Exact Aggregation Strategies for Semi-Markov Performance Models

Jeremy T. Bradley Nicholas J. Dingle William J. Knottenbelt

Department of Computing, Imperial College London
180 Queen's Gate, London SW7 2AZ, United Kingdom
{jb,njd200,wjk}@doc.ic.ac.uk

ABSTRACT
Semi-Markov modelling paradigms are more expressive than their Markovian counterparts but are equally vulnerable to the state space explosion problem. This paper addresses this issue by presenting an exact state-by-state aggregation algorithm for semi-Markov models. Empirical evidence shows that the computational complexity of our method depends critically on the order in which the states are aggregated. We investigate different state-ordering strategies and find one that aggregates a large proportion (circa 75%) of our example state spaces at minimal cost. This leaves a significantly smaller model which can then be analysed for performance-related quantities such as passage-time quantiles. We demonstrate our technique on several examples, including a 541 280 state model which is reduced to 173 101 states.

1. INTRODUCTION
Semi-Markov processes (SMPs) are expressive tools for modelling telecommunication and computer systems; they are a generalisation of Markov processes that allow for arbitrarily distributed sojourn times. In generating an SMP from a high-level model, the possible combinations of event interleavings can result in a combinatorial explosion in the number of states in the underlying SMP. One approach to addressing this state space explosion problem is to use aggregation techniques to remove single states or groups of states and aggregate their effect into the remainder of the model. Aggregation reduces the time and space requirements of subsequent performance computations, since the complexity of solution is usually at least quadratic in the size of the SMP. Aggregation is thus worthwhile if the time spent performing the aggregation and solving the reduced model is less than the time taken to solve the original unaggregated model directly.

Many techniques exist in the Markovian domain for exact and approximate aggregation (for example, lumpability [1], aggregation-disaggregation [2], aggregation of hierarchical models [3]) but to date analogous work on semi-Markov aggregation algorithms has been very limited [4].

In this paper, we present a state-by-state aggregation algorithm for semi-Markov processes. We also present a method for representing the Laplace transforms of the state-holding-time density functions of an SMP which requires constant space. This is an important implementation issue because it avoids replicating the structural complexity of the unaggregated model in the distribution representation of the aggregated model (as seen in phase-type representation of general distributions [5]).

For the example semi-Markov models analysed here, we observe that the space and time complexity of our algorithm varies according to the order in which states are aggregated. We find an ordering which gives us the ability to aggregate a large proportion of states cheaply, halting before the aggregation process becomes too expensive.

The remainder of this paper is organised as follows. In Section 2 we briefly describe semi-Markov processes and illustrate the calculations required to generate passage-time performance quantities. Such measures are assuming increasing importance as key quality of service metrics in benchmarks and Service Level Agreements. Section 3 outlines the aggregation algorithm and discusses efficient distribution representation. Finally, Section 4 presents an example semi-Markov system derived from a Petri net model. We perform aggregations on different model sizes to evaluate different state-aggregation ordering strategies and show our algorithm's efficacy in models of up to 541 280 states.

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

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

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

$$Z(t) = X_{N(t)} \qquad (2)$$

where $N(t) = \max\{n : T_n \leq t\}$, the number of transitions by time t. Thus $Z(t)$ represents the state of the system at time t. We consider time-homogeneous SMPs, in which $R(n, i, j, t)$ is independent of any previous state except the last. Thus R becomes independent of n:

$$\begin{aligned} R(i, j, t) &= \mathbb{P}(X_{n+1} = j, T_{n+1} - T_n \leq t \mid X_n = i) \\ &= p_{ij} H_{ij}(t) \end{aligned} \qquad (3)$$

for any $n \geq 0$, where $p_{ij} = \mathbb{P}(X_{n+1} = j \mid X_n = i)$ is the state transition probability between states i and j and $H_{ij}(t) = \mathbb{P}(T_{n+1} - T_n \leq t \mid X_{n+1} = j, X_n = i)$, is the sojourn time distribution in state i when the next state is j. We use $\tilde{\pi}$ to be the steady-state vector of the embedded discrete-time Markov chain (DTMC), p_{ij}.

2.2 First passage-times

Consider a finite, irreducible, continuous-time semi-Markov process with n states $\{1, 2, \ldots, n\}$. If $Z(t)$ denotes the state of the SMP at time t ($t \geq 0$), then the first passage-time from a source state i at time t into a non-empty set of target states $\vec{j}$ is:

$$P_{i\vec{j}}(t) = \inf\{u > 0 : Z(t+u) \in \vec{j}, N(t+u) > 0, Z(t) = i\} \tag{4}$$

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

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

$P_{i\vec{j}}$ is a random variable with an associated probability density function $f_{i\vec{j}}(t)$ such that:

$$\mathbb{P}(t_1 < P_{i\vec{j}} < t_2) = \int_{t_1}^{t_2} f_{i\vec{j}}(t)\,\mathrm{d}t \qquad : 0 \leq t_1 < t_2 \tag{6}$$

