Configurable Numerical Analysis for Stochastic Systems

Kristóf Marussy*, Attila Klenik*, Vince Molnár†, András Vörös†, Miklós Telek‡, István Majzik*
*Budapest University of Technology and Economics,
Budapest, Hungary
†MTA-BME Lendület Cyber-Physical Systems Research Group,
Budapest, Hungary
Email: vori@mit.bme.hu
‡MTA-BME Information Systems Research Group,
Budapest, Hungary

***Abstract*—Stochastic aspects of complex systems require more and more involved analysis approaches. Answering reachability and related analysis questions can often be reduced to steady-state, transient, reward or sensitivity value analysis of stochastic models. In this paper we introduce a configurable stochastic analysis framework which supports the user to combine explicit, symbolic and numerical algorithms to efficiently compute the measures of stochastic models. Beyond the well-known algorithms from the field, we also developed an experimental version of an Induced Dimensionality Reduction Stabilized numerical solver to compute steady-state probabilities of Markovian models. As far as we know, this is the first attempt to exploit this algorithm in stochastic analysis. We have conducted experiments on different combinations of the algorithms on various models to assess their advantages and disadvantages in the analysis. This information can be used by the user to choose the combination of algorithms being efficient solving the analysis problem.**

I. Introduction

Uncertainty of complex, asynchronous or hybrid systems can be captured by stochastic models. Stochastic Petri nets provide a proper means to describe and analyse the stochastic behavior of Markovian processes. Various analysis questions regarding reachability can be reduced to the steady-state or transient behavior of the models. Additionally, the computation of more complex measures such as rewards and expected values gives us qualitative information about the reachable states. Unfortunately, stochastic analysis of complex systems not only needs efficient methods for computing the large number of possible states, but storing the representation of the stochastic behaviors and computing the measures with numerical methods yields additional challenges. Moreover, the characteristics of the stochastic models such as stiffness highly influence the efficiency of the algorithms: no single best algorithm is known that can efficiently evaluate performance measures for a diverse set of models.

In this paper we consider a four-step stochastic analysis approach: (1) state space exploration constructs and stores the discrete behavior of the model in symbolic and explicit data structures. (2) The descriptor generation step builds the necessary representation of the stochastic behavior in various matrix-based representations. (3) Numerical solver algorithms compute transient and steady-state probabilities and (4) reward calculations provide the performance measures of interest.

We propose a configurable stochastic analysis framework to compute the various measures of complex stochastic models. A large set of state space exploration, matrix representation and numerical algorithms are available in the framework. The user can choose arbitrary combination of the algorithms to solve the analysis problems. In order to assess the advantages of the various algorithms, we composed a set of benchmark models and problems to measure and evaluate the algorithms. We hope that the users of the framework benefit from these measurement as it can provide information about the characteristics of the various algorithms. Beside the traditional numerical algorithms we implemented a new Induced Dimensionality Reduction Stabilized numerical solver [1] and we extended it to handle the specialities of Markovian models. We hope that we did the first steps towards an adaptive framework which will be able to solve stochastic analysis problems efficiently by choosing the proper combination of algorithms.

Our framework, which is integrated with PetriDotNet 1.5 tool, is available at our homepage[1].

The rest of the paper is structured as follows. Section II overviews the background needed to understand configurable stochastic analysis which is introduced in Section III. Section IV summarizes the algorithms of the framework and Section V introduces the modified version of the IDR(s)STAB(ℓ) numerical solver. Measurements are presented in Section VI and the last section concludes our work.

II. Background

A. Continuous-time Markov Chains

Continuous-time Markov Chains (CTMCs) are mathematical tools for describing the behavior of systems in continuous time. The stochastic behavior of the system in the future is assumed to only depend on its current state.

A time-homogeneous continuous-time Markov chain is a stochastic process $X(t) \in S$, $t > 0$, where S is a set of states.

[1] https://inf.mit.bme.hu/en/petridotnet/stochasticanalysis

In this work we will restrict our attention to finite state cases where $n = |S| < \infty$.

The state probabilities at time τ may be collected into a row vector $\boldsymbol{\pi} \in \mathbb{R}^n$ by introducing a bijection $\iota\colon S \to \{0, 1, \ldots, n-1\}$ to number the states. The time evolution of $X(t)$ from an initial state probability distribution $\boldsymbol{\pi}_0$ is described by the initial value problem

$$\pi(t)[\iota(s)] = \Pr(X(t) = s),\ \frac{\partial \boldsymbol{\pi}}{\partial t} = \boldsymbol{\pi} Q,\ \boldsymbol{\pi}(0) = \boldsymbol{\pi}_0, \quad (1)$$

where $Q \in \mathbb{R}^{n \times n}$ is the *infinitesimal generator matrix* of $X(t)$.

The matrix Q describes the stochastic behavior of the system. Off-diagonal entries $q[x, y]$ are the rates of exponentially distributed transition firing times from the state $\iota^{-1}(x)$ to $\iota^{-1}(y)$, while diagonal entries are selected such that rows of Q sum to zero, i.e. $Q\mathbf{1}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}}$.

B. Analysis Tasks

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. Formally, $f\colon S \to \mathbb{R}$ and

$$R(t) = f(X(t)),\ Y(t) = \int_0^t R(\tau)\, d\tau.$$

The instantaneous and accumulated rewards R and Y may be of interest at some fixed time t for *transient analysis* or stationary reward rates $R(\infty) = \lim_{t\to\infty} R(t)$ may be sought.

The expected values $\mathbb{E}R(t)$ and $\mathbb{E}Y(t)$ are calculated as

$$\mathbb{E}R(t) = \sum_{s\in S} \pi(t)[\iota(s)] f(s),\ \mathbb{E}Y(t) = \sum_{s\in S} L(t)[\iota(s)] f(s), \quad (2)$$

where $\boldsymbol{\pi}(t)$ is the probability vector of $X(t)$ at time $t \le \infty$ and $\mathbf{L}(t) = \int_0^t \boldsymbol{\pi}(\tau)\, d\tau$ is the *accumulated probability vector*.

Probability vectors $\boldsymbol{\pi}(t)$ and accumulated vectors $\mathbf{L}(t)$ at time t can be obtained by solving the initial value problem (1). The stationary solution $\boldsymbol{\pi} = \boldsymbol{\pi}(\infty)$ of (1), if exists, satisfies the linear equation system

$$\boldsymbol{\pi} Q = \mathbf{0},\ \boldsymbol{\pi}\mathbf{1}^{\mathrm{T}} = 1. \quad (3)$$

Therefore, reward calculation requires both differential equation solvers for transient solutions of (1) and linear equation solvers for steady-state solutions of (3).

Another type of analysis concerns the time to reach a state partition $D \subset S$, often called time-to-first-failure (on the basis of its usage in dependability analysis)

$$\mathrm{TFF} = \inf\{t \ge 0 : X(t) \in D\}.$$

The mean, denoted by $\mathrm{MTFF} = \mathbb{E}[\mathrm{TFF}]$, can be calculated as

$$-\boldsymbol{\pi}_U Q_{UU}^{-1} \mathbf{1}^{\mathrm{T}} = -\boldsymbol{\gamma}\mathbf{1}^{\mathrm{T}},\ \boldsymbol{\gamma} Q_{UU} = \boldsymbol{\pi}_U. \quad (4)$$

The vector $\boldsymbol{\pi}_U$ and the matrix Q_{UU} are the initial distribution vector $\boldsymbol{\pi}$ and the generator matrix Q with their entries corresponding to states in D removed, which are the parameters of the *phase-type distribution* of TFF.

