Contents

Chapter 3

Overview of the approach

3.1 General 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 [18] 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.

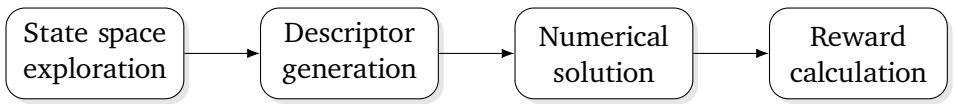

Figure 3.1 The general stochastic analysis workflow.

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 [1, 3], these analytic steps are called as subrouties to evaluate propositions. In the sythesis and optimization of stochastic models [9], 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 [10] 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 preformed 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 impelemtations are also possible, albeit they are missing from the current version of our framework.

3.2 Our workflow

Our implementation of the general stochastic analysis workflow is illustrated in Figure 3.2.

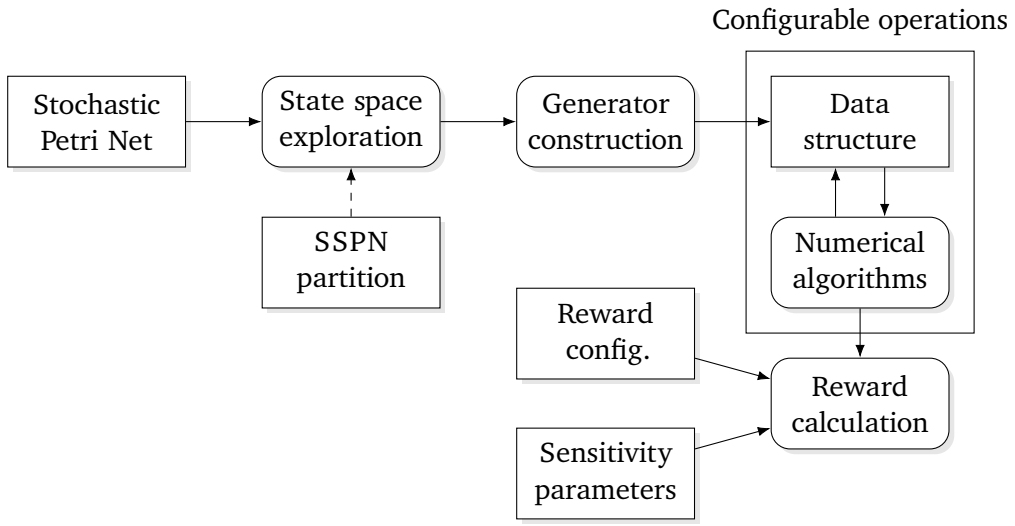

Figure 3.2 Configurable stochastic analysis workflow.

Table 3.1 Linear equation solvers supported by our framework.

	see	memory usage	parallel impl.	uses inner solver	block matrix
LU decomposition	p. 52	very high	–	–	–
Power method	p. 55	moderate	✓	–	✓
Jacobi over-relaxation	p. 56	moderate	✓	–	✓
Gauss–Seidel over-relaxation	p. 56	very low	–	–	✓
BiCGSTAB	p. 60	high	✓	–	✓
Group Jacobi	p. 58	moderate	✓	✓	required
Group Gauss–Seidel	p. 58	low	–	✓	required

The workflow is fully *configurable*, which means that the modeler may combine the available algorithms for the analysis steps arbitrarily. This is achieved by a layered architecture as shown in Figure 3.3.

- The model state space may be explored either by an explicit state space traversal, or by symbolic saturation [10]. As symbolic methods are usually much faster and use significantly less memory than explicit enumeration, they are the recommended approach for stochastic analysis. However the explicit algorithms are not sensitive to the structure of the model, they provide a robust solution as long as the state