In general, the Laplace transform of $f_{i\vec{j}}$, $L_{i\vec{j}}(s)$, can be computed by solving a set of n linear equations:

$$L_{i\vec{j}}(s) = \sum_{k \notin \vec{j}} r^*_{ik}(s) L_{k\vec{j}}(s) + \sum_{k \in \vec{j}} r^*_{ik}(s) \tag{7}$$

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

$$r^*_{ik}(s) = \int_0^\infty e^{-st}\,\mathrm{d}R(i,k,t) \tag{8}$$

Eq. (7) has matrix-vector form $A\tilde{x} = \tilde{b}$. Systems of linear equations of this form can be solved by a variety of numerical techniques, for example Successive Over-Relaxation or Conjugate Gradient Squared [6]. Note that in the general semi-Markov case, the elements of A become arbitrary complex functions with no standard form and care needs to be taken when storing such functions for eventual numerical inversion (see Section 3.5).

When there are multiple source states, denoted by the vector $\vec{i}$, the Laplace transform of the passage-time distribution at steady-state is:

$$L_{\vec{i}\vec{j}}(s) = \sum_{k \in \vec{i}} \alpha_k L_{k\vec{j}}(s) \tag{9}$$

where the weight α_k is the probability at equilibrium that the system is in state $k \in \vec{i}$ at the starting instant of the passage. As the starting vector of states does not necessarily include all the states, α_k is the renormalised version of the embedded DTMC steady-state probability vector, $\tilde{\pi}$:

$$\alpha_k = \frac{\pi_k}{\sum_{j \in \vec{i}} \pi_j} \tag{10}$$

3. ITERATIVE STATE SPACE AGGREGATION

3.1 Motivation

In order to control the state space explosion which occurs when generating the state transition matrix for a semi-Markov process, we have developed an exact aggregation algorithm that acts on the semi-Markov state space directly. The aim is to apply such aggregation before performing any passage-time analysis and thus reduce the calculation time required to solve the system of linear equations of Eq. (7).

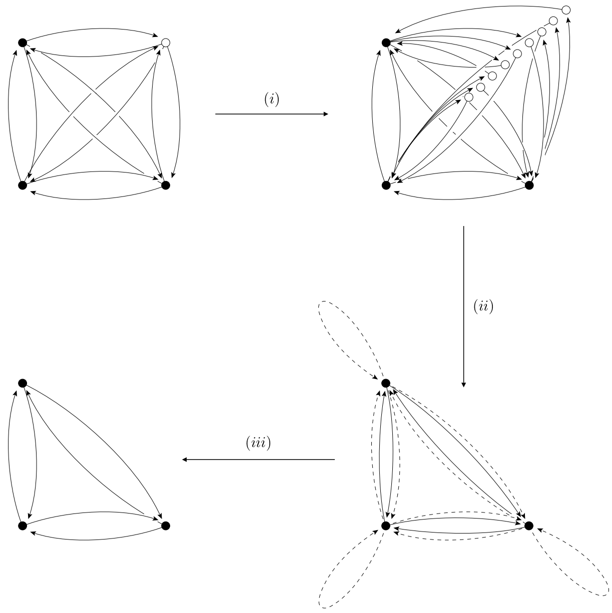

Figure 1: Reducing a complete 4 state graph to a complete 3 state graph.

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).

Of particular importance is the order in which states are chosen to be aggregated. Theoretically, any state could be operated on, providing it is not a source or destination state of a passage-time quantity. A detailed discussion of potential state ordering strategies is presented in Section 3.4. Before proceeding to this, however, we outline the basic steps of our algorithm.

3.2 Basic Reduction Steps

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, $Y = X_1 + X_2$ 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

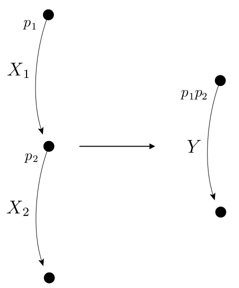

Figure 2: Aggregating sequential transitions in an SMP.

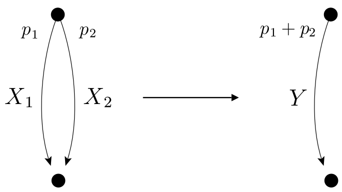

Figure 3: Aggregating branching transitions in an SMP.

path probability of p_1p_2.

Branch reduction

In Fig. 3, 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:

$$L_Y(s) = \frac{p_1}{p_1+p_2} L_{X_1}(s) + \frac{p_2}{p_1+p_2} L_{X_2}(s) \qquad (11)$$

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. 4, 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. 4, 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 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 $q_i = p_i/(1-p_n)$.

