Abstraction-Based Segmental Simulation of Chemical Reaction Networks

Computational Methods in Systems Biology (CMSB 2022)

Martin Helfrich  Milan Češka  Jan Křetínský  Štefan Martiček

September 14, 2022
Chemical Reaction Networks (CRN):
- Model real-world biochemical systems
- Many applications (e.g. in medicine & molecular programming)
Introduction

Chemical Reaction Networks (CRN):
- Model real-world biochemical systems
- Many applications (e.g. in medicine & molecular programming)

Transient analysis:
- "How does the system evolve?"
- Hard because of complex dynamics, state-space explosion, stochasticity, stiffness, and multimodality
- Two approaches:
  - Direct (numerical)
  - Indirect (using many trajectories)
Chemical Reaction Networks (CRN):
- Model real-world biochemical systems
- Many applications (e.g. in medicine & molecular programming)

Transient analysis:
- "How does the system evolve?"
- Hard because of complex dynamics, state-space explosion, stochasticity, stiffness, and multimodality
- Two approaches:
  - Direct (numerical)
  - Indirect (using many trajectories)

This Work
- **Goal**: compute many simulations fast
Introduction

Chemical Reaction Networks (CRN):
- Model real-world biochemical systems
- Many applications (e.g. in medicine & molecular programming)

Transient analysis:
- "How does the system evolve?"
- Hard because of complex dynamics, state-space explosion, stochasticity, stiffness, and multimodality
- Two approaches:
  - Direct (numerical)
  - Indirect (using many trajectories)

This Work
- **Goal**: compute many simulations fast
- **Idea**: using memorization
## CRNs

### Example: Viral Infection

<table>
<thead>
<tr>
<th>Species</th>
<th>RNA, DNA, V, P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state</td>
<td>(1 × RNA)</td>
</tr>
<tr>
<td>End time</td>
<td>200s</td>
</tr>
</tbody>
</table>
| Reactions     | DNA + P $^{0.00001125 \cdot DNA \cdot P}$ V  
RNA $^{1000 \cdot RNA}$ RNA + P  
DNA $^{0.025 \cdot DNA}$ DNA + RNA  
RNA $^{1 \cdot RNA}$ DNA + RNA  
RNA $^{0.25 \cdot RNA}$  
P $^{1.9985 \cdot P}$  |

- Evolution governed by **Chemical Master Equation**
- Gives rise to discrete-space continuous-time Markov chain (**CTMC**)
Gillespie’s stochastic simulation algorithm (SSA) [3]

- Sample one reaction at a time
Gillespie’s stochastic simulation algorithm (SSA) [3]

- Sample one reaction at a time

$t = 0s$  

Start in initial state.
Gillespie’s stochastic simulation algorithm (SSA) [3]

- Sample one reaction at a time

Compute rate of all reactions according to their propensity functions.
Gillespie’s stochastic simulation algorithm (SSA) [3]

- Sample one reaction at a time

Time until the next reaction: $\Delta t \sim EXP(0.5+2+1.5)$
Probability of reactions: $\frac{0.5}{4}, \frac{2}{4}, \frac{1.5}{4}$
Gillespie’s stochastic simulation algorithm (SSA) [3]

- Sample one reaction at a time

$t = 0.18s$
Gillespie’s stochastic simulation algorithm (SSA) [3]

- Sample one reaction at a time

Time until the next reaction: $\Delta t \sim EXP(1+2+2)$
Probability of reactions: $\frac{1}{5}, \frac{2}{5}, \frac{2}{5}$
Gillespie’s stochastic simulation algorithm (SSA) [3]

- Sample one reaction at a time

Time until the next reaction: $\Delta t \sim EXP(4+2+3+1)$

Probability of reactions: $\frac{4}{10}, \frac{2}{10}, \frac{3}{10}, \frac{1}{10}$
Gillespie’s stochastic simulation algorithm (SSA) [3]

- Sample one reaction at a time
- May take a long time

\[ r_{\text{init}} = 0.5 \]
\[ r_1 = 2 \]
\[ r_2 = 1.5 \]
\[ r_3 = 1 \]
\[ r_4 = 2 \]
\[ r_5 = 2 \]
\[ r_6 = 4 \]
\[ r_7 = 2 \]
\[ t = 0.47 \text{s} \]

\[ \Delta t \sim \text{EXP}(4+2+3+1) \]

Probability of reactions:
\[ 4 \times 10^{-1}, 2 \times 10^{-1}, 3 \times 10^{-1}, 1 \times 10^{-1} \]
Precompute \( k \) short trajectories (called segments) for each state.  
→ Simulate by sampling segments instead of single reactions.
Precompute $k$ short trajectories (called segments) for each state. → Simulate by sampling segments instead of single reactions.
Precompute $k$ short trajectories (called segments) for each state.
→ Simulate by sampling segments instead of single reactions.
Precompute $k$ short trajectories (called segments) for each state. → Simulate by sampling segments instead of single reactions.