Often the behavior of the system depends on some parameters $\boldsymbol{\theta} \in \mathbb{R}^m$. The *sensitivity* analysis of the aforementioned measures to changes in the parameters may reveal performance or reliability bottlenecks and help designers in achieving desired performance measures and robustness values. Formally, the gradients $\nabla_{\boldsymbol{\theta}} \mathbb{E}R$, $\nabla_{\boldsymbol{\theta}} \mathbb{E}Y$ or $\nabla_{\boldsymbol{\theta}}\,\mathrm{MTFF}$ are the solutions of the equations arising after taking derivatives of both sides of (1), (3) and (4), respectively [2]. Sensitivities of the performance measures to the individual parameters $\theta[i]$ are the components of the gradient vectors.

As it can be seen above, common analysis tasks arising in the performance and reliability analysis of stochastic models can be reduced to the solution of linear equation systems and linear initial value problems. In stochastic model checking, where the desired system behaviors are expressed in stochastic temporal logics [3], [4], these analytic steps are called as subrouties to evaluate propositions. In the synthesis and optimization of stochastic models [5], analysis tasks are executed as part of the fitness functions.

C. Higher Level Formalisms

While reward processes based on continuous-time Markov chains allow the study of dependability or reliability, the explicit specification of stochastic processes and rewards is often cumbersome. More expressive formalisms include queueing networks, stochastic process algebras such as PEPA [6], [7], stochastic automata networks [8], Stochastic Activity Networks [9] and stochastic Petri nets.

Stochastic Petri Nets (SPN) extend Petri nets by assigning exponentially distributed random firing delays to transitions [10]. After the delay associated with an enabled transition is elapsed the transition fires *atomically* and transitions delays are reset.

D. General Workflow

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

1) State Space Exploration: The reachable state space S of the higher level model, such as a stochastic automata network or stochastic Petri net is explored to enumerate the possible behaviors of the model. If the model is hierarchically partitioned, this step includes the exploration of the local state spaces of the components as well as the possible global combinations of states.

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 executred to obtain steady-state solutions $\boldsymbol{\pi}$, transient solutions $\boldsymbol{\pi}(t)$, $\mathbf{L}(t)$ or MTFF measures from the matrix Q.

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 manipulation, for example the calculation of the ratio of an instantaneous and accumulated reward, may be provided to the modeller.

E. 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 states can grow exponentially.

Methods such as the *saturation* algorithm [11] 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 necessitates 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 storage 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 modeller to select the algorithm suitable for the properties of the model, as well as the decomposition method and hardware environment. The measurements of Section VI try to provide insights for the user when and how to use the available large set of algorithms efficiently.

III. Configurable Stochastic Analysis Framework

A. High Level Architecture

The configurable analysis framework has a multi-layered architecture in which the layers correspond to the tasks arising during stochastic analysis: state space exploration of the model and storing the possible states, stochastic descriptor generation, numerical solution of the equations and reward calculation. This design, shown in Fig. 1, facilitates the clean separation of concerns throughout the workflow requiring data compatibility only between the adjacent layers.

Several algorithms were implemented for each sub-task with the possibility of further customization thus providing a comprehensive set of tools for the modeller to handle a wide range of stochastic models with different characteristics.

B. Configurable Workflow

Our framework provides 2 algorithms for state space exploration, 3 state space storage schemes, 3 generator matrix representations, 7 linear equation solvers for steady state analysis and 2 linear differential equation solvers for transient analysis. The main (and computationally most intensive) task of the workflow is the calculation of the appropriate probability vector $\boldsymbol{\pi}(t)$ based on the generator matrix Q in order to evaluate the required performance measures $\mathbb{E}R(t)$ and $\mathbb{E}Y(t)$.

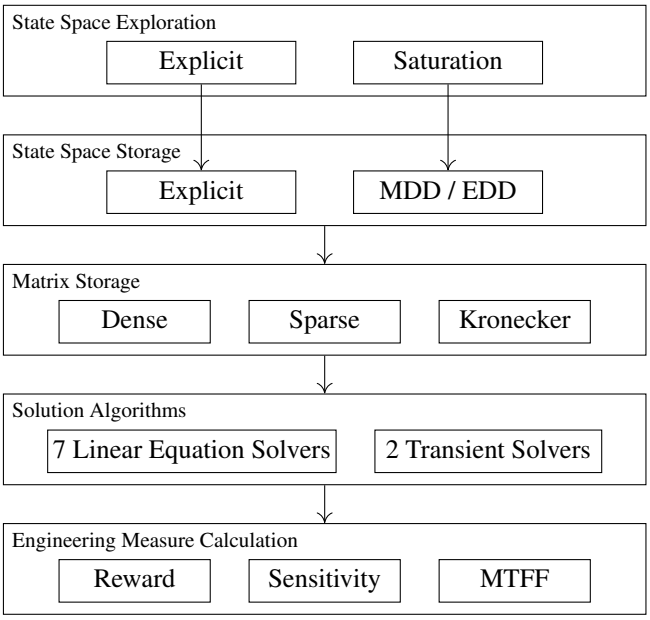

Fig. 1. Configurable stochastic analysis architecture.

Various algorithms were implemented in our framework to assemble Q from both explicit and symbolic state space representations using several storage techniques. Our compositional vector-matrix library provides means to easily assemble complex matrix structures while maintaining a unified interface towards the various numerical solvers operating on Q. This results in an almost complete (and transparent) compatibility between the analysis layers further facilitating the integration of new algorithms, such as the one presented in Section V, into the framework. This transparency makes it possible to select the best suited state space exploration technique for the model while independently selecting the best suited numerical solver for the generator matrix.

As multiple generator matrix representations are available in the framework, numerical solution algorithms must handle a variety of input formats. However, linear algerba libraries often support only specific matrix formats. Instead of integrating solvers from multiple libraries to cover all the matrix representations we implemented generic solvers from scratch. This allows the use of a single code base for numerical algorithms, as well as transparent mixing of storage schemes.

As no single best algorithm is known that can efficiently evaluate performance measures for models with various characteristics, more than 100 combinations of the implemented algorithms provide the variety and the flexibility to find the most efficient configuration for the problem at hand. Our measurements help the user to choose the configuration promising the best performance.

IV. Algorithms

A. State Space Exploration and Storage

Explicit state space exploration of high level models construct a *state space graph* by storing individual states of

the models. For Petri nets, it is implemented by repeatedly applying the transition firing rule until no transition can be fired that leads to a new state. While arbitrarily complex transition logic can be executed, the usefulness of explicit exploration is limited due to the large memory requirements of the state space graph.

Symbolic methods, such as the *saturation* algorithm can handle states spaces with up to 10^{200} states if the models are partitioned into components. Multivalued Decision Diagrams (MDD) provide compact storage for the reachable states of the model. Moreover, we implemented the mapping $\iota\colon S \to \mathbb{N}$ between the model states and state indices with Edge-valued Decision Diagrams (EDD) to facilitate vector and matrix indexing during CTMC analysis.

B. Matrix Construction and Storage

1) Dense and Sparse Matrices: Given a CTMC with a finite state space of size n, storage of the infinitesimal generator Q as a two-dimensional dense matrix requires $O(n^2)$ memory. Because the memory requirements of dense storage quickly become impractical, we implemented sparse matrices in the Compressed Column Storage (CCS) format [12, Section 4.3.1], reducing memory requirements to be proportional to the transitions in the system.

