Abstraction-based segmental simulation of reaction networks using adaptive memoization
—
Supplementary Material

Contents

A Details to chemical reaction networks

We give a more detailed formal definition of Chemical Reaction Networks (CRN). A CRN $\mathcal{N} = (\Lambda, \mathcal{R})$ is a pair of finite sets, where Λ is a set of *species*, $|\Lambda|$ denotes its size, and $\mathcal{R}$ is a set of reactions. Species in Λ interact according to the reactions in $\mathcal{R}$. A *reaction* $\tau \in \mathcal{R}$ is a triple $\tau = (r_\tau, p_\tau, k_\tau)$, where $r_\tau \in \mathbb{N}^{|\Lambda|}$ is the *reactant complex*, $p_\tau \in \mathbb{N}^{|\Lambda|}$ is the *product complex* and $k_\tau \in \mathbb{R}_{>0}$ is the coefficient associated with the rate of the reaction. r_τ and p_τ represent the stoichiometry of reactants and products. A reaction $\tau_1 = ([1,1,0],[0,0,2],k_1)$ is written as $\tau_1 : \lambda_1 + \lambda_2 \xrightarrow{k_1} 2\lambda_3$.

Under the usual assumption of mass action kinetics, the time-evolution of CRNs is governed by the Chemical Master Equation (see e.g. [2]) that leads to a (potentially infinite) discrete-space, continuous-time Markov chain (CTMC) $\mathbf{X}(\mathbf{t}) = (X_1(t), X_2(t), \ldots, X_{|\Lambda|}(t))_{t\geq 0}$ describing how the probability of the copy-numbers of each species evolve in time. The *state change* associated with the reaction τ is defined by $\upsilon_\tau = p_\tau - r_\tau$, i.e., the state $\mathbf{X}$ is changed to $\mathbf{X}' = \mathbf{X} + \upsilon_\tau$. A reaction can only happen (is enabled) in a state $\mathbf{X}$ if all reactants are present in sufficient numbers. The transition rate corresponding to a reaction τ is given by a *propensity function* that, in general, depends on the stoichiometry of reactants, their populations, and the coefficient k_τ. We focus on mass action kinetics where the propensity function is defined as $a_\tau(X) = k_\tau \prod_{i=1}^{|\Lambda|} \binom{X_i}{r_{\tau_i}}$ but our approach can principally handle also alternative kinetics including Michaelis–Menten and Hill kinetics.

B Population level abstraction for segmental simulations

A population-level abstraction partitions the state space of a system into regions called abstract states. While in principle one can use any population-level abstraction for segmental simulation, inconsistencies like negative copy-numbers, jump over abstract states, and the application of disabled reactions can arise if we do not choose the abstraction carefully.

Example Consider the system visualized in Fig. 1 with only the reaction $r : 2X \to \emptyset$ and the partitioning into three abstract states $a_1, a_2, a3$: $a_1 = \{0,1\}$ with representative 0, abstract state $a_2 = \{2,3,4\}$ with representative 3 and $a_3 = \{5,6,...\}$ with representative 10. In the segmental simulation, the representative's segments are applied to each concrete state of the abstract state where a segment is a sequence of reactions until a different abstract state is reached. The only possible segment starting at the representative of a_3 is the sequence $10 \xrightarrow{r} 8 \xrightarrow{r} 6 \xrightarrow{r} 4$ leading to abstract state a_2. However, when we apply this segment to the concrete state 5 of the same abstract state, we get $5 \xrightarrow{r} 3 \xrightarrow{r} 1 \xrightarrow{r} -1$. This is not consistent with the model as negative copy numbers are reached. Further, when applying the same segment to the concrete state 7 we get $7 \xrightarrow{r} 5 \xrightarrow{r} 3 \xrightarrow{r} 1$ reaching abstract state a_1. While this is a feasible sequence of the system, this segmental step jumped over the abstract state a_2, effectively ignoring the local dynamics.

Interval population-level abstraction In this work, we only consider population-level abstractions that are the result of partitioning each dimension using intervals. For each dimension, we define consecutive intervals that are annotated with a representative. We write $[x, y, z]$ with $x \leq y \leq z$ to denote the

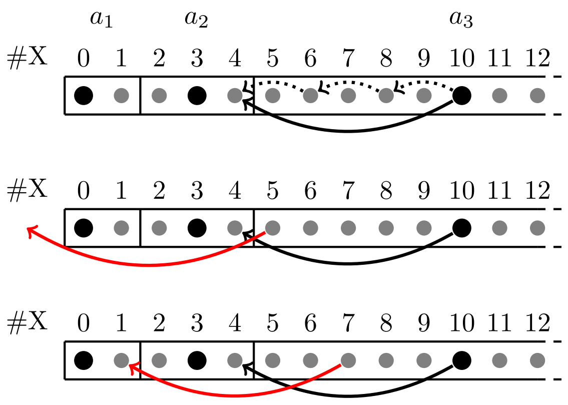
a_1
a_2
a_3
#X 0 1 2 3 4 5 6 7 8 9 10 11 12
#X 0 1 2 3 4 5 6 7 8 9 10 11 12
#X 0 1 2 3 4 5 6 7 8 9 10 11 12

Figure 1: Problematic segmental simulation steps because of unsuitable abstraction. (top) Population abstraction with three abstract states a_1, a_2, a_3. Circles are concrete states. The representatives are bigger and black. The solid arrow is a segment for a_3 consisting of three reactions drawn as dotted arrows. (middle) Applying the segment to concrete state 5 leads to a negative copy number. (bottom) Applying the segment to concrete state 7 jumps over a_2.

interval containing the values $\{x, x+1, ..., z-1, z\}$ with representative y, e.g., $[3, 4, 6] = \{3, 4, 5, 6\}$ with representative 4. An abstract state in a d-dimensional system is a hyper-rectangle that is described by d intervals, one for each dimension. The representative of an abstract state is the concrete state that corresponds to the representatives of all its intervals. E.g., the abstract state in a two dimensional system for intervals $[3, 4, 6]$ and $[10, 21, 42]$ contains all concrete states (x, y) with $3 \leq x \leq 6$ and $10 \leq y \leq 42$ and has representative $(4, 21)$.

Table 1: Exponential population-level abstraction for the Predator-Prey model with population growth factor c=1.5.

level	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
min	0	1	3	6	11	19	31	49	76	117	179	272	442	622	937
rep	0	1	4	8	14	24	39	62	96	147	225	341	516	779	1173
max	0	2	5	10	18	30	48	75	116	178	271	441	621	936	1409

The abstraction function for a given population-level abstraction maps each concrete state to its abstract state denoted as the vector of levels, e.g., for the interval abstraction in Tab. 1 the concrete state $(16, 269)$ is mapped to abstract state $(4, 10)$ because 16 is in the interval 4 and 269 is in level 10.