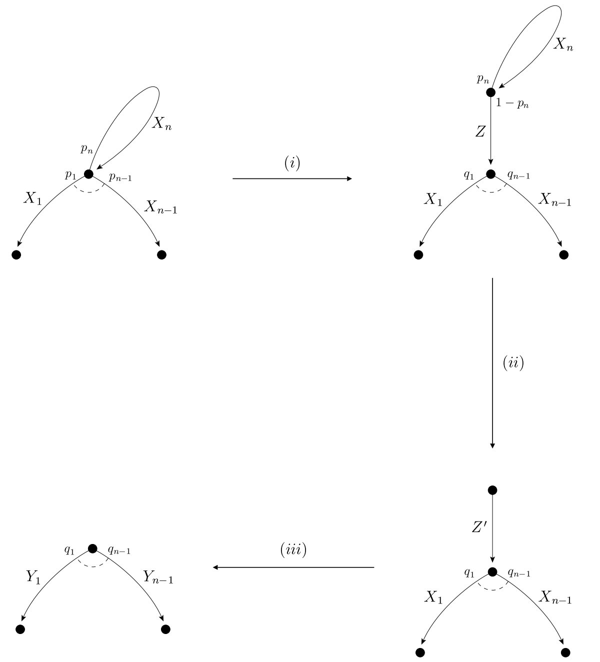

Figure 4: The three-step removal of a cycle from an SMP.

$$\begin{array}{lcl}
\mathit{aggregate_smp} & : & SMP \times S \rightarrow SMP \\
\mathit{aggregate_smp} & & (M, i) = \mathit{fold}(set, M', vs) \text{ where} \\
\end{array}$$

$$\begin{array}{rcl}
vs & = & \mathit{aggregate_cycles}(M', ts_c), \\
M' & = & \mathit{fold}(set, M, us), \\
us & = & \mathit{aggregate_branches}(M, ts_b), \\
ts_b & = & ts \setminus ts_c, \\
ts_c & = & \{((i,j),t) \,.\, ((i,j),t) \in ts, i = j\}, \\
ts & = & \mathit{aggregate_sequences}(M, i)
\end{array}$$

Figure 5: The $\mathit{aggregate\_smp}$ function

In step (ii) of Fig. 4, we aggregate the delay of the cycle into the instantaneous transition creating a new transition with distribution Z'. By treating the system as a random geometric sum of the random variable X_n, we can write:

$$L_{Z'}(s) = \frac{1-p_n}{1-p_n L_{X_n}(s)} \qquad (12)$$

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:

$$q_i = \frac{p_i}{1-p_n} \qquad (13)$$

and:

$$L_{Y_i}(z) = \frac{1-p_n}{1-p_n L_{X_n}(z)} L_{X_i}(z) \qquad (14)$$

3.3 Main aggregation algorithm

In this section, we define the aggregation function, $\mathit{aggregate\_smp}$, which is defined on an SMP, M, for a state i being aggregated.

We represent M by the tuple (S, P, L) where S is the finite set of states, P is the underlying DTMC and L is the state holding-time distribution matrix whose entries are the Laplace transforms of the state sojourn-time densities. In an irreducible semi-Markov process, the aggregation procedure can be applied to any state, while in a transient SMP, it may be applied to any state except the absorbing or initial states. Reading from the bottom up in the function definition of Fig. 5:

1. $ts = aggregate\_sequences(M, i)$:
   Find all valid paths of length two that have i as the centre state. Form a set of single transitions using the sequential aggregation described in Section 3.2.
2. $ts_b = ts \setminus ts_c,\ ts_c = \{((i, j), t) \ . \ ((i, j), t) \in ts, i = j\}$:
   Separate the transition set ts into a set of cycles, ts_c, and a set of branching transitions to other states, ts_b.
3. $us = aggregate\_branches(M, ts_b)$:
   Where transitions in M are duplicated by transitions in ts_b, these are aggregated and placed in us using the branching aggregation from Section 3.2. Where a new transition that is not present in M in described ts_b then that transition is written unchanged into us.
4. $M' = fold(set, M, us)$:
   Overwrite all the transitions in M that are present in us.
5. $vs = aggregate\_cycles(M', ts_c)$:
   Perform cyclic aggregation on the transitions in ts_c, using the method from Section 3.2.
6. $aggregate\_smp(M, i) = fold(set, M', vs)$:
   Finally, overwrite the transitions of M' with the replacement vs transitions and return the final SMP.

The result is that M' will have a disconnected state, i, which can be removed from the transition matrices.

Definitions of the three functions that implement the reductions of Section 3.2 (i.e. $aggregate\_sequences$, $aggregate\_branches$ and $aggregate\_cycles$) are given in Appendix A.

The algorithm, as outlined, removes a single state and reduces the SMP to a normal form, which has no same-state cycles, after each aggregation. If aggregating many states consecutively, one optimisation is to perform this reduction of same-state cycles once only after the last state has been aggregated, rather than after every state aggregation.