Our framework can construct generator matrices based on both explicit and symbolic representation of the state space.

2) Block Kronecker Decomposition: Memory requirements can be further reduced using the Kronecker product operation to construct larger matrices from smaller ones, which can be stored as sparse matrices [13].

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

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

Block Kronecker decomposition represents the off-diagonal part Q_O of the infinitesimal generator matrix Q as a block matrix $Q_O \in \mathbb{R}^{(n_0+n_1+\cdots+n_{k-1}) \times (n_0+n_1+\cdots+n_{k-1})}$, that is, $Q_O[x, y] \in \mathbb{R}^{n_x \times n_y}$. Hence the state space S of the CTMC is partitioned into *macro states* $S = S_0 \sqcup S_1 \sqcup \cdots \sqcup S_{k-1}$ such that $|S_x| = n_x$. The whole generator matrix can be expressed as $Q = Q_O + Q_D$, where the diagonal part is $Q_D = \operatorname{diag}\{-Q_O \mathbf{1}^{\mathrm{T}}\}$.

The individual blocks are sums of Kronecker products

$$Q_O[x, y] = \sum_{t \in T} \bigotimes_{j=0}^{J-1} Q_t^{(j)}[x, y],$$

where the index t ranges over the possible transitions in the high level model. The matrix $Q_t^{(j)}[x, y] \in \mathbb{R}^{n_x^{(j)} \times n_y^{(j)}}$ describes the effects of the transition t on the jth component of the high level model that causes the state of the CTMC to shift from the xth macro state to the yth. Notice that the macro states must be selected to ensure $n_x = n_x^{(0)} n_x^{(1)} \cdots n_x^{(J-1)}$.

Macro states and the block Kronecker generator matrix can be constructed from the explicit representation of the state space [13]. However, the explicit decomposition algorithm requires storing the *potential* state space, which is the Cartesian product of local state spaces of the components. Therefore, the size of the models that can be decomposed is limited.

We developed an approach to perform block Kronecker decomposition based on an idea of Buchholz [14] that works with the symbolic MDD representation of the state space only. Hence block Kronecker generator matrices can be used in our framework even for larger models without storing the state space or the potential state space explicitly.

The SHUFFLE algorithm [15], which we implemented in our framework, allows efficient evaluation of vector-matrix products where the matrix is stored as a Kronecker product. Recent developments include the SPLIT [16] algorithm, which allows parallel implementation while retaining the beneficial properties of the SHUFFLE algorithm. Implementation of SPLIT and the development of heuristics for its acceleration are in the scope of our ongoing work.

If symbolic state space storage, symbolic macro state composition and block Kronecker generator matrix is used, the main memory bottleneck in the stochastic analysis workflow becomes the storage of the intermediate and solution vectors instead of the state space and the generator matrix. Thus, models with up to 10^6–10^8 states can be handled, depending on the intermediate storage used by the numerical algorithms.

C. Linear Equation Solvers

1) Direct Solution: LU decomposition is a direct method for solving linear equations with forward and backward substitution, i.e. iteration is not required to reach a given precision.

To solve the equation $\mathbf{x}A = \mathbf{x}LU = \mathbf{b}$, the decomposition computes the lower triangular matrix L and upper triangular matrix U such that $A = LU$. Forward substitution is applied 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 [17, Section 2.3.1], which ensures that the diagonal of the U matrix is uniformly 1.

The case when $\mathbf{b} = \mathbf{0}$ and Q is not of full rank, which arises in the steady-state analysis of CTMCs (3), can be handled during back substitution.

2) Iterative Algorithms: Iterative methods express the solution of the linear equation $\mathbf{x}A = \mathbf{b}$ as a recurrence $\mathbf{x}_k = f(\mathbf{x}_{k-1})$, where $\mathbf{x}_0$ is an initial guess vector. The process is assumed to have converged if subsequent iterates are sufficiently close, i.e. the stopping criterion at the kth iteration becomes

$$\|\mathbf{x}_k - \mathbf{x}_{k-1}\| \leq \tau \tag{5}$$

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.

Power iteration [18, Section 10.3.1] is one of the simplest iterative methods for Markovian analysis. Its iteration function has the form

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) = \mathbf{x}_{k-1} + \alpha^{-1}(\mathbf{x}_{k-1}A - \mathbf{b}), \tag{6}$$

where $\alpha \geq \max_i |a[i,i]|$. In steady-state analysis, where A is the infinitesimal generator Q of a CTMC and $\mathbf{b} = \mathbf{0}$, the recursion (6) can be interpreted as taking time steps of length $1/\alpha$ according to (1) until steady-state is reached.

Convergence of power iteration is usually slow, but guaranteed for CTCMs if a steady-state solution exists.

Jacobi iteration [18, Section 10.3.2–3] applies the recursion

$$x_k[i] = \frac{1}{a[i,i]}\left(b[i] - \sum_{j=0}^{i-1} x_{k-1}[j] - \sum_{j=i+1}^{n-1} x_{k-1}[j]\right), \tag{7}$$

while *Gauss–Seidel iteration* uses

$$x_k[i] = \frac{1}{a[i,i]}\left(b[i] - \sum_{j=0}^{i-1} x_k[j] - \sum_{j=i+1}^{n-1} x_{k-1}[j]\right). \tag{8}$$

Gauss–Seidel iteration cannot easily be parallelized, because calculation of successive elements $x[0], x[1], \ldots$ depends 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 A, which makes the algorithm suitable for very large vectors and memory-constrained situations. In addition, convergence is often significantly faster.

3) Group Iterative Algorithms: Group Jacobi and *group Gauss–Seidel* iterations [18, Section 10.4] are block versions of (7) and (8).

A block structure is supposed for $\mathbf{x}, \mathbf{b} \in \mathbb{R}^{n_0+n_1+\cdots+n_{k-1}}$ and the matrix $A \in \mathbb{R}^{(n_0+n_1+\cdots+n_{k-1})\times(n_0+n_1+\cdots+n_{k-1})}$, thus $\mathbf{x}[i], \mathbf{x}[b] \in \mathbb{R}^{n_i}$ and $A[i,j] \in \mathbb{R}^{n_i \times n_j}$. Therefore (7) and (8) can be replaced by

$$\mathbf{x}_k[i]A[i,i] = \mathbf{b}[i] - \sum_{j=0}^{i-1} \mathbf{x}_{k-1}[j] - \sum_{j=i+1}^{n-1} \mathbf{x}_{k-1}[j], \tag{9}$$

$$\mathbf{x}_k[i]A[i,i] = \mathbf{b}[i] - \sum_{j=0}^{i-1} \mathbf{x}_k[j] - \sum_{j=i+1}^{n-1} \mathbf{x}_{k-1}[j], \tag{10}$$

respectively. The inner linear equations (9) and (10) may be solved by any algorithm, for example, LU decomposition or iterative methods. The choice of the inner algorithm may significantly affect performance and convergence behavior.