Abstraction suitable for segmental simulation For a population-level abstraction to be suitable for segmental simulation, it must hold that applying the representative's segments to any corresponding abstract state

1. does not apply reactions that are disabled, and
2. does not lead to a non-neighboring abstract state.

Note (1) implies that the same set of reactions is enabled in all concrete states of an abstract state and negative copy numbers cannot be reached. Further, to ensure that segmental simulation adequately approximates the dynamics of the system, it must hold that the states within each abstract state emit a similar probability space of the trajectories.

Exponential population-level abstraction Let $\mathcal{N} = (\Lambda, \mathcal{R})$ be a CRN and $c \in \mathbb{R}_{\geq 1}$ be the population growth factor. For each species $s \in \Lambda$, we first compute $m_s = \max_{\tau \in \mathcal{R}}(r_\tau)$ the highest multiplicity of s in any reaction's reactant complex. If s does not react, i.e., $m_s = 0$, then we do not split the dimension of s into multiple segments as the number of s molecules is not important for reusing segments.[1] Otherwise, we add the intervals $[0,0,0]$, $[1,1,1]$,..., $[m_s{-}1, m_s{-}1, m_s{-}1]$ and define the following intervals iteratively: After the interval $i = [x, y, z]$ with $x \leq y \leq z$ and size $|i| := z - x + 1$ we add the interval $i' = [x', y', z']$ where $x' := z{+}1$ and $y' := z{+}|i|$ and $z' := \lceil c \cdot |i| \rceil$. Intuitively, the next interval starts after the previous interval, it has the desired size of $\lceil c \cdot |i| \rceil$, and its representative is the largest value that does not enable jumps over the previous interval. An example of an exponential abstraction is given in Tab. 1. In case we want to force additional user-defined levels, e.g., for more precision in a certain range of copy numbers, we can generate intervals using a similar (but more complicated) heuristic or with a constraint solver.

C Improving memory consumption

C.1 Adaptive memoization

Because memory is limited, one needs to decide for which abstract states one wants to remember segments to achieve the best possible speedup. Or in other words, we want to use the available memory as efficiently as possible.

The memory efficiency of an abstract state depends on the following three values: (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.

$$\text{efficiency} = \frac{\text{speedup} \times \text{frequency}}{\text{memory}}$$

As it is infeasible to compute the efficiency directly, we need to approximate it. Because this involves the generation of multiple segments to judge the expected

[1] For some applications like measuring the accuracy it can be important to force a partition by setting $m_s := \max(m_s, 1)$.

speedup and thus takes significant time, we need to approximate the efficiency on-the-fly.

For a given memory limit, our adaptive memorization works as follows: A small fraction, typically 10%, of the memory is used for the identification of the most frequently visited abstract states. Specifically, we will use a least-frequently-used cache with dynamic aging (LFU-DA) [7]. This data structure is ideal for our use case as it supports all operations in amortized O(1) and updates usage counters that allow estimating the relative frequencies of abstract states. Dynamic aging regularly halves all usage counters, effectively reducing the impact of old usage data. This ensures that the frequencies adapt in cases where the predicted behavior of segmental simulation changes significantly, e.g., if the memory is used for simulations with different initial state or end time. Therefore, the adaptive memoization allows us to effectively apply the segmental simulation in more advanced experimental scenarios where initial conditions or simulation time is not fixed [5].

The rest of the available memory is used to store segments for each abstract state in the cache and all data needed for the efficiency estimation. This ensures that the majority of the memory is used to speed up the simulation. To estimate the expected speedup of an abstract state, we measure the average time to generate a new segment and compare it to the average time to reuse a saved segment.[2]Additionally, we keep track of the memory spent on each abstract state. Once the total used memory exceeds the defined memory limit, we evaluate the efficiency of all abstract states in the LFU-DA cache in order to free a specified fraction of the available memory, typically 15%. This is achieved by marking inefficient abstract states as *inactive* and removing their stored segments. In an inactive abstract state, we do not apply segmental simulation and instead evolve the system using SSA to reduce the error. If an inactive state becomes efficient enough, e.g., because it is visited more frequently, it will be *reactivated* in the next efficiency evaluation in order to collect new segments.

C.2 Segment-distribution approximation

When we reuse segments, we effectively sample from approximation of the actual segment distribution. If we sample from more segments we reduce the corresponding error, however, this costs more memory per abstract state and delays the simulation speedup. Up to this point, we required constant number of segment k before we start reusing. We will now generalize and allow users to define a memory function $f(x)$ that determines the number of generated segments to sample from when visiting an abstract state for the x-th time. The memory function for a constant k is $f(x) = k$. A family of functions that allows reuse earlier while generating an infinite amount of segments in the limit

[2] If efficiency is estimated using measured run times, the result of a segmental simulation does not just depend on the seed used for random number generation but also on the hardware. In cases where this is a problem, we propose to instead use the number of reactions to judge the expected speedup. Beware that this might not be a good metric if advanced base simulators are used.

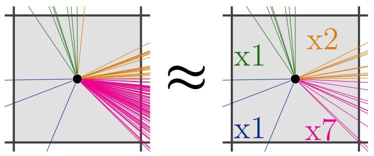

Figure 2: (left) Approximation of a segment distribution made up of 100 summaries. All summaries with the same direction have the same color, e.g., there are 20 orange summaries in direction $(+1, +1)$ or northeast. (right) A very similar but much more memory-efficient approximation with 30 summaries. 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.

is $f(x) = \log(sx + 1)/s$ for some $s > 0$. However, any other non-decreasing function is adequate.

While this determines the number of segments to sample from, it is not necessarily useful to save all these segments to memory. In fact, when all segments are similar, it is much more memory efficient if we just save one of them. However, if we generate a segment that corresponds to an unlikely event, it will differ from the known segments and should be saved. Thus, we classify segments according to their direction defined as the sign of the effect on each species, e.g., a segment that changes the state by $\Delta(+3, -1, 0)$ has direction $(1, -1, 0)$. Rather than saving all segments, we only save a small number, usually 10, segments per direction per abstract state. If we generated a segment for a direction with enough examples, we discard it and instead increase the likelihood of segments in this direction by increasing their weight when sampling a segment for reuse as visualized in Fig. 2.

