A Comparative Study of Stochastic Analysis Techniques

Monika Heiner
Computer Science Department
BTU Cottbus, Germany
monika.heiner@tu-cottbus.de

Christian Rohr
Magdeburg Centre for Systems Biology (MaCS)
Magdeburg, Germany
rohr@mpi-magdeburg.mpg.de

Martin Schwarick
Computer Science Department
BTU Cottbus, Germany
ms@informatik.tu-cottbus.de

Stefan Streif
Magdeburg Centre for Systems Biology (MaCS)
Magdeburg, Germany
streif@mpi-magdeburg.mpg.de

ABSTRACT

Stochastic models are becoming increasingly popular in Systems Biology. They are compulsory, if the stochastic noise is crucial for the behavioural properties to be investigated. Thus, substantial effort has been made to develop appropriate and efficient stochastic analysis techniques. The impressive progress of hardware power and specifically the advent of multicore computers have ameliorated the computational tractability of stochastic models. We report on a comparative study focusing on the three base case techniques of stochastic analysis: exact numerical analysis, approximative numerical analysis, and simulation. For modelling we use extended stochastic Petri nets, which allows us to take advantage of structural information and to complement the stochastic analyses by qualitative analyses, belonging to the standard body of Petri net theory.

1. MOTIVATION

Biochemical systems are inherently governed by stochastic laws. Consequently, stochastic models are an adequate choice for their thorough investigation. But due to the computational expense for their analysis, the considerations are often restricted to the averaged case and continuous models are used instead. However, stochastic models are unavoidable if the stochastic noise is crucial for the behavioural properties to be investigated.

In the light of this demand, substantial effort has been made to develop appropriate stochastic analysis techniques and to implement them by sophisticated data structures and efficient algorithms. The impressive progress of hardware power and specifically the advent of multicore computers have stirred new hope for the computational tractability of stochastic models. We performed a comparative study to characterize the current state with focus on the three base case techniques of stochastic analyses.

- **Exact numerical analysis** considers the complete set of reachable states (state space). This requires generally finite state spaces. Time-bounded as well as time-unbounded properties can be determined.
- **Approximative numerical analysis** neglects states below a specified probability threshold. Thus, only a subset of all reachable states is constructed, which also works for infinite state spaces. Time-bounded properties can be approximated with the chosen accuracy.
- **Simulation** generates a finite number of finite random walks through the state space. This double approximation does not care about the size of the state space, but is restricted to time-bounded properties.

For modelling we use stochastic Petri nets (SPN), which allows us to take advantage of structural information and to complement the stochastic analyses by qualitative analyses, belonging to the standard body of Petri net theory [HGD08]. We employ a toolkit consisting of the modelling tool Snoopy [RMH10], providing also approximative numerical analysis and simulation, the qualitative analysis tool Charlie [Fra09], and the model checker IDD-MC which outperforms comparable tools as demonstrated in [HST09, SH09].

This paper is organised as follows. We start off with an overview on the main features of our modelling language. In Section 3 we recall the basic principles of the three base case techniques, discuss some implementation issues, specifically the parallelization, and give their individual pros and cons. In Section 4 we compare the techniques in terms of efficiency by checking temporal queries against three stochastic models of our benchmark testsuite. We summarize the lessons learnt from our computational experiments in a couple of user guidelines. Finally, we refer briefly to some related work, before concluding with an outlook on future directions.

The main contributions of our paper are:

- presentation of the first toolkit supporting the three base case techniques, including parallelized exact numerical analysis and parallelized simulation,
- first comparative study of the three base case techniques of stochastic analysis, characterizing the current state of tractability,
- user guidelines, helping specifically the uninitiated reader to navigate through stochastic analysis techniques,
- identification of the potential to enhance capabilities and performance of stochastic analyses.

CMSB '10, September 29 – October 1, Trento, Italy

2. MODELLING WITH SPN

We assume basic understanding of standard Petri nets. To be self-contained we recall incrementally the main aspects of extended stochastic Petri nets relevant for this paper.

A biochemically interpreted **Petri Net** $\mathcal{PN}$ is a 4-tuple (P, T, f, s_0). As usual, $P = \{p_1, p_2, \dots p_m\}$ denotes the set of places, modelling the biochemical species, $T = \{t_1, t_2, \dots t_n\}$ the set of transitions, modelling the biochemical reactions, $f : ((P \times T) \cup (T \times P)) \to \mathbb{N}$ the arc weight function, and $s_0 : P \to \mathbb{N}$ the initial state (marking), defining the initial tokens on each place. The state of a Petri net can change by the firing of transitions. The set of all states $s' \in \mathbb{N}^{|P|}$, reachable from s_0 by the firing of a transition word $w \in T^*$, written as $s_0 \xrightarrow{w} s'$, builds the state space and is denoted with S. For more details see [HGD08].

Stochastic Petri Nets $\mathcal{SPN}$ build on $\mathcal{PN}$, but transitions have an exponentially distributed firing delay, characterized by the firing rate λ. The firing rates are typically transition-specific and state-dependent and defined by propensity (hazard) functions. We denote the propensity function for the transition t_j with h_j. We deal with biologically interpreted stochastic Petri nets; thus we consider besides arbitrary arithmetic functions specifically propensity functions representing *mass action semantics* and *level interpretation semantics*. All these functions have in common that the domain is restricted to the pre-places of the corresponding transition; see [GHL07].

Additionally, there are five special arc types: *read, inhibitor, equal, reset, and modifier arc*. They always go from a place to a transition. The first three arcs establish additional side conditions for the enabledness of a transition; but upon firing, the marking on the tested place is not changed. Contrary, the reset arc does not influence the enabledness, but upon firing all tokens on the tested place are removed. The modifier arc does neither restrict the enabledness nor does it change the tokens upon firing. But the firing rate may depend on the current marking of the tested place.

Special arcs enhance the modelling comfort, but bring the Turing power (with exception of the read and modifier arcs), which destroys the general decidability of non-trivial behavioural properties. In bounded models, special arcs can be simulated by standard arcs and some kind of unfolding of the tested places. However, this might lessen the analysis efficiency as discussed in [SH09]. Special arcs do not cause any trouble for dynamic, i.e. state-space-related analysis techniques. The special arcs as supported in our toolkit do not destroy the locality principle, and modifier arcs have actually been introduced to keep the locality, which is crucial for the efficiency of our analysis tools. For more details see [HST09, RMH10].

Generalised Stochastic Petri Nets $\mathcal{GSPN}$ build on $\mathcal{SPN}$ enriched by *immediate transitions*, which have a zero firing delay. Thus, they fire immediately after getting enabled and always prior to stochastic transitions. Consequently, getting enabled and the firing itself coincide, if not prevented by another competing immediate transition. A cyclic system behaviour involving only the firing of immediate transitions corresponds to an infinite behaviour without time progress – we get a new type of modelling fault, the time deadlock. Immediate transitions may help to avoid stiff systems by using them for transitions with extremely high rates (non-significant delay) compared to the other transitions in the system.

Extended Stochastic Petri Nets $\mathcal{XSPN}$ build on $\mathcal{GSPN}$ enriched by deterministically timed transitions, which come in two flavours.