4) Krylov Subspace Methods: Projectional methods comprise another class of linear equation solvers. The approximate solution $\mathbf{x}_k$ satisfies the Petrov–Galerkin conditions $\mathbf{x}_k \in \mathcal{K}_k$, $\mathbf{r}_k = \mathbf{b} - \mathbf{x}_k A \perp \mathcal{L}_k$ [19, Section 5.1.1], 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 methods correspond to

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

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

Bi-Conjugate Gradient Stabilized (BiCGSTAB) [19, Section 7.4.2] [20] is a Krylov subspace method where [21]

$$\mathcal{L}_k = \mathcal{K}_k(A^{\mathrm{T}}, \tilde{\mathbf{r}}_0) \cdot (\Omega_k(A)^{\mathrm{T}})^{-1}, \tag{11}$$

$$\Omega_k(A) = \Omega_{k-1}(A) \cdot (I - \omega_k A). \tag{12}$$

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.

The property (11) allows efficient implementation of the recursive update of the solution vector. BiCGSTAB is a "short recurrence", i.e. the number of allocated intermediate vectors does not depend on the number of iterations taken.

We selected BiCGSTAB as the first Krylov subspace solver integrated into our framework because of its good convergence behavior 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.

D. Transient solvers

1) Uniformization: The *uniformization* or *randomization* method solves the initial value problem (1) by computing

$$\boldsymbol{\pi}(t) = \sum_{k=0}^{\infty} \boldsymbol{\pi}_0 P^k e^{-\alpha t} \frac{(\alpha t)^k}{k!}, \tag{13}$$

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

The accumulated probability vector $\mathbf{L}(t)$ can be computed by integrating both sides of (13) [22] yielding

$$\mathbf{L}(t) = \frac{1}{\alpha}\sum_{k=0}^{\infty} \boldsymbol{\pi}_0 P^k \left(1 - \sum_{l=0}^{k} e^{-\alpha t}\frac{(\alpha t)^l}{l!}\right). \tag{14}$$

Both (13) and (14) can be realized as finite sums by left and right cutoff of the summation indices. The Poisson weights and the cutoff points are determined with Burak's algorithm [23].

Steady-state detection is performed to stop iteration if the iterate $\boldsymbol{\pi}_0 P^k$ reaches the steady-state vector $\boldsymbol{\pi}$. Hence the number of matrix multiplications can be reduced by early exit.

2) Implicit Integration: A weakness of the uniformization algorithm is the poor tolerance of *stiff* Markov chains. The CTMC is called stiff if $|\lambda_{\min}| \ll |\lambda_{\max}|$, where $\lambda_{\min}$ and $\lambda_{\max}$ are the nonzero eigenvalues of the infinitesimal generator matrix Q of minimum and maximum absolute value [24]. In other words, stiff Markov chains have behaviors on drastically different timescales. For example, there could be clients that are served frequently while failures happen infrequently.

Stiffness leads to very large values of the uniformization rate α, 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 is detected.

In our configurable analysis framework, we implemented TR-BDF2 [25], which is an L-stable differential equation solver suitable for stiff Markov chains [24]. TR-BDF2 combines the trapezoid rule and the second-order backward difference rule into an implicit integration scheme, i.e. it calls a linear equation solver specified by the user as a subroutine.

V. IDR(s)STAB(ℓ) Numerical Solver

Induced Dimensionality Reduction Stabilized [1] is a Krylov subspace solver that generalizes BiCGSTAB and IDR techniques to provide faster convergence while maintaining the short recurrence property.

As the algorithm was developed recently, 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 framework. Special attention is paid to its behavior on steady-state equations with infinitesimal generator matrices and other linear systems arising from CTMC analysis.

A. Description

IDR(s)STAB(ℓ) iteration combines two generalizations of BiCGSTAB. The first idea comes from IDR(s) [26], a Krylov subspace solver based on Sonneveld subspaces. The residual $\mathbf{r}_k$ is constrained to the subspace

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

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

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

and $\tilde{\mathbf{r}}_0[i]$ is the ith row of $\widetilde{R}_0$. Higher values of s, i.e. higher dimensional initial "shadow" spaces, may accelerate convergence, at the cost of allocating additional intermediate vectors.

The second generalization, which is called BiCGSTAB(ℓ) [27], replaces the stabilizer polynomial Ω_k from (12) with

$$\Omega_k(A) = \Omega_{k-1}(A) \cdot \left(I - \textstyle\sum_{j=1}^{\ell} \gamma[j-1]A^j\right), \qquad (15)$$

i.e. degree of the stabilizer polynomial Ω is increased by ℓ instead of 1 in every iteration according to $\vec{\gamma} \in \mathbb{R}^{\ell}$.

The higher-order stabilization improves convergence behavior with unsymmetric 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 in 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 configurable stochastic analysis, as different settings of (s, ℓ) bring the power of multiple algorithms to the modellers' disposal.

B. Implementation

The pseudocode of IDR(s)STAB(ℓ) is shown in Fig. 2. Our modifications, described in detail in the next subsection, are shaded with darker background.

We follow the notation of [1]: $(X, Y, \ldots, W)$ and $(X; Y; \ldots; W)$ denote the matrices obtained by writing

1: Select the initial guess $\mathbf{x}$ and the "shadow" residuals $\widetilde{R}_0$
2: $\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{x}A$, $R \leftarrow (\mathbf{r}_0)$
   ▷ Create initial $U = (U_0, U_1) = (U_0, U_0A)$
3: **for** $q = 0, 1, \ldots, s-1$ **do**
4: **if** $q = 0$ **then** $\mathbf{c} \leftarrow -\mathbf{x}$ **else** $\mathbf{c} \leftarrow \mathbf{u}_0$
5: **if** $q = 0$ **then** $\mathbf{u}_0 \leftarrow \mathbf{r}_0$ **else** $\mathbf{u}_0 \leftarrow \mathbf{u}_1$
6: $\vec{\mu} \leftarrow \mathbf{u}_0 U_0[0{:}q-1, :]^{\mathrm{T}}$, $\mathbf{u} \leftarrow (\mathbf{u}_0, \mathbf{u}_0A) - \vec{\mu} U[0{:}q-1, :]$
7: $C[q, :] \leftarrow \left(\mathbf{c} - \vec{\mu} C[0{:}q-1, :]\right)/\|\mathbf{u}_0\|_2$
8: $\mathbf{u} \leftarrow \mathbf{u}/\|\mathbf{u}_0\|_2$, $U[q, :] \leftarrow \mathbf{u}$
9: **end for**
10: **while** $\|\mathbf{r}_0\| > tol$ **do**
11: **for** $j = 1, 2, \ldots, \ell$ **do** ▷ An IDR step
12: $\sigma \leftarrow U_j^{\mathrm{T}} \widetilde{R}_0$, $\vec{\alpha} \leftarrow (\widetilde{R}_0 \mathbf{r}_{j-1}^{\mathrm{T}})\sigma^{-1}$, $\mathbf{x} \leftarrow \vec{\alpha} U_0$
13: $\mathbf{r} \leftarrow \mathbf{r} - \vec{\alpha}(U_1, \ldots, U_j)$, $\mathbf{r} \leftarrow (\mathbf{r}, \mathbf{r}_{j-1}A)$
14: **for** $q = 0, 1, \ldots, s-1$ **do**
15: **if** $q = 0$ **then** $\mathbf{c} \leftarrow -\mathbf{x}$ **else** $\mathbf{c} \leftarrow \mathbf{v}_{q-1}$
16: **if** $q = 0$ **then** $\mathbf{u} \leftarrow \mathbf{r}$ **else** $\mathbf{u} \leftarrow (\mathbf{u}_1, \ldots, \mathbf{u}_{j+1})$
17: $\vec{\beta} \leftarrow (\widetilde{R}_0 \mathbf{u}_j^{\mathrm{T}})\sigma^{-1}$, $\mathbf{u} \leftarrow \mathbf{u} - \vec{\beta} U$, $\mathbf{u} \leftarrow (\mathbf{u}, \mathbf{u}_j A)$
18: $\mathbf{c} \leftarrow \mathbf{c} - \vec{\beta} C$
19: $\vec{\mu} \leftarrow \mathbf{u}_j V_j[0{:}q-1, :]^{\mathrm{T}}$, $\mathbf{u} \leftarrow \mathbf{u} - \vec{\mu} V[0{:}q-1, :]$
20: $\mathbf{c} \leftarrow \mathbf{c} - \vec{\mu} D[0{:}q-1, 0]$, $t \leftarrow \|\mathbf{u_j}\|_2$
21: **if** $t < \epsilon$ and $\mathbf{b} = \mathbf{0}$ **then return** $\mathbf{c}$
22: $D[q, :] \leftarrow \mathbf{c}/t$, $\mathbf{u} \leftarrow \mathbf{u}/t$, $V[q, :] \leftarrow \mathbf{u}$
23: **end for**
24: $C \leftarrow D$, $U \leftarrow V$
25: **end for**
   ▷ The polynomial step
26: $\vec{\gamma} \leftarrow \arg\min_{\vec{\gamma}} \left\|\mathbf{r}_0 - \vec{\gamma}\,[\mathbf{r}_1, \ldots, \mathbf{r}_\ell]\right\|_2$
27: $\mathbf{x} \leftarrow \vec{\gamma}\,[\mathbf{r}_0, \ldots, \mathbf{r}_{\ell-1}]$, $\mathbf{x} \leftarrow \vec{\gamma}\,[\mathbf{r}_1, \ldots, \mathbf{r}_\ell]$
28: $C \leftarrow C - \sum_{j=1}^{\ell} \gamma[j-1] U_{j-1}$
29: $U \leftarrow (U_0 - \sum_{j=1}^{\ell} \gamma[j-1] U_j, U_1 - \sum_{j=1}^{\ell} \gamma[j-1] U_{j+1})$
30: **end while**
31: **return** $\mathbf{x}$