D Abstraction-based hybrid simulation

Reaction classification The reaction classification is done per abstract state. First, each species is assigned a target classification according to the size of the interval in the corresponding dimension. If the interval is larger than some parameter t_{fast}, typically 400, the target is fast, otherwise, it is slow. Next, each reaction is classified according to the slowest species affected by the reaction. Finally, we compare the combined speed of all slow reactions with the combined speed of all fast reactions. If the propensity of the fast reactions is not significantly larger, typically o_{fast}=4 times larger, we instead classify every reaction as slow to avoid the overhead needed to approximate the ODE. Further, instead of classifying into just slow and fast reactions, we add a third class of reactions with a medium speed that is evolved using τ-leaping. Typically, we use the parameters t_{medium}=5 and o_{medium}=2.

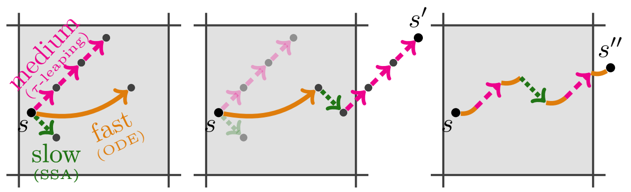

Figure 3: A hybrid simulation step starting in state s of an abstract state. (left) For every speed, the effects are first calculated separately. (middle) Then, the effects are combined leading out of the abstract state to state s'. (right) As the combined effect was too large, the events are replayed in random order. Between every discrete reaction, there is a continuous evolution. The first state outside of the abstract state is $s'' \neq s'$.

Hybrid step To perform one step of the hybrid simulation, we first determine the time $\Delta t = \min(\Delta t_{\text{slow}}, \Delta t_{\text{medium}}, \Delta t_{\text{fast}})$ for the next hybrid simulation step. Here, Δt_{slow} is the time of the next slow reaction according to SSA, $\Delta t_{\text{medium}} := \tau$ according to the τ-leaping approach of [1] and Δt_{fast} is the time needed for the fast reactions to change the abstract state according to the ODE. Next, we evolve the state according to the ODE for fast reactions up to time Δt. Finally, we sample the discrete reactions that occur in this step and apply their effect. The number of occurrences for medium reactions is determined by τ-leaping for $\tau := \Delta t$. In case $\Delta t < \Delta t_{\text{SSA}}$, the next slow reaction was too late. Otherwise, the next slow reaction occurs and is sampled according to SSA.[3] This process is illustrated in Fig. 3.

Overshooting Our hybrid simulation needs to handle cases where a single hybrid step is significantly larger than expected. This is typical whenever τ-leaping is used as it is unlikely but possible to sample very large values from the Poisson distributions. This could lead to negative copy numbers. A common solution for these rare events in τ-leaping is to discard the step and retry with a smaller τ. In our hybrid simulation approach we reclassify reactions once we leave the abstract state. Thus, we also consider steps too large if they do not stop right after the abstract state border. As this is quite common, we do not discard the sampled step but replay all reactions in random order until the abstract state is left (see right part of Fig. 3).

E Segmental simulation parameters

We explain the hyper-parameters of the simulation methods presented in this work and note their default values.

[3] The interaction between slow and medium reactions is analogous to critical and non-critical reactions in [1].

- *population-level growth factor* $c \in \mathbb{R}_{\geq 1}$ (default: c=1.5).
  It determines the exponential growth of the population-level abstraction (see Sec. B). Intuitively, each dimension is partitioned into intervals starting with intervals of size one for small copy-numbers, and the n-th interval is c times as large as interval $(n-1)$. For c=1, every concrete state is in a different abstract state.

- *τ-leap rate tolerance* $\varepsilon \in \mathbb{R}_{>0}$ (default: ε=0.03) A bound for the acceptable relative change in propensities is used to determine the next time window in τ-leaping as defined by [1]. Intuitively, we sample and apply all reactions that occur in the next τ seconds at once because the rate of each reaction does not change by more than a factor of ε.

- *segmental memory function* $f : \mathbb{N} \mapsto \mathbb{R}_{\geq 0}$ (default: $f(x) = \log(sx+1)/s$ with $s = 0.0359$)
  A function determining the number of effective segments to sample the next segment from if x is the number of visits to the current abstract state. The function typically should be monotonic and non-decreasing. Example: If the current abstract state was visited 1000 times, then $f(1000) = \log(0.0359 \cdot 1000 + 1)/0.0359 = 100.491$ and the next segment is chosen out of 101 segments.

- *memory limit in bytes* $m \in \mathbb{N}$ (default: m=5.000.000, or 5GB)
  The segmental simulation will use at most this much memory for storing segments.

- *memory fraction for abstract state cache* $m_{\text{cache}} \in \mathbb{R}$ (default: $m_{\text{cache}} = 0.1$)
  The segmental simulation will reserve this fraction of the total memory to store the known abstract states. If the cache is full and a new abstract state is hit, the least frequently used abstract state and all its segments will be removed from memory.

- *memory freeing fraction* $m_{\text{free}} \in \mathbb{R}$ (default: $m_{\text{free}} = 0.85$)
  If segmental simulation hits the memory limit because of too many saved segments, it removes the segments of the least efficient abstract states. Specifically, the memory reserved for segments is $m_{\text{seg}} := m \cdot (1 - m_{\text{cache}})$ bytes. After the memory freeing, at most $m_{\text{seg}} \cdot m_{\text{free}}$ bytes are used.

- *maximum frequency in LFU DA cache* $f_{\max} \in \mathbb{N}$ (default: $f_{\max} = 1000$)
  The LFU DA cache counts the number of uses of each element in order to keep the most frequently used abstract states in the cache. To avoid overflow of this counter, we limit it to $f_{\max}$. Elements with maximal usage count can only be increased after the next dynamic aging event.

- *maximum frequency* $f_{\text{DA}} \in \mathbb{R}$ (default: $f_{\text{DA}} = 0.1$)
  A dynamic aging event happens once the average use counter in the cache exceeds $f_{\max} \cdot f_{\text{DA}}$. This halves all usage counters. If an abstract state is

not used anymore, but the memory is full, dynamic aging will reduce the abstract state's counter until it is removed from memory.

- *minimum interval size for ODE* $t_{\text{fast}} \in \mathbb{N}$ (default: t_{fast}=400).
  In the abstraction-based hybrid simulation, a species can only be treated as continuous if the interval of the current abstract state in the corresponding dimension has at least size t_{fast}. This makes sure that only large copy numbers are evolved in a deterministic manner. In abstraction-based τ-leaping (TAU) we set $t_{\text{fast}}=\infty$ to disable ODE.

