distribution has Laplace transform $p_1 L_{X_1}(s) + p_2 L_{X_2}(s)$.

Cycle Reduction

When there is a state with at least one out-transition and a transition to itself, as shown in Fig. 3, we can remove the cycle by making its stochastic effect part of the out-going transitions.

Consider a state transition system as being in the first stage of Fig. 3, with $(n-1)$ out-transitions and probability p_i of departure along edge i. Each out-transition has an associated sojourn X_i; the cycle probability is p_n with sojourn X_n.

The first step, (i), is to isolate the cycle and treat it separately from the branching out-transitions. We do this by rewriting the system to include an instantaneous delay and extra

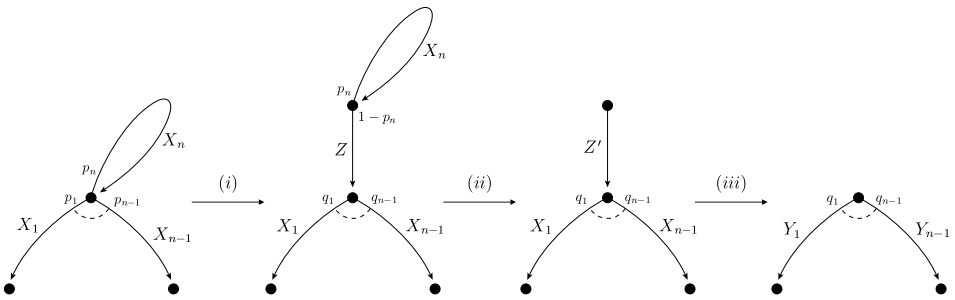


Fig. 3. The three-step removal of a cycle from an SMP.

state immediately after the cycle, $Z \sim \delta(0)$; the introduction of an extra state is only to aid our visualisation of the problem and is not necessary (or indeed performed) in the actual aggregation algorithm. Clearly the instantaneous transition will be selected with probability $(1 - p_n)$. We now have to renormalise the p_i probabilities on the branching state to become $q_i = p_i/(1 - p_n)$.

In step (ii) of Fig. 3, we aggregate the delay of the cycle into the instantaneous transition creating a new transition with distribution Z'. By treating the system as a geometric sum of the random variable X_n, we can write:

$$L_{Z'}(s) = \frac{1 - p_n}{1 - p_n L_{X_n}(s)}$$

In stage (iii) of the process, the Z' delay can be sequentially convolved with the X_i sojourns to give us our final system.

In summary, we have reduced an n-out-transition state where one of the transitions was a cycle to an $(n - 1)$-out-transition state with no cycle such that:

$$q_i = \frac{p_i}{1 - p_n}$$

and:

$$L_{Y_i}(z) = \frac{1 - p_n}{1 - p_n L_{X_n}(z)} L_{X_i}(z)$$

2.4. Case Study Semi-Markov Models

Throughout this paper we use three semi-Markov models as running examples. The Courier model [21] represents the ISO Application, Session and Transport layers of the Courier sliding-window communication protocol. The Voting model is a model of a distributed voting system with voters, failure-prone voting booths and failure-prone central servers [6, 9]. The Web-server model represents a web content authoring system, and contains a number of clients, authors web servers and a write buffer [9]. All three models were originally represented in a high-level Semi-Markov Stochastic Petri Net (SM-SPN) [7] form, from which semi-Markov processes of varying sizes can easily be generated. Further detail can be found in [14].

3. Partition Aggregation

Fig. 4 shows the number of non-zeros in the transition matrix as the matrix is aggregated. The solid line shows the progression of aggregation by the original statewise algorithm outlined in

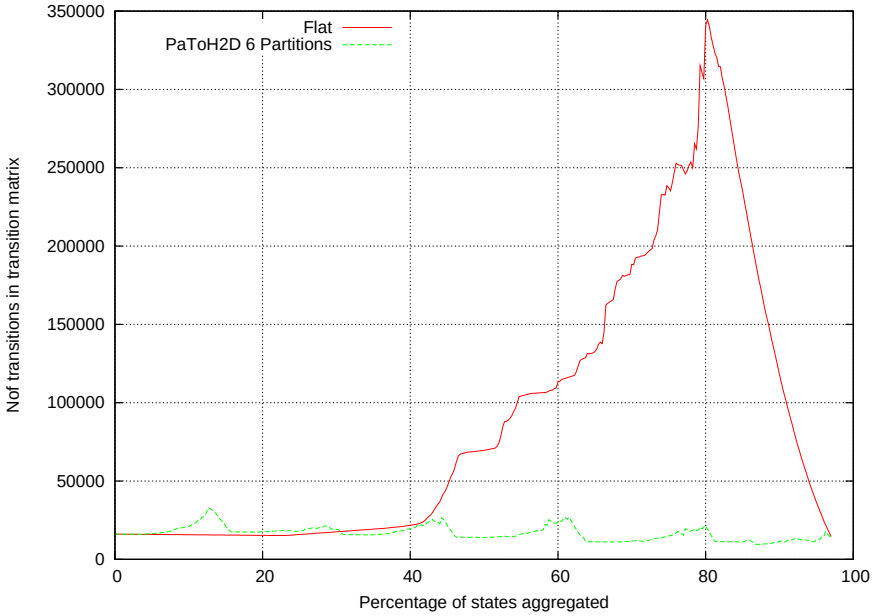


Fig. 4. The effect of partition aggregation compared to flat aggregation of the 4 050 state Voting model.

