BMC Bioinformatics

RESEARCH **Open Access**

Abstraction-based segmental simulation of reaction networks using adaptive memoization

Martin Helfrich[1], Roman Andriushchenko[2], Milan Češka[2*], Jan Křetínský[1,3], Štefan Martiček[2] and David Šafránek[3]

*Correspondence: ceskam@fit.vutbr.cz

[1] Department of Computer Science, Technical University of Munich, Garching b., Munich, Germany
[2] Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic
[3] Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract

Background: Stochastic models are commonly employed in the system and synthetic biology to study the effects of stochastic fluctuations emanating from reactions involving species with low copy-numbers. Many important models feature complex dynamics, involving a state-space explosion, stiffness, and multimodality, that complicate the quantitative analysis needed to understand their stochastic behavior. Direct numerical analysis of such models is typically not feasible and generating many simulation runs that adequately approximate the model's dynamics may take a prohibitively long time.

Results: We propose a new memoization technique that leverages a population-based abstraction and combines previously generated parts of simulations, called *segments*, to generate new simulations more efficiently while preserving the original system's dynamics and its diversity. Our algorithm adapts online to identify the most important abstract states and thus utilizes the available memory efficiently.

Conclusion: We demonstrate that in combination with a novel fully automatic and adaptive hybrid simulation scheme, we can speed up the generation of trajectories significantly and correctly predict the transient behavior of complex stochastic systems.

Keywords: Reaction networks, Stochastic simulation, Population abstraction, Memoization

Background

Stochastic models of chemical reaction networks (CRNs) enable assessing of the cell-to-cell variability in the system's dynamics. This is critical in systems involving low numbers of molecules (e.g., the process of gene regulation [1]). Exact stochastic simulation algorithms (SSA) [2] allow to obtain statistics about the dynamics by sampling the possible trajectories. SSA can be computationally demanding (requiring very large numbers of trajectories) especially in cases the system is stiff or the long-term distribution of the dynamics is not normal.

To address this problem, many approximate simulation approaches have been developed ranging from *τ-leaping* methods [3, 4] and various *hybrid simulation schemes* [5–7] distinguishing slow and fast reactions to recent attempts leveraging the *deep learning* paradigm [8, 9]. Despite these advances, running thousands of simulations to approximate the stochastic behavior of complex systems still may take many hours [7]. This performance gap is also manifested in the existing tools supporting stochastic simulations, such as Stochkit [10], COPASI [11], StochPy [12], iBioSim [13], CERENA [14], or Cain [15]. In addition, there have been proposed several techniques aiming at parallelisation of SSA on multi-core or GPU platforms [16, 17]. While a coarse-grained parallelization is straightforward, an efficient utilization of massively parallel architectures (like modern GPUs) as well as parallelization of a single simulation is challenging.

Alternatively, constructing simplifying abstractions of the model allows us to avoid simulations and apply efficient numerical analysis methods instead [18–21]. However, abstractions may not preserve the complex dynamics, e.g., oscillations in the notorious, tiny (two-species) predator-prey system [22].

In [23], we have uniquely *combined* the simulation methods with the abstraction methods. The key idea is to leverage memoization, a general optimization technique that stores partial results to speed up the consequent computation. In particular, we use SSA but when simulating from a current state we reuse previously generated pieces of runs, called *segments*, that start in "similar enough" states, i.e. states that share many similar trajectories . Thus, rather than spending time simulating a whole new run, we quickly stitch together the already available segments. The segmentation of the trajectories is feasible due to the memorylessness of the underlying stochastic process. On the one hand, a large variety of segments ensures good correspondence to the original probability space of runs. On the other hand, fewer segments result in better memory requirements and speed up due to the massive reuse of the segments. The sweet spot can be achieved and the key is to define the abstract states via *interval abstraction* on the population levels, which groups together states with similar population levels inducing similar behavior. This allows us to achieve significant speedup with respect to state-of-the-art approaches while approximating the dynamics of the analyzed system very precisely.

The proposed abstraction-based segmental simulation has, however, two key *limitations*: (1) The **memory requirements** grow exponentially with the dimension (i.e. number of species) of the model and thus, for large models, it is not feasible to store segments for all abstract states. (2) For some stochastic models, even generating a segment may take prohibitively much **time**, e.g., if there are high-population species where generating each segment requires the simulation of a large number of reactions.

In this paper, we address both limitations by introducing a novel *segmental hybrid simulation method* that significantly improves the performance. In particular, (1) we reduce the memory requirements by introducing a novel *adaptive memoization strategy* that learns on-line the abstract states, in which storing previously simulated segments is beneficial. This allows us to apply segmental simulation to large models (more than 20 species) where the original method fails due to memory requirements. (2) We propose a *hybrid simulation scheme* that allows us to incorporate the memoization approach with other efficient simulation techniques to produce "local" simulations within the segments. In contrast to the existing adaptive hybrid methods [7], we naturally apply the partitioning per abstract state resulting in a *fully automatic and adaptive hybrid scheme* that allows us, for the first time, to effectively combine SSA, τ-leaping and continuous deterministic simulation for a general class of stochastic CRNs. An overall illustration of the proposed approach is shown in Fig. 1.

We evaluate our approach on a wide class of models taken from the literature. We investigate the performance (i.e., speedup against baseline SSA) and accuracy, considering both qualitative and quantitative aspects of the system dynamics. The results show that the new method significantly improves the performance of the existing approximate simulation methods as well as the original method from [23] (for some models over one order of magnitude) while preserving the accuracy.

The adaptive memoization and a novel representation of segment distributions significantly reduce the memory requirements, allowing us to apply the segmental approach to high-dimensional models where the original method failed (e.g., on a 23-dimensional model of *E. coli*). As such, our improved approach further shifts the current capabilities of computational analysis of complex stochastic systems. Most importantly, our method can be seen as an optimization applicable to a general class of stochastic simulation methods, hence can be used to provide speed ups to state-of-the-art techniques available currently or developed in future.

Furthermore, we showcase the use of the proposed approach to generate inputs for the learning framework presented in [24]: we show that the time for input generation can be significantly reduced while marginally sacrificing the precision of the resulting network, if at all. In particular, the speedup achieved by our method means that input generation now takes less time than training and is no longer the bottleneck of the learning process.

Finally, the stored abstract states and their segments approximate the local behavior of the system and thus provide a very useful artifact that can be used for a subsequent

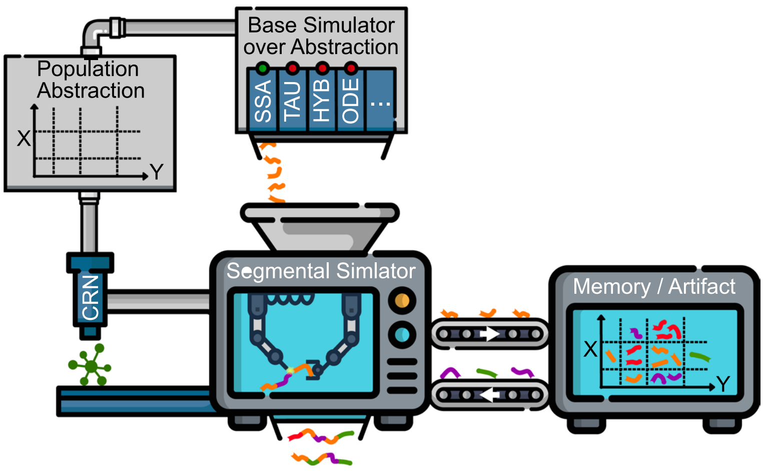

Fig. 1 A high-level scheme of the proposed abstraction-based segmental simulation. The base simulator offers various algorithms that produce short, local simulations (called segments) over a given abstract state as defined by a population-abstraction. In particular, it includes a novel hybrid simulation algorithm that leverages the abstraction to effectively combine SSA, τ-leaping, and ODE-based deterministic simulation. Using an adaptive memoization technique, the segmental simulator determines abstract states to be explored, compactly stores the generated segments to the memory, and combines the segments to generate new simulation of the entire system more efficiently. The segmental simulator also returns the memory filled with the segments providing an artifact speeding up a future analysis of the model

analysis. In particular, it can significantly speedup the simulations from different initial states allowing to analyze the impact of different initial conditions [25].

Methods

Stochastic chemical reaction networks

Stochastic modeling in systems and synthetic biology has become an indispensable tool to quantitatively understand the intrinsically noisy dynamics within living cells involving low copy-number species [1]. We focus on *Chemical Reaction Networks* (CRNs) that provide a versatile and widely used language for modelling and analysis of stochastic biochemical systems [26] as well as for high-level programming of molecular devices [27]. The proposed simulation algorithm, however, can be applied also to alternative population models.

