SZAKDOLGOZAT FELADAT

Marussy Kristóf
mérnök informatikus hallgató részére

Konfigurálható numerikus módszerek sztochasztikus modellekhez

A kritikus rendszerek – biztonságkritikus, elosztott és felhő-alapú alkalmazások – helyességének biztosításához szükséges a funkcionális és nemfunkcionális követelmények matematikai igényességű ellenőrzése. Számos, szolgáltatásbiztonsággal és teljesítményvizsgálattal kapcsolatos tipikus kérdés általában sztochasztikus analízis segítségével válaszolható meg.

A kritikus rendszerek elosztott és aszinkron tulajdonságai az állapottér robbanás jelenségéhez vezetnek. Emiatt méretük és komplexitásuk gyakran megakadályozza a sikeres sztochasztikus analízist, melynek számításigénye nagyban függ a lehetséges viselkedések számától. A modellek komponenseinek jellegzetes időbeli viselkedése és leginkább eltérő karakterisztikája a számításigény további jelentős növekedését okozhatja.

A szolgáltatásbiztonsági és teljesítményjellemzők kiszámítása markovi modellek állandósult állapotbeli és tranziens megoldását igényli. Számos eljárás ismert ezen problémák kezelésére, melyek eltérő reprezentációkat és numerikus algoritmusokat alkalmaznak; ám a modellek változatos tulajdonságai miatt nem választható ki olyan eljárás, mely minden esetben hatékony lenne. A hallgató feladata áttekinteni az irodalmat és megvizsgálni az ismert algoritmusokat.

A feladat megoldása a következő lépésekből áll:

1. Mutassa be az irodalomban ismert, markovi sztochasztikus rendszerek állandósult állapotbeli és tranziens viselkedésének vizsgálatára alkalmas numerikus algoritmusokat.
2. Az irodalom alapján implementáljon kiválasztott tranziens és állandósult állapotbeli analízis algoritmusokat.
3. Hasonlítsa össze futási idő és tárhely komplexitás szempontjából az implementált algoritmusokat.
4. Értékelje a megoldást és vizsgálja meg a továbbfejlesztési lehetőségeket.

Tanszéki konzulens: Vörös András, tudományos segédmunkatárs
Molnár Vince, doktorandusz
Külső konzulens: dr. Telek Miklós, egyetemi tanár

Budapest, 2015. október 7.

Dr. Jobbágy Ákos
egyetemi tanár
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057, Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Kristóf Marussy

Configurable Numerical Solutions for Stochastic Models

BSc Thesis

Supervisors:

dr. Miklós Telek
Vince Molnár
András Vörös

Budapest, 2015

Contents

Összefoglaló TODO

A kritikus rendszerek – biztonságkritikus, elosztott és felhőalkalmazások – helyességének biztosításához szükséges a funkcionális és nemfunkcionális követelmények matematikai igényességű ellenőrzése. Számos, szolgáltatásbiztonsággal és teljesítményvizsgálattal kapcsolatos tipikus kérdés általában sztochasztikus analízis segítségével válaszolható meg.

A kritikus rendszerek elosztott és aszinkron tulajdonságai az *állapottér robbanás* jelenségéhez vezetnek. Emiatt méretük és komplexitásuk gyakran megakadályozza a sikeres sztochasztikus analízist, melynek számításigénye nagyban függ a lehetséges viselkedések számától. A modellek komponenseinek jellegzetes időbeli viselkedése a számításigény további jelentős növekedését okozhatja.

A szolgáltatásbiztonsági és teljesítményjellemzők kiszámítása markovi modellek állandósult állapotbeli és tranziens megoldását igényli. Számos eljárás ismert ezen problémák kezelésére, melyek eltérő reprezentációkat és numerikus algoritmusokat alkalmaznak; ám a modellek változatos tulajdonságai miatt nem választható ki olyan eljárás, mely minden esetben hatékony lenne.

A markovi analízishez szükséges a modell lehetséges viselkedéseinek, azaz állapotterének felderítése, illetve tárolása, mely szimbolikus módszerekkel hatékonyan végezhető el. Ezzel szemben a sztochasztikus algoritmusokban használt vektor- és indexműveletek szimbolikus megvalósítása nehézkes. Munkánk célja egy olyan, integrált keretrendszer fejlesztése, mely lehetővé teszi a komplex sztochasztikus rendszerek kezelését a szimbolikus módszerek, hatékony mátrix reprezentációk és numerikus algoritmusok előnyeinek ötvözésével.

Egy teljesen szimbolikus algoritmust javasolunk a sztochasztikus viselkedéseket leíró mátrix-dekompozíciók előállítására a szimbolikus formában adott állapottérből kiindulva. Ez az eljárás lehetővé teszi a temporális logikai kifejezéseken alapuló szimbolikus technikák használatát.

A keretrendszerben megvalósítottuk a konfigurálható sztochasztikus analízist: megközelítésünk lehetővé teszi a különböző mátrix reprezentációk és numerikus algoritmusok kombinált használatát. Az implementált algoritmusokkal állandósult állapotbeli költség- és érzékenység analízis, tranziens költséganalízis és első hiba várható bekövetkezési idő analízis végezhető el sztochasztikus Petri-háló (SPN) alapú markovi költségmodelleken. Az elkészített eszközt integráltuk a PetriDotNet modellező szoftverrel. Módszerünk gyakorlati alkalmazhatóságát szintetikus és ipari modelleken végzett mérésekkel igazoljuk.

Abstract Ensuring the correctness of critical systems – such as safety-critical, distributed and cloud applications – requires the rigorous analysis of the functional and extra-functional properties of the system. A large class of typical quantitative questions regarding dependability and performability are usually addressed by stochastic analysis.

Recent critical systems are often distributed/asynchronous, leading to the well-known phenomenon of *state space explosion*. The size and complexity of such systems often prevents the success of the analysis due to the high sensitivity to the number of possible behaviors. In addition, temporal characteristics of the components can easily lead to huge computational overhead.

Calculation of dependability and performability measures can be reduced to steady-state and transient solutions of Markovian models. Various approaches are known in the literature for these problems differing in the representation of the stochastic behavior of the models or in the applied numerical algorithms. The efficiency of these approaches are influenced by various characteristics of the models, therefore no single best approach is known.

The prerequisite of Markovian analysis is the exploration of the state space, i.e. the possible behaviors of the system. Symbolic approaches provide an efficient state space exploration and storage technique, however their application to support the vector operations and index manipulations extensively used by stochastic algorithms is cumbersome. The goal of our work is to introduce a framework that facilitates the analysis of complex, stochastic systems by combining the advantages of symbolic algorithms, compact matrix representations and various numerical algorithms.

We propose a fully symbolic method to explore and describe the stochastic behaviors. A new algorithm is introduced to transform the symbolic state space representation into a decomposed linear algebraic representation. This approach allows leveraging existing symbolic techniques, such as the specification of properties with *Computational Tree Logic* (CTL) expressions.

The framework provides configurable stochastic analysis: an approach is introduced to combine the different matrix representations with numerical solution algorithms. Various algorithms are implemented for steady-state reward and sensitivity analysis, transient reward analysis and mean-time-to-first-failure analysis of stochastic models in the *Stochastic Petri Net* (SPN) based Markov reward model formalism. The analysis tool is integrated into the PetriDotNet modeling application. Benchmarks and industrial case studies are used to evaluate the applicability of our approach.

Hallgatói nyilatkozat

Alulírott **Marussy Kristóf** szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2015. december 6.

.....
Marussy Kristóf

Chapter 1

Introduction

Chapter 2

Background

In this section we overview the basic formalisms and scope of our work. **TODO**

2.1 Continuous-time Markov chains

Continuous-time Markov chains are mathematical tools for describing the behavior of systems in countinous time where the stochastic behavior of the system only depends on its current state.

> **Definition 2.1** A *Continuous-time Markov Chain* (CTMC) $X(t) \in S, t \geq 0$ over the finite state space $S = \{0, 1, \ldots, n-1\}$ is a continuous-time random process with the *Markovian* or memoryless property:
>
> $$\begin{aligned}\mathbb{P}(X(t_k) = x_k \mid X(t_{k-1}) = x_{k-1}, X(t_{k-2}) = x_{k-2}, \ldots, X(t_0) = x_0) \\ = \mathbb{P}(X(t_k) = x_k \mid X(t_{k-1}) = x_{k-1}),\end{aligned}$$
>
> where $t_0 \leq t_1 \leq \cdots \leq t_k$ and $X(t_k)$ is a random variable denoting the current state of the CTMC at time t_k. A CTMC is said to be *time-homogenous* if it also satisfies
>
> $$\mathbb{P}(X(t_k) = x_k \mid X(t_{k-1}) = x_{k-1}) = \mathbb{P}(X(t_k - t_{k-1}) = x_k \mid X(0) = x_{k-1}),$$
>
> i.e. it is invariant to time shifting.

In this report we will restrict our attention to time-homogenous CTMCs over finite state spaces. The state probabilities of these stochastic processes at time t form a finite-dimensional vector $\boldsymbol{\pi}(t) \in \mathbb{R}^n$,

$$\pi(t)[x] = \mathbb{P}(X(t) = x)$$

that satisfies the differential equation

$$\frac{\mathrm{d}\boldsymbol{\pi}(t)}{\mathrm{d}t} = \boldsymbol{\pi}(t)Q \tag{2.1}$$

for some square matrix Q. The matrix Q is called the *infinitesimal generator matrix* of the CTMC and can be interpreted as follows:

- The diagonal elements $q[x,x] < 0$ describe the holding times of the CTMC. If $X(t) = x$, the *holding time* $h_x = \inf\{h > 0 : X(t) = x, X(t+h) \neq x\}$ spent in state x is exponentially distributed with rate $\lambda_x = -q[x,x]$. If $q[x,x] = 0$, then no transitions are possible from state x and it is said to be *absorbing*.
- The off-diagonal elements $q[x,y] \geq 0$ describe the state transitions. In state x the CTMC will jump to state y at the next state transition with probability $-q[x,y]/q[x,x]$. Equivalently, there is expontentially distributed countdown in the state x for each $y : q[x,y] > 0$ with *transition rate* $\lambda_{xy} = q[x,y]$. The first countdown to finish will trigger a state change to the corresponding state y. Thus, the CTMC is a transition system with exponentially distributed timed transitions.
- Elements in each row of Q sum to 0, hence it satisfies $Q\mathbf{1}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}}$.

For more algebraic properties of infinitesimal generator matrices, we refer to Plemmons and Berman [49] and Stewart [64].

A state y is said to be *reachable* from the state x ($x \rightsquigarrow y$) if there exists a sequence of states

$$x = z_1, z_2, z_3, \ldots, z_{k-1}, z_k = y$$

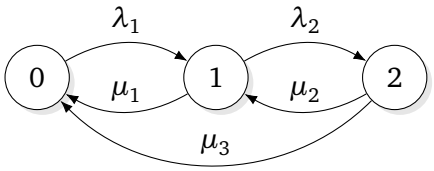
such that $q[z_i, z_{i+1}] > 0$ for all $i = 1, 2, \ldots, k-1$. If y is reachable from x for all $x, y \in S$ y, the Markov chain is said to be *irreducible*.

The *steady-state probability distribution* $\boldsymbol{\pi} = \lim_{t\to\infty} \boldsymbol{\pi}(t)$ exists and is independent from the *initial distribution* $\boldsymbol{\pi}(0) = \boldsymbol{\pi}_0$ if and only if the finite CTMC is irreducible. The steady-state distribution satisfies the linear equation

$$\frac{\mathrm{d}\boldsymbol{\pi}}{\mathrm{d}t} = \boldsymbol{\pi}Q = \mathbf{0}, \quad \boldsymbol{\pi}\mathbf{1}^{\mathrm{T}} = 1. \tag{2.2}$$

Example 2.1 Figure 2.1 shows a CTMC with 3 states. The transitions from state 0 to 1 and from 1 to 2 are associated with exponentially distributed countdowns with rates λ_1 and λ_2 respectively, while transitions in the reverse direction have rates μ_1 and μ_2. The transition form state 2 to 0 is also possible with rate μ_3.

The rows (corresponding to source states) and columns (destination states) of the infinitesimal generator matrix Q are labeled with the state numbers. The diagonal element $q[1,1]$ is $-\lambda_2 - \mu_1$, hence the holding time in state 1 is exponentially



$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{pmatrix} -\lambda_1 & \lambda_1 & 0 \\ \mu_1 & -\lambda_2 - \mu_1 & \lambda_2 \\ \mu_3 & \mu_2 & -\mu_2 - \mu_3 \end{pmatrix} \end{array}$$

Figure 2.1 Example CTMC with 3 states and its generator matrix.

distributed with rate $\lambda_2+\mu_1$. The transition from state 1 to 0 is taken with probability $-q[1,0]/q[1,1] = \mu_1/(\lambda_2+\mu_1)$, while the transition to 2 is taken with probability $\lambda_2/(\lambda_2+\mu_1)$.

The CTMC is irreducible, because every state is reachable from every other state. Therefore, there is a unique steady-state distribution $\boldsymbol{\pi}$ independent from the initial distribution $\boldsymbol{\pi}_0$.

2.1.1 Markov reward models

Continuous-time Markov chains may be employed in the estimation of performance measures of models by defining *rewards* that associate *reward rates* with the states of a CTMC. The reward rate random variable $R(t)$ can describe performance measures defined at a single point of time, such as resource utilization or probability of failure, while the *accumulated reward* random variable $Y(t)$ may correspond to performance measures associated with intervals of time, such as total downtime.

Definition 2.2 A *Continuous-time Markov Reward Process* over a finite state space $S = \{0, 1, \ldots, n-1\}$ is a pair $(X(t), \mathbf{r})$, where $X(t)$ is a CTMC over S and $\mathbf{r} \in \mathbb{R}^n$ is a *reward rate vector*.

The element $r[x]$ of the reward vector is a momentary reward rate in state x, therefore the reward rate random variable can be written as $R(t) = r[X(t)]$. The accumulated reward until time t is defined by

$$Y(t) = \int_0^t R(\tau)\,\mathrm{d}\tau.$$

The computation of the distribution function of Y(t) is a computationally intensive task (a summary is available at [51, Table 1]), while its mean, $\mathbb{E}\,Y(t)$, can be computed efficiently as discussed below.

Given the initial probability distribution vector $\boldsymbol{\pi}(0) = \boldsymbol{\pi}_0$ the expected value of the

reward rate at time t can be calculated as

$$\mathbb{E}R(t) = \sum_{i=0}^{n-1} \pi(t)[i]r[i] = \boldsymbol{\pi}(t)\,\mathbf{r}^{\mathrm{T}}, \tag{2.3}$$

which requires the solution of the initial value problem [32, 54]

$$\frac{\mathrm{d}\boldsymbol{\pi}(t)}{\mathrm{d}t} = \boldsymbol{\pi}(t)Q, \quad \boldsymbol{\pi}(0) = \boldsymbol{\pi}_0 \tag{2.4}$$

to form the inner product $\mathbb{E}R(t) = \boldsymbol{\pi}(t)\,\mathbf{r}^{\mathrm{T}}$.

To obtain the expected steady-state reward rate (if it exists) the linear equation (2.2) should be solved instead of eq. (2.4) in order to acquire the steady-state probability vector $\boldsymbol{\pi}$. The computation of the reward value proceeds by eq. (2.3) in the same way as in transient analysis.

The expected value of the accumulated reward is

$$\begin{aligned}
\mathbb{E}\,Y(t) &= \mathbb{E}\left[\int_0^t R(\tau)\,\mathrm{d}\tau\right] = \int_0^t \mathbb{E}[R(\tau)]\,\mathrm{d}\tau \\
&= \int_0^t \sum_{i=0}^{n-1} \pi(\tau)[i]r[i]\,\mathrm{d}\tau = \sum_{i=0}^{n-1} \int_0^t \pi(\tau)[i]\,\mathrm{d}\tau\, r[i] \\
&= \int_0^t \boldsymbol{\pi}(t)\,\mathrm{d}\tau\,\mathbf{r}^{\mathrm{T}} = \mathbf{L}(t)\,\mathbf{r}^{\mathrm{T}},
\end{aligned}$$

where $\mathbf{L}(t) = \int_0^t \boldsymbol{\pi}(t)\,\mathrm{d}\tau$ is the accumulated probability vector, which is the solution of the initial value problem [54]

$$\frac{\mathrm{d}\mathbf{L}(t)}{\mathrm{d}t} = \boldsymbol{\pi}(t), \quad \frac{\mathrm{d}\boldsymbol{\pi}(t)}{\mathrm{d}t} = \boldsymbol{\pi}(t)Q, \quad \mathbf{L}(0) = \mathbf{0}, \quad \boldsymbol{\pi}(0) = \boldsymbol{\pi}_0. \tag{2.5}$$

Example 2.2 Let c_0, c_1 and c_2 denote operating costs per unit time associated with the states of the CTMC in Figure 2.1. Consider the Markov reward process $(X(t), \mathbf{r})$ with reward rate vector

$$\mathbf{r} = \begin{pmatrix} c_0 & c_1 & c_2 \end{pmatrix}.$$

The random variable $R(t)$ describes the momentary operating cost, while $Y(t)$ is the total operating expenditure until time t. The steady-state expectation of R is the average maintenance cost per unit time of the long-running system.

2.1.2 Sensitivity

Sensitivity analysis is widely used to assess the robustnes of information systems. Consider a reward process $(X(t), \mathbf{r})$ where both the infinitesimal generator matrix $Q(\theta)$

and the reward rate vector $\mathbf{r}(\theta)$ may depend on some *parameters* $\theta \in \mathbb{R}^m$. The *sensitivity* analysis of the rewards $R(t)$ may reveal performance or reliability bottlenecks of the modeled system and help designers in achieving desired performance measures and robustnes values.

Definition 2.3 The *sensitivity* of the expected reward rate $\mathbb{E}R(t)$ to the parameter $\theta[i]$ is the partial derivative

$$\frac{\partial\, \mathbb{E}R(t)}{\partial\, \theta[i]}.$$

Considering parameters with high absolute sensitivity the model reacts to the changes of those parameters more prominently, therefore they can be promising directions of system optimization.

To calculate the sensivity of $\mathbb{E}R(t)$, the partial derivative of both sides of eq. (2.3) is taken, yielding

$$\frac{\partial\, \mathbb{E}R(t)}{\partial\, \theta[i]} = \frac{\partial \boldsymbol{\pi}(t)}{\partial\, \theta[i]}\mathbf{r}^{\mathrm{T}} + \boldsymbol{\pi}(t)\left(\frac{\partial \mathbf{r}}{\partial\, \theta[i]}\right)^{\mathrm{T}} = \mathbf{s}_i(t)\,\mathbf{r}^{\mathrm{T}} + \boldsymbol{\pi}(t)\left(\frac{\partial \mathbf{r}}{\partial\, \theta[i]}\right)^{\mathrm{T}},$$

where $\mathbf{s}_i$ is the sensitivity of $\boldsymbol{\pi}$ to the parameter $\theta[i]$.

