Imperial College London

TOSME 2021

TauSSA: Simulating Markovian Queueing Networks with Tau Leaping

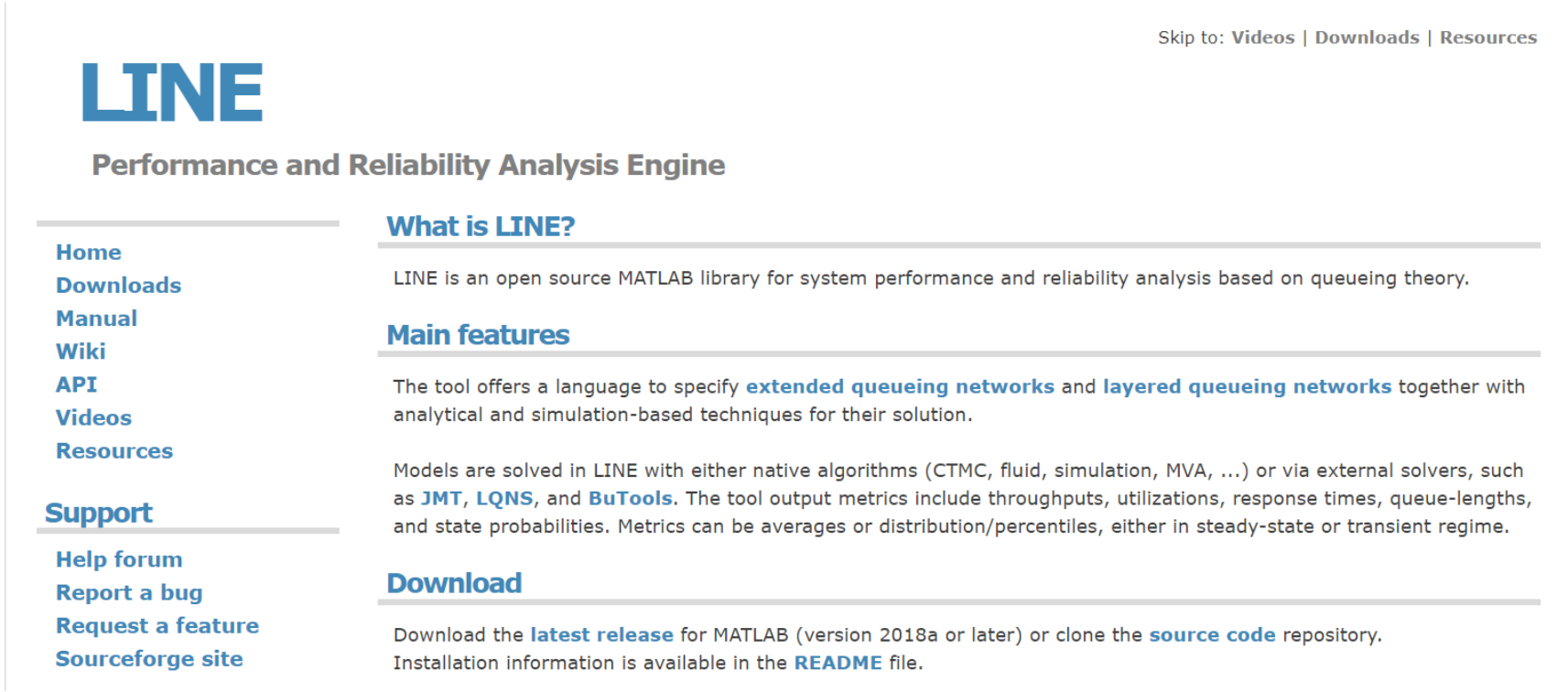
Matthew Sheldon
Department of Computing
Imperial College London
matthew.sheldon20@imperial.ac.uk

Joint work with:
G. Casale

Imperial College London

LINE Solver (line-solver.sf.net)

- MATLAB library for system performance and reliability modelling based on queueing theory
- Ver 2.0: evolution into a multi-paradigm solver