Fig. 2. The modified IDR(s)STAB(ℓ) algorithm. The objects U_j, V_j, $\mathbf{r}_{j-1}$ and $\mathbf{u_j}$ are related to $U = (U_0, U_1, \ldots, U_j)$, $V = (V_0, V_1, \ldots, V_{j+1})$, $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{j-1})$ and $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_j)$, respectively. Note that the sizes of U, $\mathbf{r}$ and $\mathbf{u}$ change during the IDR step. Our modifications for stochastic analysis are shown shaded.

$X, Y, \ldots, W$ side by side and below each other, respectively. We write $W[a{:}b, :]$ to refer to the a–bth rows of W and $W[a, :]$ to refer to the ath row of A. The algorithm works with the matrices $\widetilde{R}_0, C, D, U_0, U_1, \ldots, U_{\ell+1}, V_0, V_1, \ldots, V_{\ell+1} \in \mathbb{R}^{s\times n}$ and the vectors $\mathbf{x}, \mathbf{c}, \mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{\ell-1}, \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_\ell \in \mathbb{R}^n$. In addition, smaller vectors $\vec{\alpha}, \vec{\beta}, \vec{\gamma}, \vec{\mu}$ store linear combination coefficients and the matrix $\sigma \in \mathbb{R}^{s\times s}$ is used in projections. Following the recommendation of [28], we initialize the rows of $\widetilde{R}_0$ to an orthonormalized set of random vectors.

The algorithm repeats IDR steps and polynomial steps until convergence. The IDR step fills the matrix V with intermediate vectors. The polynomial step selects the coefficient vector $\vec{\gamma}$ of the stabilizer polynomial Ω_k in (15).

The IDR step performs sequences of projections called repetition steps in a loop with $j = 1, 2, \ldots, \ell$. An example for $j = 2$ and $s = 3$ is shown in Fig. 3. The applied projections are of the form

$$\Pi_i = I - A^{j-i} \widetilde{R}_0^{\mathrm{T}} \sigma^{-1} U_i,\ \sigma = \widetilde{R}_0 U_j^{\mathrm{T}} \qquad (16)$$

for $i = 0, 1, \ldots, j$.

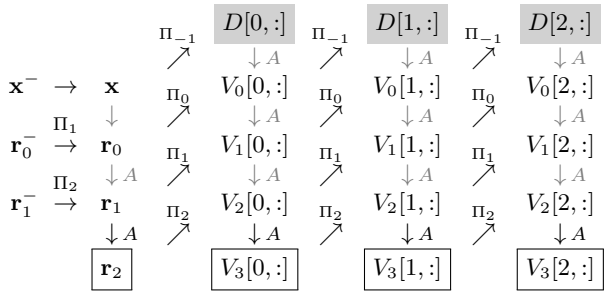

Fig. 3. Repetition step for IDR(s)STAB(ℓ) extended for stochastic analysis with $j = 2$ and $s = 3$. Quantities with a superscript $^-$ refer to the original values of the vectors before the repetition step.

First the residuals are updated by $\mathbf{r}_i = \mathbf{r}_i^- \Pi_i$, where $\mathbf{r}_i^-$ refers to the initial value of the residual at the beginning of the repetition step. Then the projections $V_{i-1}[0,:] = \mathbf{r}_i \Pi_i$ and $V_{i-1}[q,:] = V_i[q-1,:]\Pi_i$ fill V with new vectors such that the rows of V_i form the basis of Krylov subspace $\mathcal{K}_s(A\Pi_i, \mathbf{r}_i\Pi_i)$. In the jth repetition step, $\mathbf{r}_j$ is set to $\mathbf{r}_{j-1}A$ and V_{j+1} is set to V_jA such that the relationships $\mathbf{r}_{i+1} = \mathbf{r}_iA$ and $V_{i+1} = V_1A$ hold throughout the iteration.

For a more detailed description of IDR(s)STAB(ℓ), we refer to [1] and our technical report [29].

C. Observations and Proposed Modifications

Unfortunately, our initial experiments with IDR(s)STAB(ℓ) in Markovian steady-state analysis did not lead to success when one of the parameters s or ℓ were set to values strictly larger than 1. The only case that managed to decrease the norm of the residual reliably, $s = \ell = 1$, is equivalent to BiCGSTAB, but less numerically stable.

We have identified the following numerical problems in the original version of the algorithm that lead to breakdown:

1) Collapse of the V_j Spaces: Recall that the rows of V_j are basis vectors of the Krylov subspace $\mathcal{K}_s(A\Pi_j, \mathbf{r}_j\Pi_j)$. A numerical breakdown may occur if this space is not of dimension s, i.e. $\mathbf{u}_j$ is a zero vector. This happens when the matrix A is not of full rank and results in a division by zero in line 22 of Fig. 2. The breakdown is especially problematic in CTMC steady state analysis, because the generator matrix $Q \in \mathbb{R}^{n\times n}$ is of rank at most $n-1$.

It can be seen [29] that if $\mathbf{u}_j = \mathbf{0}$, then $\mathbf{u}_i = \mathbf{0}$ for all $0 < i < j$. Thus, we added another vector $\mathbf{c}$ and matrices C and D to the algorithm, which serve as a -1th row of U and V, respectively. This results in the extended projections shown in Fig. 3, where $\Pi_{-1} = I - A^{j-i}\widetilde{R}_0^{\mathrm{T}}\sigma^{-1}D$.