Formally, CRN is a collection of reactions of the form $r_1 + \cdots + r_m \xrightarrow{k} p_1 + \cdots + p_n$, transforming reactants into products under a certain *rate*. The stochastic evolution of a CRN is governed by the Chemical Master Equation (CME) that describes how the probability of the copy-numbers $\mathbf{X}(\mathbf{t}) = (X_1(t), \ldots, X_s(t))_{t\geq 0}$ of each of s species evolves in time [28]. Under the assumption of exponentially distributed waiting times, the process is well-represented by an infinite continuous-time Markov chain (CTMC),[1] where reaction rates are given by a *propensity function* that takes into account the stoichiometry of reactants, their populations, and the rate coefficient k, e.g. using mass-action kinetics.[2]

Computational analysis of CRNs

Population-level abstraction. To make the numerical analysis of complex CRNs computationally feasible, various (adaptive) population abstractions [19, 21] have been proposed to reduce the state-space of the underlying CTMC. The key idea is to divide the *concrete states* of the underlying CTMC into hyperrectangles called *abstract states*. In each abstract state, one concrete state is chosen as its *representative* (typically a central state of the hyperrectangle).

For the abstraction to adequately approximate the dynamics of the original CRN, the states within each abstract state must emit a similar probability space of the trajectories, i.e. similar runs from any two states in the abstract state must have a similar probability. Our proposed abstraction, described in Supplementary Material B, is designed to guarantee this at least at the qualitative level since, when simulating a jump from the abstract state, we ensure that (1) no reaction can be executed that is disabled in some state within the abstract state, and (2) the jump cannot lead to a non-neighboring abstract state. At the quantitative level, it is possible to compute the worst-case probability that the two states in the abstract state exhibit dissimilar runs. Nevertheless, computing this probability for a given abstract state is computationally hard, let alone constructing an abstract state that *a priori* ensures the worst-case probability. Instead, it has been empirically observed that, for adequately large abstract states, the interval abstraction on the population levels groups together states with a similar value of the propensity function, suggesting that states in the abstract state indeed exhibit similar behavior [21, 30]. We

[1] For the formal definition of CRN and the underlying CTMC refer to Supplementary Material A.

[2] Our approach can also be used with Michaelis-Menten equation or Hill kinetics.

specifically use the *exponential abstraction* generated by partitioning each dimension into intervals of exponentially increasing size. This is particularly important for CRNs where the population in one dimension varies significantly in time. Figure 2 (right) shows an exponential abstraction with a growth factor 1.5.

Simulation of CRNs. The widely used Gillespie's stochastic simulation algorithm (SSA) [31] produces statistically correct trajectories of the given CRN, i.e., sampled according to the underlying CTMC. SSA repeatedly applies one reaction at a time while keeping track of the elapsed time. This can be computationally demanding if the number of reactions per trajectory is large (typically in stiff systems with high differences in rates). To

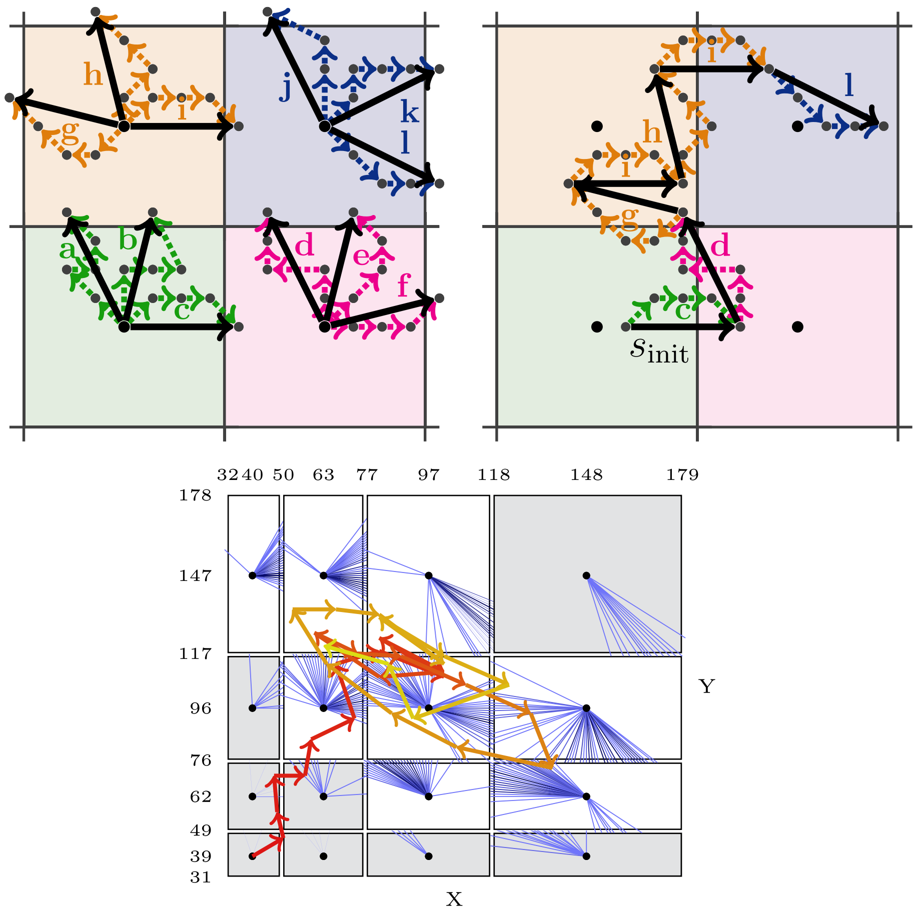

Fig. 2 (Top left) The four squares illustrate four neighbouring abstract states. In each abstract state, 3 segments starting in their respective centers are shown. Each segment represents a sequence of reactions drawn as dotted arrows. The difference between the starting point and the endpoint of the segment is called a *summary* and is drawn as unbroken black line. (Top right) Applying the segments *c*, *d*, *g*, *i*, *h*, *i* and *l* to the initial state s_{init} forms a possible segmental simulation. (bottom) Visualization of the state-space for the two-species Brusselator model [29] with reactions $\emptyset \xrightarrow{100} X, X \xrightarrow{1} \emptyset, X \xrightarrow{1} Y$, and $2X + Y \xrightarrow{1E-4} 3X$ that is partitioned into 16 abstract states. The representative of each abstract state is drawn as black circle. The segment distribution starting at each representative is approximated with at most k=100 summaries drawn as blue lines leading to the last state of the segment. A segmental simulation starting in (40, 39) is visualized as continuous path of summaries that changes its color according to the elapsed time. We observe the typical behavior of the Brusselator model: An evolution in the direction of the state (100, 100) followed by oscillation around this point. As a result of the lazy approach, there are fewer than k=100 saved summaries for the rarely visited outer abstract states. Abstract states close to the center are visited frequently making them more efficient for adaptive segmental simulation (see Sect. 3.2)

reduce the simulation time, several approximate simulation schemes have been developed, in particular, *τ-leaping* methods [3, 4], where all reaction within the next τ seconds are sampled at once, and various *hybrid simulation schemes* [5–7], where large copy-numbers are treated as continuous and deterministic. We adapt these schemes and integrate them into the abstraction-based segmental simulation (see Sect. 3.3).

Abstraction-based segmental simulation

The key idea of our abstraction-based segmental simulation [23] is to precompute, for each abstract state s, a list of k randomly chosen short trajectories, called *segments*. Each segment starts at the representative of s, a concrete state in its center. Recall our assumption that all concrete states in s emit a similar probability space of trajectories and thus we can use the k segments starting at the representative to approximate the local behavior. Figure 2 (top left) illustrates the situation for $k{=}3$ precomputed segments. Each segment is terminated when it leaves the abstract state, since the copy number of at least one species can at this point change significantly, possibly inducing significantly different behavior.

In a *segmental simulation*, we sample a segment for the current representative and apply the relative effect of the segment to the current concrete state. In contrast to sampling a segment for the current concrete state, this allows us to reuse the segments within the abstract state. Figure 2 (top right) illustrates the segmental simulation. In the initial state, the segment c was randomly chosen among the segments a, b, c belonging to the same abstract state. After applying the effect of the segment c, the system moves to the next abstract state where we sample from the segments d, e, f, and so on. As illustrated in the top left abstract state, applying a segment might not change the current abstract state or might do so only temporarily (see the segments l and h, respectively).

Algorithm

In this section, we first describe a general scheme for the simulation algorithm leveraging the segments defined over the population abstraction. The scheme extends the original algorithm we presented in [23]. In subsequent subsections, we present the key improvements, (i) adaptive and approximate memoization strategy and (ii) abstraction-based hybrid simulation, together allowing us to successfully apply our approach to high-dimensional models with high-population species. Finally, we discuss the trade-offs between the efficiency and precision of the simulation.

Segmental simulation algorithm

Precomputing the segments for all abstract states is typically not feasible due to the computational overhead as well as due to restrictions on the available memory. Therefore, we generate the segments on the fly, i.e., we generate and store each segment the first time it is needed. The number of segments generated for the given abstract state depends on the frequency the state is visited: We generate more segments for frequently visited states where the segments are reused many times and thus we improve the accuracy without much overhead. Note that segments can be reused already for a single simulation if that simulation visits the same abstract state multiple times.