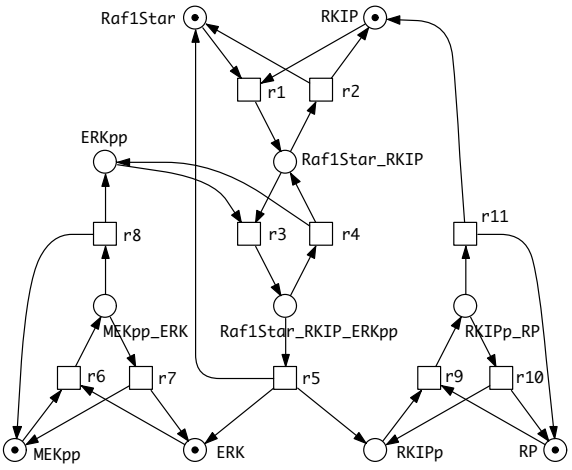
Deterministic transitions fire after a deterministic firing delay. The delay is always relative to the time point where a transition gets enabled. Deterministic transitions may be useful to reduce networks, e.g. by replacing a linear sequence of stochastic transitions by one deterministic transition with the delay set to the sum of the expectation values of the transition sequence.

Scheduled transitions fire according to a schedule specifying absolute time points of the simulation time. A schedule can specify just a single time point, or equidistant time points within a given interval, triggering the potential firing (if the transition is enabled) once or periodically. They support the straightforward modelling of wet-lab experiment scenarios. The core model can be disturbed at well-defined time points as it is done experimentally with the actual biological system under investigation in the wet-lab. Scheduled transitions can be simulated by net components deploying immediate and deterministic transitions, see [HLGM09].

There is a standard firing rule, applying equally to all $\mathcal{XSPN}$ transition types; specifically, a transition may lose its enabledness while waiting for the delay to expire, and the firing itself does never consume time.

$\mathcal{SPN}$s enjoy the Markovian property; thus, their semantics is defined by a Continuous Time Markov Chain (CTMC). The semantics of $\mathcal{GSPN}$ can still be reduced to CTMC, while the unrestricted use of deterministic transitions in $\mathcal{XSPN}$ destroys the Markovian property [Ger01].

Petri nets are famous for their graphical representation, but can equally be specified in a textual style. Snoopy supports the design of larger nets by logical nodes (connecting separated net parts) and macro nodes (defining hierarchical subnets). The graph structure may contribute to the understanding of the network.



$$\text{Raf1Star} + \text{RKIP} \overset{r1,r2}{\leftrightarrow} \text{Raf1Star\_RKIP}$$
$$\text{Raf1Star\_RKIP} + \text{ERKpp} \overset{r3,r4}{\leftrightarrow} \text{Raf1Star\_RKIP\_ERKpp}$$
$$\text{Raf1Star\_RKIP\_ERKpp} \overset{r5}{\to} \text{Raf1Star} + \text{ERK} + \text{RKIPp}$$
$$\text{ERK} + \text{MEKpp} \overset{r6,r7}{\leftrightarrow} \text{MEKpp\_ERK} \overset{r8}{\to} \text{ERKpp} + \text{MEKpp}$$
$$\text{RKIPp} + \text{RP} \overset{r9,r10}{\leftrightarrow} \text{RKIPp\_RP} \overset{r11}{\to} \text{RKIP} + \text{RP}$$

Figure 1: Petri net for one of our benchmarks (ERK) together with its textual description by a set of reaction equations.

A Petri net can be executed by playing the token game. The animation of the token flow allows to experience the net behaviour. See Figure 1 for an example, which will later serve as one of our benchmarks.

3. ANALYSIS OF SPN

3.1 Preliminaries

Qualitative analysis. Petri nets enjoy a rich body of qualitative analysis techniques; see [HGD08] for an overview. Here we confine ourselves to the static decision of boundedness, which is crucial for our discussion. A net is bounded, if the maximal number of tokens on each place does not exceed some constant in all reachable states. To decide it, we introduce the incidence matrix $\mathbf{C} : P \times T \to \mathbb{Z}$, with $\mathbf{C}(p,t) = f(t,p) - f(p,t)$. The matrix column $\mathbf{C}(.,t_j)$ is denoted with Δt_j. A net is structural bounded (bounded for any initial state) iff there is a solution for $\mathbf{x} \cdot \mathbf{C} \leq 0$, $\mathbf{x} > 0$, which can efficiently be checked [SR99].

Bounded Petri nets have a finite state space and the reachability relation can be represented by a finite reachability graph, where nodes are labelled with states, and edges with the transitions which are responsible for the state change. Algorithm 1 does a simple construction of the reachability graph starting with the initial state. It obviously terminates only if the Petri net is bounded. In practice the state space explosion problem calls for symbolic state space representations and the use of more sophisticated construction principles such as saturation [Tov08].

Algorithm 1 Reachability graph construction.

Require: $\mathcal{PN}/\mathcal{SPN}$ with initial state s_0
1: *stateSet* $S := E := \emptyset$
2: *stateSet* $U := \{s_0\}$
3: **while** $U \neq \emptyset$ **do**
4: $s := selectOneOf(U)$
5: $U := U \setminus \{s\}$
6: $S := S \cup \{s\}$
7: **for all** transitions $t_j \in T$ enabled at s **do**
8: $s' := s + \Delta t_j$
9: **if** $s' \notin S \cup U$ **then**
10: $U := U \cup \{s'\}$ ▷ new node
11: **end if**
12: $E := E \cup \{(s, t_j, s')\}$ ▷ new edge
13: **end for**
14: **end while**

Quantitative analysis. The reachability graph does not contain any timing information; thus, it is not sufficient for quantitative analysis of an $\mathcal{SPN}$. But replacing each edge label (transition name) by the state-dependent rate of the firing transition defines a CTMC, which is a 3-tuple $(S, \mathbf{R}, s_0)$ with S denoting the set of reachable states of the underlying net, $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ the rate function, and s_0 the initial state. $\mathbf{R}$ is usually represented by a real-valued matrix with the non-zero entries defined by the state-dependent firing rates; assuming no parallel transitions, i.e.

$$\mathbf{R}(s,s') = \begin{cases} h_j(s) & \exists\, t_j \in T : s \xrightarrow{t_j} s' \\ 0 & \text{otherwise} \,. \end{cases} \tag{1}$$

The exit rate $E(s) = \Sigma_{s' \in S} \mathbf{R}(s,s')$ is the sum of entries of the matrix row indexed with s. A state s with $E(s) = 0$ is an absorbing state. The probability of a transition t enabled in state s to fire (which results in state s') within n time units is $1 - e^{-\mathbf{R}(s,s') \cdot n}$. The number of non-zero entries in each row is bounded by the number of transitions in the SPN. Thus, the rate matrix of a CTMC is always very sparse.

Having the CTMC, one can reason about the transient probability $\pi(\boldsymbol{\alpha}, s, \tau)$ to be in a certain state s at a certain time point τ starting with a probability distribution $\boldsymbol{\alpha}$. The state-specific transient probability $\pi(\boldsymbol{\alpha}, s, \tau)$ can be generalized to $\boldsymbol{\pi}(\boldsymbol{\alpha}, \tau)$ for all reachable states. The transient probability for infinite time is called the steady state probability $\boldsymbol{\pi}(\boldsymbol{\alpha})$.