Section 2.3 on the whole state space. The dashed line shows the progression when partitioning of the transition matrix has been applied prior to aggregation.

Fig. 4 illustrates the problem encountered in applying the exact state-by-state aggregation algorithm sequentially across the *flat* state space of an SMP with 4 050 states. The transition matrix initially contains approximately 15 000 non-zeros, but by the time that approximately 80% of the states have been aggregated (circa 3 200 states) the number of non-zeros in the transition matrix has increased to nearly 350 000 even though the dimensions of the matrix have been reduced dramatically. This is important since it is the absolute number of non-zeros that determines the storage requirements and run-time performance of our performance analysis algorithms.

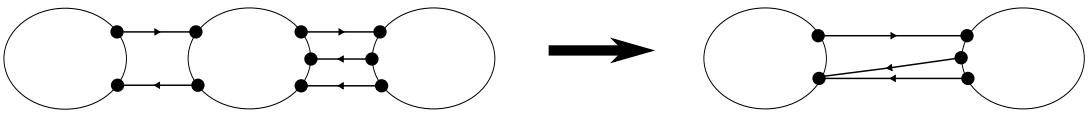


Fig. 5. Partition aggregation.

To avoid this dramatic peak in non-zeros, we propose *partition aggregation*. As shown in Fig. 5, the state space of the SMP is divided into a number of partitions and the states within each of these are aggregated together, leaving only the transitions between the states on the boundaries of each partition. The result of this can be seen in the lower curve in Fig. 4; the peak in the number of non-zeros now occurs for each partition, but each peak is an order of magnitude smaller than the peak in non-zeros which occurs when aggregating the entire state space sequentially.

3.1. Partitioning Techniques

Central to this new aggregation technique is the ability to partition the SMP's state space effectively. We divide n non-source and non-target states into k partitions, such that $k|n$. Inspired

by our experiences in parallelising sparse matrix–vector multiplication, we consider the following three partitioning techniques:

Row striping. The simplest partitioning strategy is to divide the matrix into blocks of contiguous rows such that each block contains approximately the same number of non-zeros. For k partitions and n matrix rows, the first partition contains the first n/k matrix rows, the second is assigned the next n/k rows and so on. This scheme has the advantage of being very easy to compute and also of achieving good load balance.

Graph partitioner. In a row-striped decomposition, the the $n \times n$ sparse transition matrix $\mathbf{P}$ of an SMP can be represented as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where each row i $(1 \leq i \leq n)$ in the matrix corresponds to vertex $v_i \in \mathcal{V}$ in the graph. The corresponding weight w_i of vertex v_i is the total number of non-zeros in row i. For the edge-set $\mathcal{E}$, edge e_{ij} connects vertices v_i and v_j with weight $w_{ij} = 1$ if either one of $p_{ij} > 0$ or $p_{ji} > 0$, and with weight $w_{ij} = 2$ if both $p_{ij} > 0$ and $p_{ji} > 0$ [12]. Graph partitioners try to minimise the number of edges which span two partitions (these are said to be *cut*) while balancing the number of non-zero elements in each partition. We use the METIS sequential k-way graph partitioning library [15].

Hypergraph partitioner. A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined by a set of vertices $\mathcal{V}$ and a set of nets (or hyperedges) $\mathcal{N}$, where each net is a subset of the vertex set $\mathcal{V}$ [4]. A hypergraph is therefore a generalised graph data structure in which edges can connect arbitrary non-empty subsets of vertices. In the context of a row-wise decomposition of a sparse matrix, matrix row i $(1 \leq i \leq n)$ is represented by a vertex $v_i \in \mathcal{V}$ while column j $(1 \leq j \leq n)$ is represented by net $N_j \in \mathcal{N}$ [12]. The vertices contained within net N_j correspond to the row numbers of the non-zero elements within column j, i.e. $v_i \in N_j$ if and only if $p_{ij} \neq 0$. Weights are assigned to vertices in the same manner as to the vertices of a graph The weight of all nets is one, with an individual net's contribution to the hyperedge cut being defined as one less than the number of different partitions spanned by that net. The overall objective of a hypergraph partitioning is to minimise the hyperedge cut while maintaining a balance criterion. In this paper we use the PaToH library [13] to perform hypergraph partitioning.

We distinguish between 1D hypergraph partitioning, where the hypernets either represent the successor states of each state (rows) or the predecessor states of each state (columns) and the 2D approach, where we use both successor and predecessor hypernets. Note that our definition of 2D hypergraph partitioning differs slightly from the definition commonly found in the literature, where each non-zero matrix element becomes a vertex in the 2D hypergraph. In our case 2D simply implies that we use information from both rows and columns of the SMP transition matrix to construct hypernets.

We now investigate how the choice of the partitioner affects the number of non-zeros created in the transition matrix during exact state-by-state aggregation of partitions. Recall that our idea is to partition the state space of the SMP and run the exact state aggregation algorithm on each partition separately, and thus avoid the dramatic increase in non-zeros observed (and hence the amount of memory required) when aggregating the unpartitioned state space.