- *minimum interval size for τ-leaping* $t_{\text{medium}} \in \mathbb{N}$ (default: t_{medium}=5).
  In the abstraction-based hybrid simulation, a species can only be evolved using τ-leaping if the interval of the current abstract state in the corresponding dimension has at least size t_{medium}. This makes sure that species with very few molecules are evolved using SSA and thus only change one reaction at a time.

- *overhead factor for ODE* $o_{\text{fast}} \in \mathbb{R}_{\geq 1}$ (default: o_{fast}=4)
  If the total propensity of all fast reactions is not at least o_{fast}, then they get classified as SSA instead to circumvent the overhead of solving an ODE for a few reactions.

- *overhead factor for τ-leaping* $o_{\text{medium}} \in \mathbb{R}_{\geq 1}$ (default: o_{medium}=2)
  If the total propensity of all medium reactions is not at least o_{medium}, then they get classified as SSA instead to mitigate the overhead of performing a τ-leaping step.

- *ODE solver* s_{ODE} (default: DormandPrince54Integrator)
  To evolve the CRN deterministically, we numerically solve the underlying system of ODEs. We make use of the Apache commons library for ODEs and use the Apache Commons math library for JAVA. The default Dormand-Prince solver is an embedded Runge-Kutta integrator of order 5(4) with automatic step size control and uses the following parameters: minimum step size 1.0E-12, maximum step size 100, absolute tolerance 1.0E-3, relative tolerance 1.0E-8.

F Discussion about theoretical accuracy

SEG has the following two error sources.

Segment distribution approximation error: By reusing a finite number of saved segments, we effectively sample an approximation of the actual segment distribution starting at the representative of an abstract state. If the number of saved segments is too small, the abstract might miss important local behavior or skew the probabilities of events. However, the error decreases when the number of segments is increased and vanishes when the number of segments approaches infinity.

Abstraction error: Recall that SEG does not sample the segment distribution for the current state but instead samples the distribution for the representative of the current abstract state. Because the propensities, and thus the rates of reactions, are different for different states, this inherently introduces an error. The abstraction error is reasonably small in practice as the segment distributions for states within one abstract state are quite similar: Consider that within any abstract state, the propensity and thus rate for a mass-action reaction varies by at most the factor c^r where c is the growth factor of the exponential abstraction and r is the number of reactants. Decreasing the size of abstract states decreases the abstract error and it vanishes for c=1, where every state corresponds to a different abstract state.

While SEG changes the probability measure over the space of runs, we make sure that it never produces spurious behavior, i.e., every simulated reaction was enabled. This is achieved by choosing a suitable population-level abstraction, as described in Sec. B.

G Earth Mover's Distance (EMD)

To assess the accuracy of a simulation method, we want to compare the transient distribution approximated by generating a large number of simulations with the true transient distribution. Thus, we need a metric that measures the similarity of arbitrary, possibly high-dimensional, distributions. One such metric is the *first Wasserstein distance* or Kantorovich–Rubinstein metric, also known as *Earth Mover's Distance.*

General definition of Earth Mover's Distance (EMD) Let M be a compact metric set. The *Earth-Mover's Distance* (EMD) of the probability distributions μ and ν on M is defined as

$$\mathrm{EMD}(\mu, \nu) = \inf_{\pi \in \Pi(\mu,\nu)} \mathbb{E}_{(x,y)\sim\pi}[||x - y||]$$

where $\Pi(\mu, \nu)$ is set of all joint distributions $\pi(x, y)$ with first marginal μ and second marginal ν [6]. When comparing transient distributions of a CRN with n dimensions, then $M = \mathbb{R}^n$. Intuitively, when we interpret both distributions as piles of dirt, then the EMD is the minimal cost of transforming one into the other when the cost is the amount of dirt moved times the distance it is moved.

Computing the EMD Even in the case where the distributions have finite support, e.g., because we approximated a transient distribution using a finite number of simulations, one must solve a large transportation problem using linear programming to compute the EMD. This makes computing the general EMD infeasible in practice. However, for distributions with finite support in one dimension, there is a simple algorithm that computes the EMD in (quasi-)linear time (see Alg. 1). Thus, we instead use the *total EMD*

Algorithm 1 Computing EMD in 1D with finite supports

```
Require: probability distributions $\mu, \nu$ with finite supports $S_\mu, S_\nu \subset \mathbb{R}$
Ensure: $d = \mathrm{EMD}(\mu, \nu)$
1: $S := S_\mu \cup S_\nu$
2: $d := 0.0$, $movingmass := 0.0$, $last := 0.0$
3: for $x \in \textsc{sortAscending}(S)$ do
4:     $d := d + |movingmass \cdot (last - x)|$
5:     $movingmass := movingmass - \mu(x) + \nu(x)$
6:     $last := x$
7: end for
```

$$\textsc{total}\mathrm{EMD}(\mu, \nu) = \sum_{1 \le i \le n} EMD(\mu_i, \nu_i)$$

where μ_i and ν_i are the projections of the distributions to dimension i and n is the number of species.

Total Level EMD In our work, we report the EMD not for the concrete transient distribution but instead for the transient distribution over the abstract domain. Intuitively, we are interested in the general level of the species, not their exact value. The total level EMD is

$$\textsc{totalLevel}\mathrm{EMD}(\mu, \nu) = \textsc{total}\mathrm{EMD}(a(\mu), a(\nu))$$

where a is the abstraction function extended to modify the support of probability distributions.

H Additional experimental evaluation results

Figs. 4 and 5 compare plots of SSA simulations to both segmental simulation approaches. Fig. 6 shows the speedup we achieve when generating an increasing number of simulations with segmental simulation instead of SSA. Tab. 2 gives details about the memory usage when generating 10.000 simulations. A more detailed speedup comparison of segmental simulation and the hybrid simulation approach of [4] is given in Tab. 3.