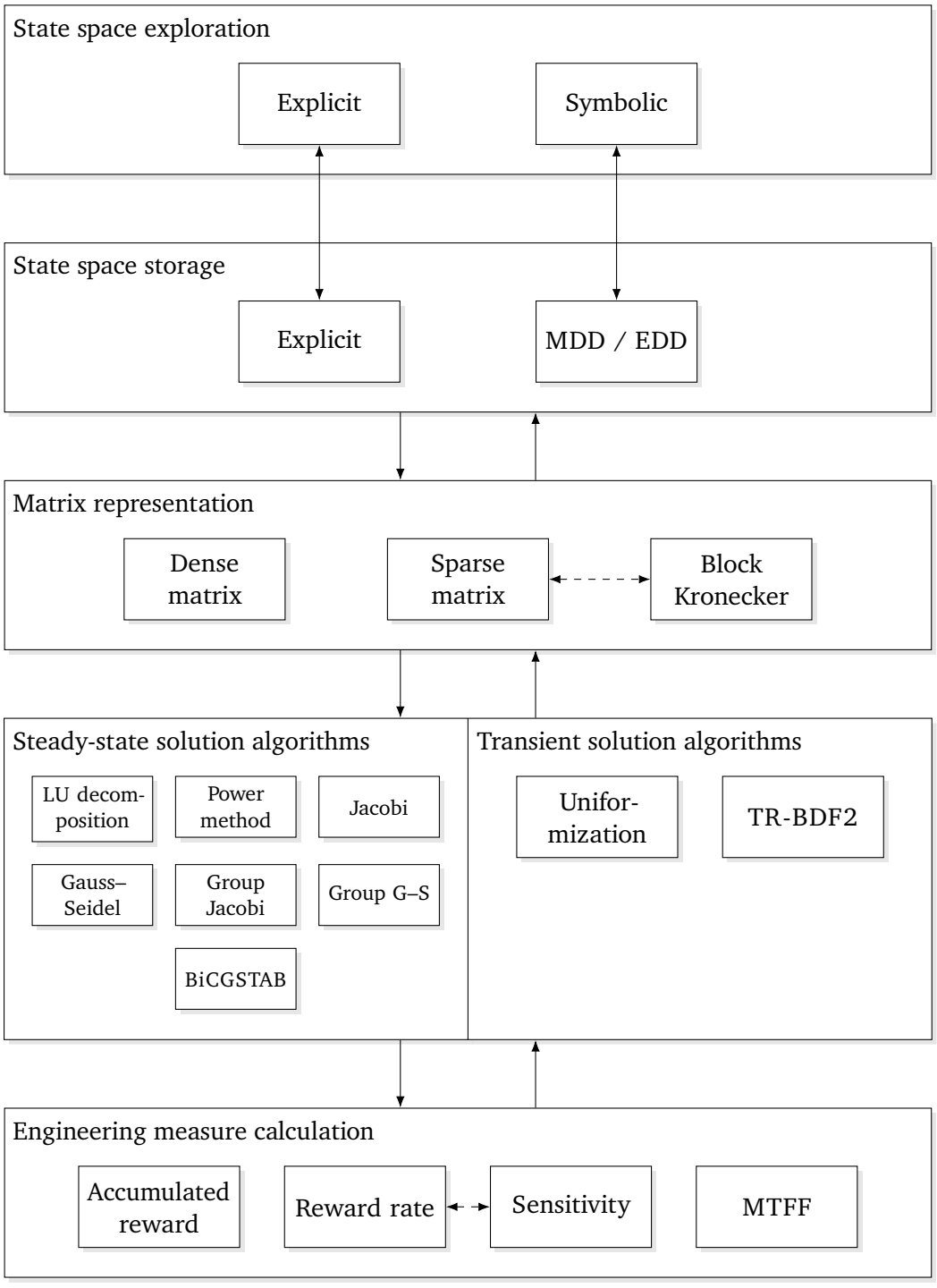

Figure 3.3 Architecture of the configurable stochastic analysis framework.

Table 3.2 Transient solvers supported by our framework.

	see	instantenous distribution	accumulated distribution	uses inner solver	block matrix
Uniformization	p. 61	✓	✓	–	✓
TR-BDF2	p. 62	✓	not impl.	✓	not impl.

space fits into memory. In addition they are provided for benchmarking and software redundancy reasons too.

The algorithms operating on a superposed SPN receive the model and a decomposition as an input. Partitions needed for the decomposition may be provided by the user as part of the model or generated on the fly.

- The generator matrix may be stored in sparse matrix representation or decomposed into block Kronecker form [7]. The matrix can be build from both explicitly or symbolically stored state spaces.

  To facilitate block Kronecker matrix generation, we propose a purely symbolic algorithm. The developed solution avoids any overheads of explicit state space operations.