Fig. 6 compares the number of non-zeros in the transition matrices of the three semi-Markov models when their state spaces are partitioned using these three techniques and then aggregated. We conclude that PaToH, which only uses the rows of the matrix as hypernets for partitioning, gives the worst results of all partitioners we tested as it leads to the largest number of non-zeros being created. For the Courier model, PaToH yields the worst matrix fill-in, while for the larger Voting and Web-server models, it either took too long to complete or exhausted the available memory on the test machine. The naïve row striping yielded good results in the Web-server and Courier model, but in the slightly more dense Voting model it performed much worse than METIS and PaToH2D. In general, we conclude that is very difficult to reliably use any one of

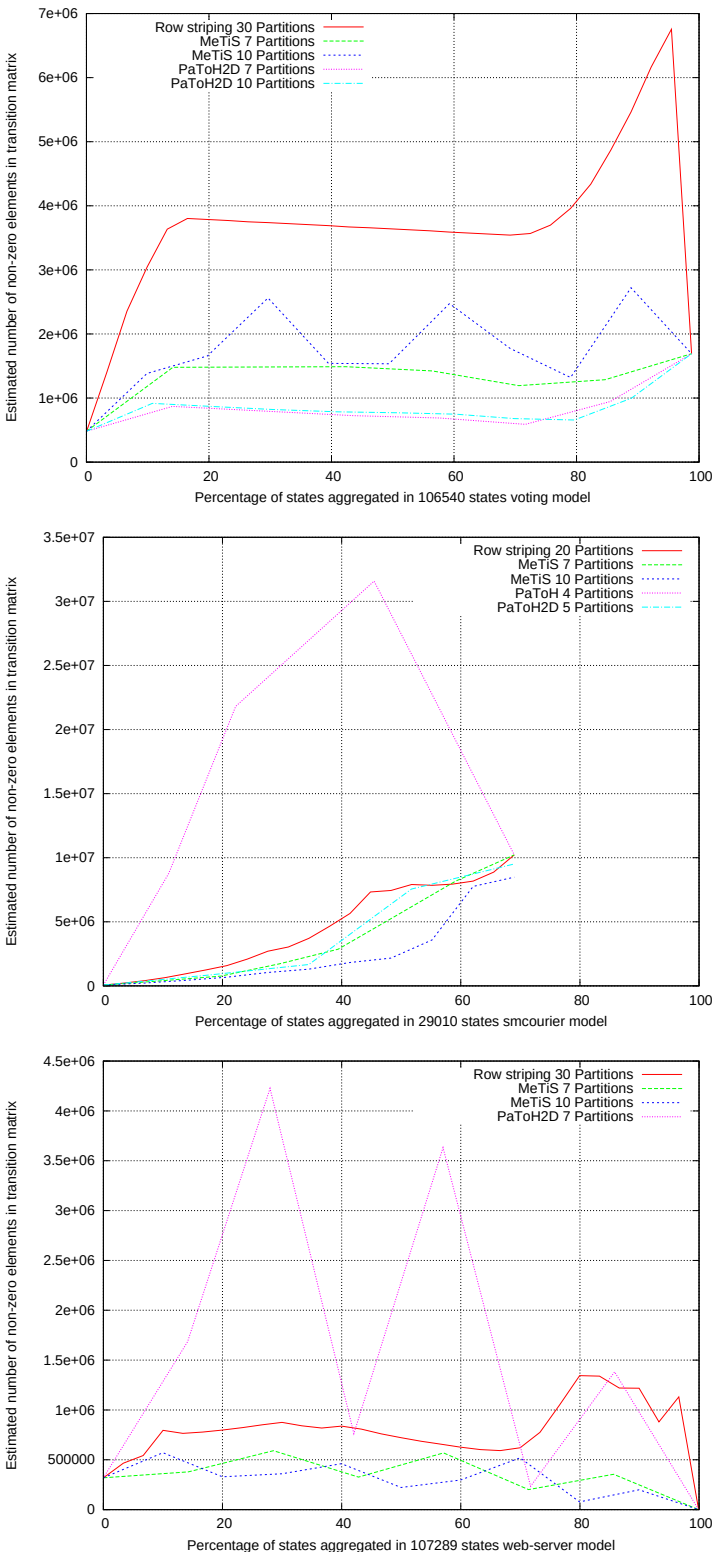


Fig. 6. The number of transitions in the transition matrix of different models during aggregation when partitioned with the three partitioners.

these techniques to produce the best partitions; the choice of best partitioner varies depending on the model and the number of partitions required. This inspires our alternative barrier partitioning approach discussed in Section 5 below.

3.2. Partition Ordering

Our prior work on exact state aggregation [8] has shown the importance of carefully choosing the order in which states should be aggregated. The same also applies to selecting the order in which partitions should be aggregated. Inspired by the state selection criteria in [8], we now compare two potential methods for partition order selection.

Fewest-Paths-First (FPF) partition sort. Suppose a partition has m predecessor states, i.e. states that lie outside the partition but have outgoing transitions to states in the partition, and n successor states, i.e. states that lie outside the partition and have incoming transitions from states in the partition. The number of transitions from the predecessor to the successor states in the SMP transition matrix after the aggregation of the partition is mn if all m predecessor states can reach all n successor states via paths through the partition. The FPF-value of a partition is:

$$mn - \textit{outgoing transitions}$$

where *outgoing transitions* is the total number of outgoing transitions from states in the partition. To choose a partition for aggregation using FPF sort we simply greedily select the one with the lowest FPF-value.