3.4 State-ordering Issues for Aggregation

As mentioned in Section 1, the order in which the states of a semi-Markov system are aggregated can have an enormous effect on the space and time demands of our algorithm. In our discussion of various state-ordering strategies, we will focus on two key metrics by which the practicality of these strategies can be judged:

1. the density of non-zero elements in the matrices P and L, also known as the matrix *fill-in*. For an $N \times N$ sparse matrix with r non-zero elements, the density is given by r/N^2.
2. the computational cost of the aggregation process as given by the number sequential, branch and cycle reduction operations performed.

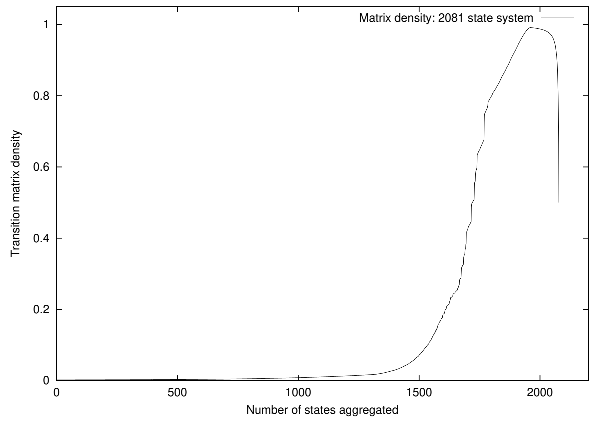

Figure 6: Complete aggregation of a $2\,081$ state semi-Markov system to two states.

Intuitively, there is a tension between these two metrics: strategies designed to reduce fill-in should result in more branching aggregation, and hence more computation, while strategies designed to reduce computation should tend to increase density faster.

3.4.1 Matrix density

Clearly, the density of non-zero elements in the matrix will increase and approach 1.0 as the states are removed and further non-zero transitions are created. If a system is aggregated so that only two states remain (with no same-state cycles) then the density will initially approach 1.0 and then decay to 0.5 in the final stages. This decrease in density occurs because the final transition matrix will only have non-zeros in elements $L_{12}(s)$ and $L_{21}(s)$ and zeros in $L_{11}(s)$ and $L_{22}(s)$. This can be observed in Fig. 6, which shows the matrix density for a $2\,081$ state semi-Markov process as it is aggregated down to two states.

3.4.2 Computation

In terms of computational cost, for a state with m predecessor states and n successor states, there are mn convolution operations and as many as mn branching aggregations to perform. For an $N \times N$ transition matrix with high density, this will give us $O(N^3)$ operations to perform to aggregate $O(N)$ states. For a sparse transition matrix with low density, m and n may be $O(1)$ rather than $O(N)$ and if the newly created transitions do not coincide with existing transitions then this gives us a lower bound of mn convolutions to perform.

As these semi-Markov models are typically generated from formalisms such as Petri nets and process algebras (where the number of potential successors of a state is almost always limited), their initial matrices tend to be sparse. As aggregation proceeds, however, these matrices will become increasingly dense, with the computational-cost consequences that this entails. In order to gain any benefit from aggregation, therefore, it may be necessary to curtail the process before the benefits gained from reducing the state space size are outweighed by the cost of performing the aggregation.

Before we examine some specific state-ordering strategies, we need to deal with one final implementation issue: that of distribution representation.

3.5 Distribution Representation

The key to practical analysis of semi-Markov processes lies in the efficient representation of the general distribution functions. Without care the structural complexity of the SMP can be recreated within the representation of the distribution functions. This is certainly true of the manipulations performed in the aggregation calculations of Section 3.2.

Many techniques have been used for representing arbitrary distributions – two of the most popular being *phase-type distributions* and the *vector-of-moments* method. These methods suffer from, respectively, exploding representation size under composition and containing insufficient information to produce accurate answers after large amounts of composition.

As all our distribution manipulations take place in Laplace-space, we link our distribution representation to the Laplace inversion technique that we ultimately employ. We use two different transform inversion algorithms: the Euler technique [7] and the modified Laguerre method [8] with modifications summarised in [9]. In our experience, these inversion techniques can produce accurate results for a large variety of models, even those which contain discontinuous distributions [10, 11].

Both algorithms work on the same general principle of sampling the transform function $f^*(s)$ at n points, $s_1, s_2, \ldots, s_n$ and generating values of $f(t)$ at m user-specified t-points $t_1, t_2, \ldots, t_m$. In the Euler inversion case $n = km$, where k can vary between 15 and 50, depending on the accuracy of the inversion. In the modified Laguerre case, $n = 400$ and, crucially, is independent of m. Whichever algorithm is chosen, however, the important issue is that calculating $s_i, 1 \leq i \leq n$ and storing the Laplace transforms of all the sojourn-time density functions sampled at these points, will be sufficient to provide a complete inversion.