If a homogeneous linear equation $\boldsymbol{\pi}Q = \mathbf{0}$ is solved with a rank deficient Q, the vector $\mathbf{c}$ is a nonzero solution when the orthonormalization of V_j breaks down. Hence it can be normalized to obtain a probability distribution $\boldsymbol{\pi} = \mathbf{c}/\mathbf{c}\mathbf{1}^{\mathrm{T}}$.

2) Singular Projection Matrix: The matrix σ may become singular such that the projections (16) cannot be computed. If a singular σ is detected, the algorithm must stop. In addition, if the determinant of σ is extremely small, but nonzero, numerical errors may accumulate rapidly that cause divergence of the norm $\|\mathbf{x}\|$ of the solution vector.

Despite our attempts, we did not manage to modify IDR(s)STAB(ℓ) to handle singular or nearly singular σ. More careful choice of the stabilizing polynomial Ω_k may be a possible remedy. Choosing $\vec{\gamma}$ though means other than the minimization of the residual norm $\|\mathbf{r}_0\|_2$ may result in better converge behavior [30], [31].

VI. Measurements

A. Benchmark Setup

We developed our configurable stochastic analysis framework as a module of the PetriDotNet modeling tool. Thus, stochastic Petri nets were selected as the input high-level formalism for the analysis.

1) Models: We used the following three models in the benchmarks with manual model partitioning.

- *SharedResource* is a scalable synthetic model representing processors executing jobs while using global resources in a mutually exclusive way.
- The *kanban* SPN from [11] describes a manufacturing process with a scalable number of resources.
- *VclCloud* is a cloud performability model [32] representing a cloud architecture with physical and virtual machines serving incoming jobs using warm and cold spare resources in case of increasing load. We modified some aspects of the model since our library currently doesn't support the GSPN formalism.

We also used Petri nets available from the website of the Model Checking Contest (MCC) [33]. From scalable families of models nets of various sizes were obtained, while colored Petri nets were unfolded. Only deadlock-free models were used to ensure that the steady-state distribution is well-defined.

For each Petri net from the MCC and the *SharedResource* family, the SPNs were generated to study the effect of transition rates on the analysis. In the *Sym* version all transition rates are equal. In the *Asym* version transition rates were randomly selected from the interval $[0.1, 2.5]$, while in the *Vasym* version the rate interval is $[0.001, 250]$. Transition rates in the *kanban* and *VclCloud* family were predefined because they were already SPNs.

The models and the PetriDotNet 1.5 tool are available at our homepage[2].

2) Algorithms: Symbolic state space exploration and storage were used during the benchmarks due to the extreme memory requirements of explicit processing. For Kronecker matrices, multiplication was performed with Shuffle.

Both steady-state and transient analysis workflows were executed to obtain probability distribution vectors. *BiCGSTAB* and *Gauss–Seidel iteration* with both sparse and block Kronecker matrices were used for steady-state analysis, as well as *Group Gauss–Seidel iteration* on block matrices with *BiCGSTAB* and *Jacobi iteration* as the inner solver. In addition, transient analysis was performed by uniformization.

[2]https://inf.mit.bme.hu/en/petridotnet/stochasticanalysis

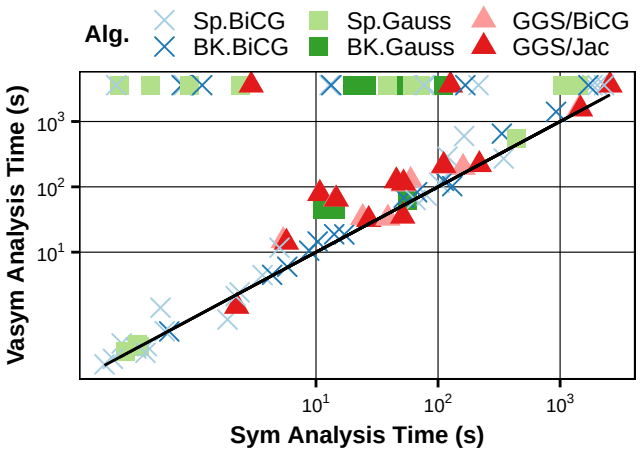

Fig. 4. Effects of changing transition rates in steady-state analysis.

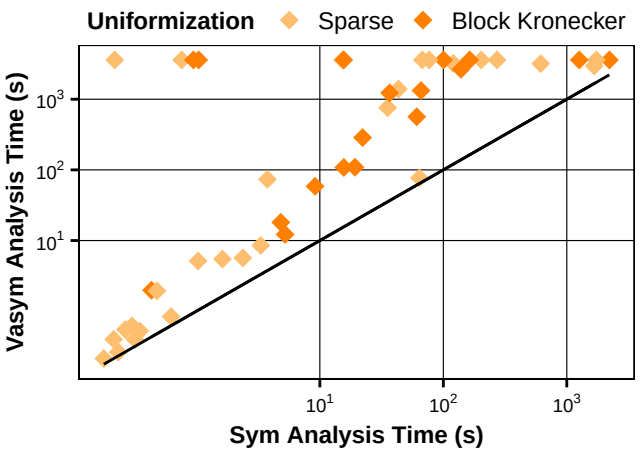

Fig. 5. Effects of changing transition rates in transient analysis.

The execution time was constrained to 1 hour in all cases.

B. Effects of Algorithm Selection

The collected benchmark results regarding the steady-state linear equation solvers are presented in Table I. The data in the cell is the number of fastest or least memory cases/number of solved cases form.

BiCGSTAB appears to be a remarkably robust solver for examined problems. In the *Sym* and *Asym* variations of models, BiCGSTAB was the fastest algorithm in all cases where it converged successfully, except for a few instances of the *RwMutex* family, where Gauss–Seidel iteration was the fastest. Moreover, in the *Vasym* variation BiCGSTAB was the fastest in all cases. However, other algorithms required less memory for many models. The high number of BiCGSTAB wins in terms of memory usage are due to the constrained execution time which prevented other algorithms from finishing. Therefore, further measurements are to be conducted with extended time limits to study more memory efficient algorithms with slower convergence properties.

Fig. 4 shows the numerical sensitivity of the linear equation solvers to the modification of transition rates. Failed executions are shown at the maximum time limit. In most cases, either the run time of the algorithms is unaffected by the *Sym* and *Vasym* variations, or solution fails to converge in time on *Vasym*.

The same analysis is repeated for uniformization in Fig. 5. It is apparent that uniformization is strongly affected by the stiffness of model, as expected from (13).

C. Effects of Model Partitioning

Symbolic state space exploration of Petri nets requires partitioning the model into groups of places i.e. components. The behavior of symbolic processing heavily depends on the selected partitions [11].

Block Kronecker decomposition also constructs the local transition matrices $Q_t^{(j)}$ from a partitioned model. Unfortunately, the optimal partitioning for symbolic algorithms and Kronecker decomposition is usually different. While saturation

TABLE I
SUMMARY OF STEADY-STATE ALGORITHM BENCHMARKS.

		BiCGSTAB		Gauss–Seidel		Group G–S	
Best		Sparse	BK	Sparse	BK	BiCG.	Jacobi
Time	Sym	21/29	3/18	5/14	0/7	0/10	0/14
	Asym	26/30	1/21	3/13	0/9	0/16	0/18
	Vasym	22/23	1/16	0/7	0/4	0/13	0/14
Mem.	Sym	18/29	1/18	6/14	1/7	0/10	3/14
	Asym	19/30	0/21	3/13	2/9	0/16	6/18
	Vasym	16/23	2/16	3/7	1/4	0/13	1/14

based symbolic state space exploration can often benefit from small groups of places, for example every-1 partitioning, i.e. every place is a separate partition, this decomposition can lead to a very large number of small matrices that fails to provide the desired reduction of matrix storage requirements.

