Aggregation Techniques for Passage Time Calculations in Large semi-Markov Models

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

Imperial College London

June 18, 2009
Generating SMPs

Use high-level models such as process algebras or Petri nets to generate SMPs. Why?

- Can’t design SMPs with millions of states by hand
- Low-level models hard to understand and difficult to reason about
- Can automatically generate kernel from high-level model even if state space is vast
Generating SMPs

Use high-level models such as process algebras or Petri nets to generate SMPs. Why?

- Can’t design SMPs with millions of states by hand
- Low-level models hard to understand and difficult to reason about
- Can automatically generate kernel from high-level model even if state space is vast
Generating SMPs

Use high-level models such as process algebras or Petri nets to generate SMPs. Why?

- Can’t design SMPs with millions of states by hand
- Low-level models hard to understand and difficult to reason about
- Can automatically generate kernel from high-level model even if state space is vast
A simple semi-Markov stochastic Petri net

Marking (2,0,0,0)

Marking (1,1,0,0)
A simple semi-Markov stochastic Petri net

- Marking (2,0,0,0)
- Marking (1,1,0,0)
E-voting: a semi-Markov stochastic Petri net

\[ (t_5, 1.0, 2, 0.8 \times \text{uni}(1.5, 10, s) + 0.2 \times \text{erl}(0.2, 5, s)) \]

\[ (t_3, 0.05, 1, \exp(1.5, s)) \]

\[ (t_4, 1.0, 1, \text{uni}(1.0, 2.0, s)) \]

\[ (t_6, 0.01, 1, \exp(1.0, s)) \]

\[ (t_8, 1.0, 2, \text{gam}(0.1, 12.337, s)) \]

\[ (t_7, 1.0, 1, \text{uni}(2.0, 3.0, s)) \]
Response time analysis in the real world

Response time quantiles in practice: “In 90% of all cases the fire engine arrives in less than 11 minutes”

Figure: NFIRS 5.0 data for 2001 and 2002[1]
Response time analysis in semi-Markov models

- In performance analysis we look at large discrete space continuous time semi-Markov models
- We call the response time the first-passage time
How do we compute the passage time analysis in a SMP

Available methods:

- DES (Discrete event simulation) (approximate)
- Fluid analysis (approximate)
- Solving a set of linear equations in Laplace space (exact)
How do we compute the passage time analysis in a SMP

Available methods:

- DES (Discrete event simulation) (approximate)
- Fluid analysis (approximate)
- Solving a set of linear equations in Laplace space (exact)
How do we compute the passage time analysis in a SMP

Available methods:

- DES (Discrete event simulation) (approximate)
- Fluid analysis (approximate)
- Solving a set of linear equations in Laplace space (exact)
The kernel of a semi-Markov process is defined as

\[ R(n, i, j, t) = P(X_{n+1} = j \land T_{n+1} - T_n \leq t \mid X_n = i) \]

In a time homogeneous SMP this becomes

\[ R(i, j, t) = p_{ij} H_{ij}(t) \]
First-passage time distribution

Passage time random variable

\[ P_{s\vec{t}} = \inf\{u > 0 \mid Z(u) \in \vec{t} \land Z(0) = s\} \]

Passage time CDF

\[ F_{s\vec{t}}(t_1) = P(P_{s\vec{t}} < t_1) = \int_0^{t_1} f_{s\vec{t}}(t) \, dt \]
Exact numerical calculation of first-passage time distribution

- Convolutions are hard to compute in real space → in Laplace space it is straightforward
- Use numerical Laplace inversion to retrieve $f_{st}(t)$
- Memory efficient as we only need to compute one Laplace transform sample at a time
- Computation of $L_{st}(x)$ for a single $x$-point entails solving $|S|$ linear equations in Laplace space
- There exists a simple and efficient iterative solver
Convolutions are hard to compute in real space $\rightarrow$ in Laplace space it is straightforward

Use numerical Laplace inversion to retrieve $f_{st}(t)$

Memory efficient as we only need to compute one Laplace transform sample at a time

Computation of $L_{st}(x)$ for a single $x$-point entails solving $|S|$ linear equations in Laplace space

There exists a simple and efficient iterative solver
Exact numerical calculation of first-passage time distribution

- Convolutions are hard to compute in real space → in Laplace space it is straightforward
- Use numerical Laplace inversion to retrieve $f_{st}(t)$
- Memory efficient as we only need to compute one Laplace transform sample at a time
  - Computation of $L_{st}(x)$ for a single x-point entails solving $|S|$ linear equations in Laplace space
- There exists a simple and efficient iterative solver
Convolutions are hard to compute in real space → in Laplace space it is straightforward

Use numerical Laplace inversion to retrieve $f_{st}(t)$

Memory efficient as we only need to compute one Laplace transform sample at a time