Core features

I. Object-oriented language to model extended and layered queueing networks (EQNs / LQNs)
II. Model specification fully decoupled from analysis
III. Seamless integration with JMT, LQNS, BUTools
IV. Multiple releases
    I. MATLAB source release
    II. Royalty-free binary release (Docker)

Line in numbers:

- 40+ algorithms
- 13 types of analyses
- 13 sched./routing strategies
- 100+ lang. classes
- 14 node types
- 4 metrics

TauSSA

Simulator for Queueing Networks

- Partial Java-based redevelopment of core LINE classes
- Both traditional SSA (Gillespie's Algorithm) and Tau Leaping
- Markovian phase-type distributions and Markov Arrival Processes
- Up to 100x the performance of SolverSSA in LINE
  - 25%-50% more savings with Tau Leaping

https://github.com/imperial-qore/line-solver-java

TauSSA

```
Metrics Summary
-----------------------------------

Node         Class       Q       U       RpT     RsT     T
Delay        Closed1     2.45035 0.00000 2.19462 2.18926 1.22912
Delay        Closed2     5.94069 0.00000 1.98377 1.97954 4.21884
Queue        Closed1     0.54935 0.16192 0.51336 0.51294 1.22837
Queue        Closed2     2.05851 0.61216 0.71094 0.71043 4.21837
Total solving time: 48
Total time: 213
```

Motivation

- Queueing networks are often tractable with analytic methods..
  - But analytic methods often introduce many assumptions
- More general classes of models can be studied with simulation
- Need for accurate and fast simulations.

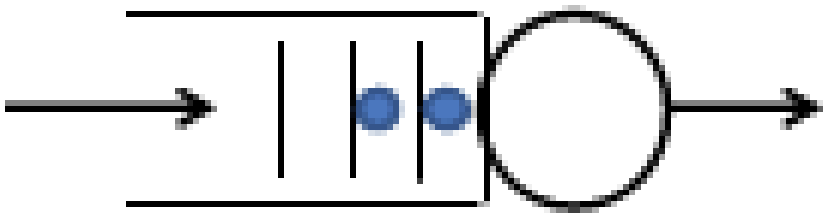
Tau Leaping

- Introduced by Gillespie, 2001
- Created to avoid expensive rate calculations in simulated chemical reactions
- Idea: find all events in interval (t, t+τ) and simulate simultaneously
- Question: can this be used to accelerate queueing simulations?

Tau Leaping

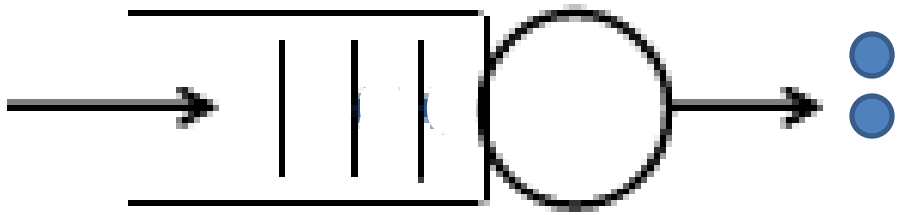
1. The system begins in state x_0 at time 0. Select a step size $\tau > 0$.
2. The rate $a_j(x)$, for each possible transition j out of state x, is calculated.
3. Generate a Poisson random variable $n_j^\tau \sim \text{Poisson}\,(a_j(x), \tau)$ for each transition j.
4. Update t to $t + \tau$, and x based on:
    1. The number of repetitions n_j^τ,
    2. The nature of the transition,
    3. The state strategy (more on this later)
5. Return to step 2