In transient analysis, the sensitivity vector $\mathbf{s}_i$ is the solution of the initial value problem

$$\frac{\mathrm{d}\mathbf{s}_i(t)}{\mathrm{d}t} = \mathbf{s}_i(t)Q + \boldsymbol{\pi}(t)V_i, \quad \frac{\mathrm{d}\boldsymbol{\pi}(t)}{\mathrm{d}t} = \boldsymbol{\pi}_i(t)Q, \quad \mathbf{s}_i(0) = \mathbf{0}, \quad \boldsymbol{\pi}(0) = \boldsymbol{\pi}_0,$$

where $V_i = \partial Q(\boldsymbol{\theta})/\partial\, \theta[i]$ is the partial derivative of the generator matrix [52]. A similar initial value problem can be derived for the sensitivity of $\mathbf{L}(t)$ and $Y(t)$.

To obtain the sensitivity $\mathbf{s}_i$ of the steady-state probability vector $\boldsymbol{\pi}$, the system of linear equations

$$\mathbf{s}_i Q = -\boldsymbol{\pi} V_i, \quad \mathbf{s}_i \mathbf{1}^T = 0 \tag{2.6}$$

is solved [10].

Another type of sensitivity analysis considers *unstructured* small perturbations of the infinitesimal generator matrix Q instead of dependecies on parameters [30, 38]. This latter, unstructured analysis may be used to study the numerical stability and conditioning of the solutions of the Markov chain.

2.1.3 Time to first failure

Computing the first time of a system failure (provided it was fully operational when it was started) has many applications in reliability engineering.

Let $D \subsetneq S$ be a set of *failure states* of the CTMC $X(t)$ and $U = S \setminus D$ be a set of operating states. We will assume without loss of generality that $U = \{0, 1, \ldots, n_U - 1\}$ and $D = \{n_U, n_U + 1, \ldots, n - 1\}$.

The matrix

$$Q_{UD} = \begin{pmatrix} Q_{UU} & \mathbf{q}_{UD}^{\mathrm{T}} \\ \mathbf{0} & 0 \end{pmatrix}$$

is the infinitesimal generator of a CTMC $X_{UD}(t)$ in which all the failures states D were merged into a single state n_U and all outgoing transitions from D were removed. The matrix Q_{UU} is the $n_U \times n_U$ upper left submatrix of Q, while the vector $\mathbf{q}_{UD} \in \mathbb{R}^{n_U}$ is defined as

$$q_{UD}[x] = \sum_{y \in D} q[x, y].$$

If the initial distribution $\boldsymbol{\pi}_0$ is 0 for all failure states (i.e. $\pi_0[x] = 0$ for all $x \in D$), the *Time to First Failure*

$$TFF = \inf\{t \geq 0 : X(t) \in D\} = \inf\{t \geq 0 : X_{UD}(t) = n_U\}$$

is *phase-type distributed* with parameters $(\boldsymbol{\pi}_U, Q_{UU})$ [47], where $\boldsymbol{\pi}_U$ is the vector containing the first n_U elements of $\boldsymbol{\pi}_0$. In particular, the *Mean Time to First Failure* is computed as follows:

$$MTFF = \mathbb{E}[TFF] = -\boldsymbol{\pi}_U Q_{UU}^{-1} \mathbf{1}^{\mathrm{T}}. \tag{2.7}$$

The probability of a D'-mode failure ($D' \subset D$) is

$$\mathbb{P}(X(TFF_{+0}) = y) = -\boldsymbol{\pi}_U Q_{UU}^{-1} \mathbf{q}_{UD'}^{\mathrm{T}}, \tag{2.8}$$

where $\mathbf{q}_{UD'} \in \mathbb{R}^{n_U}$, $q_{UD'}[x] = \sum_{y \in D'} q[x, y]$ is the vector of transition rates from operational states to failure states D'.

2.2 Kronecker algebra

Definition 2.4 The *Kronecker product* of matrices $A \in \mathbb{R}^{n_1 \times m_1}$ and $B \in \mathbb{R}^{n_2 \times m_2}$ is the matrix $C = A \otimes B \in \mathbb{R}^{n_1 n_2 \times m_1 m_3}$, where

$$c[i_1 n_1 + i_2, j_1 m_1 + j_2] = a[i_1, j_1] b[i_2, j_2].$$

Some properties of the Kroncker product are

1. Associativity:

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C,$$

which makes Kronecker products of the form $A^{(0)} \otimes A^{(1)} \otimes \cdots \otimes A^{(J-1)}$ well-defined.

2. Distributivity over matrix addition:

$$(A+B)\otimes(C+D)=A\otimes C+B\otimes C+A\otimes D+B\otimes D,$$

3. Compatibility with ordinary matrix multiplication:

$$(AB)\otimes(CD)=(A\otimes C)(B\otimes D),$$

in particular,

$$A\otimes B=(A\otimes I_2)(I_1\otimes B)$$

for identity matrices I_1 and I_2 with appropriate dimensions.

We will occasionally employ multi-index notation to refer to elements of Kronecker product matrices. For example, we will write

$$b[\mathbf{x},\mathbf{y}]=b[(x^{(0)},x^{(1)},\dots,x^{(J-1)}),(y^{(0)},y^{(1)},\dots,y^{(J-1)})]= a^{(0)}[x^{(0)},y^{(0)}]a^{(1)}[x^{(1)},y^{(1)}]\cdots a^{(J-1)}[x^{(J-1)},y^{(J-1)}],$$

where $\mathbf{x}=(x^{(0)},x^{(1)},\dots,x^{(J-1)})$, $\mathbf{y}=(y^{(0)},y^{(1)},\dots,y^{(J-1)})$ and B is the J-way Kronecker product $A^{(0)}\otimes A^{(1)}\otimes\cdots\otimes A^{(J-1)}$.

Definition 2.5 The *Kronecker sum* of matrices $A\in\mathbb{R}^{n_1\times m_1}$ and $B\in\mathbb{R}^{n_2\times m_2}$ is the matrix $C=A\oplus B\in\mathbb{R}^{n_1n_2\times m_1m_3}$, where

$$C=A\otimes I_2+I_1\otimes B,$$

where $I_1\in\mathbb{R}^{n_1\times m_1}$ and $I_2\in\mathbb{R}^{n_2\times m_2}$ are identity matrices.

Example 2.3 Consider the matrices

$$A=\begin{pmatrix}1&2\\3&4\end{pmatrix},\qquad B=\begin{pmatrix}0&1\\2&0\end{pmatrix}.$$

Their Kronecker product is

$$A\otimes B=\begin{pmatrix}1\cdot0&1\cdot1&2\cdot0&2\cdot1\\1\cdot2&1\cdot0&2\cdot2&2\cdot0\\3\cdot0&3\cdot1&4\cdot0&4\cdot1\\3\cdot2&3\cdot0&4\cdot2&4\cdot0\end{pmatrix}=\begin{pmatrix}0&1&0&2\\2&0&4&0\\0&3&0&4\\6&0&8&0\end{pmatrix},$$

while their Kronecker sum is

$$A\oplus B=\begin{pmatrix}1&0&2&0\\0&1&0&2\\3&0&4&0\\0&3&0&4\end{pmatrix}+\begin{pmatrix}0&1&0&0\\2&0&0&0\\0&0&0&1\\0&0&2&0\end{pmatrix}=\begin{pmatrix}1&1&2&0\\2&1&0&2\\3&0&4&1\\0&3&2&4\end{pmatrix}.$$

2.3 Continuous-time stochastic automata networks

Definition 2.6 A *Continuous-time stochastic automata network* is a triple $SAN = \left(E, \left(A^{(j)}\right)_{j=0}^{J-1}, \lambda\right)$, where
- E is a finite set of synchronizing *events*,
- $A^{(j)} = (S^{(j)}, x_0^{(j)}, E^{(j)}, T^{(j)})$ is a *stochastic automaton*, such that $E^{(j)} \subseteq E$ and $E = \bigcup_{j=0}^{J-1} E^{(j)}$,
- $\lambda : E \to \mathbb{R}^+$ is an *event rate function*.

Definition 2.7 A *stochastic automaton* is a 4-tuple $A = (S, x_0, E, T)$, where
- S is a finite set of *states*,
- $x_0 \in S$ is the *initial state*,
- E is a finite set of synchronizing events,
- $T \subset E \times S \times S \times \mathbb{R}^+$ is the *local transition relation*, such that $(e, x, y, \mu) \in T$, written as $x \xrightarrow{e,\mu} y$, denotes a transition from x to y with rate μ synchronized on the event e. It is required that $x \xrightarrow{e,\mu} y, x \xrightarrow{e,\nu} y \implies \mu = \nu$, i.e. the rate of a transition is a (partial) function of its start and end states and synchronizing event.

Parenthesised superscripts will be used to denote elements of automatons of a SAN, e.g. $A^{(j)} = (S^{(j)}, x_0^{(j)}, E^{(j)}, T^{(j)})$ is the jth automaton of *SAN* with $|S^{(j)}| = n^{(j)}$ states.

The set of *potential states* of *SAN* is

$$PS = S^{(0)} \times S^{(1)} \times \cdots \times S^{(J-1)},$$

i.e. the Cartesian product of the state spaces of the automata. Thus, *global states* are vectors $\mathbf{x} = (x^{(0)}, x^{(1)}, \ldots, x^{(J-1)})$. The initial global state is $\mathbf{x}_0 = (x^{(0)}, x^{(1)}, \ldots, x^{(J-1)})$.

The global state changes from $\mathbf{x} \in PS$ to $\mathbf{y} \in PS$ when the event $e \in E$ occurs,

$$\mathbf{x}\,[e\rangle\,\mathbf{y} \quad \iff \quad \text{for all } 0 \le j \le J-1 \begin{cases} (e, x^{(j)}, y^{(j)}, \mu^{(j)}) \in T^{(j)} & \text{if } e \in E^{(j)}, \\ x^{(j)} = y^{(j)} & \text{if } e \notin E^{(j)}. \end{cases}$$

The *support* of the event e is the set of automata which respond to it, $\operatorname{supp} e = \{j : e \in E^{(j)}\}$. If $\operatorname{supp} e = \{j\}$, e is *local* to $A^{(j)}$.

The events local to $A^{(j)}$ are $E_L^{(j)} = \{e : \operatorname{supp} e = \{j\}\}$. Events which affect other automata are $E_S^{(j)} = E^{(j)} \setminus E_L^{(j)}$ synchronizing events of $A^{(j)}$. The set of all local events is $E_L = \bigcup_{j=0}^{J-1} E_L^{(j)}$, while the set of all synchronizing events is $E_S = \bigcup_{j=0}^{J-1} E_S^{(j)} = E \setminus E_L$.

A state $\mathbf{y}$ is *reachable* from the state $\mathbf{x}$ (written as $\mathbf{x} \rightsquigarrow \mathbf{y}$) if there exists a sequence of states and events for some finite k such that

$$\mathbf{x} = \mathbf{x}_1\,[e_{i_1}\rangle\,\mathbf{x}_2\,[e_{i_2}\rangle\,\mathbf{x}_3\,[e_{i_3}\rangle \cdots [e_{i_{k-1}}\rangle\,\mathbf{x}_{k-1}\,[e_{i_k}\rangle\,\mathbf{x}_k = \mathbf{y}.$$

The state $\mathbf{y} \in PS$ is in the *reachable state space* of *SAN* if $\mathbf{x}_0 \rightsquigarrow \mathbf{y}$, hence the reachable state space is

$$RS = \{\mathbf{y} \in PS : \mathbf{x}_0 \rightsquigarrow \mathbf{y}\} \subseteq PS.$$

The term *state space explosion* refers to the phenomenon that even small models may have a very large number of states. For example, if $n^{(j)} = c$ for all $0 \leq j \leq J-1$, $|PS| = c^J$, hence RS may contain $O(c^J)$ elements.

We will assume a bijection $RS \leftrightarrow \{0, 1, \ldots, n-1\}$ between the reachable state space and the natural numbers such that $\mathbf{x}_0 \mapsto 0$. Moreover, we will assume a bijection $S^{(j)} \leftrightarrow \{0, 1, \ldots, n_j - 1\}$ such that $x_0^{(j)} \mapsto 0$. From now on, we will use natural number state indices and abstract state vectors interchangeably.

2.3.1 Stochastic automata networks as Markov chains

We associate a Markov chain $X(t)$ with a SAN as follows:

- The state space of the Markov chain is $S = \{0, 1, \ldots, n-1\}$, i.e. the reachable states RS of *SAN* according to the assumed bijection.
- The transition rate from $\mathbf{x}$ to $\mathbf{y}$ due to the event e is $\lambda(e) \cdot \prod_{j \in \operatorname{supp} e} \mu^{(j)}$, where $x^{(j)} \xrightarrow{e,\mu^{(j)}} y^{(j)}$. Thus, the infinitesimal generator matrix Q matrix of the $X(t)$ is formed by off-diagonal (Q_O) diagonal (Q_D) parts as

$$Q = Q_O + Q_D,$$

$$q_O[x, y] = \begin{cases} 0 & \text{if } x = y, \\ \displaystyle\sum_{e \in E, \mathbf{x}[e\rangle \mathbf{y}} \lambda(e) \cdot \prod_{j=0}^{J-1} \mu_e^{(j)} & \text{if } x \neq y, \text{ where } x^{(j)} \xrightarrow{e,\mu_e^{(j)}} y^{(j)}, \end{cases}$$

$$Q_D = -\operatorname{diag}\{Q_O \mathbf{1}^{\mathrm{T}}\}.$$

- The initial distribution concentrates all the probability mass at $\mathbf{x}_0$,

$$\pi_0 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \end{pmatrix},$$

  that is, $\pi_0[x] = \delta_{0,x}$.

The generator matrix requires $O(n^2)$ memory if a two-dimensional dense array format is used.

Suppose that for each event e and source state $\mathbf{x}$, $\mathbf{x}[e\rangle \mathbf{y}$ holds only for a number of different target states $\mathbf{y}$ bounded from above by $k \in \mathbb{N}$. Therefore, each row of Q contains up to $k|E| + 1$ nonzero elements including the diagonal element. This means Q

requires $O(nk|E|)$ memory if a sparse format is chosen, which is preferable over dense arrays for larger models.

Unfortunately, both of these storage methods may be prohibitively costly for large models due to state space explosion. In addition, explicit enumeration of large RS may take an extreme amout of time.

2.3.2 Kronecker generator matrices

To alleviate the high memory requirements of Q, the Kronecker decomposition for a SAN with J automata expresses the infinitesimal generator matrix of the associated CTMC in the form

$$Q = Q_O + Q_D, \quad Q_O = \bigoplus_{j=0}^{J-1} Q_L^{(j)} + \sum_{e \in E_S} \lambda(e) \bigotimes_{j=0}^{J-1} Q_e^{(j)}, \quad Q_D = -\operatorname{diag}\{Q_O \mathbf{1}^{\mathrm{T}}\}, \tag{2.9}$$

where Q_O and Q_D are the off-diagonal and diagonal parts of Q. The matrix

$$Q_L^{(j)} = \sum_{e \in E_L^{(j)}} \lambda(e) Q_e^{(j)}$$

is the *local* transition matrix of the component j, while the matrix

$$Q_e^{(j)} \in \mathbb{R}^{n_j \times n_j}, \quad q_e^{(j)}[x^{(j)}, y^{(j)}] = \begin{cases} \mu & \text{if } x^{(j)} \xrightarrow{e,\mu} y^{(j)}, \\ 0 & \text{otherwise} \end{cases}$$

describes the effects of the event e on $A^{(j)}$. $Q_e^{(j)}$ has a nonzero element for every local state transition caused by e. If $j \notin \operatorname{supp} e$, $Q_e^{(j)}$ is an $n_j \times n_j$ identity matrix.

The matrices $Q_L^{(j)}$ and $Q_e^{(j)}$ and the vector $-Q_O \mathbf{1}^{\mathrm{T}}$ together are usually much smaller than the full generator matrix Q even when stored in a sparse matrix form. Hence Kronecker decomposition may save a significant amount of storage at the expense of some computation time.

Unfortunately, the Kronecker generator Q is a $n_0 n_1 \cdots n_{J-1} \times n_0 n_1 \cdots n_{J-1}$ matrix, i.e. in encodes the state transitions in the potential state space PS instead of the reachable state space RS.

Potential Kronecker methods [16] perform computations with the $|PS| \times |PS|$ Q matrix and vectors of length $|PS|$. In addition to increasing storage requirements, this may lead to problems in some numerical solution algorithms, because the CTMC over PS is not neccessarily irreducible even if it is irreducible over RS.

In contrast, *actual Kronecker methods* [8, 16, 40] work with vectors of length $|RS|$. However, additional conversions must be performed between the actual dense indexing of the vectors and the potential sparse indexing of the Q matrix, which leads to implementation complexities and computational overhead.

A third approach, which we discuss in the next subsection, imposes a hierarchical structure on RS [6, 13, 17].

2.3.3 Block kronecker matrix composition

A *hierarchical decomposition* of the reachable state space expresses RS as

$$RS = \bigcup_{\tilde{\mathbf{x}} \in \widetilde{RS}} \bigtimes_{j=0}^{J-1} RS^{(j)}_{\tilde{x}^{(j)}}, \quad RS^{(j)} = \bigcup_{\tilde{x}^{(j)} \in \widetilde{RS}^{(j)}} RS^{(j)}_{\tilde{x}^{(j)}},$$

where $\widetilde{RS} = \{\tilde{0}, \tilde{1}_1, \ldots, \widetilde{\tilde{n}-1}\}$ a set of *global macro states*, $\widetilde{RS}^{(j)} = \{\tilde{0}^{(j)}, \tilde{1}^{(j)}, \ldots, \widetilde{\tilde{n}_j - 1}^{(j)}\}$ is the set of *local macro states* of $A^{(j)}$, and $RS^{(j)}_x = \{0^{(j)}_x, 1^{(j)}_x, \ldots, (n_{j,x}-1)^{(j)}_x\}$ are the *local micro states* in the local macro state $\tilde{x}^{(j)}$. The product symbol denotes the composition of local states into a global state vector.

The decomposition of the state space into global macro states allows Q to be expressed as a block matrix, where each matrix block is expressed using Kronecker decomposition.