Enhanced-Fewest-Paths-First (EFPF) partition sort. Despite a being a good estimator for the total number of new transitions created after the aggregation of a partition, the FPF-value does not take into account the number of *incoming transitions* from the predecessor states of the partition. Further it does not count the *existing transitions* between the predecessor and successor states of the partition. The total number of new transitions after the aggregation can thus be estimated more accurately using *enhanced-fewest-paths-first (EFPF) sort.* The EFPF-value is:

$$mn - \textit{outgoing transitions} - \textit{incoming transitions} - \textit{existing transitions}$$

Note that the EFPF-value of a partition is only an upper bound for the total number of new transitions in the transition matrix after the aggregation of a partition. This is because there may not be a path from every predecessor state to every successor state with all intermediate states on the path being internal partition states. Even for small values of m and n this may cause significant differences between the estimated and the actual number of transitions.

Even though it is more expensive to calculate, our experiments have shown that EFPF partition sort usually gives better results than FPF or picking the partitions in a random order. Fig. 7 shows one situation where this is the case, specifically for a 5-way partitioning of the 10 300 state Voting model. For this reason we confine ourselves to considering only EFPF partition sorting in the following sections.

4. Atomic Partition Aggregation

Compared to flat state-by-state aggregation, the 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 (see Fig. 8). 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

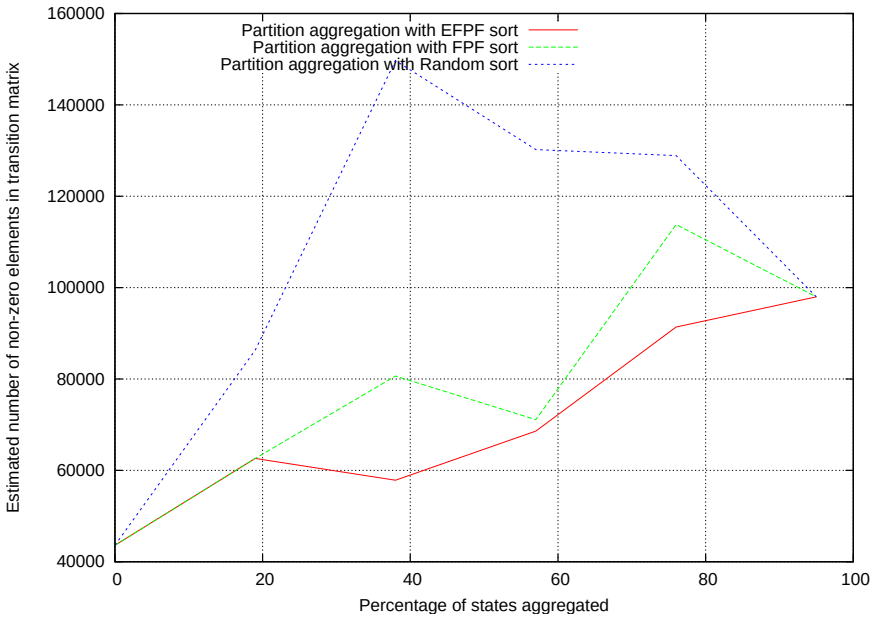


Fig. 7. Comparing EFPF partition sort with FPF partition sort.

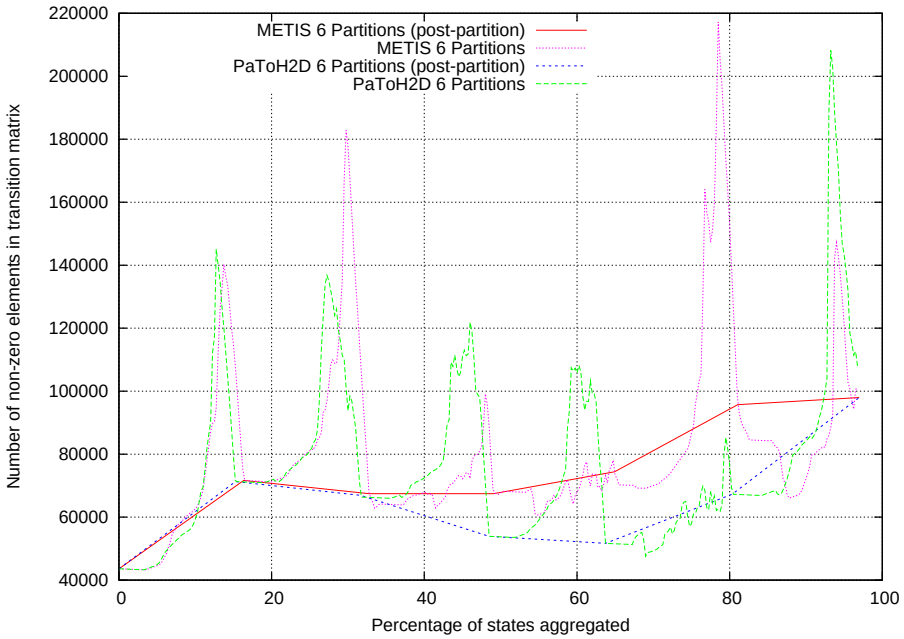


Fig. 8. State-by-state partition aggregation on 10 300 states Voting model.