Ordering Strategies



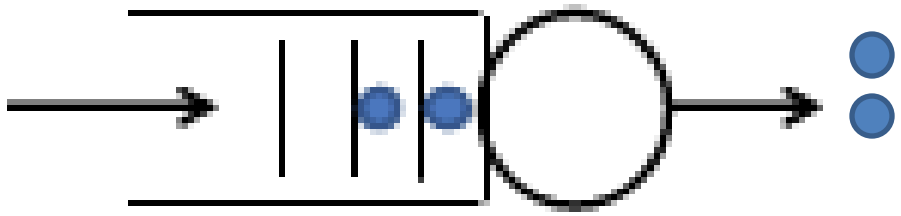
Consider a queue with **2** jobs. The tau leaping algorithm generates:
- **2** arrivals
- **3** departures

Ordering Strategies



Consider a queue with **2** jobs. The tau leaping algorithm generates:
- **2** arrivals
- **3** departures

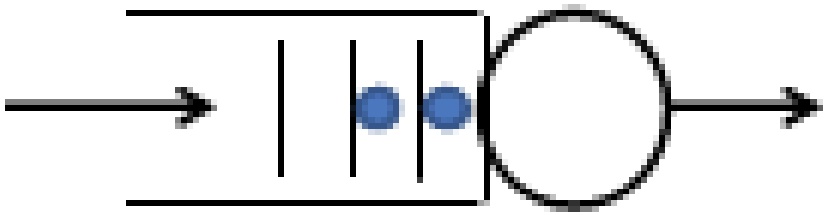
Ordering Strategies



Consider a queue with **2** jobs. The tau leaping algorithm generates:
- **2** arrivals
- **3** departures

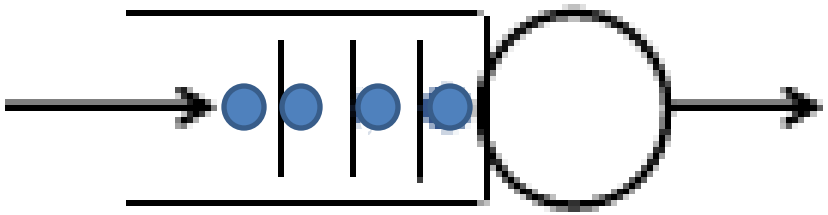
If departures are processed before arrivals, then only 2 of the three jobs can leave. Afterwards, another 2 jobs arrive. The ending state is **x=2.**

Ordering Strategies



Consider a queue with **2** jobs. The tau leaping algorithm generates:
- **2** arrivals
- **3** departures

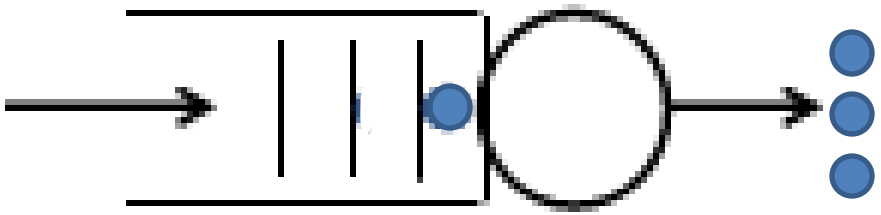
Ordering Strategies



Consider a queue with **2** jobs. The tau leaping algorithm generates:
- **2** arrivals
- **3** departures

However, if arrivals are processed first, then 4 jobs will be available for departure. All 3 available jobs are processed, and the ending state is **x=1**.