The matrices $Q^{(j)}_e[\tilde{x}^{(j)}, \tilde{x}^{(j)}]$ and $Q^{(j)}_L[\tilde{x}^{(j)}, \tilde{x}^{(j)}] \in \mathbb{R}^{n_{j,x} \times n_{n,y}}$ describe the effects of a single event $e \in E$ and the aggregated effects of local transitions on $A^{(j)}$ as its state changes from the local macro state $\tilde{x}^{(j)}$ to $\tilde{y}^{(j)}$, respectively. Formally,

$$q^{(j)}_e[\tilde{x}^{(j)}, \tilde{y}^{(j)}][a^{(j)}_x, b^{(j)}_y] = \begin{cases} \mu & \text{if } a^{(j)}_x \xrightarrow{e,\mu} b^{(j)}_y, \\ 0 & \text{otherwise,} \end{cases}$$
$$Q^{(j)}_L[\tilde{x}^{(j)}, \tilde{y}^{(j)}] = \sum_{e \in E^{(j)}_L} \lambda(e) Q^{(j)}_e[\tilde{x}^{(j)}, \tilde{y}^{(j)}].$$

In the case $j \notin \operatorname{supp} e$, we define $Q^{(j)}_e[\tilde{x}^{(j)}, \tilde{y}^{(j)}]$ as an identity matrix if $\tilde{x}^{(j)} = \tilde{y}^{(j)}$ and a zero matrix otherwise.

Let us call macro state pairs $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ *single local macro state transitions* (slmst.) at h if $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ differ only in a single index h ($\tilde{x}^{(h)} \neq \tilde{y}^{(j)}$).

The off-diagonal part Q_O of Q is written as a block matrix with $\tilde{n} \times \tilde{n}$ blocks. A single

block is expressed as

$$Q_O[\tilde{\mathbf{x}},\tilde{\mathbf{y}}] = \begin{cases} \bigoplus_{j=0}^{J-1} Q_L^{(j)}[\tilde{x}^{(j)},\tilde{x}^{(j)}] & \\ \qquad + \sum_{e\in E_S} \lambda(e) \bigotimes_{j=0}^{J-1} Q_e^{(j)}[\tilde{x}^{(j)},\tilde{x}^{(j)}] & \text{if } \tilde{\mathbf{x}} = \tilde{\mathbf{y}}, \\ I_{N_1\times N_1} \otimes Q_L^{(h)}[\tilde{x}^{(h)},\tilde{x}^{(h)}] \otimes I_{N_2\times N_2} & \\ \qquad + \sum_{e\in E_S} \lambda(e) \bigotimes_{j=0}^{J-1} Q_e^{(j)}[\tilde{x}^{(j)},\tilde{x}^{(j)}] & \text{if } (\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \text{ slmst. at } h, \\ \sum_{e\in E_S} \lambda(e) \bigotimes_{j=0}^{J-1} Q_e^{(j)}[\tilde{x}^{(j)},\tilde{x}^{(j)}] & \text{otherwise,} \end{cases} \tag{2.10}$$

where $I_1 = \prod_{f=0}^{h-1} n_{h,x^{(h)}}$, $I_2 = \prod_{f=h+1}^{J-1} n_{h,x^{(h)}}$. If $\mathbf{x} = \mathbf{y}$, the matrix block describes transitions which leave the global macro state unchanged, therefore any local transition may fire. If $(\tilde{\mathbf{x}},\tilde{\mathbf{y}})$ is slmst. at h, only local transitions on the component h may cause the global state transition, since no other local transition may affect $A^{(h)}$. In every other case, only synchronizing transitions may occur.

This expansion of block matrices is equivalent to eq. (2.9) on page 12 except the considerations to the hierarchical structure of the state space.

The full Q matrix is written as

$$Q = Q_O + Q_D, \quad Q_D = -\operatorname{diag}\{Q_O \mathbf{1}^{\mathrm{T}}\}$$

as usual.

Chapter 3

Overview

3.1 General stochastic analysis workflow

The tasks performed by stochastic analysis tools that operate on higher level formalisms can be often structured as follows (Figure 3.1):

1. *State space exploration.* The reachable state space of the higher level model, for example stochastic automata network or stochastic Petri net is explored to enumerate the possible behaviors of the model S. If the model is hierarchically partitioned, this step includes the exploration of the local state spaces of the component as well as the possible global combinations of states.

   If the set of reachable states is infinite, only special algorithms, e.g. matrix geometric methods [36] may be employed later in the workflow. In this work, we restrict our attention to finite cases.

2. *Descriptor generation.* The infinitesimal generator matrix Q of the Markov chain $X(t)$ defined over S is built. If the analyzed formalism is a Markov chain, Q is readily given. Otherwise, this matrix contains the transition rates between reachable states, which are obtained by evaluating rate expressions given in the model.

3. *Numerical solution.* Numerical algorithms are ran on the matrix Q for steady-state solutions $\boldsymbol{\pi}$, transient solutions $\boldsymbol{\pi}(t)$, $\mathbf{L}(t)$ or MTFF measures.

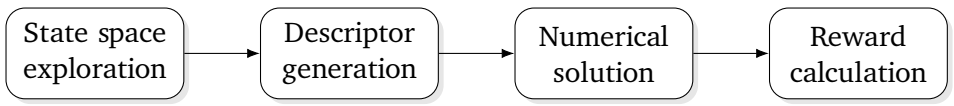


Figure 3.1 The general stochastic analysis workflow.

4. *Reward calculations.* The studied performance measures are calculated from the output of the previous step. This includes calculation of steady-state and transient rewards and sensitivities of the rewards. Additional algebraic manipulations (for example, the calculation of the ratio of an instantenous and accumulated reward) may be provided to the modeler for convenience.

In stochastic model checking, where the desired system behaviors are expressed in stochastic temporal logics [2, 9], these analytic steps are called as subrouties to evaluate propositions. In the sythesis and optimization of stochastic models [20], the workflow is executed as part of the fitness functions.

3.1.1 Challenges

The implementation of the stochastic analysis workflow poses several challenges.

Handling of large models is difficult due to the phenomenon of "state space explosion". As the size of the model grows, including the number of components, the number of reachable spaces can grow exponentially.

Methods such as the *saturation* algorithm [21] were developed to efficiently explore and represent large state spaces. However, in stochastic analysis, the generator matrix Q and several vectors of real numbers with lengths equal to the state space size must be stored in addition to the state space. This neccessitates the use of further decomposition techniques for data storage.

The convergence of the numerical methods depends on the structure of the model and the applied matrix decomposition. In addition, the memory requirements of the algorithms may constrain the methods that can be employed. As various numerical algorithms for stochastic analysis tasks are known with different characteristics, it is important to allow the modeler to select the algorithm suitable for the properties of the model, as well as the decomposition method and hardware environment.

The vector operations and vector-matrix products that are performed by the numerical algorithms can also be performed in multiple ways. For example, multiplications with matrices can be implemented either sequentially or in parallel. Large matrices benefit from parallelization, while for small matrices managing multiple tasks yields overhead. Distributed or GPU implementations are also possible, albeit they are missing from the current version of our framework.

3.2 Our workflow in PetriDotNet

"PetriDotNet is a framework for the editing, simulation and analysis of Petri nets. The framework is developed by the Fault Tolerant Systems Research Group at the Budapest University of Technology and Economics." [27]

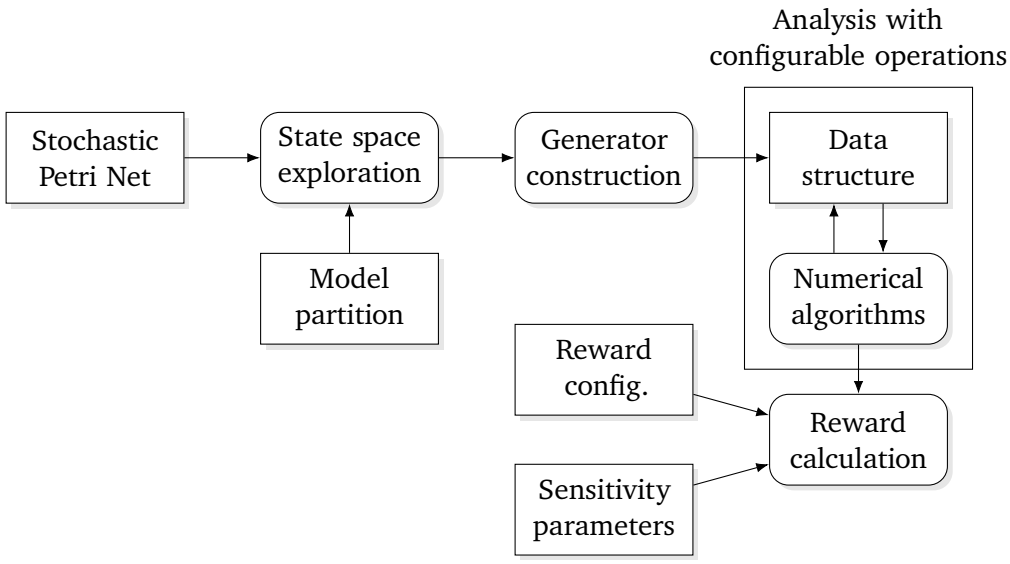


Figure 3.2 Configurable stochastic analysis workflow.

The implementation of the general stochastic analysis workflow in PetriDotNet is illustrated in Figure 3.2. The models are specified using the stochastic Petri net (SPN) formalism **TODO Cite**, while engineering measures to be calculated are expressed as SPN performance measures. Both explicit and symbolic state space exploration and storage is supported, including symbolic hierarchical state space decomposition for block Kronecker generator matrices.

The workflow is fully *configurable*, which means that the modeler may combine the available algorithms for the analysis steps arbitrarily. In addition, implementations of the linear algebra operations performed by the algorithms may be replaced at runtime.

3.3 Architecture

Figure 3.3 shows the architecture of the configurable stochastic analysis module.

- The user interacts with the stochastic *analysis workflow* runner.

  The model, its parameters and its stochastic behavior as transition rates of timed transitions is specified and engineering measures of interest (e.g. performability, availability, reliability, dependability) are defined with SPN rewards. Afterwards, the analysis workflow can be initiated by selecting the analysis type (steady-state, sensitivity, transient, MTFF), the used algorithms and the engineering measures to compute. The workflow runner instantiates and executes the components which are required to complete the analysis and displays the results.

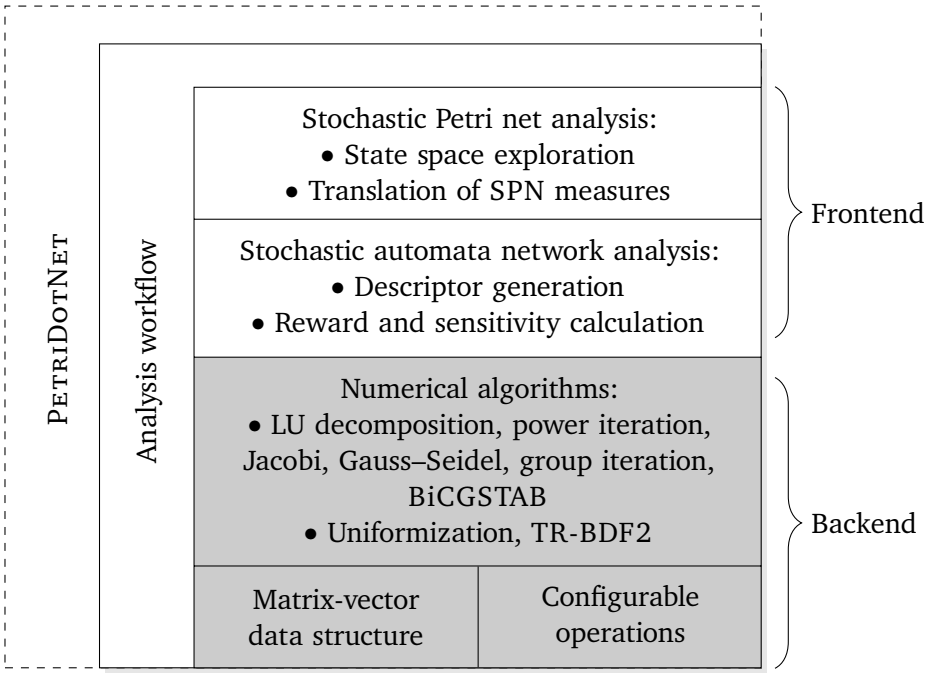


Figure 3.3 Layered architecture for configurable stochastic analysis.

Numerical analysis algorithms most suitable for the analyzed model and executing hardware may be selected by the user. Moreover, low-level linear algebra operations, for example, parallel or sequential algorithms for matrix products, may be also selected for every step in the workflow.

The stochastic analysis problem is translated into numerical problems by the "frontend" part of the analysis module:

- The *stochastic Petri net analysis* modules translate the stochastic behaviour of Petri net into generic data structures. The partition of the model defines the stochastic automata of the SAN representation of the model. The algebraic expressions that specify transition rates and rewards are evaluated, thus lower level components only work with transition rates and their derivatives.

  Symbolic state space exploration is performed by the *saturation* algorithm, which is provided by the symbolic analysis component of PetriDotNet. Petri nets with inhibitor arcs are supported, but transitions with prioirity (including non-timed transitions) cannot be used. **TODO Cite**

  Rewar expressions that refer to subsets of the reachable state space defined by Computation Tree Logic (CTL) are also evaluated by the symbolic analysis

component. Therefore, CTL rewards cannot be used with explicit state space representation algorithms.

- The *stochastic automata network analysis* module implements explicit and symbolic procedures for infinitesimal generator matrix composition and reward calculation. This component does not depend on the Petri net formalism and may be reused for different formalisms.

  The matrices Q and V_i, that is, the generator matrix and its partial derivatives may be stored as a dense or sparse array or a block Kronecker matrix using the object model defined by the matrix-vector data structure. Linear algebra operations during the generator composition, for example, calculation of the diagonal entries of Q, are performed by the operation framework supporting the data structure.

  Numerical solution algorithms, such as linear equation solvers and transient distribution integrators are called to derive the steady state and transient distributions of the Markov chain and its sensitivities.

  The final task performed by the frontend is the calculation of the reward values, which uses both linear algebra operations and symbolic iteration over the results of the CTL evaluator.

The analysis "backend" serves as a library of matrix–vector data structured, linear algebra operations and numerical solution algorithms:

- *Numerical algorithms* implement solution finding for linear equations and Markovian transient initial value problems. The algorithms are implemented generically whenever possible, so that no assumptions are made about the structure of matrices unless neccessary due to mathematical or performance reasons. This is achieved by the definition of a (non-orthogonal) set of operations on the matrix-vector data structure. The operations may be replaced at runtime for flexibility, for example, different implementations of operations may be used for different algorithms in the same workflow.

- The *matrix-vector data structure* provides an interface for storing various linear algebra objects.

  In addition to dense and sparse arrays, wrappers are provided to access parts of matrices and vectors and to build expression trees out of smaller matrices. Hence, matrices such as block Kronecker infinitesimal generators (eq. (2.10) on page 14) can be stored as a collection of small sparse matrices in a nested expression tree.

  While current the frontend only generates simple arrays and block Kronecker matrices, and descriptor format may be used as long as it can be expressed as expression trees.

The data structure only provides storage, any calls to linear algebra operations are delegated to the configurable operation context.

- The *configurable operation context* provides and dispatches the implementations of linear algebra operations, such as matrix-vector product or vector addition.

  Operations are specific to the data structure and may use multiple dispatch call semantics. For example, an operation can be defined that handles the multiplication of a block matrix and a constant vector, and stores the result in vector backed by a linear array. In addition to type information, the dispatch may use addition runtime properties, such as the length of a vector to select the appropriate implementation.

  The dispatch rules may be modified at runtime. For example, parallel execution may be replaced with sequential during the execution of algorithms that achieve parallelization through other means.

  Contrast operations with numerical algorithms, which are higher level procedures that solve a particular numerical problem on a wide range of data structures by delegation to a non-orthogonal set of specific operations.

The stochastic analysis backed, which was developed by the author, comprise the topic of present work. Figure 3.3 shows its three components shaded in gray.

Manipulations performed by the frontend components on the input Petri net models the generated descriptors are discussed in this thesis only briefly. We refer the interested reader to **TODO Cite TDK** for an overview of the whole PetriDotNet stochastic analysis component.

3.4 Current status

In this section we briefly summarize the results of the backend development effort.

3.4.1 Data structures

3.4.2 Numerical algorithms

Seven linear equation solver algorithms were implemented for steady-state, sensitivity and MTFF problems: LU decomposition, power iteration, Jacobi over-relaxation, Gauss–Seidel over-relaxation, group Jacobi, group Gauss–Seidel and BiCGSTAB. Group Jacobi and Gauss–Seidel require a block matrix, while the other algorithms may run on any matrix.

Special attention is paid to the root finding of singular systems with zero right vectors, i.e. the determination of the nullspace of a matrix for systems arising from

Table 3.1 Linear equation solvers supported by our framework.

	see	memory usage	parallel impl.	uses inner solver	block matrix
LU decomposition	p. 30	very high	–	–	–
Power method	p. 33	moderate	✓	–	✓
Jacobi over-relaxation	p. 34	moderate	✓	–	✓
Gauss–Seidel over-relaxation	p. 34	very low	–	–	✓
Group Jacobi	p. 36	moderate	✓	✓	required
Group Gauss–Seidel	p. 36	low	–	✓	required
BiCGSTAB	p. 40	high	✓	–	✓
IDR(s)STAB(l)	p. 42	Ongoing research			

Table 3.2 Transient solvers supported by our framework.

	see	instantenous distribution	accumulated distribution	uses inner solver	block matrix
Uniformization	p. 45	✓	✓	–	✓
TR-BDF2	p. 46	✓	not impl.	✓	not impl.

Markovian steady-state problems. Nonsingular problems are solved in steady-state sensitivity analysis and mean-time-to-failure analysis.

The current research and development effort focuses on the integration of a solver based on IDR(s)STAB(l) **TODO Cite**, a Krylov subspace algorithm which generalizes BiCGSTAB. As the algorithm needs adaptation for singular matrices, it is currently not suitable for production use in Markovian analysis due to numerical breakdowns and instability.

Two solution algorithms, uniformization and TR-BDF2 are available for transient analysis. Accumulated rewards can be calculated by uniformization only, while TR-BDF2 provides robustness for otherwise difficult to handle stiff Markov chains.

Important considerations in solver selection are convergence properties and memory requirements. Matrix decompositions can reduce the storage space needed by the matrix Q by orders of magnitudes. We store all elements of probability vectors explicitly. Therefore, one should pay close attention to the number of temporary vectors used in the algorithm in order to avoid excessive memory consumption.

Numerical algorithms supported by our framework are further discussed in Chapter 5. Linear equations solvers for steady-state CTMC analysis are shown in Table 3.1,

while linear solver are shown in Table 3.2.