Model checking. A more sophisticated analysis approach is model checking. It automatically determines whether a system (in our case specified by a Petri net) satisfies a specific property expressed in some kind of temporal logic. The Continuous Stochastic Logic (CSL) introduced in [ASSB00] and extended in [BHHK00] is a stochastic adaption of the Computation Tree Logic (CTL) [CGP01] to formulate properties of CTMCs. It has been used to analyze biochemical networks, e.g., in [CVGO06, GHL07, KNP08].

To give an example, the question: "What is the probability to reach within time τ a state where the sum of molecules of the species A and B is not less than *threshold*?" can be expressed by the CSL formula

$$\mathcal{P}_{=?}[\mathbf{F}_{[0,\tau]}(A + B \geq threshold)] \,. \tag{2}$$

For the comparison of the three base case techniques we confine ourselves to the following CSL formula template

$$\mathcal{P}_{=?}[\mathbf{F}_{[\tau,\tau]}(sp)] \,; \tag{3}$$

in words: "What is the probability to be at time point τ in a state satisfying some state property *sp*?" For this special case, model checking is nothing else than computing the transient probabilities of all states satisfying the state formula *sp* at time point τ and providing their sum.

In the following, a state property *sp* is restricted to qualitative conditions of states. Contrary, the CSL specification in [BHHK00] allows state properties to contain probability and steady state operators. The reason for our restriction will be explained in the next sections.

3.2 Exact Numerical Analysis

One of the standard techniques for computing transient probabilities is the uniformization method. Applying this method means to solve the uniformization equation

$$\boldsymbol{\pi}(\boldsymbol{\alpha}, \tau) = \sum_{k=0}^{\infty} \mathbf{P}^k e^{-\gamma\tau} \frac{(\gamma\tau)^k}{k!} \,. \tag{4}$$

It is based on the idea to embed a discretization of the CTMC, which is characterized by the probability matrix $\mathbf{P}$, into a Poisson process with rate γ. To compute $\boldsymbol{\pi}(\boldsymbol{\alpha}, \tau)$ requires to truncate the infinite sum in Eq. 4 to some k_{max} as shown in Algorithm 2. For more details, especially how to determine $\gamma, \mathbf{P}$ and k_{max}, see [Ste94].

The method is easy to implement, but the CTMC and at least three real-valued vectors $(\mathbf{v}, \mathbf{v}', \boldsymbol{\alpha})$ in the size of the state space have to be stored in memory. From a practical view point, the CTMC has to be finite, although the basic algorithm only requires a countable state space and an upper bound for the exit rates of the CTMC. But even for a finite CTMC, the state space explosion makes its analysis an expensive task, which is possibly practically infeasible.

Algorithm 2 Uniformization method.

Require: CTMC M, time point τ, initial distribution $\boldsymbol{\alpha}$
1: determine $\gamma, \mathbf{P}$, k_{max} depending on M and τ
2: *distribution* $\mathbf{v} := \mathbf{v}' := \alpha$
3: **for** $k := 1$ **to** k_{max} **do**
4: $\quad \mathbf{v}' := \mathbf{v}' \cdot \mathbf{P} \cdot \frac{\gamma\tau}{k}$
5: $\quad \mathbf{v} := \mathbf{v} + \mathbf{v}'$
6: **end for**
7: $\boldsymbol{\pi}(\boldsymbol{\alpha}, \tau) := e^{-\gamma\tau}\mathbf{v}$

The steady state probability can be computed by solving a linear system of equations in the size of the state space. Thus, also the steady state analysis suffers from the state space explosion and requires a finite CTMC. Iterative methods as the Jacobi or Gauss-Seidel method are the favored ones.

In both cases, the exact numerical probability computation starts with constructing the complete state space before eventually performing a repeated multiplication of a matrix and a vector over real numbers in the dimension of the system's state space. Thus, a compact encoding of the matrix is advisable, using, e.g., sparse matrices, Multi-Terminal Binary Decision Diagrams (MTBDD), or Kronecker Products [MP04]. However, the success of these techniques is bound to special conditions. For instance, a MTBDD representation requires a moderate number of distinct non-zero values in the matrix and suffers from a higher boundedness degree, caused by the increase of BDD variables [SH09]. In general both conditions do not hold for biological networks.

We use the model checker IDD-MC for doing exact numerical analysis. IDD stands for Interval Decision Diagrams which are a very efficient data structure for a symbolic representation of huge sets of integer vectors, such as states of Petri nets [Tov08]. The tool offers qualitative and quantitative state-space-based analysis, including CTL/CSL model checking and the standard CTMC analyses. It features some important characteristics, which are essential to achieve efficiency, see [SH09]. IDD-MC symbolically computes all required information on-the-fly using the reachable states and the underlying stochastic Petri net. This is done by traversing the IDD representation of the state space for all Petri net transitions. During these traversals the algorithm computes for all CTMC transitions the index of the source and the target state and the possibly state-dependent firing rate.

The implementation of the algorithm is optimized in a way that it treats all Petri net transitions at once during one traversal and that it uses a sophisticated caching technique. Moreover, symbolic analysis requires in general to find a suitable variable ordering, which is known to be a NP-complete problem. IDD-MC implements heuristics which usually generate good orders resulting in small IDDs in terms of number of nodes. Moreover, the number of IDD variables does not depend on the boundedness degree of the system.

Model checking. CSL model checking of a CTMC M can be realised by transient analysis. The basic concept is to do transient analysis for a CTMC M' which has been derived from M by making certain states absorbing, depending on the formula to be checked. IDD-MC supports full CSL for $\mathcal{SPN}$; see [BHHK00] for syntax and semantics definition.

Parallelization. To our knowledge IDD-MC is the only available tool which features symbolic, multi-threaded transient analysis to exploit workstations possessing multiple CPUs. For this purpose, it generates a state space partitioning based on the lexicographic index of the states and computes each matrix-vector multiplication concurrently by a user-specified number of threads. Multi-threading is available for all analyses using the transient and the Jacobi solver. This is similar to the explicit approach reported in [BH06].

Limitations. Exact numerical analysis is generally infeasible for infinite CTMCs and often practically infeasible for large CTMCs. Thus, approximation techniques are required to treat $\mathcal{SPN}$ models with huge or infinite states spaces.

3.3 Approximative Numerical Analysis

Approximative numerical analysis tries to overcome the problem of an unmanageable state space size by pruning insignificant states. The idea is to combine a breadth-first variant of the state space construction of Algorithm 1 with a transient analysis using uniformization. During the construction, all explored states having a probability below a specified threshold δ will be removed from the current state space. Thus, only a finite subset of a possibly infinite state space will be considered. This "sliding window" method [HMW09] can be further combined with a technique called adaptive uniformization, where the Poisson process is replaced by a birth process. This combination was first introduced in [DHMW09] as fast adaptive uniformization (FAU) and is sketched in Algorithm 3.

Algorithm 3 Fast adaptive uniformization algorithm.