Alg. 1 summarizes the general scheme for the abstraction-based segmental simulation. It reuses and fills the *memory* that potentially already contains segments for some abstract states generated in earlier simulations. As we simulate, we always compute the current abstract state *a* (L. 4) and randomly choose which of the *k* segments for *a* we want to reuse (L. 6). In case the segment was already computed, we load it from memory (L. 8). Otherwise, we generate a new segment starting at *a*'s representative *r* (L. 11) and store it. Instead of reusing segments, there is also the option to evolve the system using SSA effectively not using memoization (L. 15). As we discuss in Sect. 3.2, this typically happens if the memory was full.

Storing and applying the whole segment (i.e. the sequence of reactions) is memory- and time-intensive. Therefore, we represent each segment with a single "transition", called a *summary*. The summary captures the overall effect on the state, namely the difference between the starting state and the end state as well as the time the sequence of reactions took. Solid black arrows in Fig. 2 (top left) illustrate the state-component of the summaries for the generated segments. The segmental simulations are formed by the summaries (instead of segments) as illustrated in Fig. 2 (top right). Alg. 1 applies the segment's summary in L. 17.

Algorithm 1 Abstraction-Based Segmental Simulation

```
Require: $memory$ (mapping from abstract state to list of segments; can contain data
    from previous simulations)
1:  function SIMULATE($state$, $time_{\text{end}}$)
2:      $time := 0$
3:      while $time < time_{\text{end}}$ do
4:          $a :=$ ABSTRACTSTATE($state$)
5:          if ACTIVE($a$) then
6:              $n :=$ RANDOM($\{1, .., k\}$)                      ▷ Choose segment
7:              if $n \leq |memory(a)|$ then                       ▷ Already exists?
8:                  $segment := memory(a)$.GET($n$)                  ▷ Reuse
9:              else
10:                 $r :=$ REPRESENTATIVE($a$)                      ▷ Lazy generation
11:                 $segment :=$ NEWSEGMENTFROM($r$)
12:                 $memory(a)$.ADD($segment$)
13:             end if
14:         else                                            ▷ inactive (see Sec. 3.2)
15:             $segment :=$ NEWSEGMENTFROM($state$)
16:         end if
17:         $state := state + segment.\Delta_{\text{state}}$                ▷ Apply segment
18:         $time := time + segment.\Delta_{\text{time}}$
19:     end while
20:     return $(state, time)$
21: end function
```

Improving memory consumption

For small models (i.e., up to 5 species), it is feasible to store segments for all abstract states. However, for larger systems, the required memory quickly becomes a problem, as the number of abstract states and therefore the number of segments we need to store

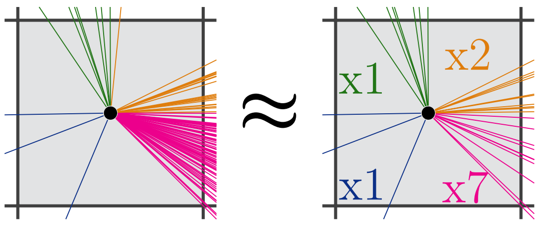

Fig. 3 (left) Approximation of a segment distribution made up of 100 summaries. All summaries with the same direction have the same color, e.g., there are 20 orange summaries in direction $(+1, +1)$ or northeast. (right) A very similar but more memory-efficient approximation with 30 summaries. For directions with many summaries, all but 10 random summaries were discarded. In order to keep the distribution similar the weights of those directions are increased. E.g., in the northeast direction the number of summaries was halved but their weight doubled

grows exponentially with the dimension of the system. Consequently, we save on memory in the two following ways.[3]

Adaptive memoization

We dynamically adjust where the memoization should be used and where we rather save on memory. Intuitively, whenever an abstract state is rarely visited, we flag it *inactive* and use standard SSA instead (L. 14). For more frequently visited, *active* states, we use our memoization to gain significant improvements (L. 5). The activity of abstract states is tracked on the fly, reflecting (1) the expected speedup we gain by reusing a segment instead of generating a new one, (2) the frequency with which future simulations visit the abstract state, and (3) the memory that is spent on the abstract state. Specifically, we use a standard method of least-frequently-used cache with dynamic aging (LFU-DA) [32], allowing for good adaptivity reflecting the behavior of the system also in more advanced experimental scenarios where initial conditions or simulation time is not fixed [25].

Segment-distribution approximation

In our original approach [23], we used a constant number of segment k before we start reusing. To better capture the different number of visits of the individual abstract states, we define a memory function $f(x)$ determining the number of generated segments when visiting an abstract state for the x-th time. Further, rather than storing a maximum of $f(x)$ segments in each state, we dynamically reduce their number as follows. Often, we store many segments with a similar effect on the copy-numbers, see Fig. 3 (left). Since their variety affects the variety of the overall behavior only negligibly, we only preserve a few representatives of each behavior (together with the original frequency with which they should be chosen), see Fig. 3 (right).

Improving the performance

To further improve the performance of the segmental simulation, we combine it with approximate simulation methods.

Abstraction-based hybrid simulation

Hybrid simulation [5, 6] distinguishes slow reactions, which are simulated as discrete and stochastic, and faster reactions, which are treated as continuous and deterministic.

[3] We present here only the key ideas, for technical details refer to Supplementary Material C.

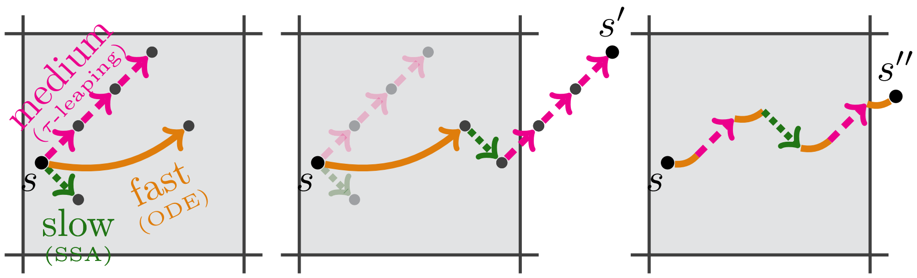

Fig. 4 A hybrid simulation step starting in state s of an abstract state. (left) For every speed, the effects are first calculated separately. (middle) Then, the effects are combined potentially leading out of the abstract state to state s'. (right) As the combined effect was too large, the events are replayed in random order. Between every discrete reaction, there is a continuous evolution. The first state outside of the abstract state is $s'' \neq s'$.

However, this classification cannot be done well a priori for the whole system, since the copy-numbers of species can vary a lot within a single simulation; hence the classification should depend on the actual state [7]. We propose a new, fully automatic, and adaptive hybrid simulation approach leveraging population abstraction that seamlessly combines with segmental simulation. Specifically, for each abstract state separately, we distinguish (i) slow reactions, to be treated by SSA, (ii) fast reactions, to be treated by ODE, and additionally (iii) medium reactions, to be treated by τ-leaping.[4] First, all three reaction types are considered separately; then, their effect is combined. If the combined effect is reaching too far into other abstract states, we need to increase the precision (preferably without re-sampling). To that end, we consider individual parts of each of the blocks (i)–(iii) and interleave them randomly, providing their mix that is more adequate within this abstract state, see Fig. 4. Upon reaching the border of the abstract state, we interrupt this sequence and continue according to the dynamics of the neighbouring abstract state. For technical details, see Supplementary Material D.

Trade-offs

Our segmental simulation faces a trade off between (i) the memory requirements, (ii) the precision of the results, and (iii) the speed up of the computation time. If memory is of no concern, e.g, because the model is small, there is just a trade off between speed and precision: storing more segments increases precision but costs time, while storing less segments results in higher speed with less precision. If memory is a limiting factor, we observe that additional memory can be utilized for increasing either precision or speed up or both in some ratio. We manage the trade off between memory, speed up and precision *locally*, i.e., for each abstract state we decide whether to generate more segments to increase the precision or to reuse previous one, and *dynamically* during runtime, e.g., if an abstract state is visited often, we allocate more memory to increase its precision according to a priori settings.

Additionally, we face a more global decision: in which abstract states to utilize the memory. When we reach the user-defined memory limit, we are forced to free a portion of the used memory. This involves determining the abstract states in which we profit

[4] The classification of the reactions in described in Supplementary Material D.

Table 1 Summary of the benchmarks including their key characteristics

Benchmark	Size	Description
Predator-Prey (PP)	2 species 3 reactions	An oscillating system with a very sensitive die-out behavior that is notoriously hard for approximation methods
Viral (VI)	2 species 6 reactions	A very stiff system with a bimodal behavior including a fast extinction that is hard to precisely quantify using approximation methods
Toggle Switch (TS)	6 species 14 reactions	A classical multi-scale system (low-population species control high-population species) with a switching behavior. Preserving the switching rate is hard for approximate methods
Repressilator (RP)	6 species 15 reactions	A classical genetic network model with a feedback loop exhibiting an oscillation with large variations in the species population. Preserving the oscillating periods is hard for approximate methods
Composition of TS and RP (TSxRP)	12 species 29 reactions	Combines the challenges in the TS and RP models
E. coli (EC)	23 species 22 reactions	A large system with a complex multi-step species conversion. Preserving the precise conversion times is hard for approximate methods

from the memory the least and deciding on how to reduce memory usage there. E.g, if an abstract state uses a lot of memory without causing a comparatively large speed up, we treat it as temporarily "inactive" so that memoization is locally switched off here. See Supplementary material C1 for the trade off strategy and E for the list of all the hyperparameters that affect this trade off. Finally, we inspect the trade off experimentally in Fig. 6.