Chapter 4

Configurable data structure and operations

TODO Ezt a fejezetet osszerakni

4.1 Matrix storage

Existing linear algebra and matrix libraries, such as [11, 26, 34, 44, 57], usually have unsatisfactory support for operations required in stochastic analysis algorithms with decomposed matrices, for example, multiplications with Kronecker and block Kronecker matrices. Therefore, we have decided to develop a linear algebra framework in C#.NET specifically for stochastic algorithms as a basis of our stochastic analysis framework.

Sparse matrices are stored in Compressed Column Storage (CCS) format, i.e. an array or values and row indices are stored for each column of the matrix, as illustrated in Figure 4.1. This facilitates multiplication from left with row vectors. To reduce pressure on the garbage collector (GC), matrices are vectors are stored in manually allocated and managed memory.

While other sparse matrix formats, such as sliced LAPACK are more amenable to parallel and SIMD processing Kreutzer et al. [41], CCS was selected due to implemen-

$$A = \begin{pmatrix} 1 & 0 & 0 & 2.5 \\ 3 & 1 & 0 & 0 \\ 4 & 0 & 0 & 1 \\ 5 & 0 & 0 & 0 \end{pmatrix} \qquad \begin{aligned} A = \{&\{(1,0),(3,1),(4,2),(5,3)\},\\ &\{(1,1)\},\\ &\{\},\\ &\{(2.5,0),(1,2)\}\} \end{aligned}$$

Figure 4.1 Compressed Column Storage of a matrix.

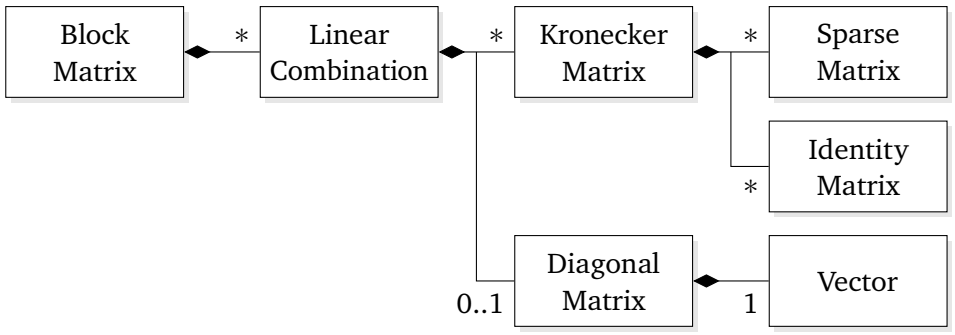


Figure 4.2 Data structure for block Kronecker matrices.

Algorithm 4.1 Parallel block vector-matrix product.

Input: block vector $\mathbf{b} \in \mathbb{R}^{n_0+n_1+\cdots+n_{k-1}}$,
block matrix $A \in \mathbb{R}^{(n_0+n_1+\cdots+n_{k-1})\times(m_0+m_1+\cdots+m_{l-1})}$
Output: $\mathbf{c} = \mathbf{b}A \in \mathbb{R}^{m_0+m_1+\cdots+m_{l-1}}$
1 **allocate** $\mathbf{c} \in \mathbb{R}^{m_0+m_1+\cdots+m_{l-1}}$
2 **parallel for** $j \leftarrow 0$ **to** $l-1$ **do**
3 $\quad \mathbf{c}[j] \leftarrow \mathbf{0}$
4 $\quad$ **for** $i \leftarrow 0$ **to** $k-1$ **do**
5 $\quad\quad \mathbf{c}[j] \leftarrow \mathbf{c}[j] + \mathbf{b}[i]A[i,j]$ // Scaled addition of vector-matrix product

tation simplicity and the small number of nonzero entries in each column of the matrix, which reduces the potential benefits of SIMD implementations.

Decomposed Kronecker and block Kronecker matrices are stored as algebraic expression trees as shown in Figure 4.2. Matrix multiplication and manipulation algorithms for expression trees are detailed in Section 4.2.

The expression tree approach allows the use of arbitrary matrix decompositions that can be expressed with block matrices, linear combinations and Kronecker products. The implementation of additional opeartional primitives is also straightforward. The data structure forms a flexible basis for the development of stochastic analysis algorithms with decomposed matrix representations.

4.2 Efficient vector-matrix products

Iterative linear equation and transient distribution solvers require several vector-matrix products per iteration. Therefore, efficient vector-matrix multiplication algorithms are

Algorithm 4.2 Product of a vector with a linear combination matrix.

Input: $\mathbf{b} \in \mathbb{R}^n$, $A = \nu_0 A_0 + \nu_1 A_1 + \cdots + \nu_{k-1} A_{k-1}$, where $A_h \in \mathbb{R}^{n \times m}$
Output: $\mathbf{c} = \mathbf{b}A \in \mathbb{R}^m$
allocate $\mathbf{c} \in \mathbb{R}^m$ if no target buffer is provided
$\mathbf{c} \leftarrow \mathbf{0}$
for $h \leftarrow 0$ **to** $k-1$ **do**
    $\mathbf{c} \leftarrow \nu_h \cdot \mathbf{b}A_h$ // In-place scaled addition of vector-matrix product
return $\mathbf{c}$

Algorithm 4.3 The SHUFFLE algorithm for vector-matrix multiplication.

Input: $\mathbf{b} \in \mathbb{R}^{n_0 n_1 \cdots n_{k-1}}$, $A = A^{(0)} \otimes A^{(1)} \otimes \cdots \otimes A^{(k-1)}$, where $A^{(h)} \in \mathbb{R}^{n_h \times m_h}$
Output: $\mathbf{c} = \mathbf{b}A \in \mathbb{R}^{m_0 m_1 \cdots m_{k-1}}$
$n \leftarrow n_0 n_1 \cdots n_{k-1}, \quad m \leftarrow m_0 m_1 \cdots m_{k-1}$
$tempLength \leftarrow \max_{h=-1,0,1,\ldots,k-1} \prod_{f=0}^{h} m_f \prod_{f=h+1}^{k-1} n_f$
allocate $\mathbf{x}, \mathbf{x}'$ with at least *tempLength* elements
$\mathbf{x}[0{:}1{:}n] \leftarrow \mathbf{b}, \quad i_{\text{left}} \leftarrow 1, \quad i_{\text{right}} \leftarrow \prod_{h=1}^{k-1} n_h$
for $h \leftarrow 0$ **to** $k-1$ **do**
    if $A^{(h)}$ is not an identity matrix **then**
        $i_{\text{base}} \leftarrow 0, j_{\text{base}} \leftarrow 0$
        for $il \leftarrow 0$ **to** $i_{\text{left}} - 1$ **do**
            for $ir \leftarrow 0$ **to** $i_{\text{right}} - 1$ **do**
                $\mathbf{x}'[j_{\text{base}}{:}m_h{:}i_{\text{right}}] \leftarrow \mathbf{x}[i_{\text{base}}{:}n_h{:}i_{\text{right}}]A^{(h)}$
                $i_{\text{base}} \leftarrow i_{\text{base}} + n_h i_{\text{right}}, \quad j_{\text{base}} \leftarrow j_{\text{base}} + m_h i_{\text{right}}$
        Swap the references to $\mathbf{x}$ and $\mathbf{x}'$
    $i_{\text{left}} \leftarrow i_{\text{left}} \cdot m_h$
    if $h \neq k-1$ **then** $i_{\text{right}} \leftarrow i_{\text{right}} / n_{h+1}$
return $\mathbf{c} = \mathbf{x}[0{:}1{:}m]$

required for the various matrix storage methods (i.e. dense, sparse and block Kronecker matrices) to support configurable stochastic analysis.

Our data structure supports run-time reconfiguration of operations, for example, to switch between parallel and sequential matrix multiplication implementations for different parts of an algorithm, depending on the characteristics of the model and the hardware which runs the analysis.

Implemented matrix multiplication for the data structure (see Figure 4.2 on page 24) routines are

- Multiplication of vectors with dense and sparse matrices. Sparse matrix multiplication may be parallelized by splitting the columns of the matrix into chunck and submitting each chunk to the executor thread pool.

  Operations with vectors and sparse matrices are implemented in an `unsafe`[1] context. The elements of the data structures are not under the influence of the Garbage Collector runtime, but stored in natively allocated memory. This allows the handling of large matrices without adversely impacting the performance of other parts of the program, albeit the cost of allocations in increased.

- Multiplication with block matrices by delegation to the constituent blocks of the matrix (Algorithm 4.1 on page 24). The input and output vectors are converted to block vectors before multiplication. If parallel execution is required, each block of the output vector can be computed in a different task, since it is independent from the others.

- Multiplication by a linear combination of matrices is delegated to the constituent matrices (Algorithm 4.2 on page 25). An in-place scaled addition of vector-matrix product to a vector operation is required for this delegation. To facilitate this, each vector-matrix multiplication algorithm is implemented also as an in-place addition and in-place scaled addition of vector-matrix product, and the appropriate implementation is selected based on the function call aruments.

- Multiplications $\mathbf{b} \cdot \mathrm{diag}\{\mathbf{a}\}$ by diagonal matrices are executed as elementwise product $\mathbf{b} \odot \mathbf{a}$. The special case of multiplication by an identity matrix is equivalent to a vector copy.

- Multiplications by Kronecker products is performed by the SHUFFLE algorithm [7, 16] as shown in Algorithm 4.3 on page 25.

  The algorithm requires access to slices of a vector, denoted as $\mathbf{x}[i_0{:}s{:}l]$, which refers to the elements $x[i], x[i+s], x[i+2s], \ldots, x[i+(l-1)s]$. Thus, slices were integrated into the operations framework as first-class elements, and multiplication algorithms are implemented with support for vector slice indexing.

[1] `https://msdn.microsoft.com/en-us/library/chfa2zb8.aspx`

Shuffle rewrites the Kronecker products as

$$\bigotimes_{h=0}^{k-1} A^{(h)} = \prod_{h=0}^{k-1} I_{\prod_{f=0}^{h-1} n_f \times \prod_{f=0}^{h-1} n_f} \otimes A^{(h)} \otimes I_{\prod_{f=h+1}^{k-1} m_f \times \prod_{f=h+1}^{k-1} m_f},$$

where $I_{a\times a}$ denotes an $a \times a$ identity matrix. Multiplications by terms of the form $I_{N\times N} \otimes A^{(h)} \otimes I_{M\times M}$ are carried out in the loop at line 8 of Algorithm 4.3.

The temporary vectors $\mathbf{x}, \mathbf{x}'$ are large enough store the results of the successive matrix multiplications. They are cached for every worker thread to avoid repeated allocations.

Other algorithms for vector-Kronecker product multiplication are the Slice [28] and Split [23] algorithms, which are more amenable to parallel execution than Shuffle. Their implementation is in the scope of our future work.

Chapter 5

Algorithms for stochastic analysis

Steady state, transient, accumulated and sensitivity analysis problems pose several numerical challanges, especially when the state space of the CTMC and the vectors and matrices involved in the computation are extemely large.

In steady-state and sensitivty analysis, linear equations of the form $\mathbf{x}A = \mathbf{b}$ are solved, such as eqs. (2.2) and (2.6) on page 4 and on page 7. The steady-state probability vector is the solution of the linear system

$$\frac{\mathrm{d}\boldsymbol{\pi}}{\mathrm{d}t} = \boldsymbol{\pi}Q = \mathbf{0}, \quad \boldsymbol{\pi}\mathbf{1}^{\mathrm{T}} = 1, \qquad \text{(2.2 revisited)}$$

where the infinitesimal generator Q is a rank-deficient matrix. Therefore, steady-state solution methods must handle various generator matrix decompositions and homogenous linear equation with rank deficient matrices. Convergence and computation times of linear equations solvers depend on the numerical properties of the Q matrices, thus different solvers may be preferred for different models.

In transient analysis, initial value problems with first-order linear differetial equations such as eqs. (2.1) and (2.5) on page 4 and on page 6 are considered. The decomposed generator matrix Q must be also handled efficiently. Another difficulty is caused by the *stiffness* of differential equations arising from some models, which may significantly increase computation times.

To facilitate configurable stochastic analysis, we developed several linear equation solvers and transient analysis methods. Where it is reasonable, the implementation is independent of the form of the generator matrix Q.

The implementation of low-level linear algebra operations is also decoupled from the numerical algorithms and data structure. This strategy enables further configurability by replacing the operations at runtime, as described in Chapter 4.

In this chapter, we describe the algorithms implemented in our stochastic analysis framework. The pseudocode of the algorithms is annotated with the low level operations performed on the configurable data structure by the high level algorithms.

Algorithm 5.1 Crout's LU decomposition without pivoting.

Input: the matrix $A \in \mathbb{R}^{n\times n}$ operated on in-place
Output: $L, U \in \mathbb{R}^{n\times n}$ such that $A = LU$, $u[i,i] = 1$ for all $i = 0, 1, \ldots, n-1$
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** i **do** $a[i,j] \leftarrow a[i,j] - \sum_{k=0}^{j-1} a[i,k]a[k,j]$
  for $j \leftarrow i+1$ **to** $n-1$ **do** $a[i,j] \leftarrow \left(a[i,j] - \sum_{k=0}^{i-1} a[i,k]a[i,j]\right) \big/ a[i,i]$
Let A_L, A_D and A_U refer to the strictly lower triangular, diagonal and strictly upper triangular parts of A, respectively.
$L \leftarrow A_L + A_D$
$U \leftarrow A_U + I$
return L, U

5.1 Linear equation solvers

5.1.1 Explicit solution by LU decomposition

LU decomposition is a direct method for solving linear equations with forward and backward substitution, i.e. it does not require iteration to reach a given precision.

The decomposition computes the lower triangular matrix L and upper triangular matrix U such that

$$A = LU.$$

To solve the equation

$$\mathbf{x}A = \mathbf{x}LU = \mathbf{b}$$

forward substitution is applied first to find $\mathbf{z}$ in

$$\mathbf{z}U = \mathbf{b},$$

then $\mathbf{x}$ is computed by back substitution from

$$\mathbf{x}L = \mathbf{b}.$$

We used Crout's LU decomposition [50, Section 2.3.1], presented in Algorithm 5.1), which ensures

$$u[i,i] = 1 \text{ for all } i = 0, 1, \ldots, n-1,$$

i.e. the diagonal of the U matrix is uniformly 1. The matrix is filled in during the decomposition even if it was initially sparse, therefore it should first be copied to a dense array storage for efficiency reasons. This considerably limits the size of Markov chains that can be analysed by direct solution due to memory requirements. Our

Algorithm 5.2 Forward and back substitution.

Input: $U, L \in \mathbb{R}^{n\times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$
Output: solution of $\mathbf{x}LU = \mathbf{b}$
1 **allocate** $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$
2 **if** $\mathbf{b} = \mathbf{0}$ **then** $\mathbf{z} \leftarrow \mathbf{0}$ // Skip forward substitution for homogenous equations
3 **else for** $j \leftarrow 0$ **to** $n-1$ **do** $z[j] \leftarrow b[j] \cdot \sum_{i=0}^{j-1} u[i,j]$
4 **if** $l[n-1,n-1] \approx 0$ **then**
5 **if** $z[n-1] \approx 0$ **then** $x[n-1] \leftarrow 0$ // Set the free parameter to 1
6 **else error** "inconsistent linear equation system"
7 **else** $x[n-1] \leftarrow z[n-1]/l[n-1,n-1]$
8 **for** $j \leftarrow n-2$ **downto** 0 **do**
9 **if** $l[j,j] \approx 0$ **then error** "more than one free parameter"
10 $x[j] \leftarrow \left(z[i] - \sum_{i=j+1}^{n-1} x[i]l[i,j]\right) \Big/ l[j,j]$
11 **return** $\mathbf{x}$

data structure allows access to upper and lower diagonal parts to matrices and linear combinations, therefore no additional storage is needed other than A itself.

The forward and back substitution process is shown in Algorithm 5.2. If multiple equations are solver with the same matrix, its LU decomposition may be cached.

Matrices of less than full rank

If the matrix Q is of rank $n-1$, the element $l[n-1,n-1]$ in Crout's LU decomposition will be 0. In this case, $x[n-1]$ is a free parameter and will be set to 1 to yield a nonzero solution vector when $z[n-1] = 0$. If $z[n-1] \neq 0$, the equation $\mathbf{x}L = \mathbf{z}$ does not have a solution and the error condition in line 6 is triggered. A matrix of rank less than $n-1$ triggers the error condition in line 9.

In practice, the algorithm can be used to solve homogenous equations in Markovian analysis, because the infinitesimal generator matrix Q of an irreducible CTMC is always of rank $n-1$. The solution vector $\mathbf{x}$ is not a probability vector in general, so it must be normalized as $\boldsymbol{\pi} = \mathbf{x}/\mathbf{x}\mathbf{1}^{\mathrm{T}}$ to get a stationary probability distribution vector.

5.1.2 Iterative methods

Iterative methods express the solution of the linear equation $\mathbf{x}A = \mathbf{b}$ as a recurrence

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}),$$

Algorithm 5.3 Basic iterative scheme for solving linear equations.

Input: matrix $A \in \mathbb{R}^{n \times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$, initial guess $\mathbf{x} \in \mathbb{R}^n$, tolerance $\tau > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$ and its residual norm
1 **allocate** $\mathbf{x}' \in \mathbb{R}^n$ // Previous iterate for convergence test
2 **repeat**
3 $\quad \mathbf{x}' \leftarrow \mathbf{x}$ // Save the previous vector
4 $\quad \mathbf{x} \leftarrow f(\mathbf{x}')$
5 **until** $\|\mathbf{x}' - \mathbf{x}\| \leq \tau$
6 **return** $\mathbf{x}$ and $\|\mathbf{x}Q - \mathbf{b}\|$

where $\mathbf{x}_0$ is an initial guess vector. The iteration converges to a solution vector when $\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}$ exists and $\mathbf{x}$ equals the true solution vector $\mathbf{x}^*$. The iteration is illustrated in Algorithm 5.3.

The process is assumed to have converged if subsequent iterates are sufficiently close, i.e. the stopping criterion at the kth iteration is

$$\|\mathbf{x}_k - \mathbf{x}_{k-1}\| \leq \tau \tag{5.1}$$

for some prescribed tolerance τ. In our implementation, we selected the L^1-norm

$$\|\mathbf{x}_k - \mathbf{x}_{k-1}\| = \sum_i \left| x_k[i] - x_{k-1}[i] \right|$$

as the vector norm used for detecting convergence.

Premature termination may be avoided if iterates spaced $m > 1$ iterations apart are used for convergence test ($\|\mathbf{x}_k - \mathbf{x}_{k-m}\| \leq \tau$), but only at the expense of additional memory required for storing m previous iterates. In order to handle large Markov chains with reasonable memory consumption, we only used the convergence test with a single previous iterate.