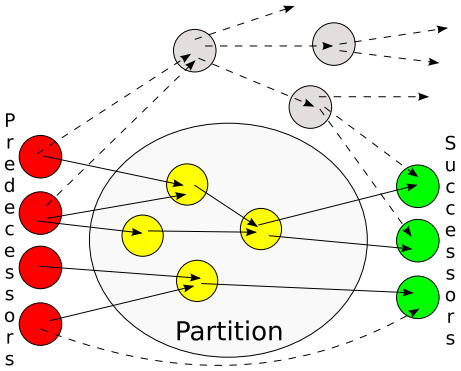


Fig. 9. Atomic aggregation.

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 sequential and branching 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 general concept is illustrated in Fig. 9. First we compute the passage time from each predecessor state p to every successor state s including only paths whose intermediate states lie entirely in the partition (denoted by the solid arcs in Fig. 9). In a second step we aggregate the passage time and the probability of these internal transitions with the passage time and probability of the existing one-step transition from p to s (denoted by the lower dashed arc in Fig. 9), if such a transition exists, using the branch aggregation technique from Section 2.3. If this one-step transition from p to s does not exist then the transition we computed in the first step becomes the new transition from p to s.

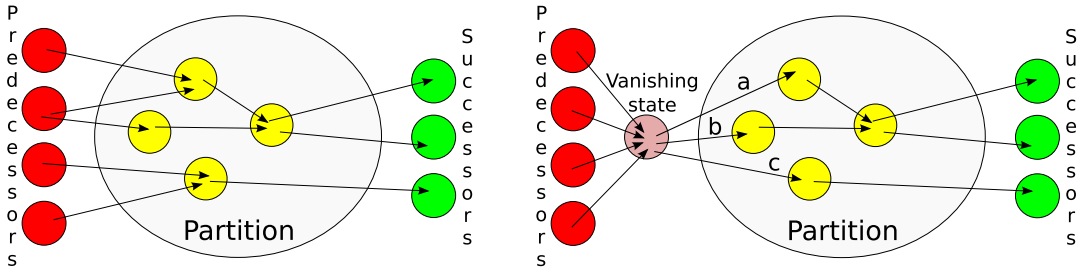
We only consider outgoing transitions from predecessor states of the partition to internal partition states. All other outgoing transitions of the predecessor states are ignored. We do not need to normalise the transition probabilities of outgoing transitions from the predecessor states. This can be formally justified by the flow conservation law, as we ensure that there are no final strongly-connected components of states within the partition.

Even though this appears to be a good strategy for aggregating an entire partition at once, it has one major disadvantage. Assume a partition has m predecessor, n successor and i internal states. In order to calculate the transition from every predecessor to every successor state using internal partition paths only, we have to solve m sets of $i + n$ linear equations.

4.1. Modified Atomic Aggregation

The main problem with atomic aggregation is that the number of linear equations to be solved to aggregate a partition depends on the number of predecessor and successor states of that partition, and that it may not be possible to find a partition of an SMP's state space that keeps the number of such states low. To overcome this, we investigate inserting extra states into the SMP to try to ensure that partitions have only one predecessor or successor state. Adding extra states was inspired by the application of hidden nodes in Bayesian inference [18].

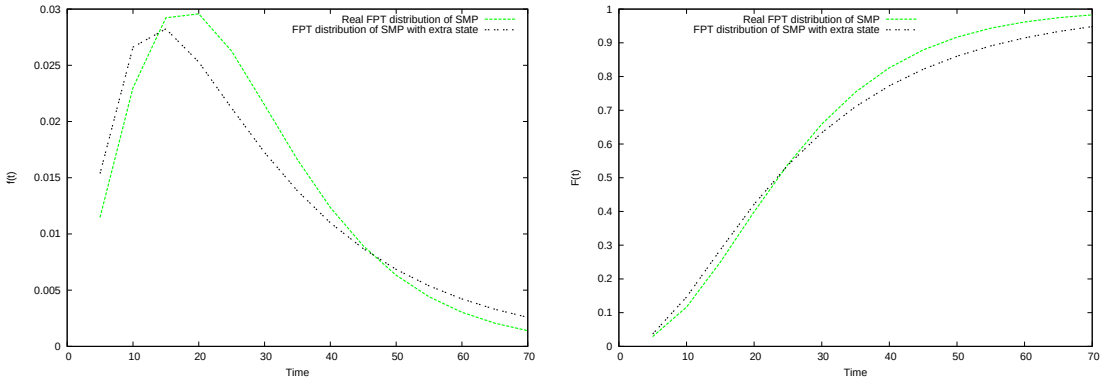
The general approach is shown in Fig. 10. Through the extra state, all four predecessor states have become connected to all partition entry states and can thereby reach each of the successor states of the partition. The number of linear equations required to aggregate the partition is



(a) Transition graph before adding extra vanishing state (b) Transition graph after adding extra vanishing state

Fig. 10. Insertion of an extra vanishing state to improve atomic aggregation.

therefore lower, but we have changed the structure of the SMP and so introduced error into any performance measures calculated upon it.