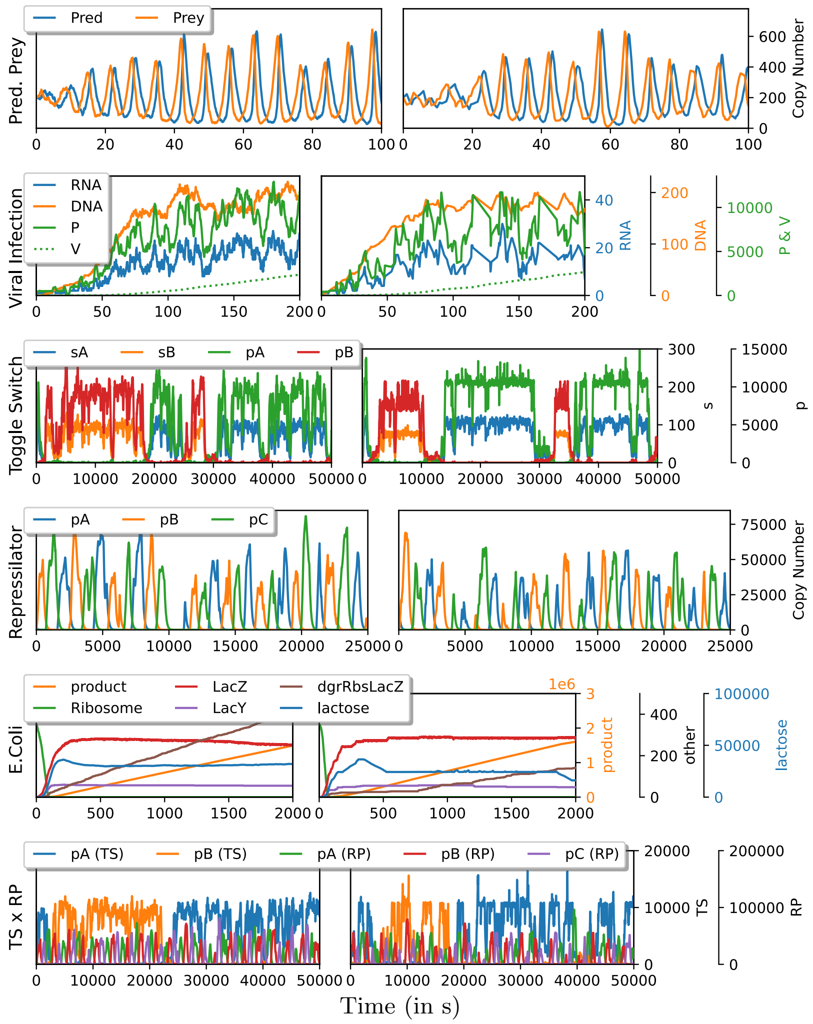

Figure 4: Comparison of SSA simulation (left) and segmental simulation using SSA (right) for different models.

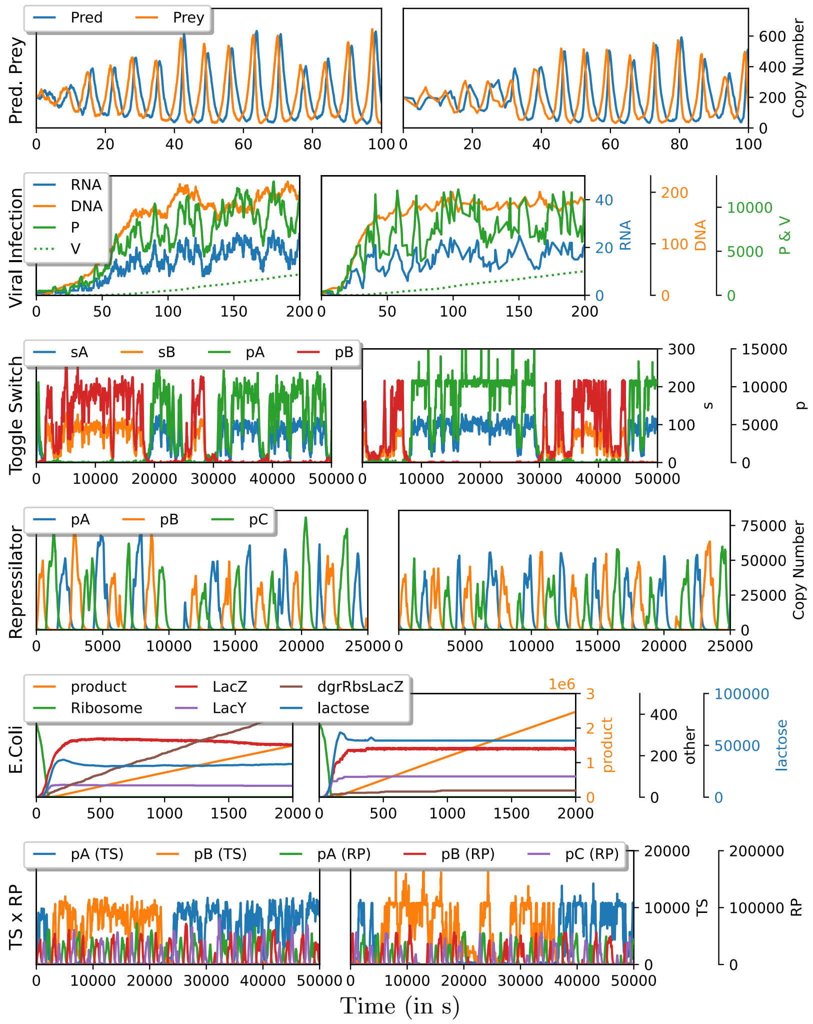

Figure 5: Comparison of the concrete simulations produced by SSA simulations (left) and hybrid SEG (right) for different models.

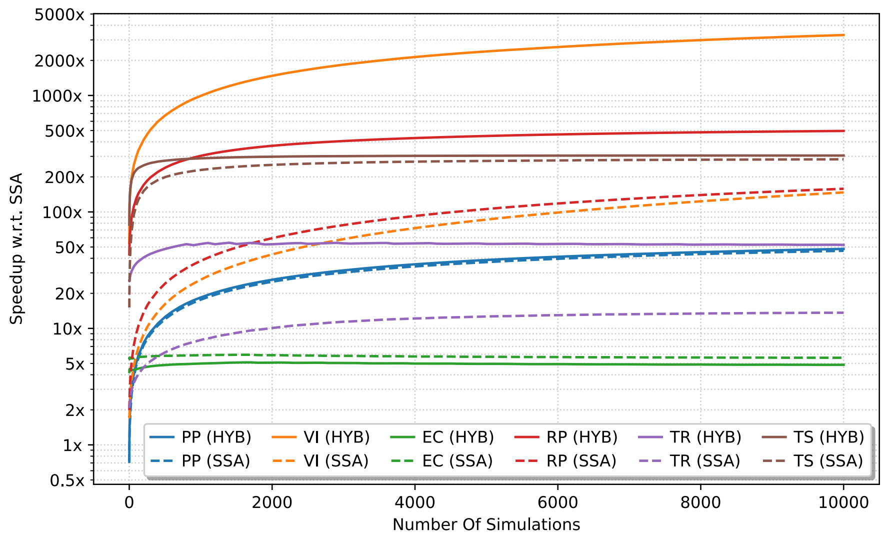

Figure 6: Speedup achieved by segmental simulation over SSA when generating a given number of simulations.