Correctness of the solution can be checked by observing the norm of the residual $\mathbf{x}_k A - \mathbf{b}$, since the error vector $\mathbf{x}_k - \mathbf{x}^*$ is generally not available. Because the additional matrix multiplication may make the latter check costly, it is performed only after detecting convergence by eq. (5.1). Unfortunately, the residual norm may not be representative of the error norm if the problem is ill-conditioned.

For a detailed discussion stopping criterions and iterate normalization in steady-state CTMC analysis, we refer to [63, Section 10.3.5].

Algorithm 5.4 Power iteration.

```
1 α^{-1} ← 1/max_i |a[i,i]|
2 repeat
3 |   x' ← xA                          ▷ VectorMatrixMultiplyFromLeft
4 |   x' ← x' + (−1) · x               ▷ In-place VectorAdd
5 |   ε ← α^{-1} ||x'||                ▷ VectorL1Norm
6 |   x ← x + α^{-1} x'                ▷ In-place VectorAdd
7 until ε ≤ τ
```

Power iteration

Power iteration [63, Section 10.3.1] is the one of the simplest iterative methods for Markovian analysis. Its iteration function has the form

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) = \mathbf{x}_{k-1} + \frac{1}{\alpha}(\mathbf{x}_{k-1}A - \mathbf{b}).$$

The iteration converges if the diagonal elements $a[i,i]$ of A are strictly negative, the off-diagonal elements $a[i,j]$ are nonnegative and $\alpha \geq \max_i |a[i,i]|$. The matrix A satisfies these properties if it is an inifinitesimal generator matrix of an irreducible CTMC. The fastest convergence is achieved when $\alpha = \min_i |a[i,i]|$.

Power iteration can be realized by replacing lines 2–5 in Algorithm 5.3 on page 32 with the loop in Algorithm 5.4.

This realization uses memory efficiently, because it only requires the allocation of a single vector $\mathbf{x}'$ in addition to the initial guess $\mathbf{x}$.

Observation 5.1 If $\mathbf{b} = 0$ and A is an inifitesimal generator matrix, then

$$\begin{aligned}\mathbf{x}_k\mathbf{1}^{\mathrm{T}} &= \left[\mathbf{x}_{k-1} + \frac{1}{\alpha}(\mathbf{x}_{k-1}A - \mathbf{b})\right]\mathbf{1}^{\mathrm{T}} \\ &= \mathbf{x}_{k-1}\mathbf{1}^{\mathrm{T}} + \frac{1}{\alpha}\mathbf{x}_{k-1}A\mathbf{1}^{T} - \mathbf{b}\mathbf{1}^{\mathrm{T}} \\ &= \mathbf{x}_{k-1}\mathbf{1}^{\mathrm{T}} + \frac{1}{\alpha}\mathbf{x}_{k-1}\mathbf{0}^{\mathrm{T}} - \mathbf{0}\mathbf{1}^{\mathrm{T}} = \mathbf{x}_{k-1}\mathbf{1}^{\mathrm{T}}.\end{aligned}$$

This means the sum of the elements of the result vector $\mathbf{x}$ and the initial guess vector $\mathbf{x}_0$ are equal, because the iteration leaves the sum unchanged.

To solve an equation of the form

$$\mathbf{x}Q = \mathbf{0}, \quad \mathbf{x}\mathbf{1}^{\mathrm{T}} = 1 \tag{5.2}$$

where Q is an infinitesimal generator matrix, the initial guess $\mathbf{x}_0$ is selected such that $\mathbf{x}_0\mathbf{1}^{\mathrm{T}} = 1$. If the CTMC described by Q is irreducible, we may select

$$x_0[i] \equiv \frac{1}{n}, \tag{5.3}$$

where n is the dimensionality of $\mathbf{x}$. After the initial guess is selected, the equation $\mathbf{x}\mathbf{1}^{\mathrm{T}}$ may be ignored to solve $\mathbf{x}Q = \mathbf{0}$ with the power method. This process yields the solution of the original problem (5.2).

Jacobi and Gauss–Seidel iteration

Jordan and Gauss–Seidel iterative methods [63, Section 10.3.2–3] repeatedly solve a system of simultaneous equations of a specific form.

In Jordan iteration, the system

$$\left.\begin{aligned} b[0] &= x_k[0]a[0,0] + x_{k-1}[1]a[1,0] + \cdots + x_{k-1}[n-1]a[n-1,0], \\ b[1] &= x_{k-1}[0]a[0,1] + x_k[1]a[1,1] + \cdots + x_{k-1}[n-1]a[n-1,1], \\ &\vdots \\ b[n-1] &= x_{k-1}[0]a[0,n-1] + x_{k-1}[1]a[1,n-1] + \cdots + x_k[n-1]a[n-1,n-1], \end{aligned}\right\}$$

is solved for $\mathbf{x}_k$ at each iteration, i.e. there is a single unknown in each row and the rest of the variables are taken from the previous iterate. In vector form, the iteration can be expressed as

$$\mathbf{x}_k = A_D^{-1}(\mathbf{b} - A_O\mathbf{x}_{k-1}),$$

where A_D and A_O are the diagonal (all off-diagonal elements are zero) and off-diagonal (all diagonal elements are zero) parts of $A = A_D + A_O$.

In Gauss–Seidel iteration, the linear system

$$\left.\begin{aligned} b[0] &= x_k[0]a[0,0] + x_{k-1}[1]a[1,0] + \cdots + x_{k-1}[n-1]a[n-1,0], \\ b[1] &= x_k[0]a[0,1] + x_k[1]a[1,1] + \cdots + x_{k-1}[n-1]a[n-1,1], \\ &\vdots \\ b[n-1] &= x_k[0]a[0,n-1] + x_k[1]a[1,n-1] + \cdots + x_k[n-1]a[n-1,n-1], \end{aligned}\right\}$$

is considered, i.e. the ith equation contains the first i elements of $\mathbf{x}_k$ as unknowns. The equations are solved for successive elements of $\mathbf{x}_k$ from top to bottom.

Jacobi over-relaxation, a generalized form of Jacobi iteraion, is realized in Algorithm 5.5. The value 1 of the over-relaxation paramter ω corresponds to ordinary Jacobi iteration. Values $\omega > 1$ may accelerate convergence, while $0 < \omega < 1$ may help diverging Jacobi iteration converge.

Jacobi over-relaxation has many parallelization opportunities. The matrix multiplication in line 4 and the vector addition in line 5 can be parallelized, as well as the for loop in line 7. Our implementation takes advantage of the configurable linear algebra

Algorithm 5.5 Jacobi over-relaxation.

Input: matrix $A \in \mathbb{R}^{n \times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$, initial guess $\mathbf{x} \in \mathbb{R}^n$, tolerance $\tau > 0$, over-relaxation parameter $\omega > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$

1 **allocate** $\mathbf{x}' \in \mathbb{R}^n$
2 Let A_O refer to the off-diagonal part of A.
3 **repeat**
4 $\quad \mathbf{x}' \leftarrow \mathbf{x}A_O$ ▷ VectorMatrixMultiplyFromLeft
5 $\quad \mathbf{x}' \leftarrow \mathbf{x}' + (-1) \cdot \mathbf{b}$ ▷ In-place VectorAdd
6 $\quad \epsilon \leftarrow 0$
7 $\quad$ **for** $i \leftarrow 0$ **to** $n-1$ **do**
8 $\quad\quad y \leftarrow (1-\omega)x[i] - \omega x'[i]/a[i,i]$
9 $\quad\quad \epsilon \leftarrow \epsilon + |y - x[i]|$
10 $\quad\quad x[i] \leftarrow y$
11 **until** $\epsilon \leq \tau$
12 **return** $\mathbf{x}$

operations framework to execute lines 4 and 5 with possible paralellization considering the structures of both the vectors $\mathbf{x}, \mathbf{x}'$ and the matrix A. However, the inner loop is left sequential to reduce implementation complexity, as it represents only a small fraction of execution time compared to the matrix-vector product.

Algorithm 5.6 shows an implementation of successive over-relaxation for Gauss–Seidel iteration, where the notation $\mathbf{a}_O[\cdot, i]$ refers to the ith column of A_O.

Gauss–Seidel iteration cannot easily be parallelized, because calculation of successive elements $x[0], x[1], \ldots$ depend on all of the prior elements. However, in contrast with Jacobi iteration, no memory is required in addition to the vectors $\mathbf{x}$, $\mathbf{b}$ and the matrix X, which makes the algorithm suitable for very large vectors and memory-constrained situations. In addition, convergence is often significantly faster.

The sum of elements $\mathbf{x}\mathbf{1}^{\mathrm{T}}$ does not stay constant during Jacobi or Gauss–Seidel iteration. Thus, when solving equations of the form $\mathbf{x}Q = \mathbf{0}, \mathbf{x}\mathbf{1}^{\mathrm{T}} = 1$, normalization cannot be entierly handled by the initial guess. We instead transform the equation into the form

$$\mathbf{x} \begin{pmatrix} q[0,0] & q[0,1] & \cdots & q[0,n-2] & 1 \\ q[1,0] & q[1,1] & \cdots & q[1,n-2] & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ q[n-2,0] & q[n-2,1] & \cdots & q[n-2,n-2] & 1 \\ q[n-1,0] & q[n-1,1] & \cdots & q[n-2,n-1] & 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}, \tag{5.4}$$

where we take advantage of the fact that the infinitesimal generator matrix is not of

Algorithm 5.6 Gauss–Seidel successive over-relaxatation.

Input: matrix $A \in \mathbb{R}^{n\times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$, initial guess $\mathbf{x} \in \mathbb{R}^n$, tolerance $\tau > 0$, over-relaxation parameter $\omega > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$

allocate $\mathbf{x}' \in \mathbb{R}^n$
Let A_O refer to the off-diagonal part of A.
repeat
  $\epsilon \leftarrow 0$
  for $i \leftarrow 0$ **to** $n-1$ **do**
    $scalarProduct \leftarrow \mathbf{x} \cdot \mathbf{a}_O[\cdot, i]$ ▷ VectorMatrixScalarProductWithColumn
    $y \leftarrow \omega(b[i] - scalarProduct)/a[i,i] + (1-\omega) \cdot x[i]$
    $\epsilon \leftarrow \epsilon + |y - x[i]|$
    $x[i] \leftarrow y$
until $\epsilon \le \tau$
return $\mathbf{x}$

full rank, therefore one of the columns is redundant and can be replaced with the condition $\mathbf{x}\mathbf{1}^{\mathrm{T}} = 1$. While this transformation may affect the convergence behavior of the algorithm, it allows uniform handling of homogenous and non-homogenous linear equations.

5.1.3 Group iterative methods

Group or *block* iterative methods Stewart [63, Section 10.4] assume the block structure for the vectors $\mathbf{x}$, $\mathbf{b}$ and the matrix A

$$\mathbf{x}[i] \in \mathbb{R}^{n_i}, \mathbf{b}[j] \in \mathbb{R}^{n_j}, A[i,j] \in \mathbb{R}^{n_i \times n_j} \text{ for all } i, j \in \{0, 1, \ldots, N-1\},$$

Infinitesimal generator matrices in the block Kronecker decomposition along with appropriately partitioned vectors match this structure (see eq. (2.10) on page 14). Each block of $\mathbf{x}$ corresponds to a group a variables that are simultaneously solved for.

Group Jacobi iteration solves the linear system

$$\left.\begin{aligned}
\mathbf{b}[0] &= \mathbf{x}_k[0]A[0,0] + \mathbf{x}_{k-1}[1]A[1,0] + \cdots + \mathbf{x}_{k-1}[n-1]A[n-1,0],\\
\mathbf{b}[1] &= \mathbf{x}_{k-1}[0]A[0,1] + \mathbf{x}_k[1]A[1,1] + \cdots + \mathbf{x}_{k-1}[n-1]A[n-1,1],\\
&\vdots\\
\mathbf{b}[n-1] &= \mathbf{x}_{k-1}[0]A[0,n-1] + \mathbf{x}_{k-1}[1]A[1,n-1] + \cdots + \mathbf{x}_k[n-1]A[n-1,n-1],
\end{aligned}\right\}$$

Algorithm 5.7 Group Jacobi over-relaxation.

Input: block matrix A, block right vector $\mathbf{b}$, block initial guess $\mathbf{n}$, tolerance $\tau > 0$, over-relaxation parameter $\omega > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$ and its residual norm

allocate $\mathbf{x}'$ and $\mathbf{c}$ with the same block structure as $\mathbf{x}$ and $\mathbf{b}$
Let A_{OB} represent the off-diagonal part of the block matrix A with the blocks along the diagonal set to zero.
repeat
    $\mathbf{x}' \leftarrow \mathbf{x}, \mathbf{c} \leftarrow \mathbf{b}$
    $\mathbf{c} \leftarrow \mathbf{c} + (-1) \cdot \mathbf{x}' A_{OB}$ ▷ AccumulateVectorMatrixMultiplyFromLeft
    parallel for $i \leftarrow 0$ **to** $N-1$ **do** // Loop over all blocks
        Solve $\mathbf{x}[i]A[i,i] = \mathbf{c}[i]$ for $\mathbf{x}[i]$
    $\epsilon \leftarrow 0$
    for $k \leftarrow 0$ **to** $n-1$ **do** // Loop over all elements
        $y \leftarrow \omega x[k] + (1-\omega)x'[k]$
        $\epsilon \leftarrow \epsilon + |y - x'[k]|$
        $x[k] \leftarrow y$
until $\epsilon \leq \tau$

while group Gauss–Seidel considers

$$\left.\begin{aligned}
\mathbf{b}[0] &= \mathbf{x}_k[0]A[0,0] &&+ \mathbf{x}_{k-1}[1]A[1,0] &&+ \cdots + \mathbf{x}_{k-1}[n-1]A[n-1,0],\\
\mathbf{b}[1] &= \mathbf{x}_k[0]A[0,1] &&+ \mathbf{x}_k[1]A[1,1] &&+ \cdots + \mathbf{x}_{k-1}[n-1]A[n-1,1],\\
&&&\vdots\\
\mathbf{b}[n-1] &= \mathbf{x}_k[0]A[0,n-1] &&+ \mathbf{x}_k[1]A[1,n-1] &&+ \cdots + \mathbf{x}_k[n-1]A[n-1,n-1].
\end{aligned}\right\}$$

Implementations of group Jacobi over-relaxation and group Gauss–Seidel successive over-relaxation are shown in Algorithms 5.7 and 5.8 on the current page and. The inner linear equations of the form $\mathbf{x}[i]A[i,i] = \mathbf{c}$ may be solved by any algorithm, for example, LU decomposition, iterative methods, or even block-iterative methods if A has a two-level block structure. The choice of the inner algorithm may significantly affect performance and care must be taken to avoid diverging inner solutions in an iterative solver is used.

In Jacobi over-relaxation, paralellization of both the matrix multiplication and the inner loop is possible. However, two vectors of the same size as $\mathbf{x}$ are required for temporary storage.

Gauss–Seidel successive over-relaxation cannot be parallelized easily. However it requires only two temporary vectors of size equal to the largest block of $\mathbf{x}$, much less than Jacobi over-relaxation. Moreover, it often requires fewer steps to converge, making

Algorithm 5.8 Group Gauss–Seidel successive over-relaxation.

Input: block matrix A, block right vector $\mathbf{b}$, block initial guess $\mathbf{n}$, tolerance $\tau > 0$, over-relaxation parameter $\omega > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$ and its residual norm

allocate $\mathbf{x}'$ and $\mathbf{c}$ large enough to store a single block of $\mathbf{x}$ and $\mathbf{b}$.
repeat
  $\epsilon \leftarrow 0$
  for $i \leftarrow 0$ **to** $N-1$ **do** // Loop over all blocks
    $\mathbf{x}' \leftarrow \mathbf{x}[i], \mathbf{c} \leftarrow \mathbf{b}[i]$
    for $j \leftarrow 0$ **to** $N-1$ **do**
      if $i \neq j$ **then** ▷ AccumulateVectorMatrixMultiplyFromLeft
        $\mathbf{c} \leftarrow \mathbf{c} + (-1) \cdot \mathbf{x}[j]A[i,j]$
    Solve $\mathbf{x}[i]A[i,i] = \mathbf{c}$ for $\mathbf{x}[i]$
    for $k \leftarrow 0$ **to** $n_i - 1$ **do**
      $y \leftarrow \omega x[i][k] + (1-\omega)x'[k]$
      $\epsilon \leftarrow \epsilon + |y - x'[k]|$
      $x[i][k] \leftarrow y$
until $\epsilon \leq \tau$

it preferable over Jacobi iteration.

Because the inner solver may be selected by the user and thus its convergence behaviour varies widely, we do not perform the transformation for homogenous equations (5.4). Instead, the normalization $\boldsymbol{\pi} = \mathbf{x}/\mathbf{x}\mathbf{1}^{\mathrm{T}}$ is performed only after finding any nonzero solution of $\mathbf{x}Q = \mathbf{0}$.

For a detailed analysis of the convergence behaviour of group iterative methods, we refer to Greenbaum [33, Chapter 14] and **TODO!!!**

5.1.4 Krylov subspace methods

Projectional iterative methods are iterative linear equation solvers that produce a sequence of approximate solutions $\mathbf{x}_k$ of the linear equation $\mathbf{x}Q = \mathbf{b}$ that satisfy the Petrov–Galerkin conditions **TODO Cite**

$$\mathbf{x}_k \in \mathcal{K}_k, \quad \mathbf{r}_k = \mathbf{b} - \mathbf{x}_k Q \perp \mathcal{L}_k, \tag{5.5}$$

where $\mathcal{K}_k$ and $\mathcal{L}_k$ are two subspaces of $\mathbb{R}^n$ and $\mathbf{r}_k$ is residual in the kth iteration.

Krylov subspace iterative methods correspond to the choice

$$\mathcal{K}_k = \mathcal{K}_k(Q, \mathbf{r}_0) = \operatorname{span}\{\mathbf{r}_0, \mathbf{r}_0 Q, \mathbf{r}_0 Q^2, \ldots, \mathbf{r}_0 Q^{k-1}\},$$

where $\mathcal{K}_k(Q, \mathbf{r}_0)$ is the *kth Krylov subspace* of Q and the initial residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{x}_0 Q$.

The smallest $m \in \mathbb{N}$ such that $\dim \mathcal{K}_m(Q, \mathbf{r}_0) = \dim \mathcal{K}_{m+1}(Q, \mathbf{r}_0)$ is called the *grade* of Q with respect to $\mathbf{r}_0$. Hence $k \leq m$ implies $\dim \mathcal{K}_k(Q, \mathbf{r}_0) = k$. Krylov subspace solvers usually suppose that the algorithm terminates at some iteration k^* such that $k^* \leq m$, therefore the dimension of $\mathcal{K}_k$ increases with each iteration. The contary situation leads to stagnation, because $\mathcal{K}_k \subseteq \mathcal{K}_{k+1}$ together with $\dim \mathcal{K}_k = \dim \mathcal{K}_{k+1}$ $(k \geq m)$ implies $\mathcal{K}_k = \mathcal{K}_{k+1}$.