(a) PDF (b) CDF

Fig. 11. Effect on the first-passage time density and distribution of adding an extra state to the Courier model with 29 010 states.

To illustrate the error in the first passage-time distribution introduced by adding an extra state to the transition matrix, we compare the results from the unmodified model with results from the same model with an extra predecessor state. Fig. 11 shows the resulting nature of the approximation to the first passage-time distribution of the original SMP when analysing the modified graph. The Kolmogorov–Smirnov statistic for the two distributions (the maximum absolute difference between the two) is 0.0573 (4 d.p.), but nevertheless the resulting pdf and cdf appear to be good approximations to the real passage-time density and distribution respectively. In a second experiment we tested the impact of adding an extra predecessor state in the 107 289 state Web-server model. In this example we achieved a better approximation with a Kolmogorov–Smirnov statistic for the two distributions of 0.0002 (4 d.p.).

Note, however, that the runtime of the passage-time analyser in both cases was twice as long for the aggregated model with the added state as for the unaggregated SMP. It was possible, however, sometimes to achieve a speed-up. The algorithm was tested on a Intel Duo Core 1.8Ghz processor with 1Gbyte RAM. For the 106 540 state Voting model the total time taken to do atomic aggregation and the subsequent passage-time analysis for 165 Laplace transform samples with convergence precision 10^{-16} was 306 seconds. The total number of complex number multiplications was 2 553 489 711. In contrast, it took 398 seconds and 3 709 928 347 complex number multiplications to do the same passage-time calculation on the initial SMP graph without aggregation.

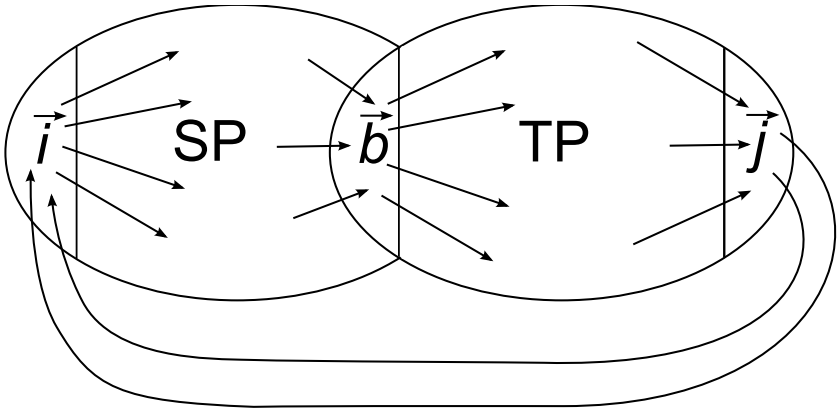


Fig. 12. A barrier partitioning, showing a set of start states $\vec{i}$ and target states $\vec{j}$ for a passage-time calculation. The remainder of the state space is split into a source partition SP, a target partition TP which contains the barrier, $\vec{b}$. Passages from SP to TP have to pass through $\vec{b}$ and cannot return, except via the target set, $\vec{j}$.

5. Barrier Partitioning

Atomic aggregation requires us to find partitions that have a low number of predecessor or successor states. As partitioners such as PaToH and METIS are not guaranteed to find such partitions, we need to investigate further partitioning methods for transition graphs of large semi-Markov models. Modified atomic aggregation of Section 4.1 attempts to solve this problem but at the expense of exact passage-time calculation.

In this section we introduce a new partitioning method called *barrier partitioning*. 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 $L_{\vec{i}\vec{j}}(s)$ (see Section 2.2). We observe that first passage-time analysis can be done forward, i.e. from each source state to the set of target states, as well as in reverse, i.e. from the set of target states to the individual source states, by transposing the SMP transition matrix and swapping source and target states. Such reverse passage-time calculation works well in Laplace space since complex multiplication is an associative operation. 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. The combined cost of doing the two separate calculations is the same as the cost of the original first passage-time calculation, but with the advantage that each of the two separate calculations requires only half the amount of memory as the original and can be performed independently and thus also in parallel.

Definition 1. 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 a source and a target state at the same time it can be split up into a target and source state, by adding an immediate transition from the new target to the new source state, without changing any measures of the SMP model represented by the new graph. We divide the state space into two partitions SP and TP. SP contains all source states and a proportion of the intermediate states such that any outgoing transitions from SP to TP go into 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 $\vec{i}$. Thus once a path has entered TP it can only ever go back to SP by going through states $\vec{j}$. Note that $\vec{b}$ and $\vec{j}$ may intersect. The resulting partitioning is a *barrier partitioning*. See Fig. 12 for a schematic representation.

Proposition 1. 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 $\vec{j}$. The result of 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 doing a first passage-time calculation from i to the set of barrier states $\vec{b}$, convolved with the first passage-time calculation from the set of barrier states $\vec{b}$ to the set of target states $\vec{j}$. In the Laplace domain this translates to:

$$L_{i\vec{j}}(s) = \sum_{b \in \vec{b}} L^R_{ib}(s) L_{b\vec{j}}(s) \tag{11}$$

where $L^R_{ib}(s)$ denotes a restricted first passage-time distribution from state i to state $b \in \vec{b}$, 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 $L^R_{ib}(s)$.