- The resulting matrices, in a possibly decomposed form, are part of a specialized data structure. Extremely large matrices may be stored with the developed decomposition algorithms (e.g. linear combinations, Kronecker products, contatenations into block structures). The data structure defines generic vector and matrix operations, as well as more specific manipulations performed by stochastic analysis algorithms.

  State space exploration and generator matrix decomposition methods are presented in Chapter 4, including our theoretical and algorithmic contribution for block Kronecker decomposition.

- The execution of the operations on the data structures can be set at runtime. This allows the use of different implementations at the different stages of the workflow, or when different algorithms are employed to calculate multiple performance measures. Whenever possible, both sequential and parallel implementations of the most common operations are available for the supported datatypes.

- Several numerical algorithms are provided for steady-state and transient analysis of Markov chains. The user can select the algorithm most suitable for the model under study. The algorithm library supports the combination of the algorithms and data structures at different levels of computations. This allows us to fine

tune the numerical solution and solve every component with the most suitable algorithm.

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 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.

3.2.1 Formalisms

Our stochastic analysis framework supports models in the Stochastic Petri Net with inhibitor arcs formalism (see Definition 2.7 on page 13). Structured models are handled as Superposed Stochastic Petri Nets (see Definition 2.9 on page 18). However, any modeling formalism can be processed by integrating the appropriate state space exploration algorithms with the workflow.

Transition rates in the SPNs can be arbitrary algebraic expressions containing references to *sensitivity variables*. These variables correspond to the parameter vector $\boldsymbol{\theta}$ of the Markov chain sensitivity analysis. However, the rate expression may not depend on the marking of the net.

Reward structures are defined as Stochastic Reward Nets (see Definition 2.8 on page 16). An SRN reward structure may be specified by composing any *reward expressions* of the forms

1. (p, w), where $p \in P$ is a place and w is a *reward weight expression*. This reward expression is equivalent to a rate reward $rr(M) = M(p) \cdot w$, i.e. the value of w is multiplied by the numer of tokens on p.

2. (t, w), where $t \in T$ is a transition and w is a reward weight expression. This is equivalent to an impulse reward $ir(t, M) = w$ gained upon the firing of t.

3. $\varphi \rightarrow w$, where φ is a Computational Tree Logic (CTL) expression and w is a reward weight expression. This is equivalent to the rate reward $rr(M) = w$ if φ holds in M, 0 otherwise.

A reward weight expression is an algebraic expression that may refer to places and transition rates in the net. Refrences to places are replaced by the number of tokens upon evaluation. For example, the reward expression (p, w) may be written as $\mathsf{true} \rightarrow p \cdot w$ or $p > 0 \rightarrow p \cdot w$ using CTL.

Reward expressions with CTL are only allowed when symbolic state spaces representation is used, as CTL evaluation[1] is performed symbolically [12].

Running example 3.1 Consider the reward structures defined over the *SharedResource* Petri net from Running example 2.6 on page 17.

The utilization of the shared resource can be described by the reward expression

$$resourceUtilization = \{(p_{S_1}, 1), (p_{S_2}, 1)\},$$

which is equivalent to the SRN reward structure

$$rr_1(M) = M(p_{S_1}) + M(p_{S_2}), \quad ir_1(t, M) \equiv 0. \tag{2.12 revisited}$$

This can be also written as

$$resourceUtilization = \{p_{S_1} > 0 \vee p_{S_2} > 0 \to 1\}$$

using CTL, because the places S_1 and S_2 are 1-bounded in the *SharedResource* model.

Completed calculations are described by

$$completedCalculations = \{(t_{s_1}, 1), (t_{r_2}, 1)\},$$

which is equivalent to the reward structure

$$rr_2(M) \equiv 0, \quad ir_2(t, M) = \begin{cases} 1 & \text{if } t \in \{t_{r_1}, t_{r_2}\}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.13 revisited}$$

3.2.2 Analysis

The framework introducted in this paper supports the configurable stochastic analysis of the following problems:

- expected steady-state reward rates $\mathbb{E}R$ for any reward structure defined by reward expressions,
- expected transient reward rates $\mathbb{E}R(t)$ and accumulated rewards $\mathbb{E}Y(t)$,
- *complex rewards*, which are algebraic expressions of mean reward rates and accumulated rewards (e.g $1 + \mathbb{E}R(t)/\mathbb{E}Y(t)$),
- sensitivity of mean steady-state reward rates and complex rewards involving steady-state rates,

[1] The symbolic state space exploration and CTL evaluation component is currently provided by the PetriDotNet [15] tool.