Table 2: Memory usage and the number of abstract states/segments in memory after 10.000 segmental simulations. Compared to [3], the memory requirements are reduced because of adaptive memory management and segment distribution approximation. This enables segmental simulation for models of any size. (*) Even more abstract states were visited but could not be saved. (**) The method is not adaptive and thus does not handle cases where the memory limit of 5GB is reached.

Model	abstract states	Segments		Memory	
		total	in approxim.	CMSB [3]	adaptive
PP	300	38,000	5,700 (15%)	2.6MB	390kB
VI	430	42,000	10,000 (24%)	3.5MB	840kB
TS	10,000	7.2E5	2.5E5 (35%)	69MB	24MB
RP	29,000	2.4E6	6.2E5 (26%)	230MB	61MB
TSxRP	6.3E6(*)	5.0E7	2.5E7 (50%)	OOM(**)	5GB
EC	4.4E6	5.5E7	1.7E6 (31%)	OOM(**)	5GB

Table 3: Runtime comparison with [4] for 100,000 simulations. Speedups are relative to the respective SSA implementation.

Model	[4]		Our results				
	SSA	Adaptive hybrid	SSA	TAU	HYB	SEG u. SSA	SEG u. HYB
RP	232 hours	3.0 hours (77x)	233 hours	30 hours (7.9x)	9.7 hours (24x)	1.5 hours (160x)	0.47 hours (500x)
TS	47 days	1.1 days (43x)	24 days	3.8 days (6.4x)	2.0 days (12x)	2.1 hours (280x)	1.9 hours (310x)

Generating training inputs for Nessie [8]. For this experiment we consider the TS model with parameterised rates: the value of each rate $r'_i, i \in [0, 13]$, is always within an order of magnitude wrt. value r_i considered in Table 6: $r'_i \in [\frac{1}{10} \cdot r_i, 10 \cdot r_i]$. Training data included 40,000 uniformly sampled parameter valuations, for which transient distributions were computed using either SSA (our implementation in SAQuaiA) or SEG+HYB with the growth factor $c = 1.1$. Given parameter valuation, from one SAQuaiA run for end time 100s, we extracted transient distributions for times 1,. . . ,100s. The resulting transient distributions are marginalized to represent distributions over the quantity of the protein sA, representing 4,000,000 training points for Nessie. Similarly, distributions for 100 and 500 random parameter valuations were used as validation and testing data, respectively; distributions for these training points were obtained by running 10,000 SSA simulations. Nessie was run with default settings. The scripts used to generate training data, the resulting neural networks and their outputs are available at `https://github.com/randriu/saquaia`.

I Models

The exact definitions of all models used in the evaluation are given in Tabs. 4 to 9. This includes their reactions together with their respective propensity functions, their initial state, and the time horizon of interest.

Table 4: Definition of the Predator Prey (PP) model.

Predator Prey	
Species (2)	$\textsc{Pred}, \textsc{Prey}$
Initial state	$(200 \times \textsc{Pred}, 200 \times \textsc{Prey})$
End time	200s
Reactions (3)	$rep : \textsc{Prey} \xrightarrow{1 \cdot \textsc{Prey}} 2\textsc{Prey}$ $eat : \textsc{Pred} + \textsc{Prey} \xrightarrow{0.005 \cdot \textsc{Pred} \cdot \textsc{Prey}} 2\textsc{Pred}$ $starve : \textsc{Pred} \xrightarrow{1 \cdot \textsc{Pred}} \emptyset$

Table 5: Definition of the Viral Infection (VI) model.

Viral Infection	
Species (4)	$\mathrm{DNA}, \mathrm{RNA}, \mathrm{P}, \mathrm{V}$
Initial state	$(1 \times \mathrm{RNA})$
End time	200s
Reactions (6)	$d0 : \mathrm{DNA} + \mathrm{P} \xrightarrow{1.125E{-}5 \cdot \mathrm{DNA} \cdot \mathrm{P}} \mathrm{V}$ $x : \mathrm{RNA} \xrightarrow{1000 \cdot \mathrm{RNA}} \mathrm{RNA} + \mathrm{P}$ $t : \mathrm{DNA} \xrightarrow{0.025 \cdot \mathrm{DNA}} \mathrm{DNA} + \mathrm{RNA}$ $p : \mathrm{RNA} \xrightarrow{1 \cdot \mathrm{RNA}} \mathrm{DNA} + \mathrm{RNA}$ $d2 : \mathrm{RNA} \xrightarrow{0.25 \cdot \mathrm{RNA}} \emptyset$ $d5 : \mathrm{P} \xrightarrow{1.9985 \cdot \mathrm{P}} \emptyset$

Table 6: Definition of the Toggle Switch (TS) model.

Toggle Switch		
Species (6)	$\textsc{m}A, \textsc{m}B, \textsc{s}A, \textsc{s}B, \textsc{p}A, \textsc{p}B$	
Initial state	$\emptyset$	
End time	50000s	
Reactions (14)	$r0 : \emptyset \xrightarrow{1} \textsc{m}A$	$r7 : \textsc{m}B + \textsc{s}A \xrightarrow{20\cdot\textsc{m}B\cdot\textsc{s}A} \textsc{s}A$
	$r1 : \emptyset \xrightarrow{1} \textsc{m}B$	$r8 : \textsc{m}A + \textsc{s}B \xrightarrow{20\cdot\textsc{m}A\cdot\textsc{s}B} \textsc{s}B$
	$r2 : \textsc{m}A \xrightarrow{0.1\cdot\textsc{m}A} \emptyset$	$r9 : \textsc{p}B \xrightarrow{0.1\cdot\textsc{p}B} \emptyset$
	$r3 : \textsc{m}B \xrightarrow{0.1\cdot\textsc{m}B} \emptyset$	$r10 : \textsc{s}A \xrightarrow{0.01\cdot\textsc{s}A} \emptyset$
	$r4 : \textsc{p}A \xrightarrow{0.1\cdot\textsc{p}A} \emptyset$	$r11 : \textsc{s}B \xrightarrow{0.01\cdot\textsc{s}B} \emptyset$
	$r5 : \textsc{m}A \xrightarrow{5\cdot\textsc{m}A} \textsc{s}A$	$r12 : \textsc{s}A \xrightarrow{10\cdot\textsc{s}A} \textsc{s}A + \textsc{p}A$
	$r6 : \textsc{m}B \xrightarrow{5\cdot\textsc{m}B} \textsc{s}B$	$r13 : \textsc{s}B \xrightarrow{10\cdot\textsc{s}B} \textsc{s}B + \textsc{p}B$

Table 7: Definition of the Repressilator (RP) model.