Computation of $L_{st}(x)$ for a single x-point entails solving $|S|$ linear equations in Laplace space

There exists a simple and efficient iterative solver
Convolutions are hard to compute in real space → in Laplace space it is straightforward

Use numerical Laplace inversion to retrieve $f_{st}(t)$

Memory efficient as we only need to compute one Laplace transform sample at a time

Computation of $L_{st}(x)$ for a single $x$-point entails solving $|S|$ linear equations in Laplace space

There exists a simple and efficient iterative solver
Weighted Laplace transform of latency pdf of transition from $i$ to $k$

$$r_{ik}^*(x) = \int_0^{\infty} e^{-xt} dR(i, k, t) \quad (1)$$

The Laplace transform of $f_{st}(t)$ is

$$L_{st}(x) = \sum_{k \in S \setminus t} r_{sk}^*(x) L_{kt}(x) + \sum_{k \in t} r_{sk}^*(x), \quad 1 \leq t \leq |S| \quad (2)$$
High-level models easily generate SMPs with several million states and transitions.

**Problem**

Large sparse adjacency matrices do not fit into memory of a single computer. Even current parallel solvers can do iterative passage time analysis on SMPs with maximum 50 million states.

**Solution**

Aggregate intermediate states to reduce size of the matrix.
Problems with FPTA in SMPs

High-level models easily generate SMPs with several million states and transitions.

**Problem**

Large sparse adjacency matrices do not fit into memory of a single computer. Even current parallel solvers can do iterative passage time analysis on SMPs with maximum 50 million states.

**Solution**

Aggregate intermediate states to reduce size of the matrix.
High-level models easily generate SMPs with several million states and transitions.

**Problem**

Large sparse adjacency matrices do not fit into memory of a single computer. Even current parallel solvers can do iterative passage time analysis on SMPs with maximum 50 million states.

**Solution**

Aggregate intermediate states to reduce size of the matrix
Exact state-by-state aggregation

Problem

- Naïve exact state-by-state aggregation reduces the number of intermediate states
- It does not affect the passage time distribution from source to state
- BUT causes an explosion in the number of transitions[2]
Exact state-by-state aggregation

Problem

- Naïve exact state-by-state aggregation reduces the number of intermediate states
- It does not affect the passage time distribution from source to state
- BUT causes an explosion in the number of transitions[2]
Naïve exact state-by-state aggregation reduces the number of intermediate states.

It does not affect the passage time distribution from source to state.

BUT causes an explosion in the number of transitions[2].
Alternative approach
Aggregation of partitions vs. normal (flat) aggregation

Guenther, Dingle, Bradley, Knottenbelt
BUT

- Iterative first passage time solver is much faster than the combined aggregation and passage time computation approach.
  - To keep density peaks low, we have to create many small partitions.
  - In larger SMPs, it was not possible to find good partitionings.

→ Exact state-by-state aggregation is not scalable.
BUT

- Iterative first passage time solver is much faster than the combined aggregation and passage time computation approach.
- To keep density peaks low we have to create many small partitions.
- In larger SMPs it was not possible to find good partitionings.

→ Exact state-by-state aggregation is not scalable.
BUT

- Iterative first passage time solver is much faster than the combined aggregation and passage time computation approach
- To keep density peaks low we have to create many small partitions
- In larger SMPs it was not possible to find good partitionings

→ Exact state-by-state aggregation is not scalable
BUT

- Iterative first passage time solver is much faster than the combined aggregation and passage time computation approach.
- To keep density peaks low we have to create many small partitions.
- In larger SMPs it was not possible to find good partitionings.

→ **Exact state-by-state aggregation is not scalable.**
Idea: Use passage time analyser to aggregate an entire partition in one go. This is called RFPTA.
Atomic partition aggregation works well if we can find large suitable partitions in the SMP

- Partitions need low number of predecessor or low number of successor states otherwise RFPTA aggregation becomes expensive
- We are not guaranteed to find such partitions

Solution
Find better ways of partitioning the state space of the SMP for passage time analysis
• Atomic partition aggregation works well if we can find large suitable partitions in the SMP

• Partitions need low number of predecessor or low number of successor states otherwise RFPTA aggregation becomes expensive

• We are not guaranteed to find such partitions

Solution

Find better ways of partitioning the state space of the SMP for passage time analysis
Atomic partition aggregation works well if we can find large suitable partitions in the SMP

Partitions need low number of predecessor or low number of successor states otherwise RFPTA aggregation becomes expensive

We are not guaranteed to find such partitions

Solution

Find better ways of partitioning the state space of the SMP for passage time analysis
Atomic partition aggregation works well if we can find large suitable partitions in the SMP

Partitions need low number of predecessor or low number of successor states otherwise RFPTA aggregation becomes expensive