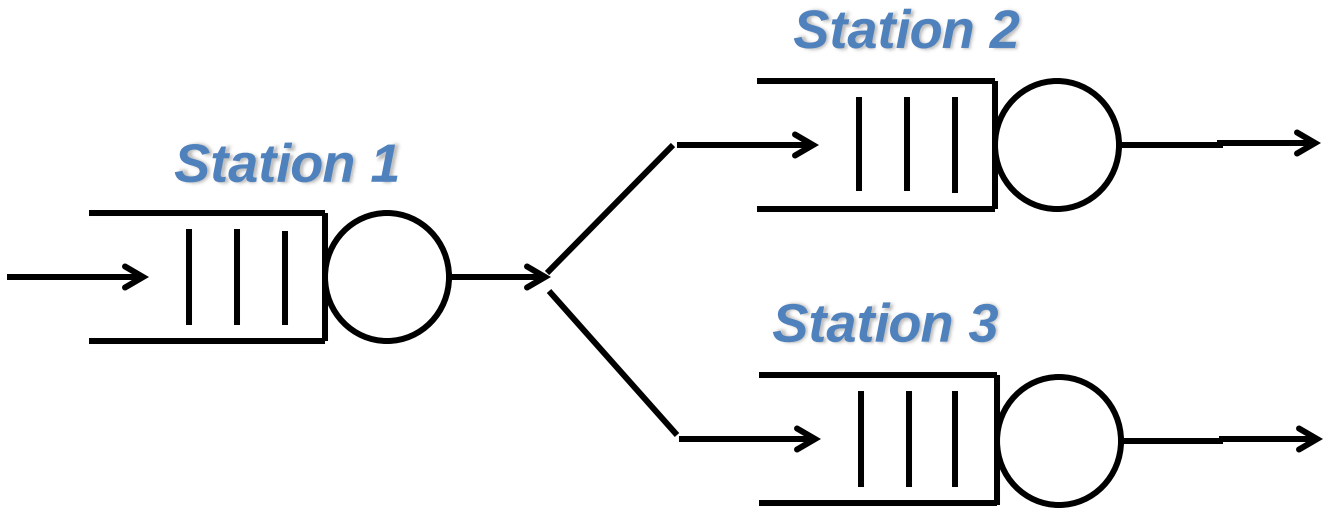
Ordering Strategies



Consider a queue with **2** jobs. The tau leaping algorithm generates:
- **2** arrivals
- **3** departures

However, if arrivals are processed first, then 4 jobs will be available for departure. All 3 available jobs are processed, and the ending state is **x=1**.

Ordering Strategies



How should the simulator order the departures from each station?

Ordering Strategies - RandomEvent

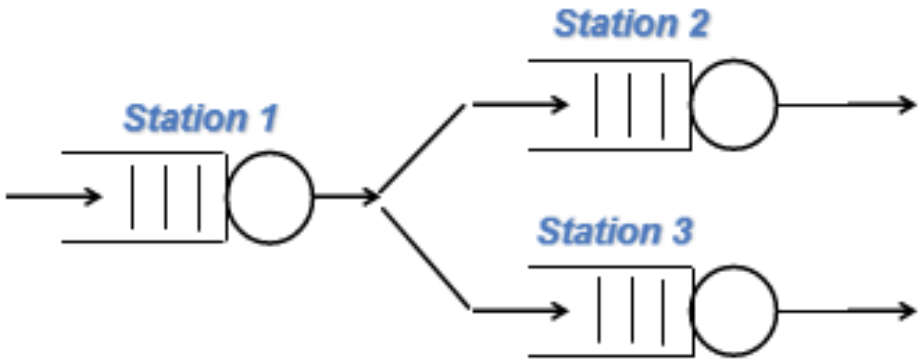
- Shuffle the event list at each iteration.
- (+) Truest to standard simulation principles
- (+) All events will be evenly selected
- (-) Possibly slower than other methods

Ordering Strategies - RandomEventFixed

- Shuffle the event list at start of simulation.
- (+) May be faster than RandomEvent with models with large numbers of nodes
- (+) Suitable for experiments with high number of runs
- (-) Bias in individual runs due to constant event order