Storing our distribution functions in this way has three main advantages. Firstly, a function has constant storage space, independent of its distribution-type. Secondly, each distribution, therefore, has the same constant storage requirement even after composition with other distributions. Finally, there is sufficient information about a distribution to determine a required passage-time density (and no more).

4. AGGREGATION STRATEGIES

4.1 Example System

To demonstrate the aggregation technique we use the example semi-Markov system shown in Fig. 7. This is specified as a semi-Markov stochastic Petri net [10] that has generally distributed transition firing times. The model represents a distributed voting system where voters cast votes through polling units, which in turn register votes with all available central voting units. Both polling units and central voting units can suffer breakdowns, from which there is a soft recovery mechanism. If, however, all the polling or voting units fail then, with high priority, a failure recovery mode is instituted to restore the system to an operational state.

There are several voters, CC, a limited number of polling units, MM, and a smaller number of central voting units, NN. The size and complexity of the underlying semi-Markov chain can be varied by altering these three parameters as shown in Table 1.

Table 1: Number of states generated by the voting system SM-SPN in terms of the number of voters (CC), polling units (MM) and central voting units (NN).

CC	MM	NN	States
11	7	4	2 081
22	7	4	4 050
60	25	4	106 540
125	40	4	541 280

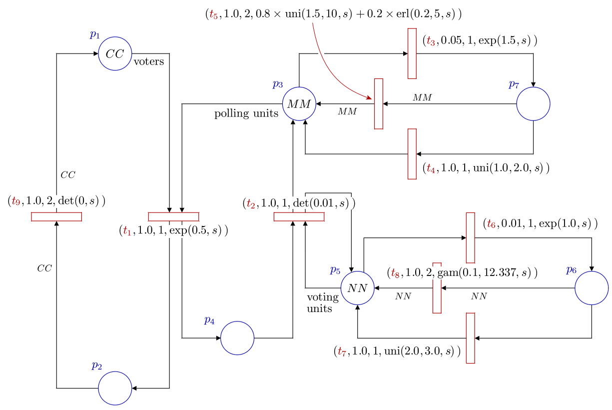

Figure 7: The semi-Markov voting model shown in stochastic Petri net form.

We generate four models from this example: 2 081 and 4 050 state models which are used to test several different aggregation strategies; and 106 540 and 541 280 state models which are aggregated using the algorithm identified as most efficient.

4.2 State-selection algorithms

Bearing in mind the transition matrix density and computational complexity issues highlighted in Section 3.4, we propose the following orderings of states for aggregation. Given that a state has m predecessor states and n successor states:

Fewest-paths-first chooses the state with lowest mn-value first. This is designed to minimise computation, as $O(mn)$ convolution and branching aggregations are required to eliminate a state.

Most-successor-states chooses the state with the highest n-value first. This is designed to reduce fill-in by targeting the rows of the transition matrix with the largest number of non-zeros for aggregation.

Most-paths-first chooses the state with highest mn-value first. This is designed to demonstrate the computationally worst-case scenario.

Random chooses an arbitrary state for aggregation, without consideration of m or n. The selection is done uniformly across the entire state space, giving us a yardstick with which to compare our other state-ordering strategies.

4.3 Comparing Aggregation Strategies

Fig. 8 shows the change in density of the transition matrix for fewest-paths-first, most-successor-states, most-paths-first and random state-selection methods when aggregating all but two states in

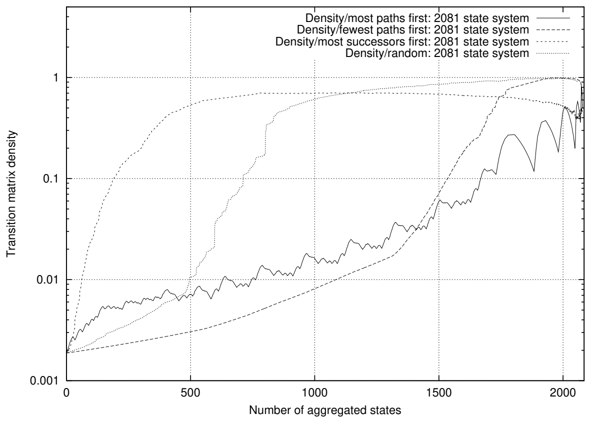

Figure 8: Transition matrix density for the $2\,081$ state model for four different state-selection policies.

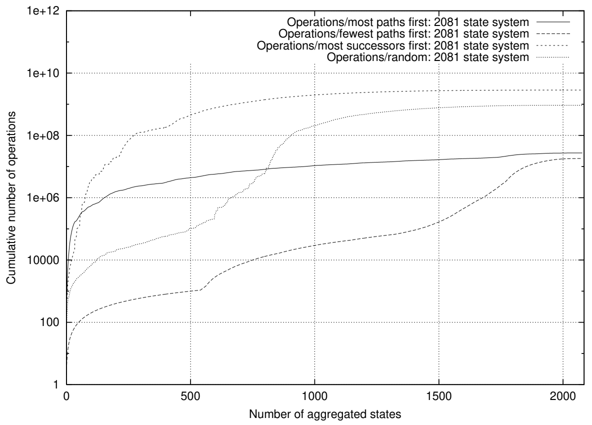

Figure 9: Computational cost for the $2\,081$ state model for four different algorithms.