The subspace $\mathcal{L}_k$ also must be a k-dimensional subspace of $\mathbb{R}^n$. Conceptually, while the Krylov subspace $\mathcal{K}_k$ "expands" in dimensionality every iteration, the subspace $\mathcal{L}_k$ likewise fills the space to make additional residuals forbidden by the Petrov–Galerkin condition (5.5).

If $Q \in \mathbb{R}^{n \times n}$ is of full rank and grade, Krylov subspace solvers find the exact solution of the linear equation in at most n iterations with exact arithmetic. The only possible orthogonal residual is the zero vector $\mathbf{0}$ if $\mathcal{L}_n = \mathbb{R}^n$ holds. While n is usually too large for this to be practical, convergence often happens with suitable accuracy after a small number of iterations.

Note that problems may arise when Q is singular, which may worsen the convergence behaviour. This is the case in CTMC analysis, where the infinitesimal generator matrix Q is of rank $n-1$.

Some Krylov subspace methods for nonsymmetric matrices in wide use are Generalized Minimum Residual (GMRES) [55], Bi-Conjugate Gradient Stabilized (BiCGSTAB) [67], Conjugate Gradient Squared (CGS) [61] and IDR(s) [62].

Generalized Minimal Residual (GMRES)

Generalized Minimal Residual (GMRES) [56, Section 6.5.1; 55] a Krylov subspace method for nonsymmetric linear systems. It is based on the choice

$$\mathcal{L}_k = \mathcal{K}_k Q = \{\mathbf{r}_0 Q, \mathbf{r}_0 Q^2, \ldots, \mathbf{r}_0 Q^k\}.$$

With this choice, the Petrov–Galerkin condition (5.5) minimizes the Euclidean norm of the residuals in each iteration, i.e.

$$\mathbf{x}_k \in \mathcal{K}_k \text{ such that } \mathbf{r}_k = \mathbf{b} - \mathbf{x}_k Q \perp \mathcal{L}_k \quad \Longleftrightarrow \quad \mathbf{x}_k = \underset{\mathbf{x} \in \mathcal{K}_k}{\arg\min} \|\mathbf{b} - \mathbf{x}Q\|_2. \tag{5.6}$$

Unfortunately, the solution of eq. (5.6) requires the storage of a basis of $\mathcal{K}_k$, which is a k dimensional subspace of $\mathbb{R}^n$. Thus, each iteration requires the allocation of an additional vector. Solution of a linear system with GMRES requires up to n additional floating-point vectors of n elements each, i.e. $O(n^2)$ floating-point numbers. This property makes GMRES a "long recurrence" algorithm.

The high memory requirements may be alleviated by discarding the basis of $\mathcal{K}_k$ and restarting the iteration from another initial guess $\mathbf{x}_0$ if no solution is obtained after ℓ iterations. The resulting algorithm is called GMRES(ℓ).

The convergence behaviour of full GMRES is often excellent. However, due to impractical memory requirements, we did not implement GMRES as a numerical solver in our framefork. We instead use BiCGSTAB and IDR(s)STAB(ℓ), Krylov subspace solvers incorporating GMRES(ℓ)-like steps.

Bi-Conjugate Gradient Stabilized (BiCGSTAB)

Bi-Conjugate Gradient Stabilized (BiCGSTAB) [56, Section 7.4.2; 67] is a Krylov subspace method where [58]

$$\mathcal{L}_k = \mathcal{K}_k(Q^{\mathrm{T}}, \tilde{\mathbf{r}}_0) \cdot (\Omega_k(Q)^{\mathrm{T}})^{-1}, \quad \Omega_k(Q) = \begin{cases} \Omega_{k-1}(Q) \cdot (I - \omega_k Q) & \text{if } k \geq 1, \\ I & \text{if } k = 0. \end{cases} \tag{5.7}$$

The *initial shadow residual* $\tilde{\mathbf{r}}_0$ must satisfy $\mathbf{r}_0 \tilde{\mathbf{r}}_0^{\mathrm{T}} \neq 0$ and must not be an eigenvector of Q^{T}. Usually, $\tilde{\mathbf{r}}_0 = \mathbf{r}_0$, which is the convention we use in our implementation.

Equivalently, BiCGSTAB is a Krylov subspace method which produces residuals

$$\mathbf{r}_k \in \mathcal{G}_k, \quad \mathcal{G}_k = \begin{cases} (\mathcal{G}_k \cap \tilde{\mathbf{r}}_0^{\perp})(I - \omega_k Q) & \text{if } k \geq 1, \\ \mathbb{R}^n & \text{if } k = 0, \end{cases} \tag{5.8}$$

where $\mathcal{A}^{\perp}$ is the set of vector orthogonal to $\mathcal{A}$. It can be shown that [60]

$$\mathcal{G}_k = \mathcal{S}(\Omega_k, Q, \tilde{\mathbf{r}}_0) = \{\mathbf{v} \cdot \Omega_k(Q) : \mathbf{v} \perp \mathcal{K}_k(Q^{\mathrm{T}}, \tilde{\mathbf{r}}_0)\},$$

where $\mathcal{S}(\Omega, Q, \tilde{\mathbf{r}}_0)$ is the Ωth *Sonneveld subspace* generated by Q and $\tilde{\mathbf{r}}_0$) of order $k = \deg \Omega$. Hence $\mathcal{L}_k = \mathcal{G}_k^{\perp}$, which makes BiCGSTAB equivalent to another Krylov subspace method, Induced Dimensionality Reduction (IDR) [60] in exact arithmetic.

BiCGSTAB is a "short recurrence", that is, the number of allocated intermediate vectors does not depend on the number of variables in equation system.

Implementation We selected BiCGSTAB as the first Krylov subspace solver integrated into our framework because of its good convergence behaviour and low memory requirements. BiCGSTAB only requires the storage of 7 vectors, which makes it suitable even for large state spaces with large state vectors.

Algorithm 5.9 shows the pseudocode for BiCGSTAB. Our implementation is based on the MATLAB code[1] by Barrett et al. [5].

[1] `http://www.netlib.org/templates/matlab/bicgstab.m`

Algorithm 5.9 BiCGSTAB iteration without preconditioning.

Input: matrix $A \in \mathbb{R}^{n \times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$, initial guess $\mathbf{x} \in \mathbb{R}^n$, tolerance $\tau > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$

allocate $\mathbf{r}, \tilde{\mathbf{r}}_0, \mathbf{v}, \mathbf{p}, \mathbf{s}, \mathbf{t} \in \mathbb{R}^n$
$\mathbf{r} \leftarrow \mathbf{b}$ ▷ VectorSet
$\mathbf{r} \leftarrow \mathbf{r} + (-1) \cdot \mathbf{x}A$ ▷ VectorMatrixAccumulateMultiplyFromLeft
if $\|\mathbf{r}\| \leq \tau$ **then**
    message "initial guess is correct, skipping iteration"
    return $\mathbf{x}$
$\tilde{\mathbf{r}}_0 \leftarrow \mathbf{r}, \mathbf{v} \leftarrow \mathbf{0}, \mathbf{p} \leftarrow \mathbf{0}, \rho' \leftarrow 1, \alpha \leftarrow 1, \omega \leftarrow 1$
while true **do**

Bi-CG step

$\rho \leftarrow \mathbf{r_0} \cdot \mathbf{r}$ ▷ VectorScalarProduct
    if $\rho \approx 0$ **then error** "breakdown: $\mathbf{r} \perp \tilde{\mathbf{r}}_0$"
    $\beta \leftarrow \rho / \rho' \cdot \alpha / \omega$
    $\mathbf{p} \leftarrow \mathbf{r} + \beta \cdot \mathbf{p}$ ▷ VectorAdd
    $\mathbf{p} \leftarrow \mathbf{p} + (-\beta\omega) \cdot \mathbf{v}$ ▷ In-place VectorAdd
    $\mathbf{v} \leftarrow \mathbf{p}Q$ ▷ VectorMatrixMultiplyFromLeft
    $\alpha \leftarrow \rho / (\tilde{\mathbf{r}}_0 \cdot \mathbf{v})$ ▷ ScalarProduct
    $\mathbf{r} \leftarrow \mathbf{s} + (-\alpha) \cdot \mathbf{s}$ ▷ VectorAdd
    if $\|\mathbf{s}\| < \tau$ **then**
        $\mathbf{x} \leftarrow \mathbf{x} + \alpha \cdot \mathbf{p}$ ▷ In-place VectorAdd
        message "early return with vanishing $\mathbf{s}$"
        return $\mathbf{x}$

GMRES(1) step

$\mathbf{t} \leftarrow \mathbf{s}A$ ▷ VectorMatrixMultiplyFromLeft
    $tLengthSquared \leftarrow \mathbf{t} \cdot \mathbf{t}$ ▷ ScalarProduct
    if $tLengthSquared \approx 0$ **then error** "breakdown: $\mathbf{t} \approx \mathbf{0}$"
    $\omega \leftarrow (\mathbf{t} \cdot \mathbf{s}) / tLengthSquared$ ▷ ScalarProduct
    if $\omega \approx 0$ **then error** "breakdown: $\omega \approx 0$"
    $\epsilon \leftarrow 0$
    for $i \leftarrow 0$ **to** $n - 1$ **do**
        $change \leftarrow \alpha p[i] + \omega s[i]$, $\epsilon \leftarrow \epsilon + |change|$. $x[i] \leftarrow x[i] + change$
    if $\epsilon \leq \tau$ **then return** $\mathbf{x}$
    $\mathbf{s} \leftarrow \mathbf{t} + (-\omega) \cdot \mathbf{r}$ ▷ VectorAdd
    $\rho' \leftarrow \rho$

The inner loop of BiCGSTAB is the composed of two procedures. The bi-conjucate gradient (Bi-CG) part in lines **??–??** calculates a residual $\mathbf{t} \in \mathcal{G}_{k-1}Q$ and its associated approximate solution $\mathbf{x} \in \mathcal{K}_k$. The GMRES(1) part in lines **??–??** selects $\omega_k \in \mathbb{R}$ and calculates a new residual $\mathbf{r} \in \mathcal{G}_k$ such that the Euclidean norm $\|\mathbf{r}\|_2$ is minimized. This part improves convergence over the original Bi-Conjucate Gradient algorithm.

Solving preconditioned equations in the form $\mathbf{x}AM^{-1} = \mathbf{b}M^{-1}$ could improve convergence, but was omitted from our current implementation. As the choice of appropriate preconditioner matrices M is not trivial [42], implementation and sudy of preconditioners for Markov chains, especially with block Kronecker decomposition, is in the scope of our future work.

Because six vectors are allocated in addition to $\mathbf{x}$ and $\mathbf{b}$, the amount of available memory may be a significant bottleneck.

Similar to Observation 5.1 on page 33, it can be seen that the sum $\mathbf{x}\mathbf{1}^{\mathrm{T}}$ stays constant throughout BiCGSTAB iteration. Thus, we can find probability vectors satisfying homogenous equations by the initialization in eq. (5.3) on page 34.

Induced Dimensionality Reduction Stabilized (IDRSTAB)

Induced Dimensionality Reduction Stabilized (IDR(s)STAB(ℓ)) [60] is Krylov subspace solver that generalizes BiCGSTAB and IDR techniques to provide converge behaviors closely matching GMRES while maintaining the short recurrence property.

As the algorithm developed relatively recently in 2010, high performance implementations of IDR(s)STAB(ℓ) are not widely available. To our best knowledge, IDR(s)STAB(ℓ) was not investigated for use in CTMC analysis despite its promising results solving differential equations arising from finite element problems. Therefore, we are currently focusing research and development effort into integrating IDR(s)STAB(ℓ) into our stochastic analysis. Special attention is paid to its behaviour on steady-state equations with infinitesimal generator matrices and other linear systems arising from CTMC analysis.

IDR(s)STAB(ℓ) merges two generalizations of BiCGSTAB:

- The first idea comes from IDR(s) [62], a Krylov subspace solver based on Sonneveld subspaces. A block version of eq. (5.8) constraints the residual $\mathbf{r}_k$

$$\mathbf{r}_k \in \mathcal{G}_k = \mathcal{S}(\Omega_k, Q, \widetilde{R}_0) = \{\mathbf{v} \cdot \Omega_k(Q) : \mathbf{v} \perp \mathcal{K}_k(Q, \tilde{R}_0)\},$$

  where $\mathcal{K}_k(Q, \tilde{R}_0)$ is the kth *row* Krylov subspace of $Q \in \mathbb{R}^{n \times n}$ with respect to $\tilde{R}_0 \in \mathbb{R}^{s \times n}$

$$\mathcal{K}_k(Q, \widetilde{R}_0) = \operatorname{span}\{\tilde{\mathbf{r}}_0[i], \tilde{\mathbf{r}}_0[i]Q, \ldots, \tilde{\mathbf{r}}_0[i]Q^{k-1} : i = 0, 1, \ldots, s-1\}$$

  and $\tilde{\mathbf{r}}_0[i]$ is the ith row $\tilde{R}_0$.

Higher values of s, i.e. higher dimensional initial shadow spaces, may accelerate convergence, at the cost of allocation additional intermediate vectors.

- The second generalization, which is called BiCGSTAB(ℓ) [59], replaces the stabilizer polynomial Ω_k from eq. (5.7) with

$$\Omega_k(Q) = \Omega_{k-1}(Q) \cdot (I - \gamma[0]Q - \gamma[1]Q^2 - \cdots - \gamma[\ell-1]Q^\ell),$$

  i.e. degree of the stabilizer polynomial Ω increases by ℓ instead of 1 every iteration. The increase is described by the vector $\vec{\gamma} \in \mathbb{R}^\ell$.

  The higher-order stabilization, also called a GMRES(ℓ) step, improves convergence behavior with unsymmetrix matrices that have complex spectrum. However, the number of intermediate vectors, thus the amount of required memory, also grows.

A single dimensional initial shadow space ($s = 1$) and first-order stabilization ($\ell = 1$) make IDR(s)STAB(ℓ) identical to BiCGSTAB. Moreover, $\ell = 1$ results is behavior equivalent to IDR(s), while $s = 1$ results in behavior equivalent to BiCGSTAB(ℓ).

These correspondences make IDR(s)STAB(ℓ) a promising candidate for use in configurable stochastic analysis, as different settings of (s, ℓ) bring the power of multiple algorithms to the modelers' disposal.

Implementation The pseudocode of our implementation of IDR(s)STAB(ℓ), which is based on the pseudocode of Sleijpen and Van Gijzen [60], is show in **TODO**.

For convenient representation of memory requirements, we employ two different typographical styles for vectors. Vectors in bold, e.g. $\mathbf{x} \in \mathbb{R}^n$ are "long" vectors, while vectors with arrows, e.g. $\vec{\gamma}$ are "short" vectors of length s or $\ell \ll n$. Storage space of long vectors dominated memory requirements and their manipulations including vector–matrix products dominate computation time.

The algorithm works with three arrays of vectors, $\mathbf{R} \in \mathbb{R}^{(\ell+1)\times n}$, $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{((\ell+2)\times s)\times n}$, i.e. $\mathbf{r}[j'], \mathbf{u}[j,q], \mathbf{v}[j,q] \in \mathbb{R}^n$ for all $j' = 0, 1, \ldots, \ell$; $j = 0, 1, \ldots, \ell+1$; $q = 0, 1, \ldots, s-1$.

The IDR part performs the projections, called a "repetition step",

$$
\begin{array}{cccccccc}
\mathbf{x}_- & \to & \mathbf{x} & & \mathbf{v}[0,0] & & \mathbf{v}[0,1] & \cdots & \mathbf{v}[0,s-1], \\
 & \Pi_1 & \downarrow & \nearrow \Pi_0 & \downarrow Q & \nearrow \Pi_0 & \downarrow Q & & \downarrow Q \\
\mathbf{r}_-[0] & \to & \mathbf{r}[0] & & \mathbf{v}[1,0] & & \mathbf{v}[1,1] & \cdots & \mathbf{v}[1,s-1], \\
 & \Pi_2 & \downarrow Q & \nearrow \Pi_1 & \downarrow Q & \nearrow \Pi_1 & \downarrow Q & & \downarrow Q \\
\mathbf{r}_-[1] & \to & \mathbf{r}[1] & & \mathbf{v}[2,0] & & \mathbf{v}[2,1] & \cdots & \mathbf{v}[2,s-1], \\
\vdots & \Pi_{j-1} & \vdots & & \vdots & & \vdots & & \vdots \\
\mathbf{r}_-[j-2] & \to & \mathbf{r}[j-2] & & \mathbf{v}[j-1,0] & & \mathbf{v}[j-1,1] & \cdots & \mathbf{v}[j-1,s-1], \\
 & \Pi_j & \downarrow Q & \nearrow \Pi_{j-1} & \downarrow Q & \nearrow \Pi_{j-1} & \downarrow Q & & \downarrow Q \\
\mathbf{r}_-[j-1] & \to & \mathbf{r}[j-1] & & \mathbf{v}[j,0] & & \mathbf{v}[j,1] & \cdots & \mathbf{v}[j,s-1], \\
 & & \downarrow Q & \nearrow \Pi_j & \downarrow Q & \nearrow \Pi_j & \downarrow Q & & \downarrow Q \\
 & & \boxed{\mathbf{r}[j]} & & \boxed{\mathbf{v}[j+1,0]} & & \boxed{\mathbf{v}[j+1,1]} & \cdots & \boxed{\mathbf{v}[j+1,s-1]}
\end{array}
\tag{5.9}
$$

for $j = 1, 2, \ldots, \ell$. After the repetition step is complete, $\mathbf{U}$ and $\mathbf{V}$ are swapped and the process starts again with increased j or the GMRES(ℓ) part commences. Symbols with a subscript "–" sign refer to vectors from the previous repetition step.

The projections Π_i $(i = 0, 1, \ldots, j)$ are defined as

$$\Pi_i = I - A^{j-i}\widetilde{R}_0\sigma^{-1}(U[i,\cdot])^{\mathrm{T}}, \quad \sigma = \widetilde{R}_0(U[j,\cdot])^{\mathrm{T}},$$

where the matrix $U[k,\cdot]$ is the $s \times n$ matrix that has the vector $\mathbf{u}[k,q]$ as its qth row. They ensure that the rows of $U[j,\cdot]$ form a basis of the Krylov subspace $\mathcal{K}_s(Q\Pi_j, \mathbf{r}Q^j\Pi_j)$ after the jth repetition step.