- mean-time to failure *MTFF* and associated failure mode probabilities.

Configurable stochastic analysis provides the combination of multiple solver and representation algorithms for the efficient computation of the introduced properties.

3.2.3 Reward and sensitivity computation

Transition and reward rates are stored as algebraic expression trees in the input SPN models. Symbolic operations, such as partial differentiation may be performed exactly on the trees using algebraic laws, as the evaluation of the expressions can be delayed.

In reward and MTFF calculations, rate expressions are evaluated by replacing sensitivity parameters with their values before the matrix Q is composed. Thus, the elements of a matrix are not expression trees, but floating point numbers and matrix generation has to be performed only when sensitivity parameters are changed.

Reward weight expressions may refer to the token counts on places, therefore they must be evaluated for every marking individually. If a CTL reward expression $\varphi \to w$ is used, evaluation is skipped in markings where φ is false.

Steady-state sensitivity calculation, shown in Figure 3.4, is the most complicated post-processing in the workflow. Partial derivatives of the transition rate expressions and reward weight expressions are taken to calculate $\partial\, \mathbb{E}R/\partial\theta[i]$ using eqs. (2.7) and (2.8).

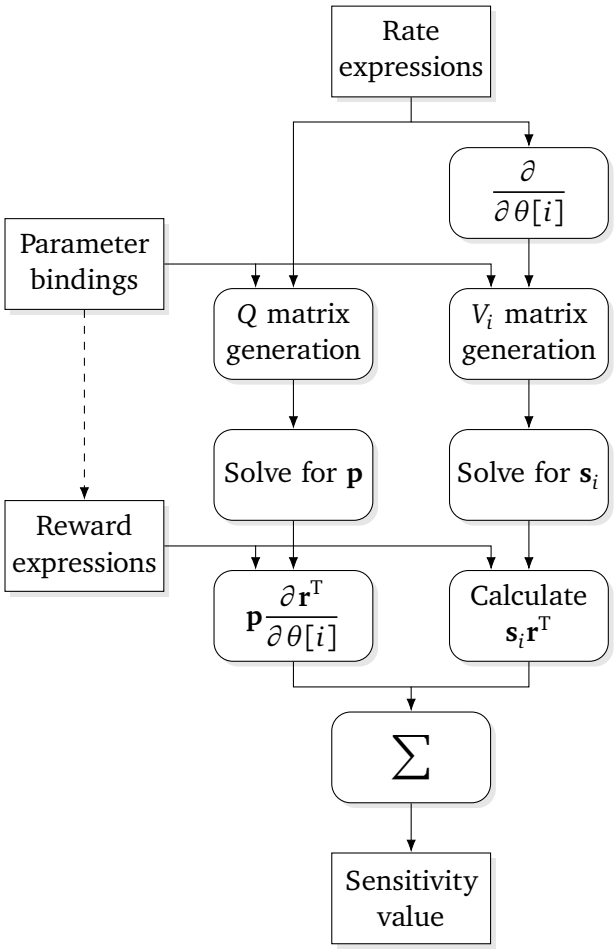

Figure 3.4 Reward and sensitivity calculation from expression tree inputs.

Chapter 7

Conclusion and future work

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 algoprithms, 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 attepmts 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) [23] and Stochastic Automata Networks (SAN) [19], as well as models with more general stochastic transition behaviors [21];
- the implementation and development of further numerical algorithms, including those that can take advantage of the various decompositions of stochastic models [5, 6, 13];
- reduction of the size of Markov chains through the exploitation of model symmetries [4, 17];
- the development of preconditioners for the available interative numerical solution methods [20];
- distributed implementations of the existing algorithms [8];
- support for fully symbolic storage and solution of Markov chains [11, 22, 24];
- the use of tensor decompositions instead of vectors to store state distributions and intermediate results to greatly reduce memory requirements of solution algorithms [2, 14, 16].

References

[1] 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.

[2] 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.

[3] 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.

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

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Dániel Darvas. *Szaturáció alapú automatikus modellellenőrző fejlesztése aszinkron rendszerekhez [in Hungarian]*. 1st prize. 2010. URL: http://petridotnet.inf.mit.bme.hu/publications/OTDK2011_Darvas.pdf.

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

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] *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.

[23] 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.

[24] 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.