Ordering Strategies - DirectedGraph

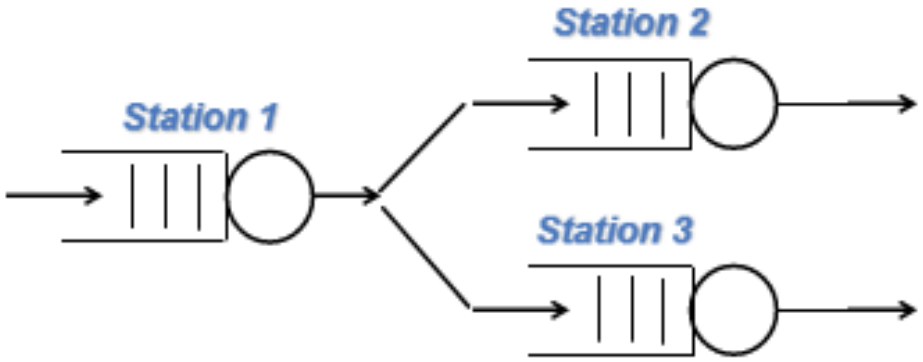
- Find a topological sorting of nodes (Kahn's algorithm), and apply events in this order.
- (+) Sensitive to the topology of the network
- (-) Creates upward bias in queue length
- (-) Potentially unsuitable for models with cycles and capacities



(1,2,3) or (1,3,2)

Ordering Strategies - DirectedCycle

- Find a topological ordering of events, and move the first event to the end at each iteration.
- (+) Sensitive to the topology of the network
- (+) Best-performing on experimental evaluation
- (-) Not sensitive to the rates of the system



First iteration: (1,2,3) or (1,3,2)
Second iteration: (2,3,1) or (3,2,1)
Third iteration: (3,1,2) or (2,1,3)

Handling Invalid States

- Tau Leaping will often request more departures than there are jobs at a given station...
- …or more arrivals than the capacity will allow.
- We propose several methods to handle these events

State Strategies - Cutoff

- After applying event, set x = max(0, min(x + d, c))
  - d: requested state change (+/-)
- (+) Simple to implement, obvious solution
- (-) Highly sensitive to event ordering