The relationships $\mathbf{r}[i+1] = \mathbf{r}iQ$, $\mathbf{u}[i+1,q] = \mathbf{u}[i,q]Q$, $\mathbf{v}[i+1,q] = \mathbf{v}[i,q]$ are maintained throughout the algorithm via the projections. This is signified by the gray $\downarrow Q$ arrows in eq. (5.9). Notice that this means $\mathbf{r}[0]Q^i = \mathbf{r}[i]$ and $U[0,\cdot]Q^i = U[i,\cdot]$.

In the case of $\mathbf{r}[j]$ and $\mathbf{v}[j+1,q]$, a matrix multiplication is performed as shown by the $\downarrow Q$ arrows. Vectors generated by matrix multiplication are shown in borders.

For improving numerical properties, Sleijpen and Van Gijzen [60] recommend performing Gram–Schmidt ortonormalization on $U[j,\cdot]$. The same subtractions and normalization operations must be performed on the rows of $U[i,\cdot]$, $i \neq j$ that are performed in $U[j,\cdot]$ in order to maintain their relationships. We realized the orthonormalization by the modified Gram–Schmidt process in lines **TODO** and **TODO**.

The storage of $\mathbf{R}$, $\mathbf{U}$ and $\mathbf{V}$ requires $(\ell+1) + 2\cdot(\ell+1)\cdot s$ vectors of length n, while the initial shadow residual matrix $\widetilde{R}_0$ requires space equal to s vectors of length n. Thus, $1 + \ell + 3s + 2\ell s$ intermediate vectors are needed in addition to the initial guess $\mathbf{x}_0$ and the right vectors $\mathbf{b}$. Although the memory requiremens of IDR(s)STAB(ℓ) are quite high, s and ℓ can be selected to ensure that the solution fits in the available memory.

A different formulation of the IDR(s)STAB(ℓ) principles is GBi-CGSTAB(s,ℓ) [65], which avoids the allocation of $\mathbf{V}$ by updating $\mathbf{U}$ in place, albeit with lesser numerical properties due to the lack of orthonormalization steps. Another variant by Aihara et al. [1] replaces some vector updates with matrix multiplications to improve accuracy.

Numerical problems and breakdown **TODO Leiras, diagram**

5.2 Transient analysis

5.2.1 Uniformization

The *uniformization* or *randomization* method solves the initial value problem

$$\frac{\mathrm{d}\boldsymbol{\pi}(t)}{\mathrm{d}t} = \boldsymbol{\pi}(t)\,Q, \quad \boldsymbol{\pi}(t) = \boldsymbol{\pi}0 \tag{2.1 revisited}$$

by computing

$$\boldsymbol{\pi}(t) = \sum_{k=0}^{\infty} \boldsymbol{\pi}_0 P^k e^{-\alpha t} \frac{(\alpha t)^k}{k!}, \tag{5.10}$$

where $P = \alpha^{-1}Q + I$, $\alpha \geq \max_i |a[i,i]|$ and $e^{-\alpha t}\frac{(\alpha t)^k}{k!}$ is the value of the Poisson probabilty function with rate αt at k.

Integrating both sides of eq. (5.10) to compute $\mathbf{L}(t)$ yields [54]

$$\begin{aligned}
\int_0^t \boldsymbol{\pi}(u)\,\mathrm{d}u = \mathbf{L}(t) &= \sum_{k=0}^{\infty} \boldsymbol{\pi}_0 P^k \int_0^t e^{-\alpha u}\frac{(\alpha u)^k}{k!}\,\mathrm{d}u \\
&= \sum_{k=0}^{\infty} \boldsymbol{\pi}_0 P^k \frac{1}{\alpha} \sum_{l=k+1}^{\infty} e^{-\alpha t}\frac{(\alpha t)^l}{l!} \\
&= \frac{1}{\alpha}\sum_{k=0}^{\infty} \boldsymbol{\pi}_0 P^k \left(1 - \sum_{l=0}^{k} e^{-\alpha t}\frac{(\alpha t)^l}{l!}\right).
\end{aligned} \tag{5.11}$$

Both eqs. (5.10) and (5.11) can be realized as

$$\mathbf{x} = \frac{1}{W}\left(\sum_{k=0}^{k_{\text{left}}-1} w_{\text{left}}\boldsymbol{\pi}_0 P^k + \sum_{k=k_{\text{left}}}^{k_{\text{right}}} w[k-k_{\text{left}}]\boldsymbol{\pi}_0 P^k\right), \tag{5.12}$$

where $\mathbf{x}$ is either $\boldsymbol{\pi}(t)$ or $\mathbf{L}(t)$, k_{left} and k_{right} are *trimming constants* selected based on the required precision, $\mathbf{w}$ is a vector of (possibly accumulated) Poisson weights and W is a scaling factor. The weight before the left cutoff w_{left} is 1 if the accumulated probability vector $\mathbf{L}(t)$ is calculated, 0 otherwise.

Eq. (5.12) is implemented by Algorithm 5.10. The algorithm performs *steady-state* detection in line 9 to avoid unneccessary work once the iteration vector $\mathbf{p}$ reaches the steady-state distribution $\boldsymbol{\pi}(\infty)$, i.e. $\mathbf{p} \approx \mathbf{p}P$. If the initial distribution $\boldsymbol{\pi}_0$ is not further needed or can be generated efficiently (as it is the case with a single initial state), the result vector $\mathbf{x}$ may share the same storing, resulting in a memory overhead of only two vectors $\mathbf{p}$ and $\mathbf{q}$.

Algorithm 5.10 Uniformization.

Input: infinitesimal generator $Q \in \mathbb{R}^{n \times n}$, initial probability vector $\boldsymbol{\pi}_0 \in \mathbb{R}^n$, truncation parameters $k_{\text{left}}, k_{\text{right}} \in \mathbb{N}$, weights $w_{\text{left}} \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}^{k_{\text{right}} - k_{\text{left}}}$, scaling constant $W \in \mathbb{R}$, tolerance $\tau > 0$
Output: instantenous or accumulated probability vector $\mathbf{x} \in \mathbb{R}^n$

1 **allocate** $\mathbf{x}, \mathbf{p}, \mathbf{q} \in \mathbb{R}^n$
2 $\alpha^{-1} \leftarrow 1/\max_i |a[i,i]|$
3 $\mathbf{p} \leftarrow \boldsymbol{\pi}_0$
4 **if** $w_{\text{left}} = 0$ **then** $\mathbf{x} \leftarrow \mathbf{0}$ **else** $\mathbf{x} \leftarrow w_{\text{left}} \cdot \mathbf{p}$ ▷ VectorScale
5 **for** $k \leftarrow 1$ **to** k_{right} **do**
6 $\quad \mathbf{q} \leftarrow \mathbf{p}Q$ ▷ VectorMatrixMultiplyFromLeft
7 $\quad \mathbf{q} \leftarrow \alpha^{-1} \cdot \mathbf{q}$ ▷ In-place VectorScale
8 $\quad \mathbf{q} \leftarrow \mathbf{q} + \mathbf{p}$ ▷ In-place VectorAdd
9 $\quad$ **if** $\|\mathbf{q} - \mathbf{p}\| \leq \tau$ **then**
10 $\quad\quad \mathbf{x} \leftarrow \mathbf{x} + \left(\sum_{l=k}^{k_{\text{right}}} w[l - k_{\text{left}}]\right) \cdot \mathbf{q}$ ▷ In-place VectorAdd
11 $\quad\quad$ **break**
12 $\quad$ **if** $k < k_{\text{left}} \wedge w_{\text{left}} \neq 0$ **then** $\mathbf{x} \leftarrow \mathbf{x} + w_{\text{left}} \cdot \mathbf{q}$ ▷ In-place VectorAdd
13 $\quad$ **else if** $k \geq k_{\text{left}}$ **then** $\mathbf{x} \leftarrow \mathbf{x} + w[k - k_{\text{left}}] \cdot \mathbf{q}$ ▷ In-place VectorAdd
14 $\quad$ Swap the references to $\mathbf{p}$ and $\mathbf{q}$
15 $\mathbf{x} \leftarrow W^{-1} \cdot \mathbf{x}$ ▷ In-place VectorScale
16 **return** $\mathbf{x}$

The weights and trimming constants may be calculated by the famous algorithm of Fox and Glynn [29]. However, their algorithm is extremely complicated due to the limitations of single-precision floating-point arithmetic [39]. We implemented Burak's significantly simpler algorithm [18] in double precision instead (Algorithm 5.11), which avoids underflow by a scaling factor $W \gg 1$.

5.2.2 TR-BDF2

A weakness of the uniformization algorithm is the poor tolerance of *stiff* Markov chains. The CTMC is called stiff if the $|\lambda_{\min}| \ll |\lambda_{\max}|$, where $\lambda_{\min}$ and $\lambda_{\max}$ are the nonzero eigenvalues of the infinitesimal generator matrix Q of minimum and maximum absolute value [53]. In other words, stiff Markov chains have behaviors on drastically different timescales, for example, clients are served frequently while failures happen infrequently.

Stiffness leads to very large values of α in line 2 of Algorithm 5.10, thus a large right cutoff k_{right} is required for computing the transient solution with sufficient accuracy. Moreover, the slow stabilization results in taking many iterations before steady-state

Algorithm 5.11 Burak's algorithm for calculating the Poisson weights.

Input: Poisson rate $\lambda = \alpha t$, tolerance $\tau > 10^{-50}$
Output: truncation parameters $k_{\text{left}}, k_{\text{right}} \in \mathbb{N}$, weights $\mathbf{w} \in \mathbb{R}^{k_{\text{right}} - k_{\text{left}}}$, scaling constant $W \in \mathbb{R}$

Calculate weights with high precision

$M_w \leftarrow 30, M_a \leftarrow 44, M_s \leftarrow 21$ // Constants determine cutoff estimation accuracy
$m \leftarrow \lfloor \lambda \rfloor, tSize \leftarrow \lfloor M_w \sqrt{\lambda} + M_a \rfloor, tStart \leftarrow \max\{m + M_s - \lfloor tSize/2 \rfloor, 0\}$
allocate tWeights $\in \mathbb{R}^{tSize}$
$tWeights[m - tStart] \leftarrow 2^{176}$
for $j \leftarrow m - tStart$ **downto** 1 **do**
  $tWeights[j-1] = (j + tStart)\, tWeights[j]/\lambda$
for $j \leftarrow m - tStart + 1$ **to** $tSize$ **do**
  $tWeights[j+1] = \lambda\, tWeights[j]/(j + tStart)$

Determine normalization constant and cutoff points

$W \leftarrow 0$
for $j \leftarrow 0$ **to** $m - tStart - 1$ **do**
  $W \leftarrow W + tWeights[j]$
$sum1 \leftarrow 0$ // Avoid adding small numbers to larger numbers
for $j \leftarrow tSize - 1$ **downto** $m - tStart$ **do**
  $sum1 \leftarrow sum1 + tWeights[j]$
$W \leftarrow W + sum1, threshold \leftarrow W\tau/2, cdf \leftarrow 0, i \leftarrow 0$
while $cdf < threshold$ **do**
  $cdf \leftarrow cdf + tWeights[i]$
  $i \leftarrow i + 1$
$k_{\text{left}} \leftarrow tStart + i, cdf \leftarrow 0, i \leftarrow tSize - 1$
while $cdf < threshold$ **do**
  $cdf \leftarrow cdf + tWeights[i]$
  $i \leftarrow i - 1$
$k_{\text{right}} \leftarrow tStart + i$

Copy weights between cutoff points

allocate $\mathbf{w} \in \mathbb{R}^{k_{\text{right}} - k_{\text{left}}}$
for $j \leftarrow k_{\text{left}}$ **to** k_{right} **do**
  $w[j - k_{\text{left}}] \leftarrow tWeights[j - tStart]$
return $k_{\text{left}}, k_{\text{right}}, \mathbf{w}, W$

detection in line 9.

Some methods that can handle stiff CTMCs efficiently are stochastic complementation [45], which decouples the slow and fast behaviors of the system, and adaptive uniformization [46], which varies the uniformization rate α. Alternatively, an L-stable differential equation solver may be used to solve eq. (2.1) on page 4, such as TR-BDF2 [4, 53].

TR-BDF2 is an implicit integrator with alternating trapezoid rule (TR) steps

$$\boldsymbol{\pi}_{k+\gamma}(2I + \gamma h_k Q) = 2\boldsymbol{\pi}_k + \gamma h_k \boldsymbol{\pi}_k Q$$

and second order backward difference steps

$$\boldsymbol{\pi}_{k+1}[(2-\gamma)I - (1-\gamma)h_k Q] = \frac{1}{\gamma}\boldsymbol{\pi}_{k+\gamma} - \frac{(1-\gamma)^2}{\gamma}\boldsymbol{\pi}_k,$$

which advance the time together by a step of size h_k. The constant $0 < \gamma < 1$ sets the breakpoint between the two steps. We set it to $\gamma = 2 - \sqrt{2} \approx 0.59$ following the recommendation of Bank et al. [4].

As a guess for the initial step size h_0, we chose the uniformization rate of Q. The kth step size $h_k > 0$, including the 0th one, is selected such that the local error estimate

$$LTE_{k+1} = \left\| 2\frac{-3\gamma^4 + 4\gamma - 2}{24 - 12\gamma} h_k \left[-\frac{1}{\gamma}\boldsymbol{\pi}_k + \frac{1}{\gamma(1-\gamma)}\boldsymbol{\pi}_{k+\gamma} - \frac{1}{1-\gamma}\boldsymbol{\pi}_{k+1} \right] \right\| \tag{5.13}$$

is bounded by the local error tolerance

$$LTE_{k+1} \leq \left(\frac{\tau - \sum_{i=0}^{k} LTE_i}{t - \sum_{i=0}^{k} k_i} \right) h_{k+1}.$$

This Local Error per Unit Step (LEPUS) error control "produces excellent results for many problems", but is usually costly [53]. Moreover, the accumulated error at the end of integration may be larger than the prescribed tolerance τ, since eq. (5.13) is only an approximation of the true error.

An implementation of TR-BDF2 based on the pseudocode of A. L. Reibman and Trivedi [53] is shown in Algorithm 5.12.

In lines 10 and 13 any linear equation solver from Section 5.1 on page 30 may be used except power iteration, since the matrices, in general, do not have strictly negative diagonals. Due to the way the matrices, which are linear combinations of I and Q, are passed to the inner solvers, our TR-BDF2 integrator is currently limited to Q matrices which are not in block form.

The vectors $\boldsymbol{\pi}_0, \boldsymbol{\pi}_k$ and $\boldsymbol{\pi}_{k+\gamma}, \mathbf{d}_{k+1}$ may share storage, respectively, therefore only 4 state-space sized vectors are required in addition to the initial distribution $\boldsymbol{\pi}_0$.

Algorithm 5.12 TR-BDF2 for transient analysis.

Input: infinitesimal generator $Q \in \mathbb{R}^{n\times n}$, initial distribution $\boldsymbol{\pi}_0$, mission time $t > 0$, tolerance $\tau > 0$
Output: transient distribution $\boldsymbol{\pi}(t)$

allocate $\boldsymbol{\pi}_k, \boldsymbol{\pi}_{k+\gamma}, \boldsymbol{\pi}_{k+1}, \mathbf{d}_k, \mathbf{d}_{k+1}, \mathbf{y} \in \mathbb{R}^n$
$maxIncrease \leftarrow 10, leastDecrease \leftarrow 0.9$
$timeLeft \leftarrow t, h \leftarrow 1/\max_i |q[i,i]|, \gamma \leftarrow 2-\sqrt{2}, C \leftarrow \left|\frac{-3\gamma^4+4\gamma-2}{24-12\gamma}\right|, errorSum \leftarrow 0$
$\boldsymbol{\pi}_k \leftarrow \boldsymbol{\pi}_0$, $\mathbf{d}_k \leftarrow \boldsymbol{\pi}_k Q$ ▷ VectorMatrixMultiplyFromLeft
while $timeLeft > 0$ **do**
  $stepFailed \leftarrow$ false, $h \leftarrow \min\{h, timeLeft\}$
  while true **do**
    TR step
    $\mathbf{y} \leftarrow 2 \cdot \boldsymbol{\pi}_k$ ▷ VectorScale
    $\mathbf{y} \leftarrow \mathbf{y} + \gamma h \cdot \mathbf{d}_k$ ▷ In-place VectorAdd
    Solve $\boldsymbol{\pi}_{k+\gamma}(2I + -\gamma hQ) = \mathbf{y}$ for $\boldsymbol{\pi}_{k+\gamma}$ with initial guess $\boldsymbol{\pi}_k$
    BDF2 step
    $\mathbf{y} \leftarrow -\frac{(1-\gamma)^2}{\gamma} \cdot \boldsymbol{\pi}_k$ ▷ VectorScale
    $\mathbf{y} \leftarrow \frac{1}{\gamma} \cdot \boldsymbol{\pi}_{k+\gamma}$ ▷ In-place VectorScale
    Solve $\boldsymbol{\pi}_{k+1}((2-\gamma)I + (\gamma-1)hQ) = \mathbf{y}$ for $\boldsymbol{\pi}_{k+1}$ with initial guess $\boldsymbol{\pi}_{k+\gamma}$
    Error control and step size estimation
    $\mathbf{y} \leftarrow -\frac{1}{\gamma}\mathbf{d}_k$ ▷ VectorScale
    $\mathbf{y} \leftarrow \mathbf{y} + \frac{1}{\gamma(1-\gamma)}\boldsymbol{\pi}_{k+\gamma} Q$ ▷ VectorMatrixAccumulateMultiplyFromLeft
    $\mathbf{d}_{k+1} \leftarrow \boldsymbol{\pi}_{k+1} Q$ ▷ VectorMatrixMultipleFromLeft
    $\mathbf{y} \leftarrow \mathbf{y} + \left(-\frac{1}{1-\gamma}\right)\mathbf{d}_{k+1}$ ▷ In-place VectorAdd
    $LTE \leftarrow 2Ch\|\mathbf{y}\|, localTol \leftarrow (\tau - errorSum)/timeLeft \cdot h$
    if $LTE < localTol$ **then** // Successful step
      $timeLeft \leftarrow timeLeft - h, errorSum \leftarrow errorSum + LTE$
      // Do not try to increase h after a failed step
      if $\neg stepFailed$ **then** $h \leftarrow h \cdot \min\{maxIncrease, \sqrt[3]{localTol/LTE}\}$
      break
    $stepFailed \leftarrow$ true, $h \leftarrow h \cdot \min\{leastDecrease, \sqrt[3]{localTol/LTE}\}$
  Swap the references to $\boldsymbol{\pi}_k, \boldsymbol{\pi}_{k+1}$ and $\mathbf{d}_k, \mathbf{d}_{k+1}$
return $\boldsymbol{\pi}_k$

The most computationally intensive part is the solution of two linear equation per every attempted step, which may make TR-BDF2 extremely slow. However, its performance does *not* depend on the stiffness of the Markov chain, which may make it better suited to stiff CTMCs than uniformization [53].