Require: $\mathcal{SPN}$ with initial state s_0; time interval $[\tau_0, \tau_{max}]$; λ_{max} as upper bound of λ, threshold δ
1: *int* $k := 0$
2: *time* $\tau := \tau_0$
3: *stateSpace* $S := \{s_0\}$
4: **while** $\tau \leq \tau_{max}$ **do**
5: $\quad \lambda := collect(S)$
6: $\quad$ **if** $\lambda > \lambda_{max}$ **then** $\quad \triangleright$ λ_{max} is to small
7: $\quad\quad$ terminate
8: $\quad$ **end if**
9: $\quad c := birthProcess(\lambda, k)$
10: $\quad S := propagate(S, \lambda, c)$
11: $\quad S := pruneStates(S, \delta)$
12: $\quad \tau := \tau + \lambda^{-1}$
13: $\quad k := k + 1$
14: **end while**

It approximates the CTMC at time point τ_{max}. At each step of the breadth-first state space construction, the current state space S never keeps states with a probability less than δ. The method *collect* determines the maximum exit rate of the states in S. The solution c of the birth process depending on λ and k is required by the method *propagate*, which realizes a complete step in the discretized CTMC. Doing so *propagate* updates the probability for all states of S and their direct successors. This will generally add new states to the current state space. Afterwards *pruneStates* removes all states with a probability below the threshold δ. A detailed description of the algorithm can be found in [DHMW09].

Snoopy's FAU implementation is extended by the handling of immediate transitions. So we can analyse not only $\mathcal{SPN}$, but also $\mathcal{GSPN}$. We have to deal with two different types of states: *tangible states*, where only stochastic transitions are enabled, and *vanishing states* that arise if an immediate transition gets enabled. Due to the higher

priority of immediate over stochastic transitions, vanishing states require instantaneous attention until no more vanishing states are left. The simulation time does not progress during the processing of vanishing states, because immediate transitions do not have a firing delay.

Model checking. Model checking Eq. 3 means for the approximative numerical analysis to check the states considered at time point τ against the state formula *sp* and to sum up the probabilities of the states where *sp* holds. For the time being the supported model checking capabilities are restricted to this special case. In principle the algorithm allows to check any unnested, time-bounded CSL formula without steady state operator by applying the technique sketched in Section 3.2.

The two numerical approaches might take advantage from a cross-fertilization. FAU, in its current version, is a one-phase model checking approach – the property is checked on-the-fly. Contrary, CSL model checking is a two-phase approach – first the state space is constructed, second the property is checked. FAU has the potential for CSL model checking in the chosen accuracy. In turn, the idea of adaptive uniformization and vanishing states can be equally applied to exact numerical analysis.

Parallelization. FAU has been just recently introduced. Thus, the potential gain by parallelization has not been explored yet. But, there exist several ideas of parallel breadth-first state space construction which seem to be promising [BH06, Kno99].

Limitations. FAU allows transient analysis of unbounded $\mathcal{GSPN}$ models. However, the technique may also exceed the limits of the physical memory, if after the pruning step too many CTMC states remain.

3.4 Simulation

In a situation where the memory effort prevents the use of numerical methods, an efficient memory-saving method is required. Instead of storing the CTMC or just a subgraph of it, the stochastic simulation algorithm (SSA) introduced in [Gil77] only creates a single finite path through the possibly infinite CTMC. The computation of such a simulation run (trajectory, path) needs only to store the current state. The basic idea is as follows.

Given the system is at time point τ in state s. The probability that a transition $t_j \in T$ will occur in the infinitesimal time interval $[\tau, \tau + \Delta\tau)$ is given by:

$$P(\tau + \Delta\tau, t_j \mid s) = h_j(s) \cdot e^{-E(s)\cdot\Delta\tau} \tag{5}$$

For each transition t_j, the rate is given by the propensity function h_j, where $h_j(s)$ is the conditional probability that transition t_j occurs in the infinitesimal time interval $[\tau, \tau + \Delta\tau)$, given state s at time τ. So, the enabled transitions in the net compete in a race condition. The fastest one determines the next state and the simulation time elapsed. In the new state, the race condition starts anew.

The SSA simulates every transition firing (basically by using Eq. 5) one at a time, and keeps track of the current system state. To determine the time increment $\Delta\tau$ and to select the next Petri net transition to fire requires to generate two random numbers $(r1, r2)$ uniformly distributed on $(0, 1)$. Different trajectories of the CTMC are obtained by different initializations of the random number generator (line 1). Reliable conclusions about the system behaviour require many simulations due to the stochastic variance. Thus, the usefulness of the simulation approach depends on the runtime for each individual simulation run. So, accelerating SSAs are desirable without changing the basic ideas of Algorithm 4.

Algorithm 4 Stochastic simulation algorithm.

Require: $\mathcal{SPN}$ with initial state s_0, time interval $[\tau_0, \tau_{max}]$

```
 1: initRand(seed)
 2: time τ := τ_0
 3: state s := s_0
 4: while τ < τ_max do
 5:     draw random numbers r_1, r_2,
          uniformly distributed on (0, 1)
 6:     Δτ = − ln (getURand()) / E (s)
 7:     r := getURand()
 8:     e := 0
 9:     for all transitions t_j ∈ T enabled at s do
10:         e := e + h_j(s)
11:         if e > r · E (s) then
12:             s := s + Δt_j
13:             break
14:         end if
15:     end for
16:     τ := τ + Δτ
17: end while
```

A crucial point is the choice of pseudo-random number generator. We decided to use the Mersenne Twister [MN98], which is one of the best performing pseudo-random number generators. Another issue is how to determine the next transition to fire. Many research has been devoted to this subject [MPC+06], but the performance gain compared to the "Gillespie" SSA is rather moderate and model-dependent.

The simulative processing of immediate, deterministic and scheduled transitions is rather straightforward, see [Ger01]. In short, the Algorithm 4 needs to be extended in two ways.

- After every firing of a Petri net transition (line 12), it needs to be checked whether immediate transitions got enabled. If so, these have to be processed until no more immediate transitions are enabled. This possibly leads to a time deadlock, if there exists a cyclic path of immediate transitions.
- Having calculated the next time step (line 6), it needs to be checked whether a deterministic or scheduled transition gets enabled in the time interval $[\tau, \tau + \Delta\tau]$. If yes, the one closest to τ is processed and the simulation time will be set to the value of this transition, i.e. $\tau = \tau_{t_j}$.

Model checking. A run fulfills the path formula $\mathbf{F}_{[\tau,\tau]}(sp)$, if the last considered state satisfies the state formula *sp*. The ratio of the number of fulfilling and total number of runs leads to an approximation of the desired probability of Eq. 3. The CSL model checking capabilities are subject to the same restrictions as FAU and are currently limited to this special case.

To achieve an appropriate accuracy of the results, one has to determine the required amount of simulation runs. The method of our choice is the confidence interval as described in [SM08]. The confidence interval contains the property of interest with some predefined probability, called confidence level. This confidence level has usually values of 90%, 95%, or 99%. Assuming 95% and an accuracy of the results of

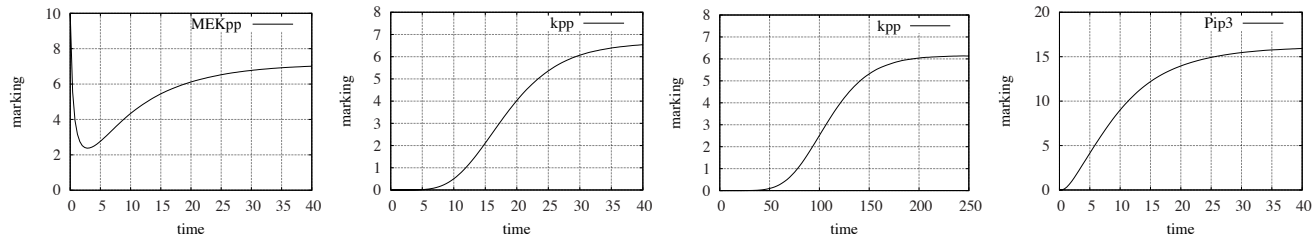


Figure 2: The plots display (from left to right) the simulation results of ERK with $N = 10$, MAPK1 and MAPK2 with $N = 8$, and ANG after 100,000 simulation runs. They are the starting point of our experiments.