Results

We focus on the following key aspects of the proposed simulation algorithm allowing us to assess its practical benefits: (1) accuracy evaluated by a general error metric as well as a model-specific characterization of the dynamics, (2) speedup achieved over our reference implementations of SSA, (3) memory footprint, and (4) comparison with the state-of-the-art simulation methods.

Benchmarks and Experimental setting

We use the following models from the literature: Viral Infection (VI) [33], Repressilator (RP) [7], Toggle Switch (TS) [7] and Predator-Prey (PP, a.k.a. Lotka-Volterra) [31]. These models are challenging due to stochasticity, multi-scale species populations, stiffness, and/or complicated transient behavior. Therefore, they are commonly used to evaluate advanced numerical as well as simulation methods. Additionally, we consider two high-dimensional models: (i) a synthetic model with 12 species, called TSxRP, where the models TS and RP run in parallel and (ii) *E. coli* Lac expression(EC) [34] with 23 species that models a complex multi-step conversion.[5] These two models represent a true challenge for simulation methods. Table 1 summarizes the key characteristics and challenges of the particular models. The definition of all models can be found in Supplementary Material I.

The proposed method is implemented within an open-source tool SAQuaiA, available at https://sequaia.model.in.tum.de/ that extends our existing tool SeQuaiA [35]. SAQuaiA includes our own competitive implementation of the direct SSA method used

[5] The initial population of Ribosomes converts into LacY and LacZ with a 1:3 split. Then, LacZ continuously produces lactose which is converted into product molecules with the help of LacZ.

as a baseline that achieves between 1×10^6 and 7×10^6 reactions per second depending on the model. Further, we use our implementation of the τ-leaping (TAU) and hybrid simulation (HYB) with the adaptive species partitioning utilizing the abstract state space. As discussed in Sect. 4.4, for some models, our implementations including the SSA baseline lag behind the performance of the existing highly optimized tools such as Stochkit [10] or Cain [15]. However, the results presented below clearly demonstrate the benefits of the proposed approach with respect to the exact SSA as well as existing approximate methods.

Segmental simulation (SEG) has multiple hyper-parameters, e.g., the population-growth factor for population levels and the number of segments that are reused per abstract state. Choosing appropriate hyper-parameters can greatly improve the accuracy and performance of SEG. In this work, we choose to evaluate our method with a set of default parameters described in Supplementary Material E that reliably produce good results for most models. The performance and accuracy of our approach based on reusing simulations may vary over multiple runs of the same experiments and thus we report the average results from 10 experiments. All experiments were run on a machine with a 2.20GHz CPU and 16 GB of RAM and can be reproduced with the artifact available at http://go.tum.de/065569.

Accuracy

From the theoretical perspective, the imprecision in SEG stems from (i) reusing a *limited number of segments*, reducing the variation in the behavior and possibly missing rarer behaviors, and (ii) discrepancy between simulating from the actual state and its *representative in the abstraction*. While both bias the probability measure on the runs, no invalid simulations are ever produced.[6] Now we focus on the practically achieved accuracy.

Experimental accuracy evaluation

We start with a detailed accuracy evaluation of our fastest approach, namely hybrid SEG, and later discuss the accuracy of all other methods. For every model, we first visually compare an SSA and a hybrid SEG simulation (see Fig. 5) and discuss how the key characterizations of the model's dynamics are qualitatively preserved. Afterwards, we consider a model-specific metric that quantitatively captures this characterization and reports how the metric is preserved. Finally, we use a general metric that compares transient distributions approximated by generating a large number of simulations with a particular simulation method. Specifically, we report the *total-level Earth Mover's distance* (EMD) between the transient distributions for our method and SSA (see Table 2).[7] It measures how similar the distributions are by combining the Earth Mover's distance (or first Wasserstein distance) of the level distribution in each dimension.[8] As total-level EMD values are difficult to interpret without context, we additionally report the total-level EMD between two different transient distributions computed with SSA. Intuitively,

[6] For a more detailed discussion about the theoretical error refer Supplementary Material F.

[7] We were not able to perform a quantitative evaluation of the accuracy for TSxTP as collecting the required statistics from SSA is computationally unfeasible (a single SSA run takes over 50 s).

[8] For a formal definition refer to Supplementary Material G.

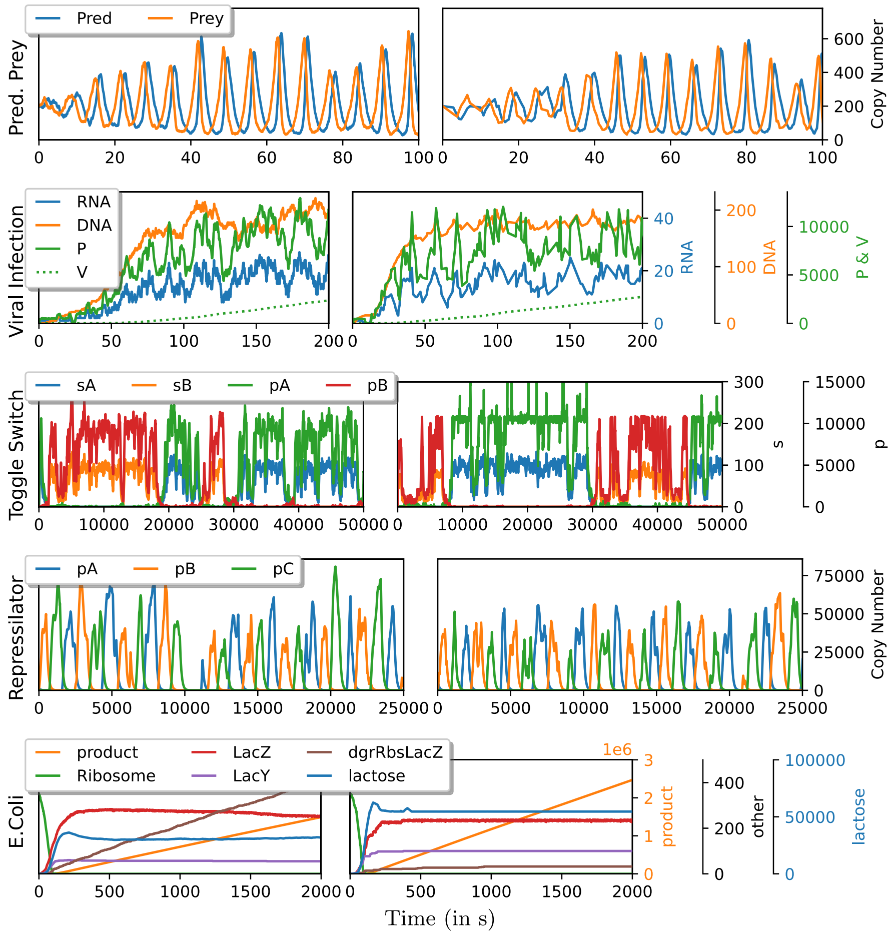

Fig. 5 Comparison of the concrete simulations produced by SSA simulations (left) and hybrid SEG (right) for different models. Similar figures for TSxRP as well as for the SSA-based version of SEG can be found in Supplementary Material H

Table 2 Accuracy of transient analysis with 10.000 simulations measured as total-level Earth Mover's distance w.r.t. SSA

Model	control	TAU	HYB	SEG using	
(dim.)	SSA			SSA	HYB
PP (2)	0.12	2.5	2.5	6.2	3.1
VI (4)	0.053	0.44	0.21	1.8	2.0
TS (6)	0.16	0.93	0.16	15	12
RP (6)	0.28	2.5	14	6.4	6.5
EC (23)	0.091	0.57*	0.20*	9.2	8.5

(*) We report the accuracy of TAU and HYB in EC for 2.591 and 5.600 simulations respectively because of timeouts

this is the total-level EMD we would expect for an accurate method. We also interpret the EMD in the context of the model-specific metrics (Fig. 6).