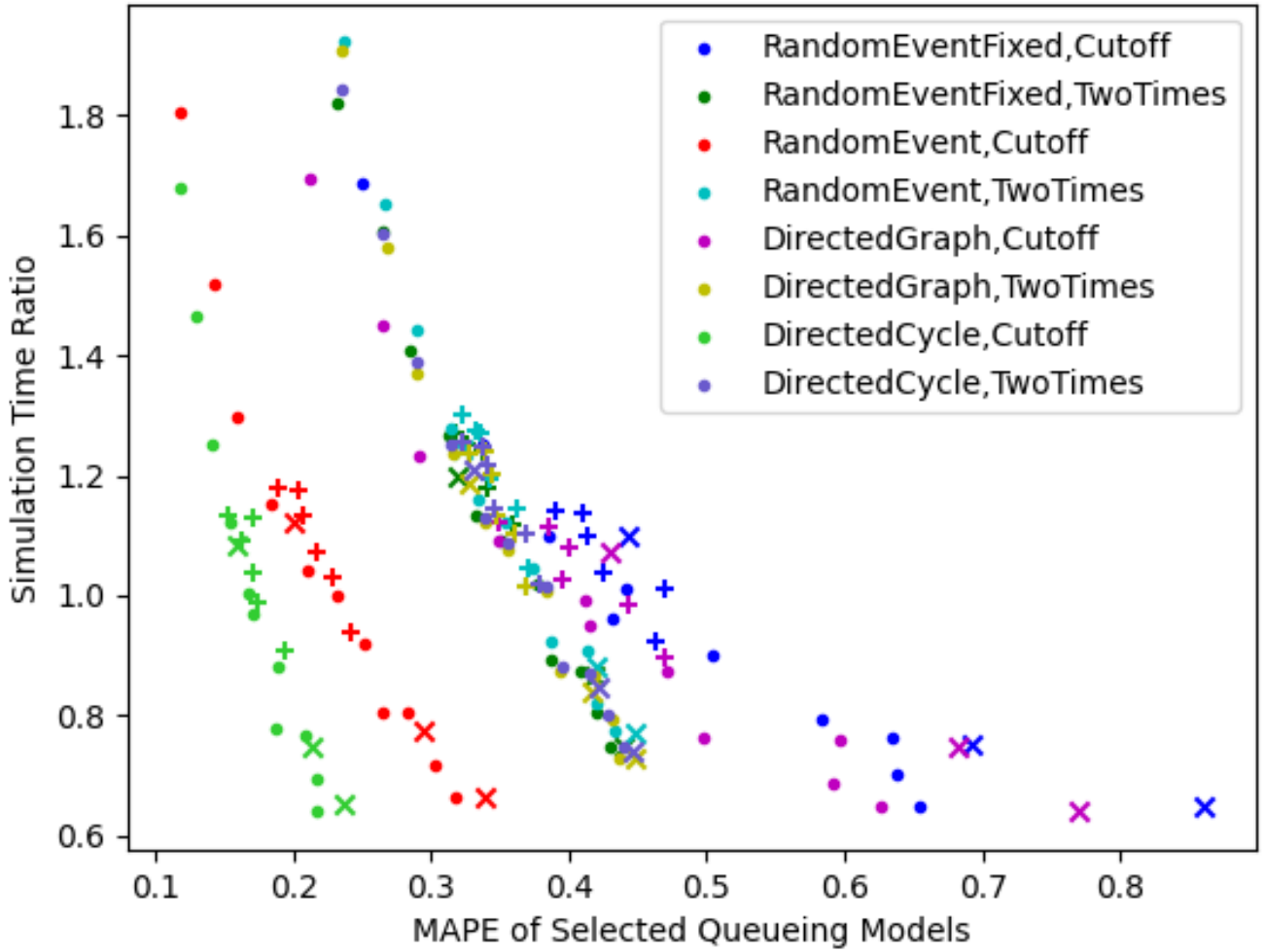
State Strategies - TwoTimes

- Pass through event list twice:
  - First pass: Apply max repetitions m that don't generate an illegal state, and update.
  - Second pass: Apply remaining repetitions n-m, and set the state accordingly with Cutoff.
- (+) Allows a closer approximation of n_j^τ events
- (+) Less sensitive to the event order
- (-) Possibly slower

Experimental Evaluation

- We analyze 500 randomly generated models
  - 5 M/M/1 Queues
  - 1 Open Class
  - Random Topologies
  - Event rates between 1 and 50
- Accuracy is measured by the MAPE of the average Q length.