Repressilator		
Species (6)	$\textsc{m}A, \textsc{m}B, \textsc{m}C, \textsc{p}A, \textsc{p}B, \textsc{p}C$	
Initial state	$(10 \times \textsc{m}A, 500 \times \textsc{p}A)$	
End time	50000s	
Reactions (15)	$spawnA : \emptyset \xrightarrow{0.1} \textsc{m}A$	$despawnC : \textsc{m}C \xrightarrow{0.01\cdot\textsc{m}C} \emptyset$
	$spawnB : \emptyset \xrightarrow{0.1} \textsc{m}B$	$degradeA : \textsc{m}A + \textsc{p}B \xrightarrow{50\cdot\textsc{m}A\cdot\textsc{p}B} \textsc{p}B$
	$spawnC : \emptyset \xrightarrow{0.1} \textsc{m}C$	$degradeB : \textsc{m}B + \textsc{p}C \xrightarrow{50\cdot\textsc{m}B\cdot\textsc{p}C} \textsc{p}C$
	$prodA : \textsc{m}A \xrightarrow{50\cdot\textsc{m}A} \textsc{m}A + \textsc{p}A$	$degradeC : \textsc{m}C + \textsc{p}A \xrightarrow{50\cdot\textsc{m}C\cdot\textsc{p}A} \textsc{p}A$
	$prodB : \textsc{m}B \xrightarrow{50\cdot\textsc{m}B} \textsc{m}B + \textsc{p}B$	$dissolveA : \textsc{p}A \xrightarrow{0.01\cdot\textsc{p}A} \emptyset$
	$prodC : \textsc{m}C \xrightarrow{50\cdot\textsc{m}C} \textsc{m}C + \textsc{p}C$	$dissolveB : \textsc{p}B \xrightarrow{0.01\cdot\textsc{p}B} \emptyset$
	$despawnA : \textsc{m}A \xrightarrow{0.01\cdot\textsc{m}A} \emptyset$	$dissolveC : \textsc{p}C \xrightarrow{0.01\cdot\textsc{p}C} \emptyset$
	$despawnB : \textsc{m}B \xrightarrow{0.01\cdot\textsc{m}B} \emptyset$	

Table 8: Definition of the Toggle Switch × Repressilator (TSxRP) model.

Toggle Switch × Repressilator		
Species (12)	$\textsc{m}, \textsc{m}', \textsc{s}, \textsc{s}', \textsc{p}, \textsc{p}', \textsc{m}A, \textsc{m}B, \textsc{m}C, \textsc{p}A, \textsc{p}B, \textsc{p}C$	
Initial state	$(10 \times \textsc{m}A, 500 \times \textsc{p}A)$	
End time	50000s	
Reactions (29)	$r0 : \emptyset \xrightarrow{1} \textsc{m}$	$r7 : \textsc{m}' + \textsc{s} \xrightarrow{20 \cdot \textsc{m}' \cdot \textsc{s}} \textsc{s}$
	$r1 : \emptyset \xrightarrow{1} \textsc{m}'$	$r8 : \textsc{m} + \textsc{s}' \xrightarrow{20 \cdot \textsc{m} \cdot \textsc{s}'} \textsc{s}'$
	$r2 : \textsc{m} \xrightarrow{0.1 \cdot \textsc{m}} \emptyset$	$r9 : \textsc{p}' \xrightarrow{0.1 \cdot \textsc{p}'} \emptyset$
	$r3 : \textsc{m}' \xrightarrow{0.1 \cdot \textsc{m}'} \emptyset$	$r10 : \textsc{s} \xrightarrow{0.01 \cdot \textsc{s}} \emptyset$
	$r4 : \textsc{p} \xrightarrow{0.1 \cdot \textsc{p}} \emptyset$	$r11 : \textsc{s}' \xrightarrow{0.01 \cdot \textsc{s}'} \emptyset$
	$r5 : \textsc{m} \xrightarrow{5 \cdot \textsc{m}} \textsc{s}$	$r12 : \textsc{s} \xrightarrow{10 \cdot \textsc{s}} \textsc{s} + \textsc{p}$
	$r6 : \textsc{m}' \xrightarrow{5 \cdot \textsc{m}'} \textsc{s}'$	$r13 : \textsc{s}' \xrightarrow{10 \cdot \textsc{s}'} \textsc{s}' + \textsc{p}'$
	$spawnA : \emptyset \xrightarrow{0.1} \textsc{m}A$	$despawnC : \textsc{m}C \xrightarrow{0.01 \cdot \textsc{m}C} \emptyset$
	$spawnB : \emptyset \xrightarrow{0.1} \textsc{m}B$	$degradeA : \textsc{m}A + \textsc{p}B \xrightarrow{50 \cdot \textsc{m}A \cdot \textsc{p}B} \textsc{p}B$
	$spawnC : \emptyset \xrightarrow{0.1} \textsc{m}C$	$degradeB : \textsc{m}B + \textsc{p}C \xrightarrow{50 \cdot \textsc{m}B \cdot \textsc{p}C} \textsc{p}C$
	$prodA : \textsc{m}A \xrightarrow{50 \cdot \textsc{m}A} \textsc{m}A + \textsc{p}A$	$degradeC : \textsc{m}C + \textsc{p}A \xrightarrow{50 \cdot \textsc{m}C \cdot \textsc{p}A} \textsc{p}A$
	$prodB : \textsc{m}B \xrightarrow{50 \cdot \textsc{m}B} \textsc{m}B + \textsc{p}B$	$dissolveA : \textsc{p}A \xrightarrow{0.01 \cdot \textsc{p}A} \emptyset$
	$prodC : \textsc{m}C \xrightarrow{50 \cdot \textsc{m}C} \textsc{m}C + \textsc{p}C$	$dissolveB : \textsc{p}B \xrightarrow{0.01 \cdot \textsc{p}B} \emptyset$
	$despawnA : \textsc{m}A \xrightarrow{0.01 \cdot \textsc{m}A} \emptyset$	$dissolveC : \textsc{p}C \xrightarrow{0.01 \cdot \textsc{p}C} \emptyset$
	$despawnB : \textsc{m}B \xrightarrow{0.01 \cdot \textsc{m}B} \emptyset$	

Table 9: Definition of the E. coli (EC) model.