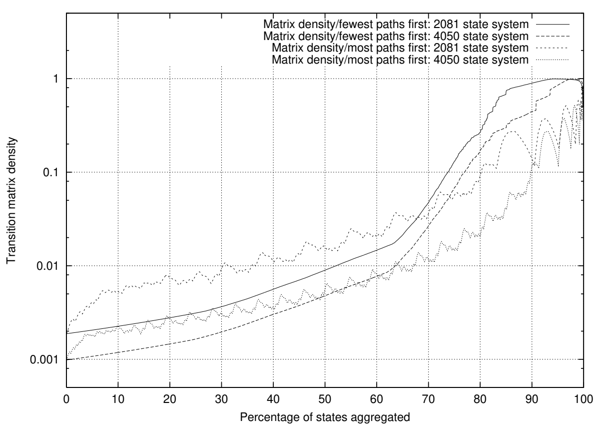

Figure 10: Transition matrix density over two different model sizes and two different algorithms.

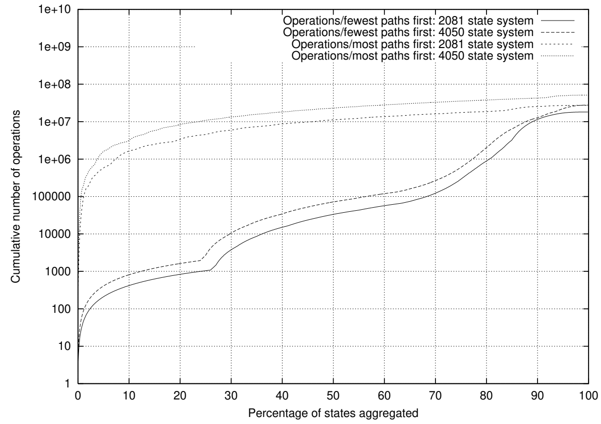

Figure 11: Computational complexity over two different model sizes and two different algorithms.

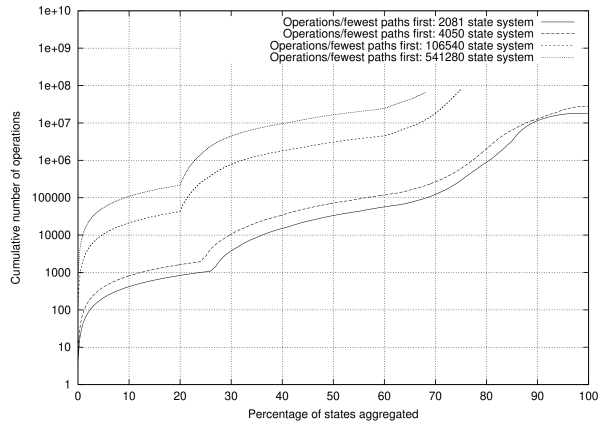

Figure 12: Computational complexity for systems with up to $541\,280$ states; fewest-paths-first algorithm only.

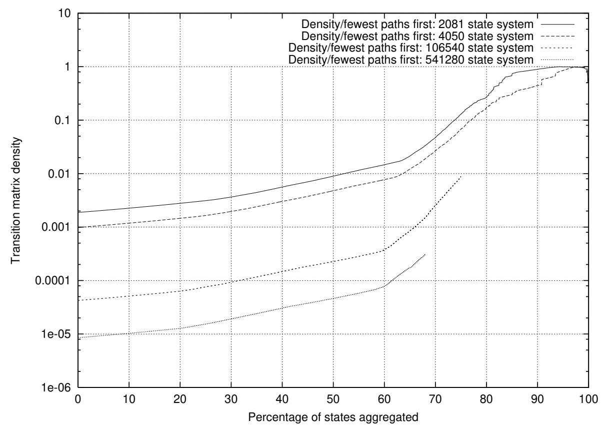

Figure 13: Transition matrix density for systems with up to $541\,280$ states; fewest-paths-first algorithm only.

the 2 081 state model. Note that, as with all other graphs in this section, the results are plotted with a logarithmic y-axis. It can be seen that the fewest-paths-first method provides the lowest density for nearly 75% of the aggregation process. The most-paths-first technique maintains a lower transition matrix density for the last 25% of the process.

The most-successor-states strategy experiences a density explosion early on, reaching 70% fill-in relatively quickly, almost certainly because it generates a large number of new non-zero elements at the start of the process. This is clearly demonstrated by its density profile when compared to that of the random state-selection method for the first half of the state-space. Once 35% of states have been aggregated, however, the desired effect is achieved as the removal of dense rows starts to lower the fill-in.