10^{-5} leads to $\approx 38,000,000$ runs. The number of required simulation runs increases exponentially with the accuracy. For example, an accuracy of 10^{-9} with the same confidence level would need about $3.8 \cdot 10^{11}$ simulation runs.

Parallelization. The individual simulation runs are constructed independently; thus parallelization is straightforward. It basically requires to organize a master thread distributing the work load on the n identical slave threads, and collecting the results.

Limitations. If there is a demand for high accuracy, the runtime may become the limiting factor.

3.5 Summary

The three base case techniques have been implemented in Snoopy and IDD-MC. IDD-MC offers exact numerical analysis and full CSL model checking for bounded $\mathcal{SPN}$ models in a multi-threaded fashion. Snoopy supports approximative numerical analysis (FAU) of $\mathcal{GSPN}$ and multi-threaded exact stochastic simulation of $\mathcal{XSPN}$; both combined with model checking of Eq. 3.

See Table 1 for a concise summary of the main characteristics discussed so far. There might already be a restricted choice for suitable analysis techniques, depending on the kind of model to be investigated and the kind of property to be checked. In the next section we compare the three base case techniques in efficiency terms.

Table 1: Comparison of the capabilities of the three base case techniques

criterion	exact	FAU	simulation
supported model class	$\mathcal{SPN}^{*)}$	$\mathcal{GSPN}$	$\mathcal{XSPN}$
state space stored	complete	partial	single state
infinite state space	–	+	+
parallelization*)	+	–	+
CSL model checking*)	full CSL	Eq. 3	Eq. 3

*) as currently supported in our toolkit

4. EXPERIMENTAL RESULTS

In this section we present selected benchmark results for the analysis techniques discussed in Section 3. The experiments were done on 2.26 GHz Apple MAC Pro with 32 GB RAM and eight physical (with hyperthreading 16 logical) cores. Simulation and IDD-MC were done on Mac OS X 10.5.8, the fast adaptive uniformization on Windows 7 64bit. [1] In the given tables, '†' means that the computation time of an experiment reached the time limit of 14 hours (applies only to IDD-MC), '−' means that the computation exceeds the available physical memory. The time limit has been enforced by the high number of IDD-MC experiments.

[1]Snoopy relies on wxWidgets. Its current version does not support 64bit Mac OS X.

4.1 Case Studies

We considered the following stochastic models of biochemical networks. For reasons of comparability of the three base case techniques we have chosen $\mathcal{SPN}$ models, which are structurally bounded.

- **ERK:** The RKIP inhibited ERK pathway published in [CSK+03], analysed with the probabilistic model checker PRISM in [CVGO06], discussed as qualitative and continuous Petri nets in [GH06], and as three related Petri net models in [HDG10]. $\mathcal{SPN}_{ERK}$ comprises 11 places and 11 transitions connected by 34 arcs.
- **MAPK:** The mitogen-activated protein kinase published in [HF96]. We consider the model in two different settings concerning the specific reaction rates: MAPK1 – the original settings [HF96], analysed with PRISM in [KNP08], and MAPK2 – the settings published in [LBS00] and used in [GHL07], [HGD08]. $\mathcal{SPN}_{MAPK}$ comprises 22 places and 30 transitions connected by 90 arcs.
- **ANG:** The model of a part of the angiogenetic process published in [NMC+09]. $\mathcal{SPN}_{ANG}$ comprises 39 places and 64 transitions connected by 185 arcs.

The ERK and the MAPK models are parameterisable concerning the initial state. The parameter N specifies a certain number of tokens which are interpreted as molecules or concentration levels. In the ERK pathway, all non-empty places carry initially N tokens; compare Figure 1. In the MAPK models, only the places k, kk and kkk carry N tokens; the remaining places carry 1 token or are empty; see supplementary material on our website `http:www-dssz.informatik.tu-cottbus.de/examples/cmsb2010`.

4.2 Experiments

We designed a couple of computational experiments to compare the three base case techniques – as implemented in our toolkit – in terms of efficiency. For reason of comparability we basically applied transient analysis using the template of Eq. 3.

Experiment 1. In order to instantiate the state property *sp* and time value τ in Eq. 3, we started with evaluating a couple of individual simulation runs to obtain an idea of

Table 2: Computation of the transient probabilities for the formulas (a - d) using exact numercial analysis. The actual computation times ($time_c$) and the total time ($time_t$) including state space generation and initialization are given for different numbers of threads. The number of required matrix-vector multiplications is given in the column k_{max}. In principle, we are able analyse MAPK models beyond $N = 12$, but this would take significantly more time than the established $14h$ limit.

model (formula)	N	k_{max}	1 thread		4 threads		16 threads	
			$time_c$	$time_t$	$time_c$	$time_t$	$time_c$	$time_t$
$ERK(a)$	20	4,013	5m53s	5m58s	1m52s	2m01s	1m01s	2m22
	30	5,881	1h38m52s	1h39m33s	29m42s	30m55s	13m49s	16m34s
	40	7,730	11h54m43s	11h58m03s	3h23m50s	3h29m55s	1h43m00s	1h59m55s
	50	9,569	†	†	†	†	7h59m05s	9h27m36s
$MAPK1(b)$	4	88,166	7m47s	7m48s	3m23s	3m24s	6m06s	6m08s
	8	137,757	†	†	5h56m58s	5h57m16s	2h55m30s	2h 55m51s
	12	187,238	†	†	†	†	†	†
$MAPK2(c)$	4	20,773	1m54s	1m55s	45s	2m25s	1m12s	1m13s
	8	37,530	†	†	1h45m26s	1h45m43s	48m29s	48m50s
	12	67,479	†	†	†	†	†	†
$ANG(d)$	4	11,143	2m31s	2m40s	1m09s	1m19s	1m05s	1m17s

the models' behaviour; the average of the runs is given in Figure 2. Based on these observations we derived the specific formulas which will be used in the following experiments.

(a) For ERK, simulation suggests to have reached a steady state at time point 40. In the steady state the amount of $MEKpp$ seems to remain between 60% and 80% of N, which motivates the formula

$$\mathcal{P}_{=?}[\mathbf{F}_{[40,40]}(MEKpp \geq N \cdot 0.6 \wedge MEKpp \leq N \cdot 0.8)].$$

(b) For MAPK1, the simulation indicates that at time point 40 at least 85% of N (the initial amount of k) has been transformed into kpp, which yields the formula

$$\mathcal{P}_{=?}[\mathbf{F}_{[40,40]}(kpp \geq N \cdot 0.85)].$$

(c) For MAPK2, using the alternative set of rates alters the temporal evolution while preserving the trajectories' shape. This suggests to adjust just the time parameter in (b), and we get

$$\mathcal{P}_{=?}[\mathbf{F}_{[250,250]}(kpp \geq N \cdot 0.85)].$$

(d) For ANG, simulation suggests that at time point 40, there will be around 16 tokens on place $Pip3$, which gives the formula

$$\mathcal{P}_{=?}[\mathbf{F}_{[40,40]}(Pip3 \geq 15 \wedge Pip3 \leq 17)].$$

Experiment 2. We used IDD-MC's CSL model checker to construct the state space (see Table 3) and to check the formulas (a - d). Table 2 presents some figures of the computation time and the total time needed for an exact transient analysis done with 1, 4 and 16 computing threads. Some of the computed probabilities can be found in Table 6.