We manually partitioned the models *SharedResource*, *SharedResource* and *kanban*. For the models of MCC, we used a compromise of automatic every-4 partitioning.

Automatic partitioning lead to long saturation times even for models of relatively modest size, as it can be seen in Fig. 6. Exceptions were the models *FMS*, *RwMutex* and *SimpleLoadBal*, where every-4 partitions coincide with the structure of the model.

Figs. 7 and 8 show the running time and memory tradeoffs associated with sparse and block Kronecker generator matrices for BiCGSTAB iteration. The behavior with other algorithms, including transient solution with uniformization, are similar. While manual partitioning was able to achieve memory savings without significantly compromising execution times, automatic partitioning resulted in larger block Kronecker generators than the original sparse matrix representations.

D. Convergence Behavior of IDR(s)STAB(ℓ)

The convergence behavior of our modified IDR(s)STAB(ℓ) implementation on various scaled versions of the *SharedResource* model is shown in Fig. 9. In total 10 000 test runs were aggregated to study different settings of s and ℓ.

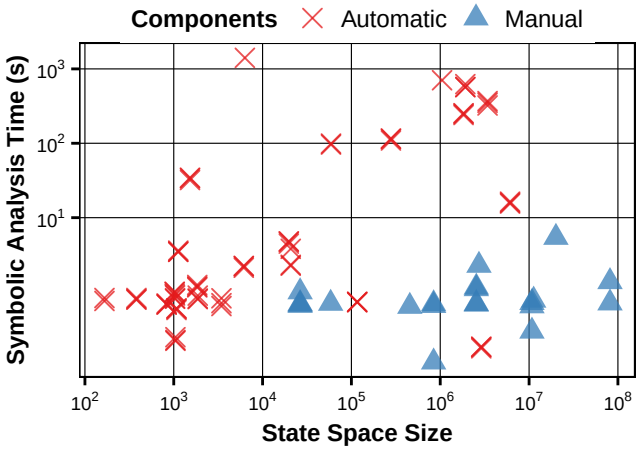

Fig. 6. Execution times of the symbolic part of the analysis.

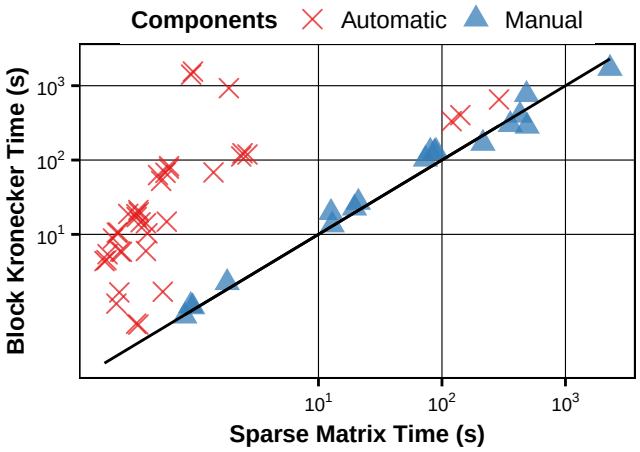

Fig. 7. Execution times of the BiCGSTAB linear equation solver.

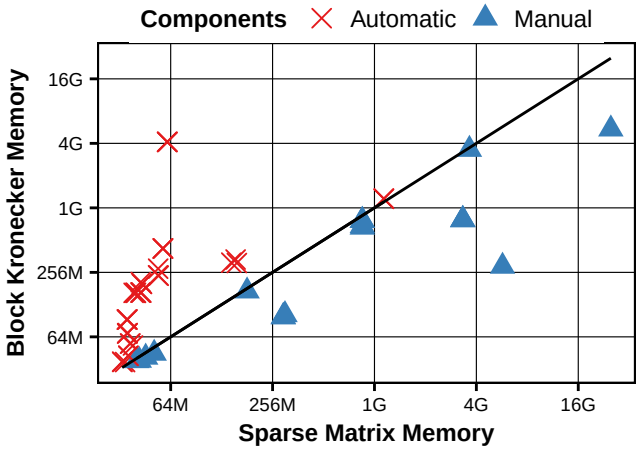

Fig. 8. Memory requirements of the BiCGSTAB linear equation solver.

Four possible outcomes were identified: "Convergence", where the algorithm finished within 200 iterations; "Breakdown", where a numerical breakdown occurred; "Divergence", where the residual norm increased until overflow and "No result" with no convergence until iteration 200.

The setting $s = 1$, $\ell = 1$, which emulates BiCGSTAB, mostly converged for small models. Before the breakdowns the residual norm was approximately $6 \cdot 10^{-7}$. This is probably the limit to the accuracy of the IDR(s)STAB(ℓ) formulation of BiCGSTAB, as the original BiCGSTAB algorithm utilizes a more stable residual update strategy.

The setting $s = \ell = 2$ results in timeout due to oscillation, whereas with higher values, such as $s = \ell = 8$, divergence usually happens rapidly. This shows that IDR(s)STAB(ℓ) is hardly able to handle problems arising in Markovian steady-state analysis even with our modifications.

VII. Conclusions and Future Work

Complex and hybrid systems have many aspects from which we studied the stochastic analysis problem in this paper. We have developed various symbolic and explicit algorithms for state space exploration. Steady-state and transient numerical solvers work on the different descriptor representations. Calculation of various measures answers the analysis questions. In order to evaluate the different configurations of the algorithms, we have conducted experiments to assess their strengths and weaknesses. In addition to the well-known solvers used in Markovian analysis, we have developed an experimental algorithm of an Induced Dimensionality Reduction Stabilized numerical solver. Some of our algorithmic improvements turned out to help the solver, however much work is left to be done to be able to fully utilize this new algorithm in Markovian stochastic analysis. In the future we plan to further extend our configurable stochastic analysis framework with new algorithms. We plan to provide heuristics for selecting the proper configuration of algorithms for the analysis problems. The long term goal is to develop an adaptive framework supporting stochastic analysis of models with various characteristics.

Acknowledgement

This work was partially supported by the ARTEMIS JU and the Hungarian National Research, Development and Innovation Fund in the frame of the R5-COP project. This research was partially performed within the framework of the grant of the Hungarian Scientific Resarch Fund (grant no. OTKA K101150).

References

[1] G. L. Sleijpen and M. B. Van Gijzen, "Exploiting BiCGstab(ℓ) strategies to induce dimension reduction," *SIAM journal on scientific computing*, vol. 32, no. 5, pp. 2687–2709, 2010.

[2] A. V. Ramesh and K. S. Trivedi, "On the sensitivity of transient solutions of Markov models," in *SIGMETRICS*, 1993, pp. 122–134.

[3] A. Bianco and L. De Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Foundations of Software Technology and Theoretical Computer Science*. Springer, 1995, pp. 499–513.

[4] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximative symbolic model checking of continuous-time Markov chains," in *CONCUR'99 Concurrency Theory*. Springer, 1999, pp. 146–161.