- *PP:* SEG shows the expected anti-cyclic oscillation between both species (see 1st plot in Fig. 5). The PREDATOR oscillation periods match closely: mean and standard deviation for SSA are 7.06*s* and 1.07*s*, while for SEG they are 6.96*s* and 1.34*s*. Similarly, for the PREDATOR peak heights: mean and standard deviation for SSA are 503 and 244 while SEG predicts 452 and 222. The die-out behavior of the model is notoriously hard to preserve for abstraction-based methods. While SEG correctly predicts that it is equally likely for either species to die out first, it underestimates the number of runs that die in the first 200s: 43% for SEG to the 71% in SSA. This explains the rather large EMDs and is expected as the emerging die-out behavior is very sensitive to any perturbations.
- *VI:* In a qualitative comparison, we find that SEG correctly predicts the typical behavior of the system: Initially, there is a small chance of 20% that the system dies out. Otherwise, all species grow until they reach a level that they oscillate around (see 2nd plot in Fig. 5). To quantify the accuracy, we compare the RNA distribution at t=200s: SSA predicts a bi-modal distribution where 20% die-out and the other cases are distributed normally with mean 17.3 and standard deviation 4.9. SEG matches this very closely with a 20% death percentage and a normal distribution around 16.7 with a standard deviation of 5.3. We find that the estimation of the death percentage is less stable when compared to SSA. This variance accounts for most of the total EMD of 2.0.
- *TS:* A qualitative comparison of simulation plots shows that both SSA and SEG show the expected switching of periods of A-species dominance and B-species dominance (see 3rd plot in Fig. 5). While SSA predicts 1.94 switches per simulation, SEG predicts only 1.03. In PB-species, we observe the effect of the abstraction-based rounding in SEG—the copy-numbers around 11000 is too dominant in SEG. As a result, the total EMD is rather large, even for 6 dimensions. While the accuracy of SEG for this model is good enough to show the general behavior of the system, it can not predict all quantitative properties precisely.
- *RP:* Visually both SSA and SEG produce the expected alternating peaks of the three P -species (see 4th plot in Fig. 5). SEG correctly predicts the oscillation periods (mean: 2520*s*, s.d. 324*s*) when compared to SSA (mean: 2580*s*, s.d. 244*s*). The peak heights (mean: 4.5*E*4, s.d. 1.5*E*4) also match with SSA (mean: 5.0*E*4, s.d. 1.4*E*4). The total EMD is low for a model with 6 species and high copy numbers. This supports that the accuracy of SEG for this model is very good.
- *EC:* Both simulation methods show an initial conversion of RIBOSOMES via multiple steps into LACY and LACZ, followed by the creation of LACTOSE and PRODUCT (see 5th plot in Fig. 5). Whereas LACTOSE levels off, the PRODUCT population grows linearly. The average time when all RIBOSOMES are converted is 98*s* for SSA, but 190*s* for SEG mostly because the conversion of the last RIBOSOMES is slower. The total EMD of 8.5 for SEG is rather small for a model with 23 dimensions. Most of the inaccuracy (74%) can be attributed to degraded species that are unable to react and thus accumulate. SEG underestimates the degradation rate as can be seen for DGRRBSLACZ. However, SEG's level EMDs for the species LACZ (0.17), LACY (0.15), LACTOSE (0.32), and PRODUCT (0.13) are very small, suggesting very good accuracy for the most interesting species.

Comparing the accuracy of the approximate methods In Table 2 we compare the accuracy using the general error metric. We observe that TAU and HYB usually exhibit a similar accuracy which is typically higher than the accuracy of SEG. In two models (TS and EC), this difference is quite significant because TAU and HYB do not introduce the previously discussed inaccuracies. In the other models, TAU and HYB also evince discrepancies from the true quantitative behavior, e.g., TAU and HYB overestimate the death percentage in PP by 15%. A notable exception is the model RP where HYB exhibits too much determinism and incorrectly predicts the peak of a specific species at t=50000 while SEG correctly predicts that all three species are equally likely to peak. Surprisingly, when we compare the two SEG approaches we find that using hybrid simulation does not considerably impact the accuracy while, as we show later, it significantly reduces the simulation time.

Key conclusions from the accuracy evaluation.

Overall, we find that SEG introduces measurable inaccuracies as it is, indeed, an approximate simulation technique. It can underestimate the likelihood of improbable events, like the switching in TS, or the effect of slow dynamics, like the degradation in EC. Hence, its total-level EMD is considerably larger than that of a control SSA and, in some models, than that of TAU and HYB. However, we find that SEG preserves the key system dynamics (the produced simulations are visually very close to SSA simulations) and, more importantly, it correctly predicts the majority of model-specific quantities. Further, recall that this evaluation was performed with a set of default hyper-parameters, that can be further optimized [23].

Speedup

Recall that our approach is based on re-using segments generated in previous simulation runs. As such, the SEG becomes faster with each subsequent simulation of the system. This can be observed in Task 1 of Fig. 7 that shows the speedup we achieve in the VI model when generating an increasing number of simulations with SEG instead of SSA. The speedup evolution for all models can be found in Supplementary Material H. Typically, SEG does not speed up the very first simulation as the memory starts empty. However, for some models segments can already be reused in the first simulation like in the oscillating TS model where the speedup for the first SSA-based SEG is already 15x. The speedup grows rapidly during the early simulations when segments for the most efficient abstract states are generated and approaches a limit where SEG is reusing as much as possible.

Table 3 shows the speedup factors for different simulation methods when generating 10.000 simulations. On its own, SEG already speeds up the simulation process significantly. For the large EC model with 23 dimensions the speedup is moderate at 5.6x while for other models, like TS, the speedup is 280x. For models where hybrid simulation is effective, like VI, TS, and RP, the speedup can be improved even more by combining SEG with our abstraction-based hybrid approach (further denoted as SEG+HYB). In VI, the speedups of HYB and SEG on their own are respectively 68x and 150x but the combination of both approaches results in a speedup of 3300x.

Table 3 Average run-time of one SSA simulation and the relative speedup w.r.t. SSA when computing 10,000 simulations with other methods

Model	**SSA**	**TAU**	**HYB**	**SEG using**	
				SSA	**HYB**
PP	0.013s	1.1x	1.2x	46x	48x
VI	0.74s	6.0x	68x	150x	3300x
TS	21s	6.4x	12x	280x	310x
RP	8.4s	7.9x	24x	160x	500x
TSxRP	52s	11x	16x	14x	52x
EC	4.9s	0.24x	0.5x	5.6x	4.9x

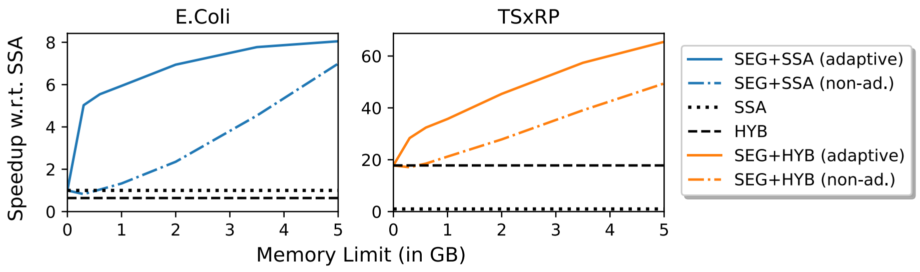

Fig. 6 Speedup achieved by SEG over SSA when generating 2000 simulations for different memory limits. Without adaptive memoization, SEG is less efficient

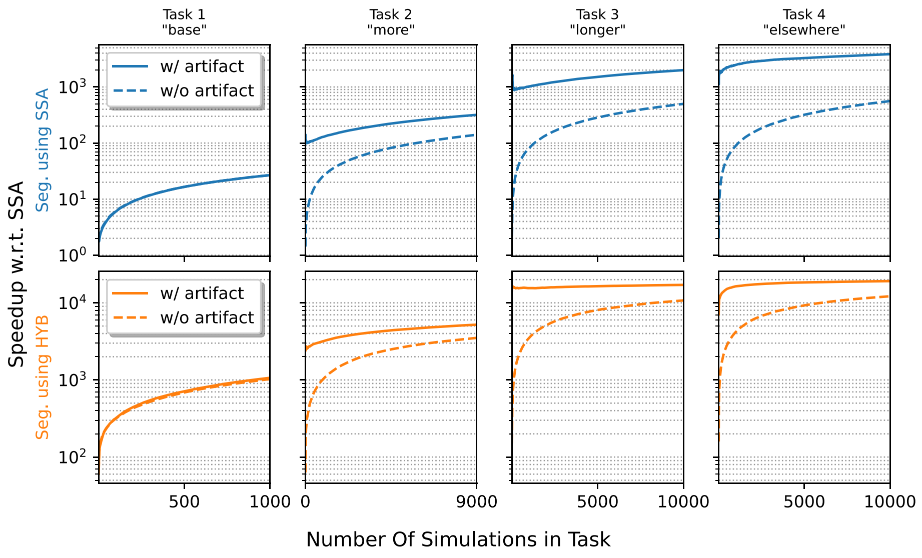

Fig. 7 Speedup per simulation task for a sequence of transient analyses in VI: (1) 1000 simulations, (2) 9000 simulations, (3) 10x longer simulation time, and (4) starting with a different initial state (5 molecules of RNA). The speedup is significantly improved if the memory between tasks is reused (w/ artifact) instead of emptied (w/o artifact)

Memory

As SEG is a memoization approach, we pay for the speedup with increased memory consumption. When generating 10.000 simulations, the segment-distribution approximation reduces the total memory consumption by about 65% to 85% when compared to the original SEG [23]. However, the speed up improves for greater memory limits, as depicted in Fig. 6, where for simplicity the accuracy is constant and the speed up factor is then dependent only on the allowed memory. Even with this reduction, we hit the memory limit of 5GB for large models like EC and TSxRP. However, in contrast to the previous implementation of SEG of [23], adaptive SEG did not run out of memory but reused segments for the most efficient abstract states. Details about the memory usage are given in Supplementary Material H.