Proof. Restricting our set of equations to consider passage times from states $i \in SP$ to the target set $\vec{j}$, by Eq. (3) 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:

$$L_{i\vec{j}}(s) = \sum_{k \in (SP \cup TP)} r^*_{ik}(s) L_{k\vec{j}}(s) \tag{12}$$

where $L_{k\vec{j}}(s)$ is equal to 1 if $k \in \vec{j} \cap \vec{b}$. 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 $TP \setminus \vec{b}$ by construction of the barrier.

$$\begin{aligned} L_{i\vec{j}}(s) &= \sum_{k \in (SP \cup \vec{b})} r^*_{ik}(s) L_{k\vec{j}}(s) \\ &= \sum_{b \in \vec{b}} r^*_{ib}(s) L_{b\vec{j}}(s) + \sum_{k \in SP} r^*_{ik}(s) L_{k\vec{j}}(s) \end{aligned} \tag{13}$$

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:

$$\begin{aligned} L_{i\vec{j}}(s) &= \sum_{b \in \vec{b}} r^*_{ib}(s) L_{b\vec{j}}(s) + \sum_{k \in SP} r^*_{ik}(s) \sum_{b \in \vec{b}} L^R_{kb}(s) L_{b\vec{j}}(s) \\ &= \sum_{b \in \vec{b}} \left[\left(\sum_{k \in SP} r^*_{ik}(s) L^R_{kb}(s) + r^*_{ib}(s) \right) L_{b\vec{j}}(s) \right] \end{aligned} \tag{14}$$

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:

$$L_{i\vec{j}}(s) = \sum_{b \in \vec{b}} L^R_{ib}(s) L_{b\vec{j}}(s) \tag{15}$$

■

Corollary 1.1. 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.

$$L^R_{i\vec{j}}(s) = \sum_{b \in \vec{b}} L^R_{ib}(s) L^R_{b\vec{j}}(s) \tag{16}$$

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

$$L^R_{b\vec{j}}(s) = L_{b\vec{j}}(s) \tag{17}$$

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

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

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. (15). ∎

Corollary 1.2. 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 $L_{\vec{b}\vec{j}}(s) = \{L_{b_1\vec{j}}(s), \ldots, L_{b_l\vec{j}}(s)\}$ then in steady-state we have:

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

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. (8). By Eq. (9) we have:

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

∎

5.1. 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 2.2) 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. 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 and TP only contains the set of target states.

In both the Voting and the Web-server model it is possible to split the state space 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 1.1 million state Voting model takes less than 10 seconds on an Intel Duo Core machine with two 1.8GHz processors and 1Gbyte of RAM. By comparison, the computation of a 2-way partitioning with PaToH2D 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.

5.2. k-way Barrier Partitioning

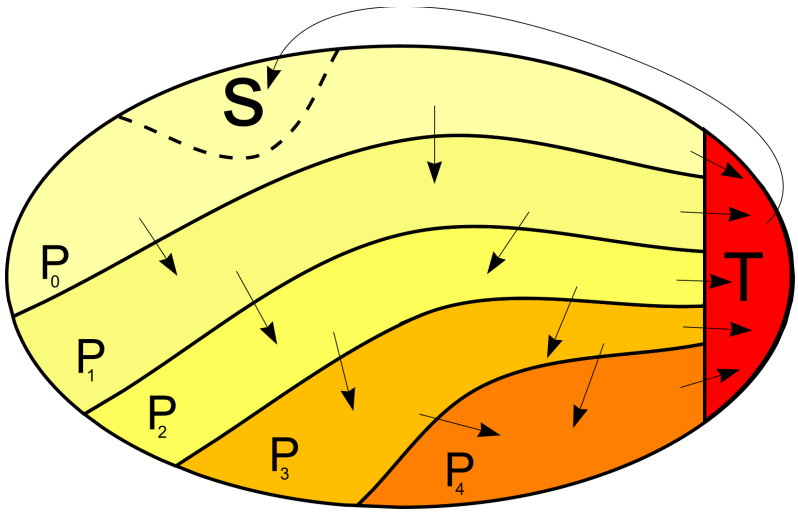


Fig. 13. A representation of k-way barrier partitioning.

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. 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} and its successor partitions P_{m+1} and T. Note that 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. This is shown in Fig. 13.

Note. Definition 2 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.

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

$$L_{i\vec{j}} = L^R_{i\vec{b}_1} \; M^R_{\vec{b}_1\vec{b}_2} \; \cdots \; M^R_{\vec{b}_{k-2}\vec{b}_{k-1}} \; L^R_{\vec{b}_{k-1}\vec{j}} \tag{20}$$

where $L^R_{i\vec{b}_1}$ is the $1 \times m_1$ row vector containing the resulting 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.

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

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

using this argument repeatedly reduces Eq. (20) to

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

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 2.2) 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 2.2). $L_{\vec{i}\vec{j}}(s)$ is computed by summing the Laplace transforms which make up this vector as in Eq. (7).

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 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.

5.3. 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 haven't been assigned to any partition. We continue partitioning the state space until all states have been assigned to a partition.

Proposition 3. 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.*

Proof. 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 partioning. 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.

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.