Multi-threading achieves a significant speed up. Using the 16 logical cores accelerates the computation by factor 7, which is close to the number of physical cores. However, Tables 3 and 2 show that runtime and state space size correlate. In our experiments the runtime increases exponentially with the number of tokens.

Experiment 3. We checked the formulas (a - d) with Snoopy's FAU implementation, using the values $\delta = 10^{-14}$ and $\lambda_{max} = 1600$.

Table 3: Comparison of the size of the complete states space S and the subset constructed by FAU in percent for different initial markings.

model	N	$\|S\|$	*percentage of* $\|S\|$
ERK	10	47,047	50.80%
	20	1,696,618	9.98%
	30	15,721,464	3.07%
	40	79,414,335	1.26%
	50	2.834E+08	$\ll$1%
	60	8.114E+08	$\ll$1%
	100	3.582E+12	$\ll$1%
	250	1.591E+13	–
	500	2.231E+14	–
MAPK1/2	4	99,535	57.29%/100.00%
	8	10,276,461	11.57%/93.81%
	12	210,211,339	2.44%/–
	100	1.125E+16	–
ANG	1	197,414	51.95%

Table 4: Comparison of the memory consumption peak of the exact and approximative numerical analysis for different initial markings.

model	N	$\mid mem \mid$	$\mid mem_{FAU} \mid$
ERK	10	115MB	40MB
	20	190MB	158MB
	30	676MB	374MB
	40	2.68GB	699MB
	50	8.98GB	1.6GB
	60	–	2.5GB
	100	–	6.9GB
MAPK1/2	4	116MB/116MB	98MB/149MB
	8	442MB/442MB	1.46GB/10.7GB
	12	6.45GB/6.45GB	5.49GB/–
ANG	1	154MB	245MB

The computed probabilities are shown in Table 6. The complete state space computed with IDD-MC is compared with the peak of the approximated one created by FAU in

Table 5: Steady state analysis for ERK. The actual computation times ($time_c$) and the total time ($time_t$) including state space generation and initialization are given for different numbers of threads. The number of required iterations is given in the column *iter*.

model	N	iter	1 thread		4 threads		16 threads		result
			$time_c$	$time_t$	$time_c$	$time_t$	$time_c$	$time_t$	
$ERK(e)$	20	485	39s	45s	11s	21s	7s	27s	0.77508
	30	736	11m54s	12m36s	3m15s	4m30s	1m38s	4m23s	0.83297
	40	987	1h29m34s	1h33m07s	25m13s	31m33s	12m06s	14m45s	0.87452
	50	1238	6h41m06s	6h57m05s	2h06m44s	2h37m22s	58m23s	2h27m50s	0.90465
	60	1,489	†	†	7h36m18s	9h18m33s	3h45m35s	8h25m15s	0.92682

Table 3 and 4. In our experiments, the state space considered by FAU increases until a peak is reached, before decreasing again. In all cases, the peak lies between $50\% - 75\%$ of the time interval. FAU may perform entirely different for the same structural model and the same initial state when changing the rate settings, see the figures for MAPK1 and MAPK2. It seems not to be possible to estimate the subset of the state space required by FAU.

Experiment 4. We used the stochastic simulation engine of Snoopy to compute the probabilities of the formulas (a - d). All simulations were done with 16 computing threads. We created 40,000,000 simulation runs per model and formula in order to achieve the desired accuracy of 10^{-5}.

The total runtime of the simulation increases linearly with the number of considered events, see Table 6. This in turn depends on the marking-dependent propensity functions. It follows that the more tokens a given SPN contains, the longer the simulation lasts. The multi-threaded simulation is about 10 times faster then the single-threaded one. It scales nearly linear with the physical cores, but not with the logical ones (no figures given due to the space limitations).

Experiment 5. We used the steady state analysis of IDD-MC to prove our conjecture concerning the ERK pathway, see (a), and we ask for

(e) the steady state probability of being in a state with an amount of $MEKpp$ between 60% and 80% of N

$$\mathcal{S}_{=?}[MEKpp \geq N \cdot 0.6 \wedge MEKpp \leq N \cdot 0.8]\,.$$

Table 5 shows that the probability converges to 1 with an increase of N, as expected. Multi-threading experiments with 1, 4 and 16 threads confirm the nearly linear speed-up in the number of physical cores for the pure probability computation ($time_c$). The actual computation takes less than half of the total time ($time_t$). The remaining time is basically spent for initialization. As more threads are used, as more time this step does cost. The relation between computation and initialization time depends on the number of iterations (column *iter*), required to compute the steady state; the number of iterations is modest for this example.

Discussion. We compared the results of the experiments 2, 3 and 4 concerning runtime and memory consumption taking into account the figures obtained with 16 threads for the exact numerical analysis. The comparison does not sufficiently consider the potential of FAU; the conclusions may change with a parallel FAU implementation.

Table 6 gives an overview on runtimes and computed probabilities. For the simulation, an increasing number of tokens in the initial state results in a modest increase of the runtime. For the numerical analysis methods, the increase is unpredictable; for the experiments considered here it is exponential and obviously model-dependent. For instance, the approximative numerical analysis shows totally different runtime behaviour for MAPK with different rate sets.

Table 4 compares the memory consumption peak of the numerical analysis methods. The memory effort of the simulation is in all cases insignificant. The approximative analysis is again strongly effected by the model settings as can be seen for MAPK1/2. Table 6 shows that an objective judgement of the capabilities of the numerical methods is not easy and seems to depend on the models. FAU outperforms the exact numerical analysis concerning memory consumption and runtime for the ERK model. For the MAPK and ANG models we observe the opposite. In general, the size of the model structure seems to have a negative influence on FAU, caused by its explicit state space representation.

4.3 Summary

We summarize our experience with the three base case techniques of stochastic analysis – as implemented in our toolkit – in efficiency terms.

The efficiency of IDD-MC relies on symbolic data structures. In turn, the efficiency of symbolic representations relies on the compression effect, which is generally not predictable. Constructing the state space and CTL model checking basically requires to juggle efficiently huge sets of integer vectors in the size of P, the number of places (variables). This often works on current computer techniques up to a state space of about 10^{20}. Contrary, the core operation of CSL model checking is a vector-matrix multiplication over real values in the size of the state space S. This works currently up to a state space of some hundred million states.

FAU has been recently introduced; we report about our very first experience in using this technique. The percentage of state space constructed depends on the formula (time frame) and the rates, and is generally not predictable. In the worst case, FAU generates the (almost) complete state space, but at higher costs than the exact numerical analysis.

Simulation is the fallback technique, if numerical methods did not succeed. It takes advantage of the increasing gap between computing power and memory supply, specifically in the era of multicore and cluster computing.

We derive the following general trends from our experimental results.