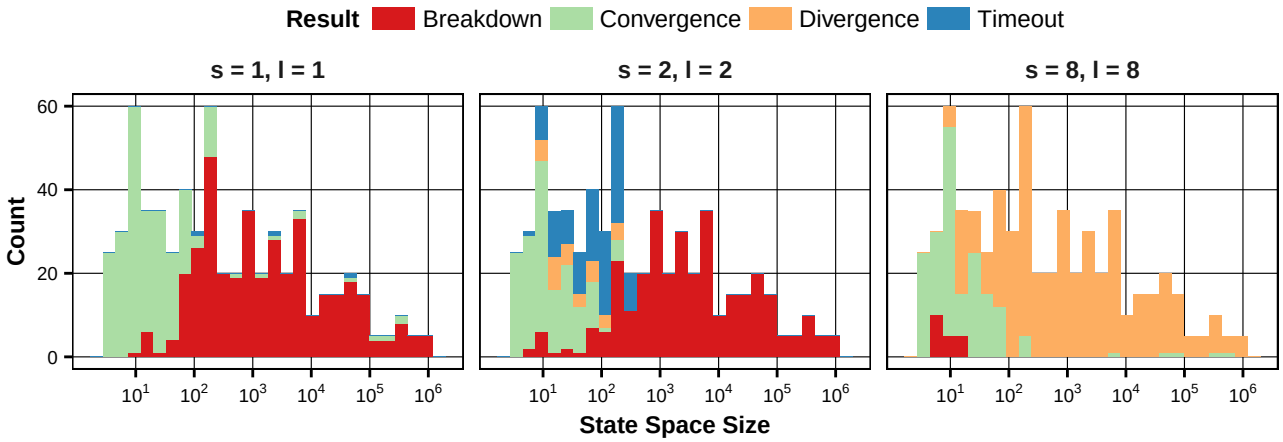

Fig. 9. Histograms of the outcomes of IDR(s)STAB(ℓ) iteration for various state space sizes and some values of the parameters (s, ℓ).

[5] K. Chatterjee, T. A. Henzinger, B. Jobstmann, and R. Singh, "Measuring and synthesizing systems in probabilistic environments," *J. ACM*, vol. 62, no. 1, pp. 9:1–9:34, 2015.

[6] S. Gilmore and J. Hillston, "The PEPA workbench: A tool to support a process algebra-based approach to performance modelling," in *Computer Performance Evaluation, Modeling Techniques and Tools, 7th International Conference, Vienna, Austria, May 3-6, 1994, Proceedings*, ser. Lecture Notes in Computer Science, G. Haring and G. Kotsis, Eds., vol. 794. Springer, 1994, pp. 353–368.

[7] S. Donatelli, "Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space," *Performance Evaluation*, vol. 18, no. 1, pp. 21–36, 1993.

[8] P. Fernandes, B. Plateau, and W. J. Stewart, "Numerical evaluation of stochastic automata networks," in *MASCOTS '95*, P. W. Dowd and E. Gelenbe, Eds. IEEE Computer Society, 1995, pp. 179–183.

[9] W. H. Sanders and J. F. Meyer, "Stochastic activity networks: Formal definitions and concepts," in *Lectures on Formal Methods and Performance Analysis*. Springer, 2001, pp. 315–343.

[10] M. A. Marsan, "Stochastic Petri nets: an elementary introduction," in *Advances in Petri Nets 1989, covers the 9th European Workshop on Applications and Theory in Petri Nets, held in Venice, Italy in June 1988, selected papers*, ser. Lecture Notes in Computer Science, G. Rozenberg, Ed., vol. 424. Springer, 1988, pp. 1–29.

[11] G. Ciardo, Y. Zhao, and X. Jin, "Ten years of saturation: A Petri net perspective," *T. Petri Nets and Other Models of Concurrency*, vol. 5, pp. 51–95, 2012.

[12] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*. Siam, 1994, vol. 43.

[13] P. Buchholz, "Hierarchical structuring of superposed GSPNs," *IEEE Trans. Software Eng.*, vol. 25, no. 2, pp. 166–181, 1999.

[14] P. Buchholz and P. Kemper, "Kronecker based matrix representations for large Markov models," in *Validation of Stochastic Systems*. Springer, 2004, pp. 256–295.

[15] A. Benoit, B. Plateau, and W. J. Stewart, "Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems," *Future Generation Comp. Syst.*, vol. 22, no. 7, pp. 838–847, 2006.

[16] R. M. Czekster, C. A. F. D. Rose, P. H. L. Fernandes, A. M. de Lima, and T. Webber, "Kronecker descriptor partitioning for parallel algorithms," in *Proceedings of the 2010 Spring Simulation Multiconference, SpringSim 2010, Orlando, Florida, USA, April 11-15, 2010*, R. M. McGraw, E. S. Imsand, and M. J. Chinni, Eds. SCS/ACM, 2010, p. 242.

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

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

[19] Y. Saad, *Iterative methods for sparse linear systems*. Siam, 2003.

[20] H. A. Van der Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems," *SIAM Journal on scientific and Statistical Computing*, vol. 13, no. 2, pp. 631–644, 1992.

[21] V. Simoncini and D. B. Szyld, "Interpreting IDR as a Petrov–Galerkin method," *SIAM Journal on Scientific Computing*, vol. 32, no. 4, pp. 1898–1912, 2010.

[22] A. Reibman, R. Smith, and K. Trivedi, "Markov and Markov reward model transient analysis: An overview of numerical approaches," *European Journal of Operational Research*, vol. 40, no. 2, pp. 257–267, 1989.

[23] M. Burak, "Multi-step uniformization with steady-state detection in nonstationary $M/M/s$ queuing systems," *CoRR*, vol. abs/1410.0804, 2014.

[24] A. L. Reibman and K. S. Trivedi, "Numerical transient analysis of markov models," *Computers & OR*, vol. 15, no. 1, pp. 19–36, 1988.

[25] R. E. Bank, W. M. C. Jr., W. Fichtner, E. Grosse, D. J. Rose, and R. K. Smith, "Transient simulation of silicon devices and circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 4, no. 4, pp. 436–451, 1985.

[26] P. Sonneveld and M. B. van Gijzen, "IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations," *SIAM Journal on Scientific Computing*, vol. 31, no. 2, pp. 1035–1062, 2008.

[27] G. L. Sleijpen and D. R. Fokkema, "BiCGstab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum," *Electronic Transactions on Numerical Analysis*, vol. 1, no. 11, p. 2000, 1993.

[28] P. Sonneveld, "On the convergence behaviour of IDR(s)," Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft Institute of Applied Mathematics, Tech. Rep., 2010.

[29] K. Marussy, "Configurable numerical solutions for stochastic models," Bachelor's Thesis, Budapest University of Technology and Economics, 2015. [Online]. Available: http://petridotnet.inf.mit.bme.hu/publications/BScThesis2015_Marussy.pdf

[30] O. Rendel and M. Jens-Peter, "Tuning IDR to fit your applications," in *Proceedings of a Workshop at Doshisha University*, 2011.

[31] G. L. Sleijpen and H. A. Van der Vorst, "Maintaining convergence properties of BiCGstab methods in finite precision arithmetic," *Numerical Algorithms*, vol. 10, no. 2, pp. 203–223, 1995.

[32] R. Ghosh, "Scalable stochastic models for cloud services," Ph.D. dissertation, Duke University, 2012.

[33] Model Checking Contest. Models for the MCC 2016. Accessed: 2016-02-02. [Online]. Available: http://mcc.lip6.fr/models.php