- much faster!
Precompute $k$ short trajectories (called segments) for each state. → Simulate by sampling segments instead of single reactions.

- much faster!
- Problem: many states → too inefficient
Abstraction-Based Segmental Simulation

- **Idea:** Do not treat every state separately!

States with similar species counts have similar propensities → they behave similarly

Population-level abstraction: split state-space into regions (called abstract states)

Population levels grow exponentially

Choose representative for each abstract state (usually center)
Abstraction-Based Segmental Simulation

- **Idea:** Do not treat every state separately!
- States with similar species counts have similar propensities
  → their behave similarly
Abstraction-Based Segmental Simulation

- **Idea**: Do not treat every state separately!
- States with similar species counts have similar propensities → their behave similarly
- **Population-level abstraction**: split state-space into regions (called abstract states)

![Diagram showing the state-space and abstraction regions](image-url)
Abstraction-Based Segmental Simulation

- **Idea**: Do not treat every state separately!
- States with similar species counts have similar propensities → their behave similarly
- **Population-level abstraction**: split state-space into regions (called abstract states)

![](image)

- Population levels grow exponentially
Abstraction-Based Segmental Simulation

- **Idea**: Do not treat every state separately!
- States with similar species counts have similar propensities → their behave similarly
- **Population-level abstraction**: split state-space into regions (called abstract states)

Population levels grow exponentially

Choose **representative** for each abstract state (usually center)
Only precompute $k$ segments for each representative.

Segments end when they leave the abstract state.

→ Intuition: "significant change"
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

MEMORY
$(k = 3)$

SIMULATION

Lazy: Do not precompute but fill memory on-the-fly!
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

MEMORY
$(k = 3)$

SIMULATION

Lazy: Do not precompute but fill memory on-the-fly!
Only precompute $k$ segments for each representative.
Only precompute $k$ segments for each representative.
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

MEMORY
$(k = 3)$

SIMULATION
Only precompute $k$ segments for each representative.
Only precompute $k$ segments for each representative.
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

MEMORY
$(k = 3)$

SIMULATION
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

MEMORY

$(k = 3)$

SIMULATION

To save memory: Work with summaries instead of segments.
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

Lazy: Do not precompute but fill memory on-the-fly!
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

Lazy: Do not precompute but fill memory on-the-fly!
Abstraction-Based Segmental Simulation

Only precompute \( k \) segments for each representative.

MEMORY
\((k = 3)\)

SIMULATION

Lazy: Do not precompute but fill memory on-the-fly!
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

**MEMORY**

$(k = 3)$

**SIMULATION**

Lazy: Do not precompute but fill memory on-the-fly!
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

Lazy: Do not precompute but fill memory on-the-fly!
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

MEMORY $(k = 3)$

SIMULATION

Lazy: Do not precompute but fill memory on-the-fly!
Abstraction-Based Segmental Simulation

Only precompute $k$ segments for each representative.

Lazy: Do not precompute but fill memory on-the-fly!
Only precompute $k$ segments for each representative.

**MEMORY**

$\{ \text{a, b, c, d, e, f, g, h, i, j, k, l} \}$

$\{ k = 3 \}$

**SIMULATION**

$\{ \text{i, g, d, c} \}$

$\text{s_{init}}$

**Lazy:** Do not precompute but fill memory on-the-fly!
There are two error sources:
There are two error sources:

1. Limited number of memorized segments:
   - Cannot faithfully represent actual segment distribution
   - Error vanishes for $k \to \infty$
There are two error sources:

1. Limited number of memorized segments:
   - Cannot faithfully represent actual segment distribution
   - Error vanishes for $k \to \infty$

2. Using representative’s segments
   - Similar species counts $\to$ similar propensities $\to$ similar segments
   - Error gets smaller if we add more population levels
## Example: Viral Infection