The **structural model size** effects performance and memory consumption of all methods. But only for the approximative numerical analysis the effect is significant because of its explicit state space representation. As more places and transitions the net contains, as more memory and finally runtime will be consumed. Thus, approximative

Table 6: Comparison of the three base case techniques. π_τ denotes the computed probability of the formulas (a - d) at time point τ. The columns *time* give the total analysis time.

model(formula)	N	Exact		FAU		Simulation	
		π_τ	time	π_τ	time	π_τ	time
ERK(a)	10	0.698561	6s	0.69856	9s	0.698468	3m20s
	50	0.911121	9h27m67s	0.911107	29m18s	0.911109	13m18s
	100	–	–	0.978311	4h10m16s	0.978451	30m43s
	1000	–	–	–	–	0.999999	5h36m41s
MAPK1(b)	4	0.128856	6m08s	0.128852	26m16s	0.128877	14m23s
	8	0.566233	2h45m51s	0.566071	9h7m5s	0.566206	16m49s
	12	†	†	0.599753	36h37m24s	0.600356	20m43s
	50	–	–	–	–	0.80188	46m51s
MAPK2(c)	4	0.202771	1m13s	0.020277	28m46s	0.02028	40m7s
	8	0.413835	48m50s	0.413422	88h20m35s	0.413757	52m42s
	12	†	†	–	–	0.417308	59m46s
	50	–	–	–	–	0.962748	2h15m4s
ANG(d)	1	0.659294	77s	0.659289	12m3s	0.659266	30m26s

numerical analysis is currently suitable for structurally small models only.

Increasing the resolution in the level semantics requires to increase the **token numbers**. Higher token numbers have two consequences. They generally involve higher state-dependent rates, which have a major impact on the runtime of all analyses. As higher the reaction rates, as more steps the system will perform in a given time interval.

Higher token numbers also cause a dramatic increase of the state space. The size of the **state space** effects the numerical analysis methods in terms of memory consumption. A look at Tables 3 and 6 may suggest the conclusion that the state space size also effects the runtime. Indeed, state space size and runtime correlate directly for numerical analyses. For simulation, the correlation between state space size and runtime is in fact caused indirectly by the higher state-dependent rates.

There are several aspects which influence the experimental costs independently of the model. Our experiments prove that **parallelization** reduces substantially the runtime for simulation and exact numerical analysis. Of course, this requires adequate hardware. Without multi-threading our simulation experiments would last 10 times longer, and our exact numerical analysis experiments would last 6 to 7 times longer, whereas the fast approximative numerical analysis is restricted to one thread at the moment. So if one runs the experiments on a machine with different capabilities, the ratio between the individual methods may differ.

If **accuracy** of the results or rare events ($rate \ll 10^{-5}$) are an issue, exact numerical analysis should be the preferred choice, if possible. It always calculates accurate results and never neglects rare events. Accuracy of the results may be a challenge for simulative analysis, where the number of required runs increases exponentially. Then parallelization is a must. In the case of the fast adaptive uniformization, increasing accuracy means to lower the threshold and to manage a larger subset of the state space.

If the analysis goes beyond the computation of transient probabilities (as characterized by Eq. 3) one is currently restricted in our toolkit to exact numerical analysis.

To sum up, there is no clear winner, but simulation can be applied in any conceivable situation. In general the best analysis technique highly depends on the given model (net class, state space size), the required accuracy, the property to be checked and on the available hardware.

We summarize the main lessons learnt by the following user guidelines.

1. Simulation is always a good choice for a very first rough estimate. It allows to get some experience of typical model behaviour in reasonable time.

2. Higher demands in accuracy and rare events call for numerical methods. Simulative analysis can only compete if parallel hardware is exploited.

3. Exact numerical analysis requires a finite state space. Structural boundedness can be decided efficiently. The size of the state space is generally not predictable.

4. It is not predictable, which of the numerical methods performs better for a given model and property.

5. An infinite state space is always manageable with simulation and maybe with approximative numerical analysis, depending on the model size, rates, and time frame.

6. Memory is always the limiting factor for the numerical methods, expected accuracy for simulation.

5. RELATED WORK

There is a multitude of tools on the market offering similar functionalities as our toolkit, with exception of FAU, for which we present the first public tool. An adequate discussion would go far beyond the given space limit; thus we only mention a few. Tools offering exact numerical analysis and stochastic simulation of CTMC include PRISM [HKNP06], SMART [CJMS06], Möbius [GKL+09], and GreatSPN [BBC+09]. Only Möbius provides an adaptive transient solver.

There are several CSL model checking tools, among them PRISM and MRMC [KZH+09]. A thorough comparison of CSL model checking tools can be found in [JKO+08]. PRISM can be compared with IDD-MC due to its CSL capabilities and the states space size manageable by its hybrid engine. However, it can not compete; see [SH09] for a comparison.

Finally, there are quite a number of commonly used simulation tools, e.g. Dizzy [ROB05], Copasi [HSG+06], and StochKit [LCPG08]. Most tools import and export SBML models, as Snoopy does as well, which allows to construct models using other software and facilitates to share models between different software tools. Usually at least one optimized variant of the Gillespie's SSA is available. Besides that, some tools (e.g. Dizzy and StochKit) allow approximative stochastic simulations by means of the tau-leaping algorithm or by further optimized algorithm variants. StochKit provides means for stochastic simulations on computing clusters, which is interesting for larger models. In addition to stochastic simulations, Copasi allows hybrid stochastic/deterministic simulations, which might be useful for analysis of reaction networks that contain both fast and slow reactions. Some tools (e.g., Dizzy, Copasi) also support the feature – as our tool Snoopy does – to translate the stochastic reaction network into a system of ordinary differential equations which can then be solved by built-in numerical integrators.

However, none of these tools supports all three base case techniques of stochastic analysis as discussed in this paper.

6. CONCLUSIONS

We present for the first time a comparative study of the three base case techniques of stochastic analysis. While we have used stochastic Petri nets to represent our models, our comparison is equally valid for any kind of stochastic models. We report of computational experiments lasting in total about 400 hours. There is no clear winner. Our results suggest that the three base case techniques are not competing, but complementing each other. It depends on the model and the type of properties to be analysed which techniques should be favoured.

The reported results may not come as a big surprise for the expert – except of the astonishing sensitivity of FAU. But this paper has been deliberately written from a user's point of view. We discuss all techniques in a detail which should allow the readers to find the most suitable analysis technique for their own settings. For this purpose, we present user guidelines summarizing our main lessons learnt.

In our approach we take advantage of complementary qualitative analysis techniques of Petri nets. The efficiency of our analysis tools relies partly on local information, which we extract from the Petri net structure. Our toolkit consists of three tools, which can also be used separately. We are not aware of any other toolkit, supporting a similar range of analysis techniques

The experience gained during the computational experiments indicate several directions for improvements. We are working on a disc-based and distributed computation of IDD-based CSL model checking, which is expected to manage more than 10^9 states. FAU has the potential for approximative stochastic model checking of time-bounded formulas, which may be accelerated by parallelization techniques. We have also a prototype of an IDD-based FAU implementation supporting $\mathcal{GSPN}$ and intend to incorporate adaptive uniformization into IDD-MC. To accelerate our simulation engines, we are experimenting with Graphics Processing Units (GPU).

Acknowledgements.

We would like to thank Verena Wolf for fruitful discussions and for the support in Snoopy's FAU implementation, and Thomas Meier for implementing Snoopy's parallel simulation engine.

7. REFERENCES

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous-time Markov chains. *ACM Trans. on Computational Logic*, 1(1), 2000.