We tested the new 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 a function of the number of state transitions, and our results suggest that this relationship is linear 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 PaToH2D takes about 60 seconds on the same machine for the Web-server model, but the resulting partitioning is not even suitable for atomic aggregation.

5.4. Evaluation

The log–log plot in Fig. 14 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 seems 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

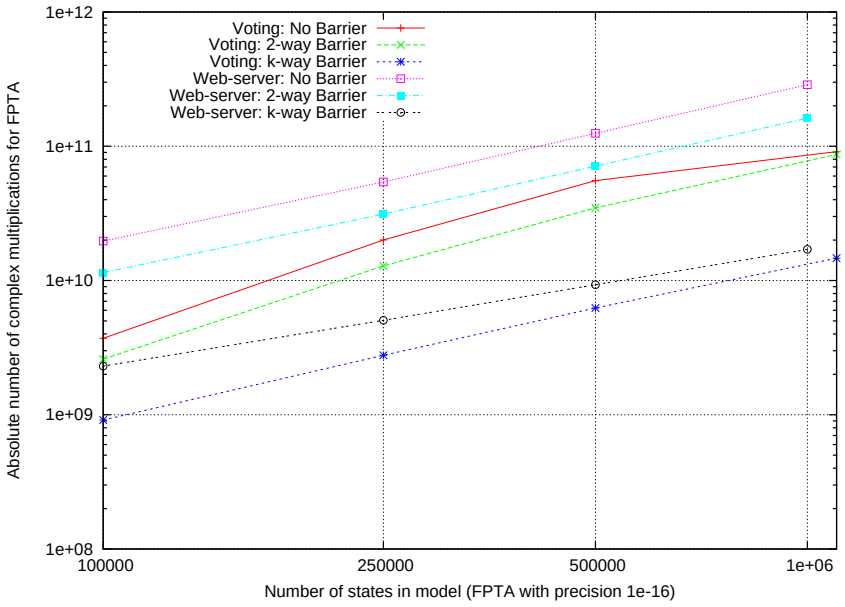


Fig. 14. Log–log comparison of the absolute number of multiplications required under different barrier aggregation strategies in the Voting and Web-server models.

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 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.

5.4.1. 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, 14] for very large SMPs.

The parallel scheme was implemented in the Semi-Markov Response Time Analyser (SMARTA) [14]. 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 partition-

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.

ing 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 [16] 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}.

6. Conclusion

In this paper we have presented a number of improved aggregation techniques for SMPs. We have shown how dividing an SMP's state space into a number of loosely-connected partitions reduces the maximum number of transitions generated during the application of our state-by-state exact aggregation algorithm. We have also devised two partition-ordering metrics (analogous to the state-ordering metrics of the exact aggregation algorithm) to determine the order in which partitions should be aggregated. Of these, we concluded that our EFPF method gave better results than the FPF method.

Even with the partition aggregation approach with improved partitioning ordering metrics, however, we could not escape the fact that many additional temporary transitions were being created during aggregation. This inspired us to propose a scheme, based on first passage-time analysis, for the atomic aggregation of partitions. Provided we find a suitable partition, atomic partition aggregation is more efficient than state-by-state aggregation of partitions. Like state-by-state aggregation, it may not always yield a speed-up in computation time of the passage-time analysis, but it can always be used to save memory as we only need to store the sub-matrix of the partition under consideration.

The biggest problem with atomic partitioning is that we may not be able to find suitable partitions using existing state-space partitioning techniques. Introducing additional vanishing states

alleviates this somewhat, but results in errors being introduced into the final calculated first passage-time distributions. We therefore developed 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. We postulate that barrier partitioning is suitable for SMPs of models with large populations of similarly operating cooperating components; this was true of the Web-server and Voting models and is fortunately a common feature of large SMP models, derived from higher-level formalisms.

For the future, it would be interesting to investigate if graph and hypergraph partitioners can be modified to produce better partitionings for atomic aggregation. This could potentially be done by finding more suitable input parameters for the PaToH and METIS partitioner. However, it is likely that there are also better algorithms and partitioning heuristics, and further research might produce partitioning strategies that extend the range of semi-Markov models for which atomic aggregation can be used.

We would also like to explore to extent to which k-way passage-time computation can be conducted in parallel. Recall from Section 5 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.

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] C. Berge. *Hypergraphs: Combinatorics of Finite Sets*. North-Holland, Amsterdam, 1989.

[5] 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)*, pages 178–187, London, April 14th–17th 2002. Springer-Verlag.

[6] 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.

[7] 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.

[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)*, pages 755–762, Montreal, Canada, 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*, 386:311–334, 2004.

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

[11] W-L. Cao and W.J. Stewart. Iterative aggregation/disaggregation techniques for nearly uncoupled Markov chains. *Journal of the ACM*, 32(3):702–719, July 1985.

[12] U.V. Catalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, July 1999.

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

[14] 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.

[15] 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.

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

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

[18] R. Neapolitan. *Probabilistic Reasoning in Expert Systems.* John Wiley, 1990.

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

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

[21] C.M. Woodside and Y. Li. Performance Petri net analysis of communication protocol software by delay-equivalent aggregation. In *Proceedings of the 4th International Workshop on Petri nets and Performance Models (PNPM'91)*, pages 64–73, Melbourne, Australia, 2–5 December 1991. IEEE Computer Society Press.