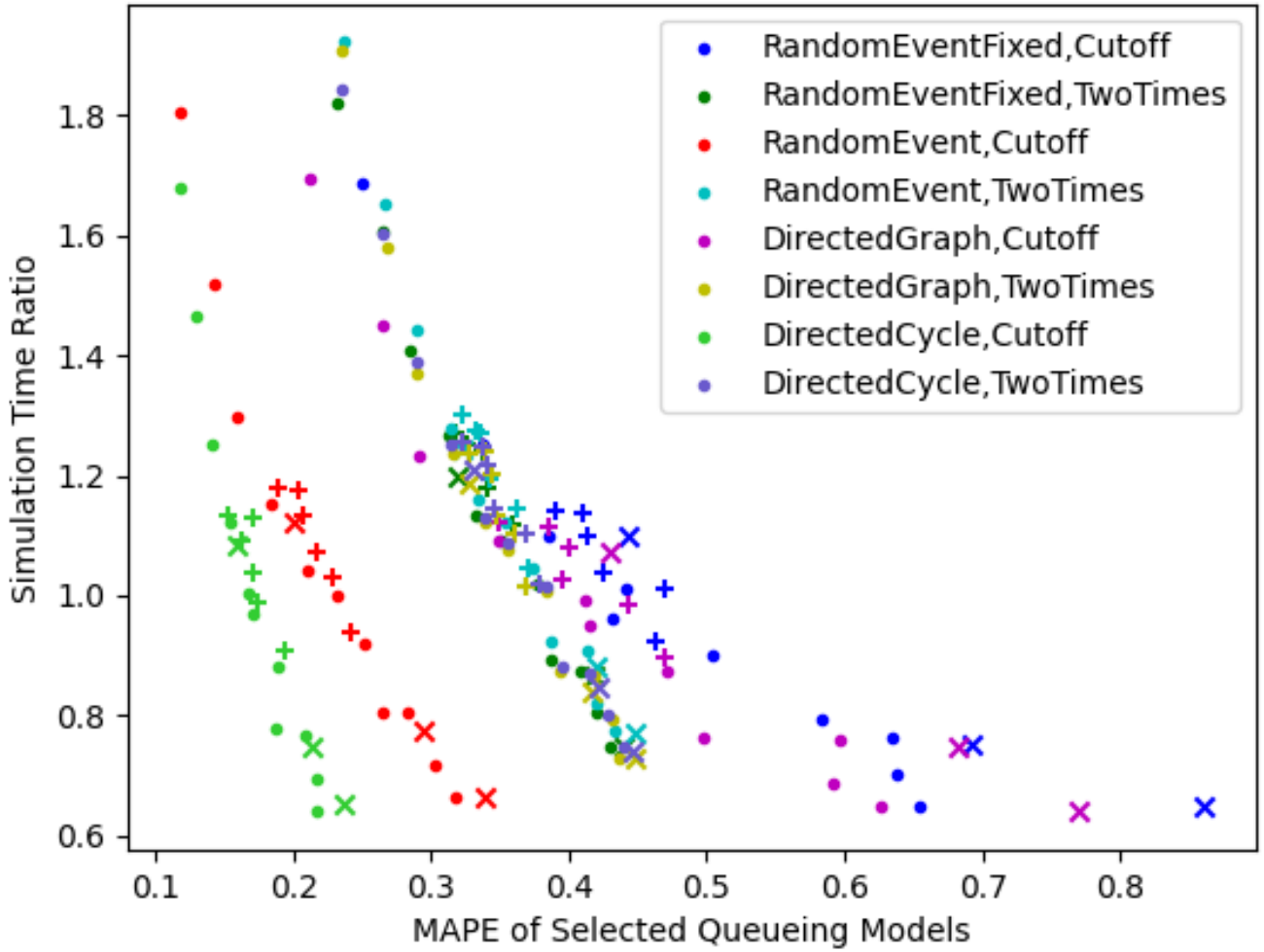
Experimental Evaluation



Tau Methods:
+: $\tau = k/\max(a_j)$, $k \in \{2.0, 2.1..2.5\}$
o: $\tau = k/\text{avg}(a_j)$, $k \in \{0.5, 0.6..1.5\}$
X: $\tau = k/\min(a_j)$, $k \in \{0.1, 0.15..0.2\}$

Experimental Evaluation

[DirectedCycle, Cutoff] dominates all other configurations.