<table>
<thead>
<tr>
<th>Species</th>
<th>RNA, DNA, V, P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state</td>
<td>(1 × RNA)</td>
</tr>
<tr>
<td>End time</td>
<td>200s</td>
</tr>
<tr>
<td>Reactions</td>
<td></td>
</tr>
<tr>
<td>DNA + P</td>
<td>$\frac{0.00001125 \cdot \text{DNA} \cdot \text{P}}{1000 \cdot \text{RNA}}$ → V</td>
</tr>
<tr>
<td>RNA</td>
<td>$\frac{1000 \cdot \text{RNA}}{0.025 \cdot \text{DNA}}$ → RNA + P</td>
</tr>
<tr>
<td>DNA</td>
<td>$\frac{0.025 \cdot \text{DNA}}{1 \cdot \text{RNA}}$ → DNA + RNA</td>
</tr>
<tr>
<td>RNA</td>
<td>$\frac{1 \cdot \text{RNA}}{0.25 \cdot \text{RNA}}$ → ∅</td>
</tr>
<tr>
<td>RNA</td>
<td>$\frac{0.25 \cdot \text{RNA}}{1.9985 \cdot \text{P}}$ → ∅</td>
</tr>
</tbody>
</table>
Evaluation - Accuracy

SSA - Simulation 1
Evaluation - Accuracy

SSA - Simulation 2

![Graph showing DNA, RNA, V, and P over time in s.](image-url)
Evaluation - Accuracy

SSA - Simulation 3
Evaluation - Accuracy

SEG - Simulation 4

Graph showing the change in RNA, DNA, V, and P over time (in s) with values ranging from 0 to 2500 for RNA and DNA, and 0 to 14000 for V and P.
Evaluation - Accuracy

SEG - Simulation 6

Graph showing the time (in s) on the x-axis and RNA, DNA, V, and P on the y-axis.
Evaluation - Accuracy

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSA</td>
<td>13.6</td>
<td>2878</td>
</tr>
<tr>
<td>SEG</td>
<td>13.5</td>
<td>2685</td>
</tr>
</tbody>
</table>
Evaluation - Performance

**Speed-up:**

<table>
<thead>
<tr>
<th>Time Per-Simulation (in s)</th>
<th>Number Of Simulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSA</td>
<td>c=1.5,k=100</td>
</tr>
<tr>
<td>c=1.3,k=100</td>
<td>c=1.5,k=1000</td>
</tr>
<tr>
<td>c=1.3,k=1000</td>
<td>c=2,k=10</td>
</tr>
<tr>
<td>c=1.3,k=1000</td>
<td>c=2,k=100</td>
</tr>
<tr>
<td>c=1.3,k=1000</td>
<td>c=1.5,k=100</td>
</tr>
<tr>
<td>c=1.5,k=10</td>
<td>c=2,k=1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time Per-Simulation (in s)</th>
<th>Number Of Simulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSA</td>
<td>c=1.5,k=100</td>
</tr>
<tr>
<td>c=1.3,k=10</td>
<td>c=1.5,k=1000</td>
</tr>
<tr>
<td>c=1.3,k=1000</td>
<td>c=2,k=10</td>
</tr>
<tr>
<td>c=1.3,k=1000</td>
<td>c=2,k=100</td>
</tr>
<tr>
<td>c=1.3,k=1000</td>
<td>c=1.5,k=100</td>
</tr>
<tr>
<td>c=1.5,k=10</td>
<td>c=2,k=1000</td>
</tr>
</tbody>
</table>
Evaluation - Performance

**Speed-up:**

- Depends on model and target accuracy

![Graph showing speed-up over increasing number of simulations](image)

- Accelerates with number of simulations
- Can already be faster than SSA in the first simulation
- Memorization: trade-off between speed and memory
Evaluation - Performance

**Speed-up:**

- Depends on model and target accuracy
- Accelerates with number of simulations
Evaluation - Performance

**Speed-up:**

- Depends on model and target accuracy
- Accelerates with number of simulations
- Can already be faster than SSA in first simulation
Evaluation - Performance

**Speed-up:**

- Depends on model and target accuracy
- Accelerates with number of simulations
- Can already be faster than SSA in first simulation
- Memorization: trade-off between speed and memory
Oversimplified comparison with other approaches:

<table>
<thead>
<tr>
<th>Approach</th>
<th>Speed-up</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSA [3]</td>
<td>1x</td>
<td>perfect</td>
</tr>
<tr>
<td>$\tau$-leaping [2]</td>
<td>$\sim5x$</td>
<td>very good</td>
</tr>
<tr>
<td>Hybrid Simulation [4]</td>
<td>$\sim50x$</td>
<td>good</td>
</tr>
<tr>
<td>Deep Learning$^1$ [1]</td>
<td>$\sim100x$</td>
<td>good</td>
</tr>
<tr>
<td>Segmental Simulation$^2$</td>
<td>$\sim200x$</td>
<td>good</td>
</tr>
</tbody>
</table>

$^1$requires precomputed data and long training period
$^2$significant memory requirement
Future Work

- Handle larger models: adaptive memory allocation
Future Work

- Handle larger models: adaptive memory allocation
- Formal error bounds
Future Work

- Handle larger models: adaptive memory allocation
- Formal error bounds
- Segmental simulation as general framework for accelerating simulations
Future Work

- Handle larger models: adaptive memory allocation
- Formal error bounds
- Segmental simulation as general framework for accelerating simulations

Thank you!
References


Importance of Concrete State Information

MEMORY

\( (k = 3) \)

SIMULATION

(this work)

SIMULATION

(previous work)
Importance of Concrete State Information

- Only abstract states $\rightarrow$ rounding
- Rounding loses progress in all but one dimension
Importance of Concrete State Information

- Only abstract states $\rightarrow$ rounding
- Rounding looses progress in all but one dimension

Example: Rounding Problem

<table>
<thead>
<tr>
<th>Species</th>
<th>ON, OFF, X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state</td>
<td>(1 $\times$ ON, 50 $\times$ X)</td>
</tr>
<tr>
<td>Reactions</td>
<td>ON $\rightarrow$ OFF + X</td>
</tr>
<tr>
<td></td>
<td>OFF $\rightarrow$ ON + X</td>
</tr>
</tbody>
</table>
Importance of Concrete State Information

- Only abstract states → rounding
- Rounding loses progress in all but one dimension

**Example: Rounding Problem**

<table>
<thead>
<tr>
<th>Species</th>
<th>ON, OFF, X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state</td>
<td>(1 × ON, 50 × X)</td>
</tr>
</tbody>
</table>
| Reactions | ON → OFF + X  
OFF → ON + X |

**SIMULATION**

X does grow. ✓
Importance of Concrete State Information

- Only abstract states $\rightarrow$ rounding
- Rounding looses progress in all but one dimension

**Example: Rounding Problem**

<table>
<thead>
<tr>
<th>Species</th>
<th>ON, OFF, X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state</td>
<td>$ (1 \times \text{ON}, 50 \times \text{X}) $</td>
</tr>
<tr>
<td>Reactions</td>
<td>ON $\rightarrow$ OFF $+$ X</td>
</tr>
<tr>
<td></td>
<td>OFF $\rightarrow$ ON $+$ X</td>
</tr>
</tbody>
</table>

SIMULATION (this work)

X does grow. ✅

SIMULATION (previous work)
Importance of Concrete State Information

- Only abstract states → rounding
- Rounding looses progress in all but one dimension

**Example: Rounding Problem**

<table>
<thead>
<tr>
<th>Species</th>
<th>ON, OFF, X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state</td>
<td>(1 × ON, 50 × X)</td>
</tr>
<tr>
<td>Reactions</td>
<td>ON → OFF + X</td>
</tr>
<tr>
<td></td>
<td>OFF → ON + X</td>
</tr>
</tbody>
</table>

**Simulation (this work)**

X does grow. ✅

**Simulation (previous work)**
Importance of Concrete State Information

- Only abstract states $\rightarrow$ rounding
- Rounding loses progress in all but one dimension

**Example: Rounding Problem**

<table>
<thead>
<tr>
<th>Species</th>
<th>ON, OFF, X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state</td>
<td>$(1 \times \text{ON}, 50 \times \text{X})$</td>
</tr>
</tbody>
</table>
| Reactions | ON $\rightarrow$ OFF $+$ X  
OFF $\rightarrow$ ON $+$ X |

<table>
<thead>
<tr>
<th>#X</th>
<th>48</th>
<th>49</th>
<th>50</th>
<th>51</th>
<th>52</th>
<th>53</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SIMULATION (this work)

X does grow. ✔

SIMULATION (previous work)