[BBC+09] S. Baarir, M. Beccuti, D. Cerotti, M. De Pierro, S. Donatelli, and G. Franceschinis. The GreatSPN tool: recent enhancements. *SIGMETRICS Perform. Eval. Rev.*, 36(4):4–9, 2009.

[BH06] A. Bell and B. R. Haverkort. Distributed disk-based algorithms for model checking very large markov chains. *Form. Methods Syst. Des.*, 29(2):177–196, 2006.

[BHHK00] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Proc. CAV 2000*, pages 358–372. LNCS 1855, Springer, 2000.

[CGP01] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2001.

[CJMS06] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. Logical and stochastic modeling with SMART. *Performance Evaluation*, 63(1), 2006.

[CSK+03] K.-H. Cho, S.-Y. Shin, H.-W. Kim, O. Wolkenhauer, B. McFerran, and W. Kolch. Mathematical modeling of the influence of RKIP on the ERK signaling pathway. In *CMSB 2003*, pages 127–141. LNCS 2602, Springer, 2003.

[CVGO06] M. Calder, V. Vyshemirsky, D. Gilbert, and R. Orton. Analysis of signalling pathways using continuous time Markov chains. *Trans. on Computat. Syst. Biol. VI, LNCS/LNBI 4220*, pages 44–67, 2006.

[DHMW09] F. Didier, T. A. Henzinger, M. Mateescu, and V. Wolf. Fast Adaptive Uniformization for the Chemical Master Equation. In *HiBi*, 2009.

[Fra09] A. Franzke. *Charlie 2.0 - a multi-threaded Petri net analyzer*. Diploma Thesis, BTU Cottbus, CS Dep., 2009.

[Ger01] R. German. *Performance analysis of communication systems with non-Markovian stochastic Petri nets*. Wiley, 2001.

[GH06] D. Gilbert and M. Heiner. From Petri nets to differential equations - an integrative approach for biochemical network analysis. In *Proc. ICATPN 2006*, pages 181–200. LNCS 4024, Springer, 2006.

[GHL07] D. Gilbert, M. Heiner, and S. Lehrack. A unifying framework for modelling and analysing biochemical pathways using Petri nets. In *Proc. CMSB 2007*, pages 200–216. LNCS/LNBI 4695, Springer, 2007.

[Gil77] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340 – 2361, December 1977.

[GKL+09] S. Gaonkar, K. Keefe, R. Lamprecht, E. Rozier, P. Kemper, and W. H. Sanders. Performance and dependability modeling with Möbius. *SIGMETRICS Perform. Eval. Rev.*, 36(4):16–21, 2009.

[HDG10] M. Heiner, R. Donaldson, and D. Gilbert. *Petri Nets for Systems Biology, in Iyengar, M.S. (ed.), Symbolic Systems Biology: Theory and Methods*. Jones and Bartlett Publishers, Inc., in Press, 2010.

[HF96] C. Huang and J. Ferrell. Ultrasensitivity in the mitogen-activated protein kinase cascade. *Proc. Natl. Acad. Sci.*, 93:10078–10083, 1996.

[HGD08] M. Heiner, D. Gilbert, and R. Donaldson. Petri nets in systems and synthetic biology. In *SFM*, pages 215–264. LNCS 5016, Springer, 2008.

[HKNP06] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. TACAS 2006*, pages 441–444. Springer, LNCS 3920, 2006.

[HLGM09] M. Heiner, S. Lehrack, D. Gilbert, and W. Marwan. Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments. pages 138–163. LNCS/LNBI 5750, Springer, 2009.

[HMW09] T. Henzinger, M. Mateescu, and V. Wolf. Sliding window abstraction for infinite Markov chains. In *Proc. CAV*, pages 337–352. Springer, LNCS 5643, 2009.

[HSG+06] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. Copasi–a complex pathway simulator. *Bioinformatics*, 22(24):3067–74, Dec 2006.

[HST09] M. Heiner, M. Schwarick, and A. Tovchigrechko. DSSZ-MC - A Tool for Symbolic Analysis of Extended Petri Nets. In *Proc. Petri Nets 2009*, pages 323–332. LNCS 5606, Springer, 2009.

[JKO+08] D. N. Jansen, J.-P. Katoen, M. Oldenkamp, M. Stoelingan, and I. Zapreev. How fast and fat is your probabilistic model checker? In *HVC 2007*, pages 69–85. Springer, LNCS 4899, 2008.

[Kno99] W. J. Knottenbelt. *Parallel Performance Analysis of Large Markov Models*. PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine. University of London., December 1999.

[KNP08] M. Kwiatkowska, G. Norman, and D. Parker. Using probabilistic model checking in systems biology. *ACM SIGMETRICS Performance Evaluation Review*, 35(4):14–21, 2008.

[KZH+09] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. The Ins and Outs of The Probabilistic Model Checker MRMC. In *Quantitative Evaluation of Systems (QEST)*, pages 167–176. IEEE Computer Society, 2009.

[LBS00] A. Levchenko, J. Bruck, and P.W. Sternberg. Scaffold proteins may biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties. *Proc Natl Acad Sci USA*, 97(11):5818–5823, 2000.

[LCPG08] H. Li, Y. Cao, L. Petzold, and D.T. Gillespie. Algorithms and software for stochastic simulation of biochemical reacting systems. *Biotechnol Prog*, 24(1):56–61, 2008.

[MN98] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[MP04] A. Miner and D. Parker. *Validation of Stochastic Systems: A Guide to Current Research*, chapter Symbolic Representations and Analysis of Large Probabilistic Systems, pages 296–338. LNCS 2925. Springer, 2004.

[MPC+06] J. M. McCollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Comput. Biol. Chem.*, 30(1):39–49, 2006.

[NMC+09] L. Napione, D. Manini, F. Cordero, A. Horvath, A. Picco, M. De Pierro, S. Pavan, M. Sereno, A. Veglio, F. Bussolino, and G. Balbo. On the Use of Stochastic Petri Nets in the Analysis of Signal Transduction Pathways for Angiogenesis Process. In *Proc. CMSB 2009*, pages 281–295. LNCS/LNBI 5688, Springer, 2009.

[RMH10] C. Rohr, W. Marwan, and M. Heiner. Snoopy–a unifying Petri net framework to investigate biomolecular networks. *Bioinformatics*, 26(7):974–975, 2010.

[ROB05] S. Ramsey, D. Orrell, and H. Bolouri. Dizzy: stochastic simulation of large-scale genetic regulatory networks. *J Bioinform Comput Biol*, 3(2):437–54, Apr 2005.

[SH09] M. Schwarick and M. Heiner. CSL model checking of biochemical networks with interval decision diagrams. In *Proc. CMSB 2009*, pages 296–312. LNCS/LNBI 5688, Springer, 2009.

[SM08] W. Sandmann and C. Maier. On the statistical accuracy of stochastic simulation algorithms implemented in Dizzy. In *Proc. WCSB 2008*, pages 153–156, 2008.

[SR99] P.H. Starke and S. Roch. *INA - The Integrated Net Analyzer*. www.informatik.hu-berlin.de/∼starke/ina.html, 1999.

[Ste94] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton Univ. Press, 1994.

[Tov08] A. Tovchigrechko. *Model Checking Using Interval Decision Diagrams*. PhD thesis, BTU Cottbus, Dep. of CS, 2008.