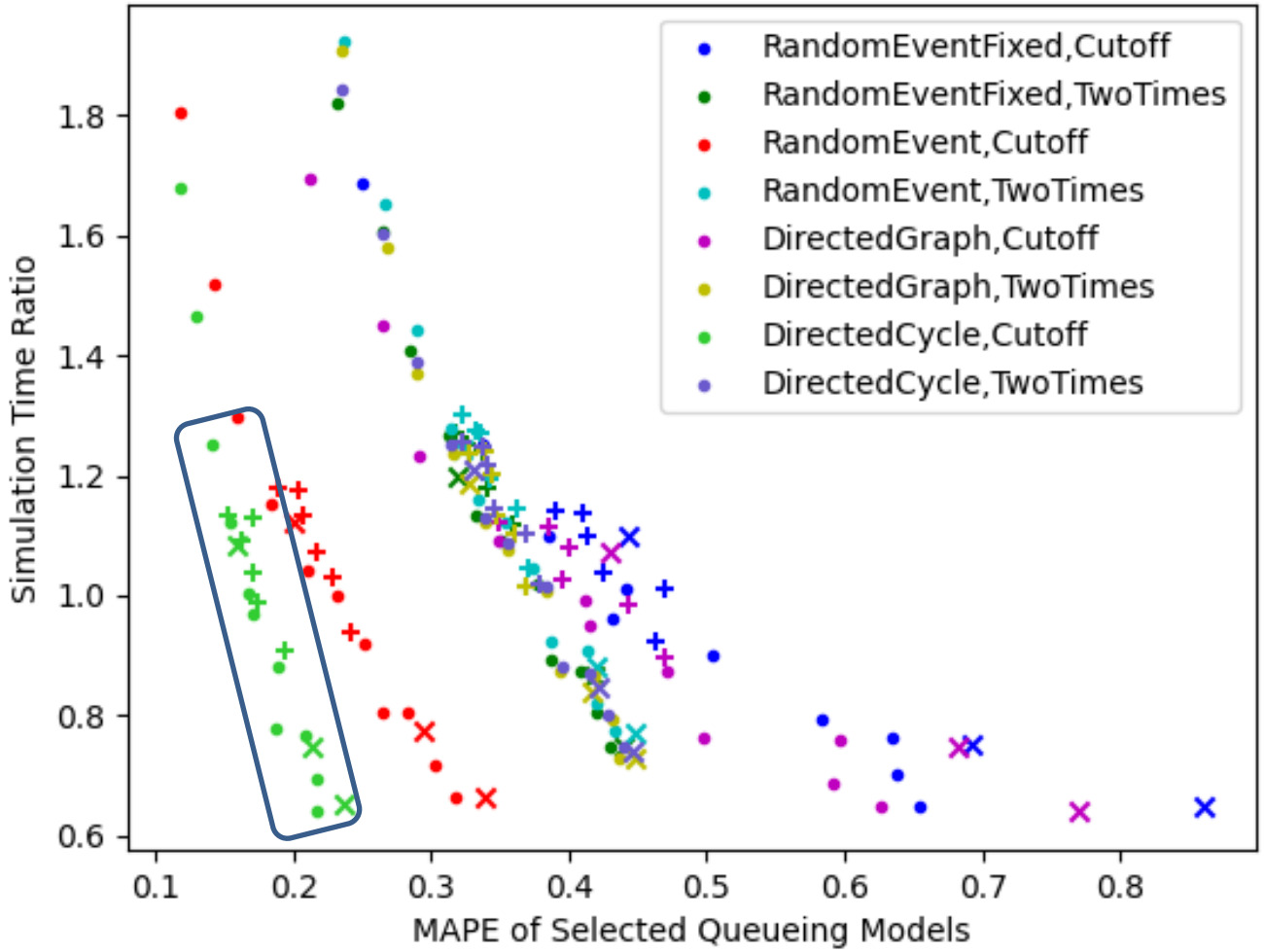


Tau Methods:
+: $\tau = k/\max(a_j)$, $k \in \{2.0, 2.1..2.5\}$
o: $\tau = k/\text{avg}(a_j)$, $k \in \{0.5, 0.6..1.5\}$
X: $\tau = k/\min(a_j)$, $k \in \{0.1, 0.15..0.2\}$

No demonstrated difference between tau calculation methods.

Experimental Evaluation

[DirectedCycle, Cutoff] dominates all other configurations.



Tau Methods:
+: $\tau = k/\max(a_j)$, $k \in \{2.0, 2.1..2.5\}$
o: $\tau = k/\text{avg}(a_j)$, $k \in \{0.5, 0.6..1.5\}$
X: $\tau = k/\min(a_j)$, $k \in \{0.1, 0.15..0.2\}$

No demonstrated difference between tau calculation methods.

Conclusion

- We have proposed TauSSA, a queueing simulator based on SSA and Tau Leaping
- Tau Leaping with TauSSA can significantly accelerate queueing simulations with attention to event ordering and τ, and accuracy tradeoffs.
- Future work
  - Automated methods to determine τ.
  - Investigation of other nodes, processing regimes, and distributions with tau leaping.

https://github.com/imperial-qore/line-solver-java