5.3 Mean time to first failure

In MTFF calculation (Section 2.1.3 on page 7), quantities of the forms

$$MTFF = -\underbrace{\boldsymbol{\pi}_U Q_{UU}^{-1}}_{\boldsymbol{\gamma}} \mathbf{1}^{\mathrm{T}}, \quad \mathbb{P}(X(TFF_{+0}) = y) = -\underbrace{\boldsymbol{\pi}_U Q_{UU}^{-1}}_{\boldsymbol{\gamma}} \mathbf{q}_{UD'}^{\mathrm{T}} \qquad \text{(2.7, 2.8 revisited)}$$

are computed, where U, D, D' are the set of operations states, failure states and a specific failure mode $D' \subsetneq D$, respectively.

The vector $\boldsymbol{\gamma} \in \mathbb{R}^{|U|}$ is the solution of the linear equation

$$\boldsymbol{\gamma} Q_{UU} = \boldsymbol{\pi}_U \tag{5.14}$$

and may be obtained by any linear equation solver.

The sets $U, D = D_1 \cup D_2 \cup \cdots$ are constructed by the evaluation of CTL expressions. If the failure mode D_i is described by φ_i, then the sets D and U are described by CTL formulas $\varphi_D = \neg\mathbf{AX}\,\mathrm{true} \vee \varphi_1 \vee \varphi_2 \vee \cdots$ and $\varphi_U = \neg\varphi_D$, where the deadlock condition $\neg\mathbf{AX}\,\mathrm{true}$ is added to make (5.14) irreducible.

After the set U is generated symbolically, the matrix Q_{UU} may be decomposed in the same way as the whole state space S. Thus, the vector-matrix operations required for solving (5.14) can be executed as in steady-state analysis.

Chapter 6

Evaluation

Chapter 7

Conclusion and future work

TODO

We have developed and presented our *configurable stochastic analysis framework* for the dependability, reliability and performability analysis of complex asynchronous systems. Our presented approach is able to combine the strength and advantages of the different algorithms into one framework. We have not only implemented a stochastic analysis library, but we integrated the various state space traversal, generator matrix representation and numerical analysis algorithms together. Various optimization techniques were used during the development and many of the algorithms are paralellized to exploit the advantages of modern mulitcore processor architectures.

From the theoretical side, we have developed an algorithm which can efficiently compile the symbolic state space representation into the complex data structure representation of the stochastic process. We have formalised our algorithm and proved its correctness. This new algorithm helps us to exploit the efficient state space representation of symbolic algorithms in stochastic analysis.

In addition we have investigated the composability of the various data storage, numerical solution and state space representation techniques and combined them together to provide configurable stochastic analysis in our framework.

Extensive investigation was executed in the field to be able to develop more than 2 state space exploration algorithms, 3 state space representation algorithms, 3 generator matrix decomposition and representation algorithms, 7 steady-state solvers, 2 transient analysis algorithms and 4 different computation algorithms for engineering measures. Our long term goal is to provide these analysis techniques also for a wider community, we have integrated our library into the PetriDotNet framework. Our algorithms are used also in the education for illustration purposes of the various stochastic analysis techniques. In addition, our tool was also used in an industrial project: one of our case-studies is based on that project. The stochastic analysis library is built from more than 50 000 lines of code. More than 70 000 generated test cases serve to ensure correctness

as much as possible. In addition, software redundancy based testing was applied to further improve the quality of our library.

Despite our attempts to be as comprehensive as possible, many promising directions for future research and development are

- more extensive benchmarking of algorithms to extend the knowledge base about the effectiveness and behavior of stochastic analysis approaches toward and adaptive framework for stochastic analysis;
- support for extended formalisms for stochastic models, such as Generalized Stochastic Petri Nets (GSPN) [66] and Stochastic Automata Networks (SAN) [37], as well as models with more general stochastic transition behaviors [43];
- the implementation and development of further numerical algorithms, including those that can take advantage of the various decompositions of stochastic models [14, 15, 24];
- reduction of the size of Markov chains through the exploitation of model symmetries [12, 35];
- the development of preconditioners for the available interative numerical solution methods [42];
- distributed implementations of the existing algorithms [19];
- support for fully symbolic storage and solution of Markov chains [22, 48, 68];
- the use of tensor decompositions instead of vectors to store state distributions and intermediate results to greatly reduce memory requirements of solution algorithms [3, 25, 31].

Acknowledgement We would like to thank IncQueryLabs Ltd. for their support during the summer internship.

References

[1] Kensuke Aihara, Kuniyoshi Abe, and Emiko Ishiwata. “A variant of IDRstab with reliable update strategies for solving sparse linear systems”. In: *Journal of Computational and Applied Mathematics* 259 (2014), pp. 244–258.

[2] Christel Baier, Joost-Pieter Katoen, and Holger Hermanns. “Approximative symbolic model checking of continuous-time Markov chains”. In: *CONCUR’99 Concurrency Theory*. Springer, 1999, pp. 146–161.

[3] Jonas Ballani and Lars Grasedyck. “A projection method to solve linear systems in tensor format”. In: *Numerical Linear Algebra with Applications* 20.1 (2013), pp. 27–43.

[4] Randolph E. Bank, William M. Coughran Jr., Wolfgang Fichtner, Eric Grosse, Donald J. Rose, and R. Kent Smith. “Transient Simulation of Silicon Devices and Circuits”. In: *IEEE Trans. on CAD of Integrated Circuits and Systems* 4.4 (1985), pp. 436–451. DOI: 10.1109/TCAD.1985.1270142.

[5] Richard Barrett, Michael W Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*. Vol. 43. Siam, 1994.

[6] Falko Bause, Peter Buchholz, and Peter Kemper. “A Toolbox for Functional and Quantitative Analysis of DEDS”. In: *Computer Performance Evaluation: Modelling Techniques and Tools, 10th International Conference, Tools ’98, Palma de Mallorca, Spain, September 14-18, 1998, Proceedings*. Vol. 1469. Lecture Notes in Computer Science. Springer, 1998, pp. 356–359. DOI: 10.1007/3-540-68061-6_32.

[7] Anne Benoit, Brigitte Plateau, and William J Stewart. “Memory efficient iterative methods for stochastic automata networks”. In: (2001).

[8] Anne Benoit, Brigitte Plateau, and William J. Stewart. “Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems”. In: *Future Generation Comp. Syst.* 22.7 (2006), pp. 838–847. DOI: 10.1016/j.future.2006.02.006.

[9] Andrea Bianco and Luca De Alfaro. “Model checking of probabilistic and nondeterministic systems”. In: *Foundations of Software Technology and Theoretical Computer Science*. Springer. 1995, pp. 499–513.

[10] James T. Blake, Andrew L. Reibman, and Kishor S. Trivedi. “Sensitivity Analysis of Reliability and Performability Measures for Multiprocessor Systems”. In: *SIGMETRICS*. 1988, pp. 177–186. DOI: 10.1145/55595.55616.

[11] BlueBit Software. *.NET Matrix Library 6.1*. Accessed October 26, 2015. URL: http://www.bluebit.gr/NET/.

[12] Peter Buchholz. “Exact and ordinary lumpability in finite Markov chains”. In: *Journal of applied probability* (1994), pp. 59–75.

[13] Peter Buchholz. “Hierarchical Structuring of Superposed GSPNs”. In: *IEEE Trans. Software Eng.* 25.2 (1999), pp. 166–181. DOI: 10.1109/32.761443.

[14] Peter Buchholz. “Multilevel solutions for structured Markov chains”. In: *SIAM Journal on Matrix Analysis and Applications* 22.2 (2000), pp. 342–357.

[15] Peter Buchholz. “Structured analysis approaches for large Markov chains”. In: *Applied Numerical Mathematics* 31.4 (1999), pp. 375–404.

[16] Peter Buchholz, Gianfranco Ciardo, Susanna Donatelli, and Peter Kemper. “Complexity of Memory-Efficient Kronecker Operations with Applications to the Solution of Markov Models”. In: *INFORMS Journal on Computing* 12.3 (2000), pp. 203–222. DOI: 10.1287/ijoc.12.3.203.12634.

[17] Peter Buchholz and Peter Kemper. “On generating a hierarchy for GSPN analysis”. In: *SIGMETRICS Performance Evaluation Review* 26.2 (1998), pp. 5–14. DOI: 10.1145/288197.288202.

[18] Maciej Burak. “Multi-step Uniformization with Steady-State Detection in Non-stationary M/M/s Queuing Systems”. In: *CoRR* abs/1410.0804 (2014). URL: http://arxiv.org/abs/1410.0804.

[19] Jaroslaw Bylina and Beata Bylina. “Merging Jacobi and Gauss-Seidel methods for solving Markov chains on computer clusters”. In: *Proceedings of the International Multiconference on Computer Science and Information Technology, IMCSIT 2008, Wisla, Poland, 20-22 October 2008*. IEEE, 2008, pp. 263–268. DOI: 10.1109/IMCSIT.2008.4747250.

[20] Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann, and Rohit Singh. “Measuring and Synthesizing Systems in Probabilistic Environments”. In: *J. ACM* 62.1 (2015), 9:1–9:34. DOI: 10.1145/2699430.

[21] Gianfranco Ciardo, Robert Marmorstein, and Radu Siminiceanu. "The saturation algorithm for symbolic state-space exploration". In: *Int. J. Softw. Tools Technol. Transf.* 8.1 (2006), pp. 4–25. DOI: http://dx.doi.org/10.1007/s10009-005-0188-7.

[22] Gianfranco Ciardo and Andrew S. Miner. "Implicit data structures for logic and stochastic systems analysis". In: *SIGMETRICS Performance Evaluation Review* 32.4 (2005), pp. 4–9. DOI: 10.1145/1059816.1059818.

[23] Ricardo M. Czekster, César A. F. De Rose, Paulo Henrique Lemelle Fernandes, Antonio M. de Lima, and Thais Webber. "Kronecker descriptor partitioning for parallel algorithms". In: *Proceedings of the 2010 Spring Simulation Multiconference, SpringSim 2010, Orlando, Florida, USA, April 11-15, 2010*. SCS/ACM, 2010, p. 242. ISBN: 978-1-4503-0069-8. URL: http://dl.acm.org/citation.cfm?id=1878537.1878789.

[24] Tugrul Dayar. *Analyzing Markov chains using Kronecker products: theory and applications*. Springer Science & Business Media, 2012.

[25] Sergey V Dolgov. "TT-GMRES: solution to a linear system in the structured tensor format". In: *Russian Journal of Numerical Analysis and Mathematical Modelling* 28.2 (2013), pp. 149–172.

[26] Extreme Optimization. *Numerical Libraries for .NET*. Accessed October 26, 2015. URL: http://www.extremeoptimization.com/VectorMatrixFeatures.aspx.

[27] Fault Tolerant Systems Research Group, Budapest University of Technology and Economics. *The PetriDotNet webpage*. Accessed October 23, 2015. URL: https://inf.mit.bme.hu/en/research/tools/petridotnet.

[28] Paulo Fernandes, Ricardo Presotto, Afonso Sales, and Thais Webber. "An Alternative Algorithm to Multiply a Vector by a Kronecker Represented Descriptor". In: *21st UK Performance Engineering Workshop*. 2005, pp. 57–67.

[29] Bennett L. Fox and Peter W. Glynn. "Computing Poisson Probabilities". In: *Commun. ACM* 31.4 (1988), pp. 440–445. DOI: 10.1145/42404.42409.

[30] Robert E Funderlic and Carl Dean Meyer. "Sensitivity of the stationary distribution vector for an ergodic Markov chain". In: *Linear Algebra and its Applications* 76 (1986), pp. 1–17.

[31] Lars Grasedyck, Daniel Kressner, and Christine Tobler. "A literature survey of low-rank tensor approximation techniques". In: *arXiv preprint arXiv:1302.7121* (2013).

[32] Winfried K. Grassmann. "Transient solutions in markovian queueing systems". In: *Computers & OR* 4.1 (1977), pp. 47–53. DOI: 10.1016/0305-0548(77)90007-7.

[33] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1997. ISBN: 9781611970937. URL: https://books.google.hu/books?id=IX9rrFe1YLQC.

[34] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. Accessed October 26, 2015. 2010. URL: http://eigen.tuxfamily.org.

[35] Serge Haddad and Patrice Moreaux. "Evaluation of high level Petri nets by means of aggregation and decomposition". In: *Petri Nets and Performance Models, 1995., Proceedings of the Sixth International Workshop on*. IEEE. 1995, pp. 11–20.

[36] Boudewijn R Haverkort. "Matrix-geometric solution of infinite stochastic Petri nets". In: *Computer Performance and Dependability Symposium, 1995. Proceedings., International*. IEEE. 1995, pp. 72–81.

[37] *International Workshop on Timed Petri Nets, Torino, Italy, July 1-3, 1985*. IEEE Computer Society, 1985. ISBN: 0-8186-0674-6.

[38] Ilse CF Ipsen and Carl D Meyer. "Uniform stability of Markov chains". In: *SIAM Journal on Matrix Analysis and Applications* 15.4 (1994), pp. 1061–1074.

[39] David N Jansen. "Understanding Fox and Glynn's "Computing Poisson probabilities". In: (2011).

[40] Peter Kemper. "Numerical Analysis of Superposed GSPNs". In: *IEEE Trans. Software Eng.* 22.9 (1996), pp. 615–628. DOI: 10.1109/32.541433.

[41] Moritz Kreutzer, Georg Hager, Gerhard Wellein, Holger Fehske, and Alan R. Bishop. "A unified sparse matrix data format for modern processors with wide SIMD units". In: *CoRR* abs/1307.6209 (2013). URL: http://arxiv.org/abs/1307.6209.

[42] Amy Nicole Langville and William J. Stewart. "Testing the Nearest Kronecker Product Preconditioner on Markov Chains and Stochastic Automata Networks". In: *INFORMS Journal on Computing* 16.3 (2004), pp. 300–315. DOI: 10.1287/ijoc.1030.0041.

[43] Francesco Longo and Marco Scarpa. "Two-layer Symbolic Representation for Stochastic Models with Phase-type Distributed Events". In: *Intern. J. Syst. Sci.* 46.9 (2015), pp. 1540–1571. DOI: 10.1080/00207721.2013.822940.

[44] Math.NET. *Math.NET Numerics webpage*. Accessed October 26, 2015. URL: http://numerics.mathdotnet.com/.

[45] Carl D Meyer. "Stochastic complementation, uncoupling Markov chains, and the theory of nearly reducible systems". In: *SIAM review* 31.2 (1989), pp. 240–272.

[46] Aad PA van Moorsel and William H Sanders. "Adaptive uniformization". In: *Stochastic Models* 10.3 (1994), pp. 619–647.

[47] M. Neuts. “Probability distributions of phase type”. In: *Liber Amicorum Prof. Emeritus H. Florin*. University of Louvain, 1975, pp. 173–206.

[48] *Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012*. IEEE Computer Society, 2012. ISBN: 978-1-4673-2346-8. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6354262.

[49] RJ Plemmons and A Berman. *Nonnegative matrices in the mathematical sciences*. Academic Press, New York, 1979.

[50] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[51] S. Rácz, Á. Tari, and M. Telek. “MRMSolve: Distribution estimation of Large Markov reward models”. In: *Tools 2002*. Springer, LNCS 2324, 2002, pp. 72–81.

[52] A. V. Ramesh and Kishor S. Trivedi. “On the Sensitivity of Transient Solutions of Markov Models”. In: *SIGMETRICS*. 1993, pp. 122–134. DOI: 10.1145/166955.166998.

[53] Andrew L. Reibman and Kishor S. Trivedi. “Numerical transient analysis of markov models”. In: *Computers & OR* 15.1 (1988), pp. 19–36. DOI: 10.1016/0305-0548(88)90026-3.

[54] Andrew Reibman, Roger Smith, and Kishor Trivedi. “Markov and Markov reward model transient analysis: An overview of numerical approaches”. In: *European Journal of Operational Research* 40.2 (1989), pp. 257–267.

[55] Youcef Saad and Martin H Schultz. “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems”. In: *SIAM Journal on scientific and statistical computing* 7.3 (1986), pp. 856–869.

[56] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[57] Conrad Sanderson. “Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments”. In: (2010).

[58] Valeria Simoncini and Daniel B Szyld. “Interpreting IDR as a Petrov–Galerkin method”. In: *SIAM Journal on Scientific Computing* 32.4 (2010), pp. 1898–1912.

[59] Gerard LG Sleijpen and Diederik R Fokkema. “BiCGstab (l) for linear equations involving unsymmetric matrices with complex spectrum”. In: *Electronic Transactions on Numerical Analysis* 1.11 (1993), p. 2000.

[60] Gerard LG Sleijpen and Martin B Van Gijzen. “Exploiting BiCGstab (ℓ) strategies to induce dimension reduction”. In: *SIAM journal on scientific computing* 32.5 (2010), pp. 2687–2709.

[61] Peter Sonneveld. “CGS, a fast Lanczos-type solver for nonsymmetric linear systems”. In: *SIAM journal on scientific and statistical computing* 10.1 (1989), pp. 36–52.

[62] Peter Sonneveld and Martin B van Gijzen. “IDR (s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations”. In: *SIAM Journal on Scientific Computing* 31.2 (2008), pp. 1035–1062.

[63] William J Stewart. *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press, 2009.

[64] Williams J Stewart. *Introduction to the numerical solutions of Markov chains*. Princeton Univ. Press, 1994.

[65] Masaaki Tanio and Masaaki Sugihara. “GBi-CGSTAB (s, L): IDR (s) with higher-order stabilization polynomials”. In: *Journal of computational and applied mathematics* 235.3 (2010), pp. 765–784.

[66] Enrique Teruel, Giuliana Franceschinis, and Massimiliano De Pierro. “Well-Defined Generalized Stochastic Petri Nets: A Net-Level Method to Specify Priorities”. In: *IEEE Trans. Software Eng.* 29.11 (2003), pp. 962–973. DOI: 10.1109/TSE.2003.1245298.

[67] Henk A Van der Vorst. “Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems”. In: *SIAM Journal on scientific and Statistical Computing* 13.2 (1992), pp. 631–644.

[68] Yang Zhao and Gianfranco Ciardo. “A Two-Phase Gauss-Seidel Algorithm for the Stationary Solution of EVMDD-Encoded CTMCs”. In: *Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012*. IEEE Computer Society, 2012, pp. 74–83. DOI: 10.1109/QEST.2012.34.