We are not guaranteed to find such partitions

Solution

Find better ways of partitioning the state space of the SMP for passage time analysis
We can show that

\[ L_{st} = \sum_{b \in \bar{b}} L_{sb}^R L_{bt} \]

and

\[ L_{st} = \sum_{b \in \bar{b}} L_{sb}^R L_{bt} = L_{sb}^R \cdot L_{bt} \]
We can show that

\[ L_{s\bar{t}} = \sum_{b \in \bar{b}} L^R_{sb} L_{bt} \]

and

\[ L_{\bar{s}\bar{t}} = \sum_{b \in \bar{b}} L^R_{sb} L_{bt} = L^R_{s\bar{b}} \cdot L_{\bar{b}t} \]
Barrier partitioning for passage time analysis

We can show that

\[ L_{s \bar{t}} = \sum_{b \in \bar{b}} L^R_{sb} L_{bt} \]

and

\[ L_{\bar{s} \bar{t}} = \sum_{b \in \bar{b}} L^R_{\bar{s}b} L_{\bar{b}t} = L^R_{\bar{s}b} \cdot L_{\bar{b}t} \]
Advantages:

- Balanced barrier partitioning has linear time complexity
- $L^R_{sb}$, $L^{-bt}$ can be computed separately and hence we only need to store half of the matrix in memory
- First-passage time computation on barrier partitionings is exact
Advantages:

- Balanced barrier partitioning has linear time complexity
- $L^R_{sb}$, $L_{bt}$ can be computed separately and hence we only need to store half of the matrix in memory
- First-passage time computation on barrier partitionings is exact
Advantages:

- Balanced barrier partitioning has linear time complexity
- $L^R_{sb}, L^R_{bt}$ can be computed separately and hence we only need to store half of the matrix in memory
- First-passage time computation on barrier partitionings is exact
K-way barrier partitioning

- Also allows exact passage time calculation
- Reduces memory requirements of the numerical solver by up to 99%
- K-way barrier partitioning still only requires linear amount of time
K-way barrier partitioning

- Also allows exact passage time calculation
- Reduces memory requirements of the numerical solver by up to 99%
- K-way barrier partitioning still only requires linear amount of time
K-way barrier partitioning

- Also allows exact passage time calculation
- Reduces memory requirements of the numerical solver by up to 99%
- K-way barrier partitioning still only requires linear amount of time
Figure: 6-way barrier partitioning of 1,100,000 states e-voting model (CC=175, MM=45, NN=5)
Our new algorithm can perform FPTA on large SMPs using only a fraction of the memory required by SMARTA.

Future work

- Can partitioning and aggregation techniques be used to improve the computation of other performance metrics such as the transient-state or the steady-state distribution?
- Can we build a new SMARTA pipeline to do iterative passage time analysis on a SMP with a billion states?
Questions?
Figure: 6-way barrier partitioning of 1,100,000 states e-voting model (CC=175, MM=45, NN=5)
Extras for Q&A: Doing passage time analysis on a k-way barrier partitioning
Extras for Q&A: Doing passage time analysis on a k-way barrier partitioning
Extras for Q&A: Doing passage time analysis on a k-way barrier partitioning
Extras for Q&A: Doing passage time analysis on a k-way barrier partitioning
Extras for Q&A: Doing passage time analysis on a k-way barrier partitioning
The set of $|S| = N$ linear equations (2) needed to compute $L_{s\bar{t}}(x)$, for all $s \in S$ can be written in matrix form as follows [3, 4]

$$
\begin{pmatrix}
1 & -r_{12}^*(x) & \ldots & -r_{1N}^*(x) \\
0 & 1 - r_{22}^*(x) & \ldots & -r_{2N}^*(x) \\
0 & -r_{32}^*(x) & \ldots & -r_{3N}^*(x) \\
\vdots & \vdots & \ddots & \vdots \\
0 & -r_{N2}^*(x) & \ldots & 1 - r_{NN}^*(x)
\end{pmatrix}
\begin{pmatrix}
L_{1\bar{t}}(x) \\
L_{2\bar{t}}(x) \\
L_{3\bar{t}}(x) \\
\vdots \\
L_{N\bar{t}}(x)
\end{pmatrix}
= 
\begin{pmatrix}
r_{1\bar{t}}^*(x) \\
r_{2\bar{t}}^*(x) \\
r_{3\bar{t}}^*(x) \\
\vdots \\
r_{N\bar{t}}^*(x)
\end{pmatrix}
$$

where $r_{st\bar{t}}^*(x) = \sum_{k \in \bar{t}} r_{sk}^*(x)$
Bibliography


J. T. Bradley, N. J. Dingle, P. G. Harrison, W. J. Knottenbelt

*Distributed Computation of Passage Time Quantiles and Transient State Distributions in Large Semi-Markov Models*

Proceedings of the 17th International Symposium on Parallel and Distributed Processing, p.281.1
April 22-26, 2003