Fig. 9 shows the cumulative number of convolution, branch elimination and cycle elimination operations taken to aggregate all but two states in the 2 081 state model. The fewest-paths-first policy maintains a very low operations count for the first 70% of the state space. The most-paths-first selection algorithm is in fact linear in the number of states aggregated, as it spreads the work evenly across states and has no computational explosion. Ultimately, the most-paths-first policy performs twice as many operations as the fewest-paths-first algorithm. The most-successor-states policy is also seen to be computationally worse (that is, result in a higher number of operations) than the random policy. For this reason it will not be considered further for the aggregation of larger state spaces.

4.4 Comparing Models of Different Size

In Fig. 10, we compare density profiles over two different sizes of model (2 081 and 4 050 states) for two different aggregation strategies: fewest- and most-paths-first algorithms. For a valid comparison to be made, we plot the density against percentage of state space aggregated. The results show that, after about 75% of states have been aggregated, the transition matrix density is lower for the larger model, right up until complete matrix fill-in is achieved.

The operational cost for different model sizes is shown in Fig. 11 for fewest- and most-paths-first aggregation techniques. For both techniques there is a small increase in the cost when aggregating larger models; but this is dwarfed by the orders-of-magnitude increase that can be seen when using the most-paths-first algorithm over the fewest-paths algorithm.

In the last two figures (Fig. 12 and Fig. 13), we consider only the fewest-paths-first algorithm, as, of all the techniques, it seems to be the one which keeps the computational cost under control the longest whilst simultaneously maintaining matrix sparsity. In addition to the two previous cases, we provide results from the aggregation of SMPs with 106 540 and 541 280 states. Fig. 12 shows the number of operations for all four state spaces, while Fig. 13 plots the density.

Our previous results suggest that our first concern should be over the amount of computation to be done – whilst matrix sparsity remains at acceptable levels when a large proportion of the state space has been aggregated, the number of operations exceeds 10^6 even when small state spaces are aggregated completely. For larger models, therefore, the process is truncated after about 75% of the state space has been aggregated, avoiding the computational explosion. Some success with this truncation can be observed as it limits the required number of aggregation operations for both 106 540 and 541 280 state systems to below 10^8 (this is of the same order as the number of operations required to aggregate the 4 050 state model completely using the most-paths-first method).

Finally, the effect on the density of stopping after aggregating 75% of the state space can be seen in Fig. 13. Again the density of the larger model remains smaller for longer and even for the 106 540 state model only reaches 0.01. Sparse-matrix solution techniques will still function well at such densities, and will benefit greatly from the reduction in the dimensions of the matrix.

5. CONCLUSION AND FUTURE WORK

In this paper we have presented an exact state-by-state aggregation algorithm that can be applied to structurally-unrestricted semi-Markov processes. To complement this, we have developed a constant space representation method for the general distributions of SMPs, which is based on the evaluation demands of numerical Laplace transform inversion techniques. We have applied our algorithm to large SMPs, reducing a 541 280 state model to 173 101 states. While, theoretically, it is possible to reduce every model to just two states, computational cost becomes very large as the transition matrices representing the SMP get less sparse.

It has been shown that the computational cost of our technique is dramatically affected by the order in which the states are aggregated. We devised a number of state-selection strategies and demonstrated that the fewest-paths-first technique (which eliminates the state with the lowest product of predecessor and successor states at each step) was best at postponing the computation explosion during the lifetime of the algorithm. By partially aggregating the state space using the latter method, both the computational cost and matrix density remain low. This yields a substantially smaller, yet still sparse, system on which to perform exact performance analysis. In our experiments we were able to aggregate up to 75% of the state space at low cost. More experimentation over a wider range of models is necessary to determine whether this limit is typical.

We intend to investigate further aggregation possibilities for SMPs. In particular, hybrid combinations of the state-selection strategies are possible. For example, we could envisage a technique which dynamically changes selection algorithm depending on the current density of the transition matrix, in order to achieve a beneficial computation/density trade-off. Further investigation is necessary to determine whether or not this scheme offers any benefits.

Where possible, the partitioning of the transition matrix into loosely coupled submatrices may also prove to be effective. As with individual states, submatrices can have m predecessor and n successors, and thus mn paths running through them. It should, therefore, be possible to aggregate these submatrices all at once by performing the necessary aggregation operations for these mn paths. For low values of m and n (i.e. a loosely coupled state space) this should provide economical state space reductions.

Finally, as the existing aggregation algorithm is iterative and only acts on local portions of the state space, it offers possibilities for efficient parallelisation.

6. 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] J. T. Bradley, "A passage-time preserving equivalence for Semi-Markov Processes," in *TOOLS 2002, Computer Performance Evaluation: Modelling Techniques and Tools* (A. J. Field, P. G. Harrison, J. T. Bradley, and U. Harder, eds.), vol. 2324 of *Lecture Notes in Computer Science*, Springer-Verlag, London, April 2002.