Figure 6 shows the speedup for different memory limits. In general, increasing the memory limit results in faster simulations as SEG can reuse segments more frequently. The speedup approaches a model-specific limit where segments are reused in all visited abstract states. Even in some large models, like EC, we can achieve most of the speedup with few but very efficient abstract states. In other models, like TRxRP, the relation of memory to speedup is closer to linear as the efficiencies of most abstract states are similar. Notice that the new adaptive property of our approach is crucial for the analysis of these large models. The simulation runs produced by SEG may change over time as new segments are added to memory. This can render large parts of the memory inefficient. Once the memory is full, the non-adaptive memory cannot be modified resulting in significantly less speedup than the adaptive memory that simply replaces the segments of the inefficient abstract states.

Reusing the segments for different simulation tasks

In the experiments described so far, the memory of SEG starts empty and is filled over time. The filled memory can be understood as an artifact speeding up future SEGs of the same model. This is the case even if future simulations start from a different initial state or end at a different time. Consider the scenario depicted in Fig. 7 where a user performs a series of transient analyses for the VI model: First they compute 1000 simulations (Task 1), then they need 9000 more simulations to estimate the likelihood of a rare event (Task 2), then they run the simulation 10 times longer to make sure the system is stable (Task 3), and finally they modify the initial state to analyze its impact on the system (Task 4). If the memory is emptied between tasks (dashed line), we observe the normal speedup increase per task. However, if the memory is reused between tasks (solid line), then the initial speedup and final speedup per task are larger as less time is spent on generating segments. Note that reusing the memory of older tasks is possible because SEG is adaptive and thus removes ineffective segments. Without adaptive memory, it could happen that all memory is used for ineffective segments resulting in no speedup.

Comparison with state-of-the-art approaches

Comparison with optimized SSA There is a long list of work that focuses on improving the exact stochastic simulation algorithm. Optimizations include a dependency graph to minimize the recalculation of propensities [36] as well as strategies that order reactions by frequency to make the sampling more efficient [37]. The state-of-the-art simulation tool Stochkit [10] implements these techniques and as a result outperforms our SSA implementation on the considered benchmarks by a factor of 6x to 18x—see the second

Table 4 Comparison with StochKit

Model	StochKit SSA's	Our speedup wrt. StochKit SSA	
	speedup wrt. our SSA	SEG	SEG+HYB
VI	6.4x	23.3x	512.8x
TS	7.5x	37.5x	41.5x
RP	8.5x	18.7x	58.5x
TSxRP	12.9x	1.1x	4.0x
EC	18.1x	0.3x	0.3x

The second column shows the speedup achieved by the optimized SSA implementation in StochKit comparing to our SSA. The last two columns show the speedup achieved by SEG SSA and SEG+HYB, respectively, comparing to SSA in StochKit. Despite the slower underlying SSA, our implementation of SEG significantly improves the state of the art on most models. Note that the speedups roughly correspond to the speedups SEG achieves with respect to our SSA (see Table 3) decreased by the factor reflecting the better SSA baseline (the second column)

column in Table 4.[9] Observe that the impact of the optimizations increases with the complexity of the models. Despite the slower base simulator, our SEG still significantly outperforms the SSA of Stochkit on a majority of the models. The last two columns of the table show the speedup achieved by our SEG using our SSA and HYB, respectively. Note that Stochkit implements a highly optimized version of TAU. On the considered models, it reduces the simulation time about a factor 4 or 5 while introducing a similar approximation error as our implementation of TAU. The exception is the EC model where it introduces a 3-times slowdown (similar as our TAU) demonstrating that this model is hard for approximate simulation techniques.

Since the proposed segmental approach is orthogonal to the SSA optimization techniques, the speed of our segmental simulation can be improved by implementing a more efficient base simulation. However, notice that while an optimized SSA implementation can make simulations significantly more efficient, segmental simulation can speed them up by multiple orders of magnitude. In the presentation of our new segmental simulation, we have introduced the tool SAQuaiA with our own SSA implementation, rather than integrated it into established simulation tools like StochKit. This in turn allows us to demonstrate the feasibility of our technique without encountering the technical complexities of adapting segmental simulation to a framework originally not designed to accommodate a memoization approach. One of the key technical challenges lies in the fundamental difference between our approach, which emphasizes efficient reuse of multiple simulations of the same system, and the typical operation of StochKit, which primarily deals with single, independent simulations. However, we certainly acknowledge that the ultimate goal, beyond the scope of this paper, is an integration of segmental simulation into the code bases of all widely-used tool-kits like StochKit and others, resulting in broader accessibility and applicability.

Comparison with advanced simulation methods

To our best knowledge, there is no available implementation of slow-scale stochastic simulations [5, 38, 39] and thus a fair comparison is problematic. Therefore, we report a comparison with the results presented in [7] describing the state-of-the-art hybrid

[9] We were not able to conduct a fair comparison on the PP model, since Stochkit requires a modification of the model preventing bursts of the PREY population when PREDATORS die out.

simulation method. The run times of the adaptive hybrid simulation (100,000 runs) on RP and TS as reported in [7] (Fig. 1) are 3.0 hours for RP and 1.1 days for TS. Respectively, this is a speedup of 77x and 43x over their SSA baseline implementation. Our SEG approach with abstraction-based hybrid simulations only needs 0.47 hours for RP and 1.9 hours for TS. This is a speedup of 500x and 310x over our SSA baseline implementation, i.e. our computational gain is around one order of magnitude larger. For more details refer to Supplementary Material H.

Comparison with approximate formulations of Chemical Master Equation

The chemical master equation is considered an accurate description of CRNs. However, solving CMEs directly is considered computationally intensive. To that end, various approximation techniques have been developed for models working with large populations of species. A classical approach builds on a deterministic approximation derived as a set of ordinary differential equations (ODEs) describing the expected value as well as some higher moments of the total number of molecules [40]. Another approach is to employ a continuous-state stochastic process to approximate the discrete-state stochastic process described by the CME. The Linear Noise Approximation (LNA) is a Gaussian process which has been derived as an approximation of the CME [41, 42] and describes the time evolution of expectation and variance of the species in terms of ODEs. Alternatively, the Langevin equation [43], a specific form of stochastic differential equations (SDEs), can be derived. Using these approximate formulations is, however, very challenging in the case of systems with low-population species as they typically cannot adequately capture the underlying stochasticity.

Inspired by hybrid simulation methods [5, 6], our abstraction framework addresses these challenges. For low-population states, it employs the precise Gillespie algorithm. However, for high-concentration states, it offers the flexibility to use various methods such as τ-leaping, ODEs, or potentially the Langevin equation. This strategy allows us to devise a hybrid method that effectively capitalizes on the strengths of these individual approaches.

Comparison with abstraction techniques

The work of [21] introduces an abstraction-based approximation technique that aims to provide precise approximations of the transient probabilities along with a formal error estimate. It was demonstrated that, for small CTMCs with up to 1M states, the method can achieve significant speedups (up to 30 times) while providing approximations that provably differ from the true distribution with 0.1% error; here error represents (the upper bound on) the total probability lost by truncation or misplaced by abstraction, and can serve as an absolute state-wise error. On the other hand, CRNs with a large set of reachable states remain out of reach for this technique unless abstraction and truncation are applied aggressively, in which case the approximation becomes too imprecise. For example, the aforementioned TS benchmark contains roughly 10^{17} states reachable in 100s and FAU (without abstraction) fails to compute the transient distribution under 6 hours. 30-fold speedup using abstraction from [21] would result in a runtime of at least several minutes. Meanwhile, the segmental simulation can generate tens of thousands of trajectories for this benchmark in a few seconds. Although the resulting distribution will most likely not fall within 0.1% error, it can still provide an adequately precise estimate of the true distribution, which suffices in many practical applications.

Recently, a population-level abstraction with overlapping intervals and an interval DTMC semantics (the timing of the reactions is ignored, only the probability is to be preserved) has been proposed [44]. Although the motivation for using such abstraction is different (this work studies how abstract interpretation can be used to design more precise discretization methods), it would be very interesting to consider the overlapping abstraction in the context of the segmental simulation. However, from the practical point of view, the overlapping abstraction is evaluated only on very small CRNs with two species and two reactions.