X does NOT grow. ✗
Lazy Algorithm

**Inputs:** $\mathcal{N}$ (CRN), $k$ (number of segments), $c$ (partitioning parameter), $t_{\text{end}}$ (time horizon), $s_{\text{init}}$ (initial state) and $m$ (number of simulations)

**Output:** list of $m$ segmental simulations

1. $\text{simulations} := [ ]$;
2. $\text{memory} := \{\}$; // mapping each abstract state to a list of segments
3. for $1$ to $m$ do
4.     $s := s_{\text{init}}$; $t := 0$; $\text{simulation} := [(s, t)]$;
5.     while $t < t_{\text{end}}$ do
6.         $a := \text{abstractState}_c(s)$;
7.         if $|\text{memory}(a)| < k$ then
8.             $\text{segment} := \text{sampleNewSegm}(\mathcal{N}, a.\text{representative})$; // sample new segment
9.             $\text{memory}(a).\text{add(}\text{segment}\text{)}$; // save it for reuse
10.        else
11.            $\text{segment} := \text{chooseUniformlyFrom}(\text{memory}(a))$; // reuse old segment
12.        end
13.        // apply segment’s relative effects
14.        $s := s + \text{segment}.\Delta_{\text{state}}$; $t := t + \text{segment}.\Delta_{\text{time}}$;
15.        $\text{simulation}.\text{add(}((s, t))\text{)}$;
16.    end
17.    $\text{simulations}.\text{add(}\text{simulation}\text{)}$;
18. end
19. return $\text{simulations}$
## More Data - Speed

<table>
<thead>
<tr>
<th>Mod.</th>
<th>SSA</th>
<th>SEG $k=10$</th>
<th>SEG $k=100$</th>
<th>SEG $k=1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$c=2$</td>
<td>$c=1.5$</td>
<td>$c=1.3$</td>
</tr>
<tr>
<td>PP</td>
<td>0.014s</td>
<td>70x</td>
<td>70x</td>
<td>70x</td>
</tr>
<tr>
<td>VI</td>
<td>0.88s</td>
<td>730x</td>
<td>380x</td>
<td>180x</td>
</tr>
<tr>
<td>TS</td>
<td>22s</td>
<td>360x</td>
<td>360x</td>
<td>340x</td>
</tr>
<tr>
<td>RP</td>
<td>9.1s</td>
<td>760x</td>
<td>540x</td>
<td>320x</td>
</tr>
</tbody>
</table>

**Table 1:** Average run-time of one SSA simulation and the speedup factor of segmental simulation when computing 10,000 simulations with different abstraction parameters.
More Data - Memory

<table>
<thead>
<tr>
<th>Mod.</th>
<th>SEG $k=10$</th>
<th>SEG $k=100$</th>
<th>SEG $k=1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$c=2$</td>
<td>$c=1.5$</td>
<td>$c=1.3$</td>
</tr>
<tr>
<td>PP</td>
<td>25kb</td>
<td>61kb</td>
<td>130kb</td>
</tr>
<tr>
<td></td>
<td>250kb</td>
<td>570kb</td>
<td>1.3mb</td>
</tr>
<tr>
<td></td>
<td>2.2mb</td>
<td>4.8mb</td>
<td>11mb</td>
</tr>
<tr>
<td>VI</td>
<td>210kb</td>
<td>730kb</td>
<td>2.0mb</td>
</tr>
<tr>
<td></td>
<td>1.8mb</td>
<td>4.8mb</td>
<td>13mb</td>
</tr>
<tr>
<td></td>
<td>11mb</td>
<td>25mb</td>
<td>53mb</td>
</tr>
<tr>
<td>TS</td>
<td>1.2mb</td>
<td>3.0mb</td>
<td>8.7mb</td>
</tr>
<tr>
<td></td>
<td>15mb</td>
<td>37mb</td>
<td>85mb</td>
</tr>
<tr>
<td></td>
<td>100mb</td>
<td>250mb</td>
<td>550mb</td>
</tr>
<tr>
<td>RP</td>
<td>3.8mb</td>
<td>12mb</td>
<td>34mb</td>
</tr>
<tr>
<td></td>
<td>43mb</td>
<td>120mb</td>
<td>300mb</td>
</tr>
<tr>
<td></td>
<td>310mb</td>
<td>760mb</td>
<td>1.0gb</td>
</tr>
</tbody>
</table>

**Table 2:** Size of segmental abstraction after 10,000 simulations for different parameters.