E. coli	
Species (23)	$\textsc{PLac}, \text{RNAP}, \textsc{PLacRNAP}, \textsc{TrLacZ1}, \textsc{RbsLacZ}, \textsc{TrLacZ2}, \textsc{TrLacY1}, \textsc{RbsLacY}, \textsc{TrLacY2}, \textsc{Ribosome}, \textsc{RbsRibosomeLacZ}, \textsc{RbsRibosomeLacY}, \textsc{TrRbsLacZ}, \textsc{TrRbsLacY}, \textsc{LacZ}, \textsc{LacY}, \textsc{dgrLacZ}, \textsc{dgrLacY}, \textsc{dgrRbsLacZ}, \textsc{dgrRbsLacY}, \textsc{lactose}, \textsc{LacZlactose}, \textsc{product}$
Initial state	$(\textsc{PLac}, 35 \times \text{RNAP}, 350 \times \textsc{Ribosome})$
End time	2000s
Reactions (22)	$r0 : \textsc{PLac} + \text{RNAP} \xrightarrow{0.17\cdot\textsc{PLac}\cdot\text{RNAP}} \textsc{PLacRNAP}$ $r1 : \textsc{PLacRNAP} \xrightarrow{10.0\cdot\textsc{PLacRNAP}} \textsc{PLac} + \text{RNAP}$ $r2 : \textsc{PLacRNAP} \xrightarrow{1.0\cdot\textsc{PLacRNAP}} \textsc{TrLacZ1}$ $r3 : \textsc{TrLacZ1} \xrightarrow{1.0\cdot\textsc{TrLacZ1}} \textsc{PLac} + \textsc{RbsLacZ} + \textsc{TrLacZ2}$ $r4 : \textsc{TrLacZ2} \xrightarrow{0.015\cdot\textsc{TrLacZ2}} \textsc{TrLacY1}$ $r5 : \textsc{TrLacY1} \xrightarrow{1.0\cdot\textsc{TrLacY1}} \textsc{RbsLacY} + \textsc{TrLacY2}$ $r6 : \textsc{TrLacY2} \xrightarrow{0.36\cdot\textsc{TrLacY2}} \text{RNAP}$ $r7 : \textsc{RbsLacZ} + \textsc{Ribosome} \xrightarrow{0.17\cdot\textsc{RbsLacZ}\cdot\textsc{Ribosome}} \textsc{RbsRibosomeLacZ}$ $r8 : \textsc{RbsLacY} + \textsc{Ribosome} \xrightarrow{0.17\cdot\textsc{RbsLacY}\cdot\textsc{Ribosome}} \textsc{RbsRibosomeLacY}$ $r9 : \textsc{RbsRibosomeLacZ} \xrightarrow{0.45\cdot\textsc{RbsRibosomeLacZ}} \textsc{RbsLacZ} + \textsc{Ribosome}$ $r10 : \textsc{RbsRibosomeLacY} \xrightarrow{0.45\cdot\textsc{RbsRibosomeLacY}} \textsc{RbsLacY} + \textsc{Ribosome}$ $r11 : \textsc{RbsRibosomeLacZ} \xrightarrow{0.4\cdot\textsc{RbsRibosomeLacZ}} \textsc{RbsLacZ} + \textsc{TrRbsLacZ}$ $r12 : \textsc{RbsRibosomeLacY} \xrightarrow{0.4\cdot\textsc{RbsRibosomeLacY}} \textsc{RbsLacY} + \textsc{TrRbsLacY}$ $r13 : \textsc{TrRbsLacZ} \xrightarrow{0.015\cdot\textsc{TrRbsLacZ}} \textsc{LacZ}$ $r14 : \textsc{TrRbsLacY} \xrightarrow{0.036\cdot\textsc{TrRbsLacY}} \textsc{LacY}$ $r15 : \textsc{LacZ} \xrightarrow{6.42E-5\cdot\textsc{LacZ}} \textsc{dgrLacZ}$ $r16 : \textsc{LacY} \xrightarrow{6.42E-5\cdot\textsc{LacY}} \textsc{dgrLacY}$ $r17 : \textsc{RbsLacZ} \xrightarrow{0.3\cdot\textsc{RbsLacZ}} \textsc{dgrRbsLacZ}$ $r18 : \textsc{RbsLacY} \xrightarrow{0.3\cdot\textsc{RbsLacY}} \textsc{dgrRbsLacY}$ $r19 : \textsc{LacZ} + \textsc{lactose} \xrightarrow{9.52E-5\cdot\textsc{LacZ}\cdot\textsc{lactose}} \textsc{LacZlactose}$ $r20 : \textsc{LacZlactose} \xrightarrow{431.0\cdot\textsc{LacZlactose}} \textsc{LacZ} + \textsc{product}$ $r21 : \textsc{LacY} \xrightarrow{14.0\cdot\textsc{LacY}} \textsc{LacY} + \textsc{lactose}$

References

[1] Yang Cao, Daniel T Gillespie, and Linda R Petzold. Efficient step size selection for the tau-leaping simulation method. *The Journal of chemical physics*, 124(4):044109, 2006.

[2] Daniel T Gillespie. A rigorous derivation of the chemical master equation. *Physica A: Statistical Mechanics and its Applications*, 188(1-3):404–425, 1992.

[3] Martin Helfrich, Milan Češka, Jan Křetínský, and Štefan Martiček. Abstraction-based segmental simulation of chemical reaction networks. In *Computational Methods in Systems Biology (CMSB)*, pages 41–60. Springer, 2022.

[4] Benjamin Hepp, Ankit Gupta, and Mustafa Khammash. Adaptive hybrid simulations for multiscale stochastic reaction networks. *The Journal of chemical physics*, 142(3):034118, 2015.

[5] Simeone Marino, Ian B Hogue, Christian J Ray, and Denise E Kirschner. A methodology for performing global uncertainty and sensitivity analysis in systems biology. *Journal of theoretical biology*, 254(1):178–196, 2008.

[6] Svetlozar T Rachev. The monge–kantorovich mass transference problem and its stochastic applications. *Theory of Probability & Its Applications*, 29(4):647–676, 1985.

[7] John T. Robinson and Murthy V. Devarakonda. Data cache management using frequency-based replacement. In *Conference on Measurement and Modeling of Computer Systems*, page 134–142. Association for Computing Machinery, 1990.

[8] Augustinas Sukys, Kaan Öcal, and Ramon Grima. Approximating solutions of the chemical master equation using neural networks. *Iscience*, 25(9), 2022.