Comparison with the deep learning approaches

A *deep learning* paradigm has been recently introduced to further shift the scalability of CRN analysis. A neural-based generator, in the form of a Generative Adversarial Network (GAN), can be learned from a set of stochastic simulations of the given CRN [8]. The generator is able to efficiently produces trajectories having a similar distribution as the trajectories in the original CRN. To compare the performance and accuracy, we consider a simpler variant of our TS model as in [8]. The segmental simulation achieves a similar performance and accuracy as the neural-based generator (for more details, see [23]). In the recent work [45], the GAN approach was improved using a conditional score-based diffusion model that typically offers a better accuracy and stability of the solution, but generates the new samples slower. To further automatize the design of the neural-based generators, a learning framework for the neural network architecture and the transition kernel has been proposed in [46]. In [9], the authors go even further and learn from the simulations an estimator of a require statistic (e.g. the expectation of a species population at the given time) over the original CRN. All these approaches share the principal limitation: they typically require a large number of simulations of the original CRN to learn the generator/estimator accurately. This indeed impractical for complex CRNs.

A different approach (called Nessie) trying to mitigate this limitation has been recently proposed in [24]. Instead of learning the transition kernels or estimators of the statistics, Nessie directly learns marginal probability distributions, in the form of mixtures of negative binomials, that approximate the solution of the given CRN at different times and rate parameters. The experiments show that in many cases only a moderate number of simulations (e.g. 500 or 1000) per training point (representing the given simulation time and CRN parameters) is sufficient to accurately approximate the target distribution. On the other hand, if the CRN has a complicated temporal behavior, e.g. an oscillation, this approach struggles to predict the behavior adequately.

To gain another perspective on the versatility of our method, we compare the outcome of the learning process in Nessie for training inputs (i.e. the simulations) obtained using the standard SSA and using our SEG+HYB approach. Namely, for the TS model with parameterised rates, every training point (specific time and rate values) is associated with a transient distribution obtained by either (i) 500 SSA simulations (SSH-500), (ii) 500 SEG+HYB simulations (SEG+HYB-500) or (iii) 10,000 SEG+HYB simulations (SEG+HYB-10k). To compare the transient distributions predicted by Nessie with the reference distribution (obtained by 10,000 SSA simulations), we use two metrics as in [24]: mean Kullback-Leibler divergence (KL) and mean Hellinger distance (HD). Generating 4,000,000 training points (40,000 parameter valuations for times 1,..., 100 s) for

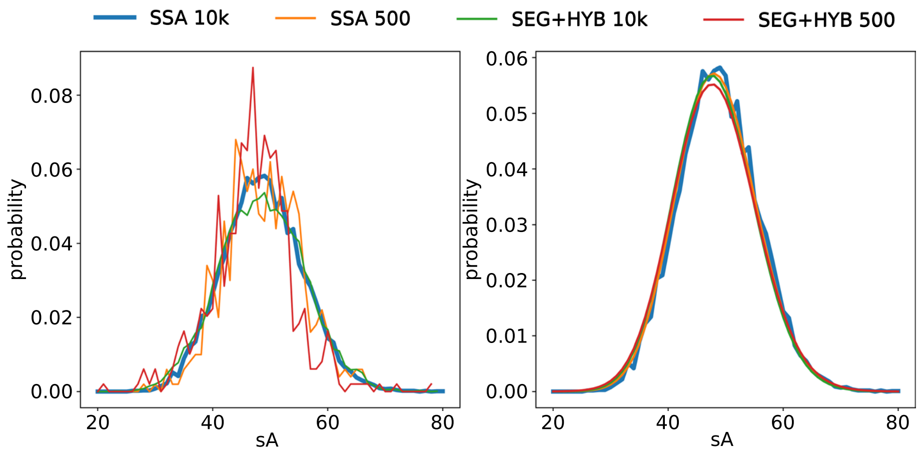

Fig. 8 Left: Transient distributions (Nessie inputs) of the species sA in the TS model (for an arbitrary training point) obtained using a different number of SSA and SEG+HYB simulations, respectively; 10,000 SSA simulations (bold blue) provide an adequate approximation of the true distribution. Right: The distributions predicted by Nessie [24] trained on different inputs

this model using SSA-500 would take about 8 days on a common workstation with 16 CPUs;[10] the training would take about 5 hours (16 CPUs) and the resulting neural network achieves KL=0.065 and HD=0.059. Using SEG+HYB-10k reduces the time for input generation 10-fold without affecting the training time, and the resulting network predicts distributions with precision comparable to that of SSA-500: KL=0.050 (better) and HD=0.069 (worse). Finally, using SEG+HYB-500 reduces the time for input generation even further (50-fold compared to SSA-500) and the precision of the resulting neural network is only slightly worse: KL=0.080 and HD=0.089. We conclude that the framework presented in [24] can benefit greatly from the proposed segmental simulations by significantly cutting down the time for input generation while slightly sacrificing the precision, if at all. In particular, the 50-fold speedup enabled by SEG+HYB-500 means that input generation now takes less time than training and is thus no longer the bottleneck of the learning process.

Figure 8 illustrates the comparison of the inputs and outputs of the neural network for a randomly selected training point. As expected, SEG+HYB-500 produces the least accurate estimate of transient distributions, while SEG+HYB-10k produces the most accurate ones (compared to the reference distribution in bold blue). Nonetheless, for any of the three inputs, Nessie is capable of producing quite precise predictions, where the outputs of the two networks trained using SSA-500 and SEG+HYB-10k coalesce. Additional data about the setup of this experiment are documented in Supplementary Material H.

Conclusion

In this work, we present a new approach for simulation-based approximate analysis of stochastic CRNs that feature complex dynamics making their quantitative analysis computationally challenging. Our approach builds on a novel memoization technique that

[10] We ran the experiments on a computational cluster.

leverages a population-based abstraction to generate and store segments corresponding to a local behavior of the given CRN. The criteria driving the abstraction include the frequency of visiting the states, the speedup achieved by reusing segments in this state, and the memory requirements. The segments are then effectively combined to speedup the simulation of the CRN while preserving the original system dynamics and its diversity. Note that adaptive memoization treats the challenging *exploration vs. exploitation problem* since it is, in general, already infeasible to store all abstract states of a large system. The outcome of our approach is not only the set of simulations, but also the stored segments that compactly represent the local dynamics induced by the particular abstract states. The experiments over a representative set of benchmarks demonstrate that our approach is able to significantly accelerate quantitative analysis of CRNs while preserving the key qualitative as well as quantitative characteristics of the precise simulations (see Tables 2 and 3 reporting the main experimental evaluation results).

A key advantage of the proposed methods is their orthogonality to the existing optimization and approximation techniques developed for stochastic simulations. In particular, the state-of-the-art frameworks like StochKit can greatly benefit from integrating our approach. Since our technique can theoretically embed any efficient simulation technique for computing segments, it can significantly speed up the practical simulation tasks with efficient SSA or hybrid techniques. Our approach is especially advantageous in cases that require repeated simulation from various initial system states. We leave for future work utilization of our technique for *in silico* design of oscillation generators [47] or fine-tuning of strand-displacement reaction dynamics [48, 49].

Another advantage of our approach is the potential for efficient task decomposition. More specifically, it can be beneficial for a fine-grained parallelization where every single computing unit is responsible for computing segments in a dedicated abstract state (or set of states). Such a task-decomposition strategy naturally benefits from a reduced code-divergence and a coalesced memory access pattern. Even though a parallel simulation algorithm based on memoization is left as a challenge for our future work, we strongly believe that the distribution of local simulation tasks per abstract states opens a new possibility for further improvements of simulation scalability.

Our hybrid approach is inspired by existing hybrid simulation methods where reactions are partitioned into fast and slow reactions. This allows one to approximate the fast reactions by a quasi-steady-state assumption [5]. In general, hybrid approaches have proven to bring a promising step toward tackling multi-scale models that are of significant interest in systems biology. E.g., the combination of SSA and a flux-balance analysis has been recently shown [50] to provide a working solution for models in which steady-state assumption works well for *a priori* identified groups of variables. In contrast, our hybrid method provides a versatile solution for systems where such strong assumptions cannot be made.

Finally, we demonstrate that the proposed approach can substantially improve the existing deep-learning approaches. In particular, it can speed up the input generation, that is typically the bottleneck of the learning process, while preserving the precision of the resulting network.

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s12859-024-05966-5.

Supplementary file 1.

Acknowledgements
We would like to thank Vojtěch Mrázek for his help with the tool Nessie we used to demonstrate that the proposed simulation algorithm can significantly speed up deep learning of CRNs.

Author Contributions
MH contributed to the design, implementation and experimental evaluation of the proposed simulation algorithm. RA contributed to the implementation and interpretation of the experimental evaluation allowing for a comparison with the alternative approaches. MC and JK contributed to the conceptualisation of this work and to the design of the simulation algorithm. SM contributed to the implementation and experimental evaluation of the proposed simulation algorithm. DS contributed to the interpretation of the achieved results. All co-authors contributed to writing the manuscript and its revision.

Funding
This work has been supported by the Czech Science Foundation grant GJ20-02328Y, the FIT BUT internal project FIT-S-23-8151, the MUNI Award in Science and Humanities (MUNI/I/1757/2021) of the Grant Agency of Masaryk University, the German Research Foundation (DFG) projects 378803395 (ConVeY) and 427755713 (GOPro) as well as the ERC Advanced Grant 787367 (PaVeS).