[5] H. C. Bohnenkamp and B. R. Haverkort, "Stochastic event structures for the decomposition of stochastic process algebra models," in *Process Algebra and Performance Modelling Workshop* (J. Hillston and M. Silva, eds.), pp. 25–39, Centro Politécnico Superior de la Universidad de Zaragoza, Prensas Universitarias de Zaragoza, Zaragoza, September 1999.

[6] W. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[7] J. Abate and W. Whitt, "Numerical inversion of Laplace transforms of probability distributions," *ORSA Journal on Computing*, vol. 7, no. 1, pp. 36–43, 1995.

[8] J. Abate, G. L. Choudhury, and W. Whitt, "On the Laguerre method for numerically inverting Laplace transforms," *INFORMS Journal on Computing*, vol. 8, no. 4, pp. 413–427, 1996.

[9] P. G. Harrison and W. J. Knottenbelt, "Passage-time distributions in large Markov chains," in *Proceedings of ACM SIGMETRICS 2002* (M. Martonosi and E. d. S. e Silva, eds.), pp. 77–85, Marina Del Rey, USA, June 2002.

[10] 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 *Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems*, (Nice), IEEE Computer Society Press, April 2003.

[11] 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 processes," in *Proc. Numerical Solution of Markov Chains (NSMC '03)*, 2003.

APPENDIX

A. AGGREGATION FUNCTIONS

This appendix expands on and defines the key functions used by $aggregate\_smp$: $aggregate\_sequences$, $aggregate\_branches$ and $aggregate\_cycles$.

As before, we define the set SMP as the tuple (S, P, L). We also use $PD = (Prob, LaplaceT)$ to be a tuple of a probability and Laplace transform associated with a particular transition. $T = (S \times S) \times PD$, associates states i and j states to a given transition pair.

We define $i \stackrel{M}{\to} j$ to mean that there exists a transition from state i to state j in M.

A.1 Subsidiary functions

$aggregate\_sequences$ takes a state, i, to be aggregated and the SMP and returns a set of transitions which represent all the paths (from one state preceding to one state after i) which pass through i. The paths have been aggregated into single transitions using agg_seq. The number of transitions generated is equal to the product the number transitions leading into i and the number leaving the state.

$aggregate\_sequences : SMP \times S \to \mathcal{P}(T)$
$aggregate\_sequences(M, i) = \{((i, j), agg\_seq(t, t'))$ where
$(i, t) \in trans\_to(i, M), (j, t') \in trans\_from(i, M)\}$

$aggregate\_branches$ processes an SMP, M, and a set of transitions, R, which must not contain any cycles. The transitions are to be integrated into the SMP, M. If there is a pre-existing transition in M for a given member of R, then the two are combined using agg_branch, otherwise the transition in R is just returned unchanged.

$aggregate\_branches : SMP \times \mathcal{P}(T) \to \mathcal{P}(T)$
$aggregate\_branches(M, R) = \{((i, j), agg\_branch(t, t'))$ where
$((i, j), t)) \in R, t' =$ if $i \stackrel{M}{\to} j(M_P(i, j), M_L(i, j))$ else $(0, 0)\}$

$aggregate\_cycles$ processes an SMP, M, and a set of cyclic transitions, R. For each cycle defined in R from i to i, the set of out-transitions is selected from the SMP, M. For each member of that set, the aggregation function agg_cycle is applied. All the modified transitions are unified into a single set and returned.

$aggregate\_cycles : SMP \times \mathcal{P}(T) \to \mathcal{P}(T)$
$aggregate\_cycles(M, R) = \bigcup_{i:((i,i),t') \in R} X$ where
$X = \{((i, j), agg\_cycle(t, t')) \;.\; (j, t) \in trans\_from(i, M)\}$

agg_seq is used to represent two transitions in sequence as a single transition.

$agg\_seq : PD \times PD \to PD$
$agg\_seq((p, d(z)), (p', d'(z))) = (pp', d(z)d'(z))$

agg_branch is used to represent two branching transitions which terminate in the same state as a single transition.

$agg\_branch : PD \times PD \to PD$
$agg\_branch((p, d(z)), (p', d'(z))) = (p + p', f(z))$ where

$$f(z) = p/(p + p')d(z) + p'/(p + p')d'(z)$$

agg_cycle is used to represent a cycle to the same state and a leaving transition as a single transition. The first argument is the leaving transition and the second is the cyclic transition. If there is more than one out-transition then the transformation will need to be applied to each in turn.

$agg\_cycle : PD \times PD \to PD$
$agg\_cycle((p, d(z)), (p_c, d_c(z))) = (p/(1 - p_c), g(z))$ where

$$g(z) = \frac{(1 - p_c)d(z)}{1 - p_c d_c(z)}$$

A.2 Utility functions

For the definitions of $trans\_from$, $trans\_to$, set, $fold$ and $elem\_rest$ see [4].