Code and Data Availability
The proposed method is implemented within an open-source tool SAQuaiA (Simulation & Abstraction-based Quantitative Analyzer) as a branch of SeQuaiA, available at https://sequaia.model.in.tum.de/, and supplemented with an artifact (http://go.tum.de/065569) that allows to easily reproduce the experimental evaluation.

Materials Availability
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 13 August 2024 Accepted: 22 October 2024
Published online: 08 November 2024

References
1. Elowitz MB, Levine AJ, Siggia ED, Swain PS. Stochastic gene expression in a single cell. Science. 2002;297(5584):1183–6.
2. Gillespie DT. Exact stochastic simulation of coupled chemical reactions. J Phys Chem. 1977;81(25):2340–81.
3. Cao Y, Gillespie DT, Petzold LR. Efficient step size selection for the tau-leaping simulation method. J Chem Phys. 2006;124(4):044109.
4. Lester C, Yates CA, Giles MB, Baker RE. An adaptive multi-level simulation algorithm for stochastic biological systems. J Chem Phys. 2015;142(2):024113.
5. Goutsias J. Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems. J Chem Phys. 2005;122(18):184102.
6. Ganguly A, Altintan D, Koeppl H. Jump-diffusion approximation of stochastic reaction dynamics: error bounds and algorithms. Multiscale Model Simul. 2015;13(4):1390–419.
7. Hepp B, Gupta A, Khammash M. Adaptive hybrid simulations for multiscale stochastic reaction networks. J Chem Phys. 2015;142(3):034118.
8. Cairoli F, Carbone G, Bortolussi L. Abstraction of markov population dynamics via generative adversarial nets. In: Computational Methods in Systems Biology (CMSB), 2021;19–35. Springer
9. Gupta A, Schwab C, Khammash M. DeepCME: a deep learning framework for computing solution statistics of the chemical master equation. PLoS Comput Biol. 2021;17(12):1009623.
10. Sanft KR, Wu S, Roh M, Fu J, Lim RK, Petzold LR. StochKit2: software for discrete stochastic simulation of biochemical systems with events. Bioinformatics. 2011;27(17):2457–8.

11. Hoops S, Sahle S, Gauges R, Lee C, Pahle J, Simus N, Singhal M, Xu L, Mendes P, Kummer U. COPASI - a COmplex PAthway SImulator. Bioinformatics. 2006;22(24):3067–74.
12. Maarleveld TR, Olivier BG, Bruggeman FJ. Stochpy: a comprehensive, user-friendly tool for simulating stochastic biological processes. PLoS ONE. 2013;8(11):79345.
13. Myers CJ, Barker N, Jones K, Kuwahara H, Madsen C, Nguyen N-PD. ibiosim: a tool for the analysis and design of genetic circuits. Bioinformatics. 2009;25(21):2848–9.
14. Kazeroonian A, Frohlich F, Raue A, Theis FJ, Hasenauer J. Cerena: Chemical reaction network analyzer: a toolbox for the simulation and analysis of stochastic chemical kinetics. PLoS ONE. 2016;11(1):1–15.
15. Mauch S, Stalzer M. Efficient formulations for exact stochastic simulation of chemical systems. IEEE/ACM Trans Comput Biol Bioinf. 2009;8(1):27–35.
16. Klingbeil G, Erban R, Giles M, Maini PK. Stochsimgpu: parallel stochastic simulation for the systems biology toolbox 2 for matlab. Bioinformatics. 2011;27(8):1170–1.
17. Nobile MS, Cazzaniga P, Besozzi D, Pescini D, Mauri G. cutauleaping: a gpu-powered tau-leaping stochastic simulator for massive parallel analyses of biological systems. PLoS ONE. 2014;9(3):91963.
18. Munsky B, Khammash M. The finite state projection algorithm for the solution of the chemical master equation. J Chem Phys. 2006;124:044104.
19. Zhang J, Watson LT, Cao Y. Adaptive aggregation method for the chemical master equation. Int J Comput Biol Drug Des. 2009;2(2):134–48.
20. Hasenauer J, Wolf V, Kazeroonian A, Theis FJ. Method of conditional moments (MCM) for the chemical master equation. J Math Biol. 2013;1–49.
21. Abate A, Andriushchenko R, Češka M, Kwiatkowska M. Adaptive formal approximations of markov chains. Perform Eval. 2021;148:102207.
22. Singh A. Stochastic dynamics of predator-prey interactions. PLoS ONE. 2021;16(8):1–14. https://doi.org/10.1371/journal.pone.0255880.
23. Helfrich M, Češka M, Křetínský J, Martiček Š. Abstraction-based segmental simulation of chemical reaction networks. In: Computational Methods in Systems Biology (CMSB), 2022;41–60. Springer.
24. Sukys A, Öcal K, Grima R. Approximating solutions of the chemical master equation using neural networks. Iscience 2022;25(9)
25. Marino S, Hogue IB, Ray CJ, Kirschner DE. A methodology for performing global uncertainty and sensitivity analysis in systems biology. J Theor Biol. 2008;254(1):178–96.
26. Chellaboina V, Bhat SP, Haddad WM, Bernstein DS. Modeling and analysis of mass-action kinetics. IEEE Control Syst Mag. 2009;29(4):60–78.
27. Soloveichik D, Seelig G, Winfree E. DNA as a universal substrate for chemical kinetics. Proc Natl Acad Sci USA. 2010;107(12):5393–8.
28. Gillespie DT. A rigorous derivation of the chemical master equation. Physica A. 1992;188(1–3):404–25.
29. Lefever R, Nicolis G. Chemical instabilities and sustained oscillations. J Theor Biol. 1971;30(2):267–84.
30. Madsen C, Myers CJ, Roehner N, Winstead C, Zhang Z. Utilizing stochastic model checking to analyze genetic circuits. In: Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), 2012;379–386. IEEE.
31. Gillespie DT. Exact stochastic simulation of coupled chemical reactions. J Phys Chem. 1977;81(25):2340–61.
32. Robinson JT, Devarakonda MV. Data cache management using frequency-based replacement. In: Conference on Measurement and Modeling of Computer Systems, 1990;134–142. Association for Computing Machinery.
33. Srivastava R, You L, Summers J, Yin J. Stochastic vs. deterministic modeling of intracellular viral kinetics. J Theor Biol. 2002;218(3):309–21.
34. Burrage K, Tian T, Burrage P. A multi-scaled approach for simulating chemical reaction systems. Prog Biophys Mol Biol. 2004;85(2–3):217–34.
35. Češka M, Chau C, Křetínský J. SeQuaiA: A scalable tool for semi-quantitative analysis of chemical reaction networks. In: Computer Aided Verification (CAV), 2020;653–666. Springer.
36. Cao Y, Li H, Petzold L. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. J Chem Phys. 2004;121(9):4059–67.
37. McCollum JM, Peterson GD, Cox CD, Simpson ML, Samatova NF. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. Comput Biol Chem. 2006;30(1):39–49.
38. Rao CV, Arkin AP. Stochastic chemical kinetics and the quasi-steady-state assumption: application to the gillespie algorithm. J Chem Phys. 2003;118(11):4999–5010.
39. Cao Y, Gillespie DT, Petzold LR. The slow-scale stochastic simulation algorithm. J Chem Phys. 2005;122(1):014116.
40. Engblom S. Computing the moments of high dimensional solutions of the master equation. Appl Math Comput. 2006;180(2):498–515.
41. Van Kampen NG. Stochastic Processes in Physics and Chemistry vol. 1, Elsevier 1992.
42. Ethier SN, Kurtz TG. Markov Processes - Characterization and Convergence, vol. 282. Wiley; 2009.
43. Gillespie DT. The chemical Langevin equation. J Chem Phys. 2000;113(1):297–306.
44. Feret J, Salazar A. A generic framework to coarse-grain stochastic reaction networks by abstract interpretation. In: Verification, Model Checking, and Abstract Interpretation (VMCAI), 2023;228–251. Springer
45. Cairoli F, Anselmi F, d'Onofrio A, Bortolussi L. Generative abstraction of markov population processes. Theoret Comput Sci. 2023;977:114169.

46. Repin D, Petrov T. Automated deep abstractions for stochastic chemical reaction networks. Inf Comput. 2021;281:104788.
47. Cardelli L, Csikász-Nagy A. The cell cycle switch computes approximate majority. Scientific reports 2012;2.
48. Lakin MR, Youssef S, Polo F, Emmott S, Phillips A. Visual DSD: a design and analysis tool for DNA strand displacement systems. Bioinformatics. 2011;27(22):3211–3. https://doi.org/10.1093/bioinformatics/btr543.
49. Wang B, Thachuk C, Ellington AD, Winfree E, Soloveichik D. Effective design principles for leakless strand displacement systems. Proc Natl Acad Sci. 2018;115(52):12182–91.
50. Tourigny DS, Goldberg AP, Karr JR. Simulating single-cell metabolism using a stochastic flux-balance analysis algorithm. Biophys J. 2021;120(23):5231–42.